EPICS Documentation Sandbox

Timo Korhonen

Mar 06, 2024

RECORD REFERENCE:

CHAPTER

ONE

EPICS RECORD REFERENCE MANUAL

1.1 EPICS Process Database Concepts

Table of Contents

1.1.1 The EPICS Process Database

An EPICS-based control system contains one or more Input Output Controllers, IOCs. Each IOC loads one or more databases. A database is a collection of records of various types.

A Record is an object with:

- A unique name
- A behavior defined by its type
- Controllable properties (fields)
- Optional associated hardware I/O (device support)
- Links to other records

There are several different types of records available. In addition to the record types that are included in the EPICS base software package, it is possible (although not recommended unless you absolutely need) to create your own record type to perform some specific tasks.

Each record comprises a number of **fields**. Fields can have different functions, typically they are used to configure how the record operates, or to store data items.

Below are short descriptions for the most commonly used record types:

Analog Input and Output (AI and AO) records can store an analog value, and are typically used for things like set-points, temperatures, pressure, flow rates, etc. The records perform number of functions like data conversions, alarm processing, filtering, etc.

Binary Input and Output (BI and BO) records are generally used for commands and statuses to and from equipment. As the name indicates, they store binary values like On/Off, Open/Closed and so on.

Calc and Calcout records can access other records and perform a calculation based on their values. (E.g. calculate the efficiency of a motor by a function of the current and voltage input and output, and converting to a percentage for the operator to read).

1.1.2 Database Functionality Specification

This chapter covers the general functionality that is found in all database records. The topics covered are I/O scanning, I/O address specification, data conversions, alarms, database monitoring, and continuous control:

- *Scanning Specification* describes the various conditions under which a record is processed.
- *Address Specification* explains the source of inputs and the destination of outputs.
- *Conversion Specification* covers data conversions from transducer interfaces to engineering units.
- *Alarm Specification* presents the many alarm detection mechanisms available in the database.
- *Monitor Specification* details the mechanism, which notifies operators about database value changes.
- *Control Specification* explains the features available for achieving continuous control in the database.

These concepts are essential in order to understand how the database interfaces with the process.

The EPICS databases can be created by manual creation of a database "myDatabase.db" text file or using visual tools (VDCT, CapFast). Visual Database Configuration Tool (VDCT), a java application from Cosylab, is a tool for database creation/editing that runs on Linux, Windows, and Sun. The illustrations in this document have been created with VDCT.

1.1.3 Scanning Specification

Scanning determines when a record is processed. A record is *processed* when it performs any actions related to its data. For example, when an output record is processed, it fetches the value which it is to output, converts the value, and then writes that value to the specified location. Each record must specify the scanning method that determines when it will be processed. There are three scanning methods for database records:

(1) periodic,

(2) event, and

(3) passive.

Periodic scanning occurs on set time intervals.

Event scanning occurs on either an I/O interrupt event or a user-defined event.

Passive scanning occurs when the records linked to the passive record are scanned, or when a value is "put" into a passive record through the database access routines.

For periodic or event scanning, the user can also control the order in which a set of records is processed by using the PHASE mechanism. The number in the

PHAS field allows to define the relative order in which records are processed within a scan cycle:

- Records with PHAS=0 are processed first
- Then those with PHAS=1, PHAS=2, etc.

For event scanning, the user can control the priority at which a record will process. The PRIO field selects Low/Medium/High priority for Soft event and I/O Interrupts.

In addition to the scan and the phase mechanisms, there are data links and forward processing links that can be used to cause processing in other records.

Periodic Scanning

The periodic scan tasks run as close as possible to the specified frequency. When each periodic scan task starts, it calls the gettime routine, then processes all of the records on this period. After the processing, gettime is called again and this thread sleeps the difference between the scan period and the time to process the records. For example, if it takes 100 milliseconds to process all records with "1 second" scan period, then the 1 second scan period will start again 900 milliseconds after completion. The following periods for scanning database records are available by default, though EPICS can be configured to recognize more scan periods:

- 10 second
- 5 second
- 2 second
- 1 second
- .5 second
- .2 second
- .1 second

The period that best fits the nature of the signal should be specified. A five-second interval is adequate for the temperature of a mass of water because it does not change rapidly. However, some power levels may change very rapidly, so they need to be scanned every 0.5 seconds. In the case of a continuous control loop, where the process variable being controlled can change quickly, the 0.1 second interval may be the best choice.

For a record to scan periodically, a valid choice must be entered in its SCAN field. Actually, the available choices depend on the configuration of the menuScan.dbd file. As with most other fields which consists of a menu of choices, the choices available for the SCAN field can be changed by editing the appropriate .dbd (database definition) file. dbd files are ASCII files that are used to generate header files that are, in turn, are used to compile the database code. Many dbd files can be used to configure other things besides the choices of menu fields.

Here is an example of a menuScan.dbd file, which has the default menu choices for all periods listed above as well as choices for event scanning, passive scanning, and I/O interrupt scanning:

```
menu(menuScan) {
  choice(menuScanPassive,"Passive")
  choice(menuScanEvent,"Event")
  choice(menuScanI_O_Intr,"I/O Intr")
  choice(menuScan10_second,"10 second")
  choice(menuScan5_second,"5 second")
  choice(menuScan2_second,"2 second")
  choice(menuScan1_second,"1 second")
  choice(menuScan_5_second,".5 second")
  choice(menuScan_2_second,".2 second")
  choice(menuScan_1_second,".1 second")
}
```

The first three choices must appear first and in the order shown. The remaining definitions are for the periodic scan rates, which must appear in the order slowest to fastest (the order directly controls the thread priority assigned to the particular scan rate, and faster scan rates should be assigned higher thread priorities). At IOC initialization, the menu choice strings are read at scan initialization. The number of periodic scan rates and the period of each rate is determined from the menu choice strings. Thus the periodic scan rates can be changed by changing menuScan.dbd and loading this version via dbLoadDatabase. The only requirement is that each periodic choice string must begin with a number and be followed by any of the following unit strings:

- second or seconds

- minute or minutes
- hour or hours
- Hz or Hertz

For example, to add a choice for 0.015 seconds, add the following line after the 0.1 second choice:

```
choice(menuScan_015_second, " .015 second")
```

The range of values for scan periods can be from one clock tick to the maximum number of ticks available on the system (for example, vxWorks out of the box supports 0.015 seconds or a maximum frequency of 60 Hz). Note, however, that the order of the choices is essential. The first three choices must appear in the above order. Then the remaining choices should follow in descending order, the biggest time period first and the smallest last.

Event Scanning

There are two types of events supported in the input/output controller (IOC) database, the I/O interrupt event and the user-defined event. For each type of event, the user can specify the scheduling priority of the event using the PRIO or priority field. The scheduling priority refers to the priority the event has on the stack relative to other running tasks. There are three possible choices: LOW, MEDIUM, or HIGH. A low priority event has a priority a little higher than Channel Access. A medium priority event has a priority about equal to the median of periodic scanning tasks. A high priority event has a priority equal to the event scanning task.

I/O Interrupt Events

Scanning on I/O interrupt causes a record to be processed when a driver posts an I/O Event. In many cases these events are posted in the interrupt service routine. For example, if an analog input record gets its value from an I/O card and it specifies I/O interrupt as its scanning routine, then the record will be processed each time the card generates an interrupt (not all types of I/O cards can generate interrupts). Note that even though some cards cannot actually generate interrupts, some driver support modules can simulate interrupts. In order for a record to scan on I/O interrupts, its SCAN field must specify I/O Intr.

User-defined Events

The user-defined event mechanism processes records that are meaningful only under specific circumstances. User-defined events can be generated by the post_event() database access routine. Two records, the event record and the timer record, are also used to post events. For example, there is the timing output, generated when the process is in a state where a control can be safely changed. Timing outputs are controlled through Timer records, which have the ability to generate interrupts. Consider a case where the timer record is scanned on I/O interrupt and the timer record's event field (EVNT) contains an event number. When the record is scanned, the user-defined event will be posted. When the event is posted, all records will be processed whose SCAN field specifies event and whose event number is the same as the generated event. User-defined events can also be generated through software. Event numbers are configurable and should be controlled through the project engineer. They only need to be unique per IOC because they only trigger processing for records in the same IOC.

All records that use the user-defined event mechanism must specify Event in their SCAN field and an event number in their EVNT field.

Passive Scanning

Passive records are processed when they are referenced by other records through their link fields or when a channel access put is done to them.

Channel Access Puts to Passive Scanned Records

In this case where a channel access put is done to a record, the field being written has an attribute that determines if this put causes record processing. In the case of all records, putting to the VAL field causes record processing. Consider a binary output that has a SCAN of Passive. If an operator display has a button on the VAL field, every time the button is pressed, a channel access put is sent to the record. When the VAL field is written, the Passive record is processed and the specified device support is called to write the newly converted RVAL to the device specified in the OUT field through the device support specified by DTYP. Fields determined to change the way a record behaves, typical cause the record to process. Another field that would cause the binary output to process would be the ZSV; which is the alarm severity if the binary output record is in state Zero (0). If the record was in state 0 and the severity of being in that state changed from No Alarm to Minor Alarm, the only way to catch this on a SCAN Passive record is to process it. Fields are configured to cause binary output records to process in the bo.dbd file. The ZSV severity is configured as follows:

```
field(ZSV,DBF_MENU) {
  prompt("Zero Error Severity")
  promptgroup(GUI_ALARMS)
  pp(TRUE)
  interest(1)
  menu(menuAlarmSevr)
}
```

where the line "pp(TRUE)" is the indication that this record is processed when a channel access put is done.

Database Links to Passive Record

The records in the process database use link fields to configure data passing and scheduling (or processing). These fields are either INLINK, OUTLINK, or FWDLINK fields.

Forward Links

In the database definition file (.dbd) these fields are defined as follows:

```
field(FLNK,DBF_FWDLINK) {
  prompt("Forward Process Link")
  promptgroup(GUI_LINKS)
  interest(1)
}
```

If the record that is referenced by the FLNK field has a SCAN field set to "Passive", then the record is processed after the record with the FLNK. The FLNK field only causes record processing, no data is passed. In (*Figure 1*), three records are shown. The ai record "Input_2" is processed periodically. At each interval, Input_2 is processed. After Input_2 has read the new input, converted it to engineering units, checked the alarm condition, and posted monitors to Channel Access, then the calc record "Calculation_2" is processed. Calculation_2 reads the input, performs the calculation, checked the alarm condition, and posted monitors to Channel Access, then the ao record "Output_2" is processed. Output_2 reads the desired output, rate limits it, clamps the range, calls the device support for the OUT field, checks alarms, posts monitors and then is complete.

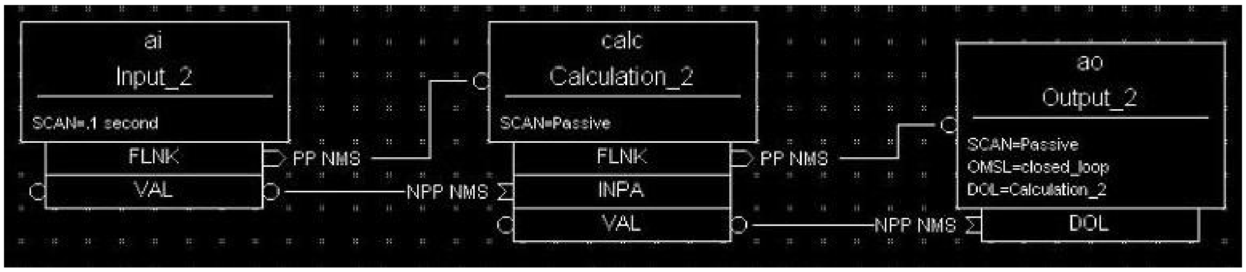


Figure 1. Input Links

Input links normally fetch data from one field into a field in the referring record. For instance, if the INPA field of a CALC record is set to Input_3.VAL, then the VAL field is fetched from the Input_3 record and placed in the A field of the CALC record. These data links have an attribute that specify if a passive record should be processed before the value is returned. The default for this attribute is NPP (no process passive). In this case, the record takes the VAL field and returns it. If they are set to PP (process passive), then the record is processed before the field is returned.

In *Figure 2*), the PP attribute is used. In this example, Output_3 is processed periodically. Record processing first fetching the DOL field. As the DOL field has the PP attribute set, before the VAL field of Calc_3 is returned, the record is processed. The first thing done by the ai record Input_3 does is to read the input. It then converts the RVAL field to engineering units and places this in the VAL field, checks alarms, posts monitors, and then returns. The calc record then fetches the VAL field field from Input_3, places it in the A field, computes the calculation, checks alarms, posts monitors, the returns. The ao record, Output_3, then fetches the VAL field from the CALC record, applies rate of change and limits, write the new value, checks alarms, posts monitors and completes.

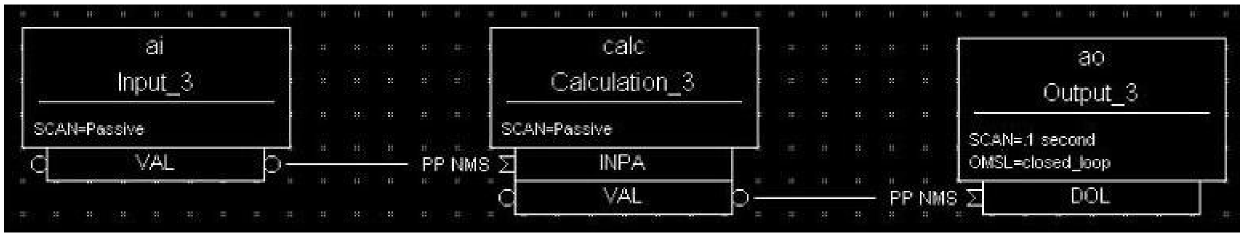


Figure 2

In *Figure 3*) the PP/NPP attribute is used to calculate a rate of change. At 1 Hz, the calculation record is processed. It fetches the inputs for the calc record in order. As INPA has an attribute of NPP, the VAL field is taken from the ai record. Before INPB takes the VAL field from the ai record it is processed, as the attribute on this link is PP. The new ai value is placed in the B field of the calc record. A-B is the VAL field of the ai one second ago and the current VAL field.

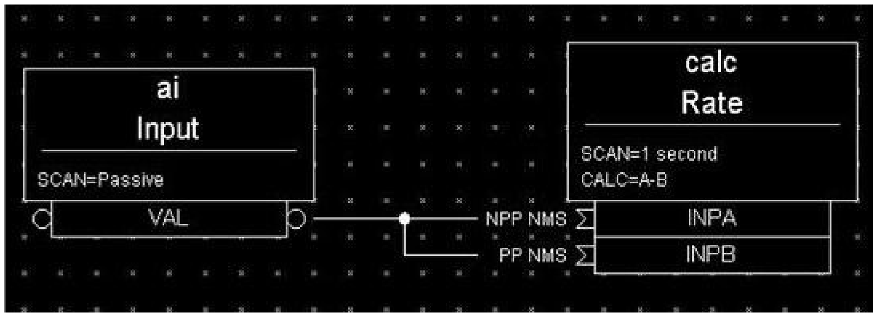


Figure 3

Process Chains

Links can be used to create complex scanning logic. In the forward link example above, the chain of records is determined by the scan rate of the input record. In the PP example, the scan rate of the chain is determined by the rate of the output. Either of these may be appropriate depending on the hardware and process limitations.

Care must be taken as this flexibility can also lead to some incorrect configurations. In these next examples we look at some mistakes that can occur.

In *Figure 4*) two records that are scanned at 10 Hz make references to the same Passive record. In this case, no alarm or error is generated. The Passive record is scanned twice at 10 Hz. The time between the two scans depends on what records are processed between the two periodic records.

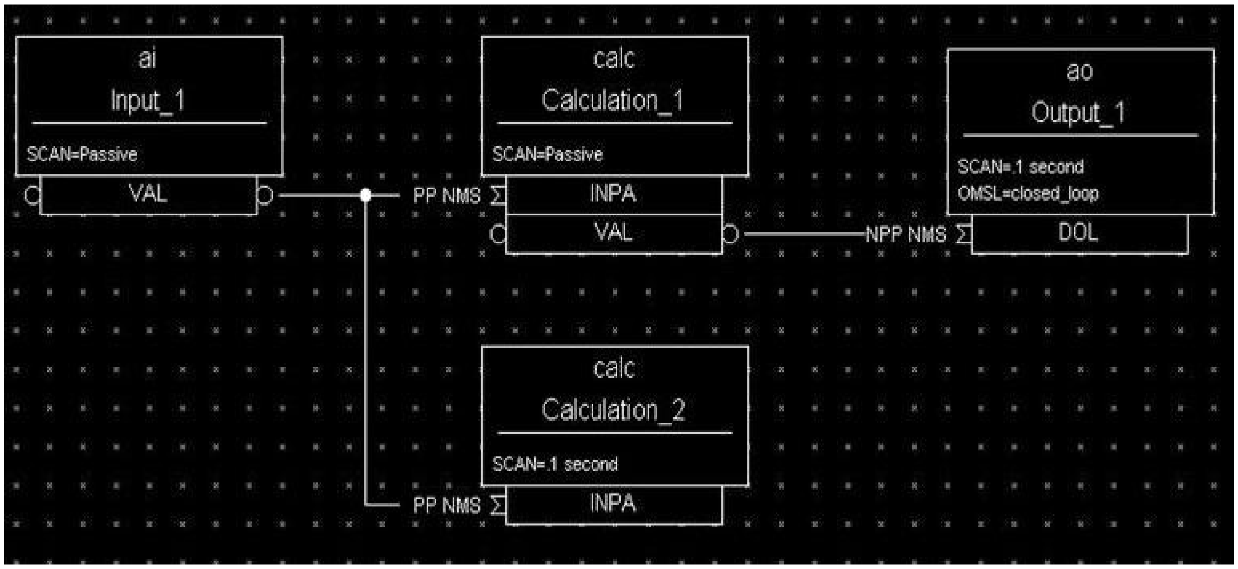


Figure 4

In *Figure 5*), several circular references are made. As the record processing is recursively called for links, the record containing the link is marked as active during the entire time that the chain is being processed. When one of these circular references is encountered, the active flag is recognized and the request to process the record is ignored.

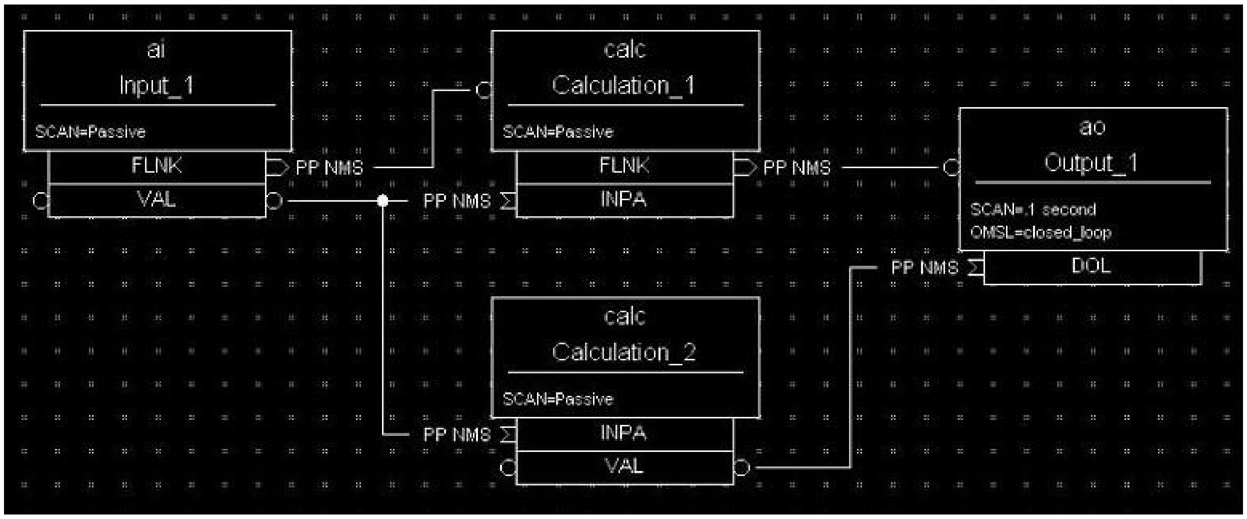


Figure 5

Channel Access Links

A Channel Access link is an input link or output link that specifies a link to a record located in another IOC or an input and output link with one of the following attributes: CA, CP, or CPP.

Channel Access Input Links

If the input link specifies CA, CP, or CPP, regardless of the location of the process variable being referenced, it will be forced to be a Channel Access link. This is helpful for separating process chains that are not tightly related. If the input link specifies CP, it also causes the record containing the input link to process whenever a monitor is posted, no matter what the record's SCAN field specifies. If the input link specifies CPP, it causes the record to be processed if and only if the record with the CPP link has a SCAN field set to Passive. In other words, CP and CPP cause the record containing the link to be processed with the process variable that they reference changes.

Channel Access Output Links

Only CA is appropriate for an output link. The write to a field over channel access causes processing as specified in *Channel Access Puts to Passive Scanned Records*.

Channel Access Forward Links

Forward links can also be Channel Access links, either when they specify a record located in another IOC or when they specify the CA attributes. However, forward links will only be made Channel Access links if they specify the PROC field of another record.

Maximize Severity Attribute

The Maximize Severity attribute is one of the following :

- NMS (Non-Maximize Severity)
- MS (Maximize Severity)
- MSS (Maximize Status and Severity)
- MSI (Maximize Severity if Invalid)

It determines whether alarm severity is propagated across links. If the attribute is MSI only a severity of INVALID_ALARM is propagated; settings of MS or MSS propagate all alarms that are more severe than the record's current severity. For input links the alarm severity of the record referred to by the link is propagated to the record containing the link. For output links the alarm severity of the record containing the link is propagated to the record referred to by the link. If the severity is changed the associated alarm status is set to LINK_ALARM, except if the attribute is MSS when the alarm status will be copied along with the severity.

The method of determining if the alarm status and severity should be changed is called ``maximize severity". In addition to its actual status and severity, each record also has a new status and severity. The new status and severity are initially 0, which means NO_ALARM. Every time a software component wants to modify the status and severity, it first checks the new severity and only makes a change if the severity it wants to set is greater than the current new severity. If it does make a change, it changes the new status and new severity, not the current status and severity. When database monitors are checked, which is normally done by a record processing routine, the current status and severity are set equal to the new values and the new values reset to zero. The end result is that the current alarm status and severity reflect the highest severity outstanding alarm. If multiple alarms of the same severity are present the alarm status reflects the first one detected.

Phase

The PHAS field is used to order the processing of records that are scanned at the same time, i.e., records that are scanned periodically at the same interval and priority, or that are scanned on the same event. In this manner records dependent upon other records can be assured of using current data.

To illustrate this we will look at an example from the previous section, with the records, however, being scanned periodically instead of passively (*Figure 6*). In this example each of these records specifies .1 second; thus, the records are synchronous. The phase sequence is used to assure that the analog input is processed first, meaning that it fetches its value from the specified location and places it in the VAL field (after any conversions). Next, the calc record will be processed, retrieving its value from the analog input and performing its calculation. Lastly, the analog output will be processed, retrieving its desired output value from the calc record's VAL field (the VAL field contains the result of the calc record's calculations) and writing that value to the location specified it its OUT link. In order for this to occur, the PHAS field of the analog input record must specify 0, the PHAS field of the calculation record must specify 1, and the analog output's PHAS field must specify 2.

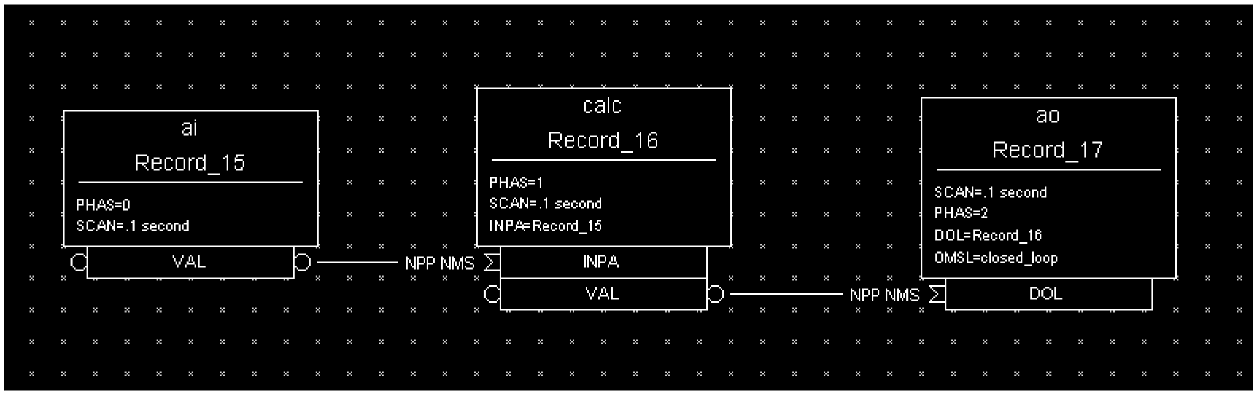


Figure 6

It is important to understand that in the above example, no record causes another to be processed. The phase mechanism instead causes each to process in sequence.

PVAccess Links

When built against Base >= 3.16.1, support is enabled for PVAccess links, which are analogous to Channel Access (CA) links. However, the syntax for PVA links is quite different.

The authoritative documentation is available in the git repository, pva2pva.

Note

The "dbjlr" and "dbpvar" IOC shell command provide information about PVA links in a running IOC.

A simple configuration using defaults is

```
record(longin, "tgt") {}
record(longin, "src") {
  field(INP, {pva:"tgt"})
}
```

This is a shorthand for

```
record(longin, "tgt") {}
record(longin, "src") {
```

(continues on next page)

(continued from previous page)

```
    field(INP, {pva:{pv:"tgt"}})
}
```

Some additional keys (beyond "pv") may be used. Defaults are shown in the example below:

```
record(longin, "tgt") {}
record(longin, "src") {
  field(INP, {pva:{
    pv:"tgt",
    field:"", # may be a sub-field
    local:false,# Require local PV
    Q:4, # monitor queue depth
    pipeline:false, # require that server uses monitor
    # flow control protocol
    proc:none, # Request record processing
    #(side-effects).
    sevr:false, # Maximize severity.
    time:false, # set record time during getValue
    monorder:0, # Order of record processing as a result #of CP and CPP
    retry:false,# allow Put while disconnected.
    always:false,# CP/CPP input link process even when # .value field hasn't changed
    defer:false # Defer put
  }})
}
```

pv: Target PV name

The PV name to search for. This is the same name which could be used with 'pvget' or other client tools.

field: Structure field name

The name of a sub-field of the remotely provided Structure. By default, an empty string "" uses the top-level Structure.

If the top level structure, or a sub-structure is selected, then it is expeccted to conform to NTScalar, NTScalarArray, or NTEnum to extract value and meta-data.

If the sub-field is an PVScalar or PVScalarArray, then a value will be taken from it, but not meta-data will be available.

local: Require local PV

When true, link will not connect unless the named PV is provided by the local (QSRV) data provider.

Q: Monitor queue depth

Requests a certain monitor queue depth. The server may, or may not, take this into consideration when selecting a queue depth.

pipeline: Monitor flow control

Expect that the server supports PVA monitor flow control. If not, then the subscription will stall (ick.)

proc: Request record processing (side-effects)

The meaning of this option depends on the direction of the link.

For output links, this option allows a request for remote processing (side-effects).

- none (default) - Make no special request. Uses a server specific default.
- false, “NPP” - Request to skip processing.
- true, “PP” - Request to force processing.
- “CP”, “CPP” - For output links, an alias for “PP”.

For input links, this option controls whether the record containing the PVA link will be processed when subscription events are received.

- none (default), false, “NPP” - Do not process on subscription updates.
- true, “CP” - Always process on subscription updates.
- “PP”, “CPP” - Process on subscription updates if SCAN=Passive

sevr: Alarm propagation

This option controls whether reading a value from an input PVA link has the addition effect of propagating any alarm via the Maximize Severity process.

- false - Do not maximize severity.
- true - Maximize alarm severity
- “MSI” - Maximize only if the remote severity is INVALID.

time: Time propagation

Somewhat analogous to sevr: applied to timestamp. When true, the record TIME field is updated when the link value is read.

Warning

> TSEL must be set to -2 for time:true to have an effect.

monorder: Monitor processing order

When multiple record target the same target PV, and request processing on subscription updates. This option allows the order of processing to be specified.

Record are processed in increasing order. monorder=-1 is processed before monorder=0. Both are processed before monorder=1.

defer: Defer put

By default (defer=false) an output link will immediately start a PVA Put operation. defer=true will store the new value in an internal cache, but not start a PVA Put.

This option, in combination with field: allows a single Put to contain updates to multiple sub-fields.

retry: Put while disconnected

Allow a Put operation to be queued while the link is disconnected. The Put will be executed when the link becomes connected.

always: CP/CPP always process

By default (always:false) a subscription update will only cause a CP input link to scan if the structure field (cf. field: option) is marked as changed. Set to true to override this, and always process the link.

Link semantics/behavior

This section attempts to answer some questions about how links behave in certain situations.

Links are evaluated in three basic contexts.

- dbPutLink()/dbScanFwdLink()
- dbGetLink() of non-CP link
- dbGetLink() during a scan resulting from a CP link.

An input link can bring in a Value as well as meta-data, alarm, time, and display/control info. For input links, the PVA link engine attempts to always maintain consistency between Value, alarm, and time. However, consistency between these, and the display/control info is only ensured during a CP scan.

1.1.4 Address Specification

Address parameters specify where an input record obtains input, where an output record obtains its desired output values, and where an output record writes its output. They are used to identify links between records, and to specify the location of hardware devices. The most common link fields are OUT, an output link, INP, an input link, and DOL (desired output location), also an input link.

There are three basic types of address specifications, which can appear in these fields: hardware addresses, database addresses, and constants.

Note: Not all links support all three types, though some do. However, this doesn't hold true for algorithmic records, which cannot specify hardware addresses. Algorithm records are records like the Calculation, PID, and Select records. These records are used to process values retrieved from other records. Consult the documentation for each record.

Hardware Addresses

The interface between EPICS process database logic and hardware drivers is indicated in two fields of records that support hardware interfaces: DTYP and INP/OUT. The DTYP field is the name of the device support entry table that is used to interface to the device. The address specification is dictated by the device support. Some conventions exist for several buses that are listed below. Lately, more devices have just opted to use a string that is then parsed by the device support as desired. This specification type is called INST I/O. The other conventions listed here include: VME, Allen-Bradley, CAMAC, GPIB, BITBUS, VXI, and RF. The input specification for each of these is different. The specification of these strings must be acquired from the device support code or document.

INST

The INST I/O specification is a string that is parsed by the device support. The format of this string is determined by the device support.

@*parm*

For INST I/O

- @ precedes optional string *parm*

VME Bus

The VME address specification format differs between the various devices. In all of these specifications the '#' character designates a hardware address. The three formats are:

#C*x* S*y* @*parm*

For analog in, analog out, and timer

- C precedes the card number *x*
- S precedes the signal number *y*
- @ precedes optional string *parm*

The card number in the VME addresses refers to the logical card number. Card numbers are assigned by address convention; their position in the backplane is of no consequence. The addresses are assigned by the technician who populates the backplane, with the logical numbers welldocumented. The logical card numbers start with 0 as do the signal numbers. *parm* refers to an arbitrary string of up to 31 characters and is device specific.

Allen-Bradley Bus

The Allen-Bradley address specification is a bit more complicated as it has several more fields. The '#' designates a hardware address. The format is:

#L*a* A*b* C*c* S*d* @*parm'*

All record types

- L precedes the serial link number *a* and is optional - default 0
- A precedes the adapter number *b* and is optional - default 0
- C precedes the card number *c*
- S precedes the signal number *d*
- @ precedes optional string *parm*

The card number for Allen-Bradley I/O refers to the physical slot number, where 0 is the slot directly to the right of the adapter card. The AllenBradley I/O has 12 slots available for I/O cards numbered 0 through 11. Allen-Bradley I/O may use double slot addresses which means that slots 0,2,4,6,8, and 10 are used for input modules and slots 1,3,5,7,9 and 11 are used for output modules. It's required to use the double slot addressing mode when the 1771IL card is used as it only works in double slot addressing mode. This card is required as it provides Kilovolt isolation.

Camac Bus

The CAMAC address specification is similar to the Allen-Bradley address specification. The '#' signifies a hardware address. The format is:

#B*a* C*b* N*c* A*d* F*e* @*parm*

For waveform digitizers

- B precedes the branch number *a* C precedes the crate number *b*
- N precedes the station number *c*
- A precedes the subaddress *d* (optional)
- F precedes the function *e* (optional)
- @ precedes optional string *parm*

The waveform digitizer supported is only one channel per card; no channel was necessary.

Others

The GPIB, BITBUS, RF, and VXI card-types have been added to the supported I/O cards. A brief description of the address format for each follows. For a further explanation, see the specific documentation on each card.

#L*a* A*b* @*parm*

For GPIB I/O

- L precedes the link number *a*
- A precedes the GPIB address *b*
- @ precedes optional string *parm*

#L*a* N*b* P*c* S*d* @*parm*

For BITBUS I/O

- L precedes the link *a*, i.e., the VME bitbus interface
- N precedes the bitbus node *b*
- P precedes the port on node *c*
- S precedes the signal on port *d*
- @ precedes optional string *parm*

#V*a* C*b* S*c* @*parm*

For VXI I/O, dynamic addressing

- V precedes the VXI frame number *a*
- C precedes the slot within VXI frame *b*
- S precedes the signal number *c*

- @ precedes optional string *parm*

#V*a* S*b* @*parm*

For VXI I/O, static addressing

- V precedes the logical address *a*
- S precedes the signal number *b*
- @ precedes optional string *parm*

Database Addresses

Database addresses are used to specify input links, desired output links, output links, and forward processing links. The format in each case is the same:

<RecordName>.<FieldName>

where RecordName is simply the name of the record being referenced, '.' is the separator between the record name and the field name, and FieldName is the name of the field within the record.

The record name and field name specification are case sensitive. The record name can be a mix of the following: a-z A-Z 0-9 _ - : . [ ] < > ;. The field name is always upper case. If no field name is specified as part of an address, the value field (VAL) of the record is assumed. Forward processing links do not need to include the field name because no value is returned when a forward processing link is used; therefore, a forward processing link need only specify a record name.

Basic typecast conversions are made automatically when a value is retrieved from another record–integers are converted to floating point numbers and floating point numbers are converted to integers. For example, a calculation record which uses the value field of a binary input will get a floating point 1 or 0 to use in the calculation, because a calculation record's value fields are floating point numbers. If the value of the calculation record is used as the desired output of a multi-bit binary output, the floating point result is converted to an integer, because multi-bit binary outputs use integers.

Records that use soft device support routines or have no hardware device support routines are called *soft records*. See the chapter on each record for information about that record's device support.

Constants

Input link fields and desired output location fields can specify a constant instead of a hardware or database address. A constant, which is not really an address, can be an integer value in whatever format (hex, decimal, etc.) or a floating-point value. The value field is initialized to the constant when the database is initialized, and at run-time the value field can be changed by a database access routine. For instance, a constant may be used in an input link of a calculation record. For non-constant links, the calc record retrieves the values from the input links, and places them in a corresponding value field. For constant links, the value fields are initialized with the constant, and the values can be changed by modifying the value field, not the link field. Thus, because the calc record uses its value fields as the operands of its expression, the constant becomes part of the calculation.

When nothing is specified in a link field, it is a NULL link. Before Release 3.13, the value fields associated with the NULL link were initialized with the value of zero. From Release 3.13 onwards, the value fields associated with the links are not initialized.

A constant may also be used in the desired output location or DOL field of an output record. In such a case, the initial desired output value (VAL) will be that constant. Any specified conversions are performed on the value before it is written as long as the device support module supports conversions (the Soft Channel device support routine does not perform conversions). The desired output value can be changed by an operator at run-time by writing to the value field.

A constant can be used in an output link field, but no output will be written if this is the case. Be aware that this is not considered an error by the database checking utilities.

1.1.5 Conversion Specification

Conversion parameters are used to convert transducer data into meaningful data. Discrete signals require converting between levels and states (i.e., on, off, high, low, etc.). Analog conversions require converting between levels and engineering units (i.e., pressure, temperature, level, etc.). These conversions are made to provide operators and application codes with values in meaningful units.

The following sections discuss these types of conversions. The actual field names appear in capital letters.

Discrete Conversions

The most simple type of discrete conversion would be the case of a discrete input that indicates the on/off state of a device. If the level is high it indicates that the state of the device is on. Conversely, if the level is low it indicates that the device is off. In the database, parameters are available to enter strings which correspond to each level, which, in turn, correspond to a state (0,1). By defining these strings, the operator is not required to know that a specific transducer is on when the level of its transmitter is high or off when the level is low. In a typical example, the conversion parameters for a discrete input would be entered as follows:

Zero Name (ZNAM): Off
One Name (ONAM): On

The equivalent discrete output example would be an on/off controller. Let's consider a case where the safe state of a device is On, the zero state. The level being low drives the device on, so that a broken cable will drive the device to a safe state. In this example the database parameters are entered as follows:

Zero Name (ZNAM): On
One Name (ONAM): Off

By giving the outside world the device state, the information is clear. Binary inputs and binary outputs are used to represent such on/off devices.

A more complex example involving discrete values is a multi-bit binary output record. Consider a two state valve which has four states-Traveling, full open, full closed, and disconnected. The bit pattern for each control state is entered into the database with the string that describes that state. The database parameters for the monitor would be entered as follows:

Number of Bits (NOBT): 2
First Input Bit Spec (INP): Address of the least significant bit
Zero Value (ZRVL): 0
One Value (ONVL): 1
Two Value (TWVL): 2
Three Value (THVL): 3
Zero String (ZRST): Traveling
One String (ONST): Open
Two String (TWST): Closed
Three String (THST): Disconnected

In this case, when the database record is scanned, the monitor bits are read and compared with the bit patterns for each state. When the bit pattern is found, the device is set to that state. For instance, if the two monitor bits read equal 10 (binary), the Two value is the corresponding value, and the device would be set to state 2 which indicates that the valve is Closed.

If the bit pattern is not found, the device is in an unknown state. In this example all possible states are defined.

In addition, the DOL fields of binary output records (bo and mbbo) will accept values in strings. When they retrieve the string or when the value field is given a string via put_enum_strs, a match is sought with one of the states. If a match is found, the value for that state is written.

Analog Conversions

Analog conversions require knowledge of the transducer, the filters, and the I/O cards. Together they measure the process, transmit the data, and interface the data to the IOC. Smoothing is available to filter noisy signals. The smoothing argument is a constant between 0 and 1 and is specified in the SMOO field. It is applied to the converted hardware signal as follows:

eng units = (new eng units × (1 - smoothing)) + (old eng units × smoothing)

The analog conversions from raw values to engineering units can be either linear or breakpoint conversions.

Whether an analog record performs linear conversions, breakpoint conversions, or no conversions at all depends on how the record's LINR field is configured. The possible choices for the LINR field are as follows:

- LINEAR
- SLOPE
- NO CONVERSION
- typeKdegF
- typeKdegC
- typeJdegF
- typeJdegC

If either LINEAR or SLOPE is chosen, the record performs a linear conversion on the data. If NO CONVERSION is chosen, the record performs no conversion on its data. The other choices are the names of breakpoint tables. When one of these is specified in the LINR field, the record uses the specified table to convert its data. (Note that additional breakpoint tables are often added at specific sites, so more breakpoint tables than are listed here may be available at the user's site.) The following sections explain linear and breakpoint conversions.

Linear Conversions

The engineering units full scale and low scale are specified in the EGUF and EGUL fields, respectively. The values of the EGUF and EGUL fields correspond to the maximum and minimum values of the transducer, respectively. Thus, the value of these fields is device dependent. For example, if the transducer has a range of -10 to +10 volts, then the EGUF field should be 10 and the EGUL field should be -10. In all cases, the EGU field is a string that contains the text to indicate the units of the value.

The distinction between the LINEAR and SLOPE settings for the LINR field are in how the conversion parameters are calculated:

With LINEAR conversion the user must set EGUL and EGUF to the lowest and highest possible engineering units values respectively that can be converted by the hardware. The device support knows the range of the raw data and calculates ESLO and EOFF from them.

SLOPE conversion requires the user to calculate the appropriate scaling and offset factors and put them directly in ESLO and EOFF.

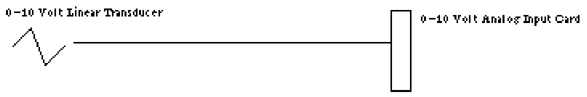
There are three formulas to know when considering the linear conversion parameters. The conversion from measured value to engineering units is as follows:

$$\text{engunits} = \text{eng units low} + \frac{\text{measured A/D counts}}{\text{full scale A/D counts}} * (\text{eng units full scale - eng units low})$$

In the following examples the determination of engineering units full scale and low scale is shown. The conversion to engineering units is also shown to familiarize the reader with the signal conversions from signal source to database engineering units.

Transducer Matches the I/O module

First let us consider a linear conversion. In this example, the transducer transmits 0-10 Volts, there is no amplification, and the I/O card uses a 0-10 Volt interface.



The transducer transmits pressure: 0 PSI at 0 Volts and 175 PSI at 10 Volts. The engineering units full scale and low scale are determined as follows:

eng. units full scale = 17.5 × 10.0
eng. units low scale = 17.5 × 0.0

The field entries in an analog input record to convert this pressure will be as follows:

LINR: Linear
EGUF: 175.0
EGUL: 0
EGU: PSI

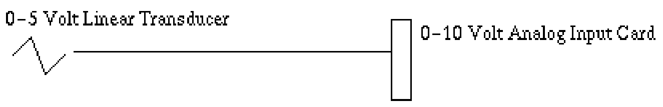
The conversion will also take into account the precision of the I/O module. In this example (assuming a 12 bit analog input card) the conversion is as follows:

$$\text{eng units} = 0 + \frac{\text{measured A/D counts}}{4095} * (175 - 0)$$

When the pressure is 175 PSI, 10 Volts is sent to the I/O module. At 10 Volts the signal is read as 4095. When this is plugged into the conversion, the value is 175 PSI.

Transducer Lower than the I/O module

Let's consider a variation of this linear conversion where the transducer is 0-5 Volts.



In this example the transducer is producing 0 Volts at 0 PSI and 5 Volts at 175 PSI. The engineering units full scale and low scale are determined as follows:

eng. units low scale = 35 × 10 eng. units full scale = 35 × 0

The field entries in an analog record to convert this pressure will be as follows:

LINR: Linear
EGUF: 350
EGUL: 0
EGU: PSI

The conversion will also take into account the precision of the I/O module. In this example (assuming a 12 bit analog input card) the conversion is as follows:

$$\text{eng units} = 0 + \frac{\text{measured A/D counts}}{4095} * (350 - 0)$$

Notice that at full scale the transducer will generate 5 Volts to represent 175 PSI. This is only half of what the input card accepts; input is 2048.

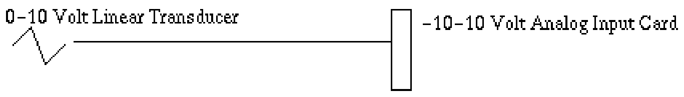
Let's plug in the numbers to see the result:

$$0 + (2048/4095) * (350 - 0) = 175$$

In this example we had to adjust the engineering units full scale to compensate for the difference between the transmitter and the analog input card.

Transducer Positive and I/O module bipolar

Let's consider another variation of this linear conversion where the input card accepts -10 Volts to 10 Volts (i.e. Bipolar instead of Unipolar).



In this example the transducer is producing 0 Volts at 0 PSI and 10 Volts at 175 PSI. The input module has a different range of voltages and the engineering units full scale and low scale are determined as follows:

eng. units full scale = 17.5 × 10 eng. units low scale = 17.5 × (-10)

The database entries to convert this pressure will be as follows:

LINR: Linear
EGUF: 175
EGUL: -175
EGU: PSI

The conversion will also take into account the precision of the I/O module. In this example (assuming a 12 bit analog input card) the conversion is as follows:

$$\text{eng units} = -175 + \frac{\text{measured A/D counts}}{4095} * (175 - (-175))$$

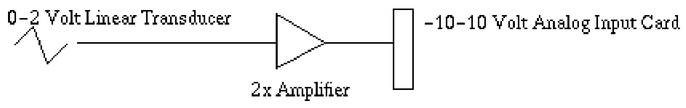
Notice that at low scale the transducer will generate 0 Volts to represent 0 PSI. Because this is half of what the input card accepts, it is input as 2048. Let's plug in the numbers to see the result:

$$-175 + (2048/4095) * (175 - (-175)) = 0$$

In this example we had to adjust the engineering units low scale to compensate for the difference between the unipolar transmitter and the bipolar analog input card.

Combining Linear Conversion with an Amplifier

Let's consider another variation of this linear conversion where the input card accepts -10 Volts to 10 Volts, the transducer transmits 0 - 2 Volts for 0 - 175 PSI and a 2x amplifier is on the transmitter.



At 0 PSI the transducer transmits 0 Volts. This is amplified to 0 Volts. At half scale, it is read as 2048. At 175 PSI, full scale, the transducer transmits 2 Volts, which is amplified to 4 Volts. The analog input card sees 4 Volts as 70 percent of range or 2867 counts. The engineering units full scale and low scale are determined as follows:

eng units full scale = 43.75 × 10
eng units low scale = 43.75 × (-10)

(175 / 4 = 43.75) The record's field entries to convert this pressure will be as follows:

LINR Linear
EGUF 437.5
EGUL -437.5
EGU PSI

The conversion will also take into account the precision of the I/O module. In this example (assuming a 12 bit analog input card) the conversion is as follows:

$$\text{eng units} = -437.5 + \frac{\text{measured A/D counts}}{4095} * (437.5 - (-437.5))$$

Notice that at low scale the transducer will generate 0 Volts to represent 0 PSI. Because this is half of what the input card accepts, it is input as 2048. Let's plug in the numbers to see the result:

$$-437.5 + (2048/4095) * (437.5 - (-437.5)) = 0$$

Notice that at full scale the transducer will generate 2 volts which represents 175 PSI. The amplifier will change the 2 Volts to 4 Volts. 4 Volts is 14/20 or 70 percent of the I/O card's scale. The input from the I/O card is therefore 2866 (i.e., 0.7 * 4095). Let's plug in the numbers to see the result:

$$-437.5 + (2866/4095) * (437.5 - (-437.5)) = 175PSI$$

We had to adjust the engineering units full scale to adjust for the difference between the transducer with the amplifier affects and the range of the I/O card. We also adjusted the low scale to compensate for the difference between the unipolar transmitter/amplifier and the bipolar analog input card.

Breakpoint Conversions

Now let us consider a non-linear conversion. These are conversions that could be entered as polynomials. As these are more time consuming to execute, a breakpoint table is created that breaks the non-linear conversion into linear segments that are accurate enough.

Breakpoint Table

The breakpoint table is then used to do a piecewise linear conversion. Each piecewise segment of the breakpoint table contains:

Raw Value Start for this segment, Engineering Units at the start.

```
breaktable(typeJdegC) {
   0.000000 0.000000
   365.023224 67.000000
   1000.046448 178.000000
   3007.255859 524.000000
   3543.383789 613.000000
   4042.988281 692.000000
   4101.488281 701.000000
}
```

Breakpoint Conversion Example

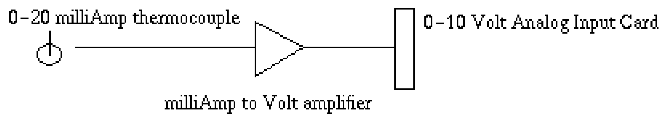
When a new raw value is read, the conversion routine starts from the previously used line segment, compares the raw value start, and either going forward or backward in the table searches the proper segment for this new raw value. Once the proper segment is found, the new engineering units value is the engineering units value at the start of this segment plus the slope of this segment times the position on this segment.

value = eng.units at segment start + (raw value - raw at segment start) * slope

A table that has an entry for each possible raw count is effectively a look up table.

Breakpoint tables are loaded to the IOC using the *dbLoadDatabase* shell function. The slope corresponding to each segment is calculated when the table is loaded. For raw values that exceed the last point in the breakpoint table, the slope of the last segment is used.

In this example the transducer is a thermocouple which transmits 0-20 milliAmps. An amplifier is present which amplifies milliAmps to volts. The I/O card uses a 0-10 Volt interface and a 12-bit ADC. Raw value range would thus be 0 to 4095.


0-20 milliAmp thermocouple
0-10 Volt Analog Input Card
milliAmp to Volt amplifier

The transducer is transmitting temperature. The database entries in the analog input record that are needed to convert this temperature will be as follows:

LINR typeJdegC
EGUF 0
EGUL 0
EGU DGC

For analog records that use breakpoint tables, the EGUF and EGUL fields are not used in the conversion, so they do not have to be given values.

With this example setup and assuming we get an ADC raw reading of 3500, the formula above would give:

Value = 524.0 + (3500 - 3007) * 0.166 = 605.838 DGC

EPICS Base distribution currently includes lookup tables for J and K thermocouples in degrees F and degrees C.

Other potential applications for a lookup table are e.g. other types of thermocouples, logarithmic output controllers, and exponential transducers. The piece-wise linearization of the signals provides a mechanism for conversion that minimizes the amount of floating point arithmetic required to convert non-linear signals. Additional breakpoint tables can be added to the predefined ones.

Creating Breakpoint Tables

There are two ways to create a new breakpoint table:

1) Simply type in the data for each segment, giving the raw and corresponding engineering unit value for each point in the following format.

```
breaktable(<tablename>) {
  <first point> <first eng units>
  <next point> <next eng units>
  <etc.> <...>
}
```

where the <tablename> is the name of the table, such as typeKdegC, and <first point> is the raw value of the beginning point for each line segment, and <first eng units> is the corresponding engineering unit value. The slope is calculated by the software and should not be specified.

2) Create a file consisting of a table of an arbitrary number of values in engineering units and use the utility called **makeBpt** to convert the table into a breakpoint table. As an example, the contents data file to create the typeJdegC breakpoint table look like this:

```
!header
"typeJdegC" 0 0 700 4095 .5 -210 760 1
!data
-8.096 -8.076 -8.057 <many more numbers>
```

The file name must have the extension .data. The file must first have a header specifying these nine things:

1. Name of breakpoint table in quotes: **"typeJdegC"**
2. Engineering units for 1st breakpoint table entry: **0**
3. Raw value for 1st breakpoint table entry: **0**
4. Highest value desired in engineering units: **700**
5. Raw value corresponding to high value in engineering units: **4095**
6. Allowed error in engineering units: **.5**
7. Engineering units corresponding to first entry in data table: **-210**
8. Engineering units corresponding to last entry in data table: **760**
9. Change in engineering units between data table entries: **1**

The rest of the file contains lines of equally spaced engineering values, with each line no more than 160 characters before the new-line character. The header and the actual table should be specified by **!header** and **!data**, respectively. The file for this data table is called typeJdegC.data, and can be converted to a breakpoint table with the **makeBpt** utility as follows:

unix% makeBpt typeJdegC.data

1.1.6 Alarm Specification

There are two elements to an alarm condition: the alarm *status* and the *severity* of that alarm. Each database record contains its current alarm status and the corresponding severity for that status. The scan task, which detects these alarms, is also capable of generating a message for each change of alarm state. The types of alarms available fall into these categories: scan alarms, read/write alarms, limit alarms, and state alarms. Some of these alarms are configured by the user, and some are automatic which means that they are called by the record support routines on certain conditions, and cannot be changed or configured by the user.

Alarm Severity

An alarm *severity* is used to give weight to the current alarm status. There are four severities:

- NO_ALARM
- MINOR
- MAJOR
- INVALID

NO_ALARM means no alarm has been triggered. An alarm state that needs attention but is not dangerous is a MINOR alarm. In this instance the alarm state is meant to give a warning to the operator. A serious state is a MAJOR alarm. In this instance the operator should give immediate attention to the situation and take corrective action. An INVALID alarm means there's a problem with the data, which can be any one of several problems; for instance, a bad address specification, device communication failure, or signal is over range. In these cases, an alarm severity of INVALID is set. An INVALID alarm can point to a simple configuration problem or a serious operational problem.

For limit alarms and state alarms, the severity can be configured by the user to be MAJOR or MINOR for the a specified state. For instance, an analog record can be configured to trigger a MAJOR alarm when its value exceeds 175.0. In addition to the MAJOR and MINOR severity, the user can choose the NO_ALARM severity, in which case no alarm is generated for that state.

For the other alarm types (i.e., scan, read/write), the severity is always INVALID and not configurable by the user.

Alarm Status

Alarm status is a field common to all records. The field is defined as an enumerated field. The possible states are listed below.

- NO_ALARM: This record is not in alarm
- READ: An INPUT link failed in the device support
- WRITE: An OUTPUT link failed in the device support
- HIHI: An analog value limit alarm
- HIGH: An analog value limit alarm
- LOLO: An analog value limit alarm
- LOW: An analog value limit alarm
- STATE: An digital value state alarm
- COS: An digital value change of state alarm
- COMM: A device support alarm that indicates the device is not communicating
- TIMEOUT: A device sup alarm that indicates the asynchronous device timed out
- HWLIMIT: A device sup alarm that indicates a hardware limit alarm
- CALC: A record support alarm for calculation records indicating a bad calculation
- SCAN: An invalid SCAN field is entered
- LINK: Soft device support for a link failed:no record, bad field, invalid conversion, INVALID alarm severity on the referenced record.
- SOFT
- BAD_SUB
- UDF
- DISABLE
- SIMM
- READ_ACCESS
- WRITE_ACCESS

There are a number of issues with this field and menu.

- The maximum enumerated strings passed through channel access is 16 so nothing past SOFT is seen if the value is not requested by Channel Access as a string.
- Only one state can be true at a time so that the root cause of a problem or multiple problems are masked. This is particularly obvious in the interface between the record support and the device support. The hardware could have some combination of problems and there is no way to see this through the interface provided.
- The list is not complete.

- In short, the ability to see failures through the STAT field are limited. Most problems in the hardware, configuration, or communication are reduced to READ or WRITE error and have their severity set to INVALID. When you have an INVALID alarm severity, some investigation is currently needed to determine the fault. Most EPICS drivers provide a report routine that dumps a large set of diagnostic information. This is a good place to start in these cases.

Alarm Conditions Configured in the Database

When you have a valid value, there are fields in the record that allow the user to configure off normal conditions. For analog values these are limit alarms. For discrete values, these are state alarms.

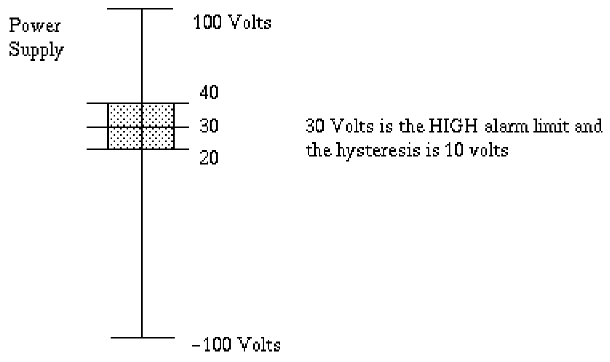
Limit Alarms

For analog records (this includes such records as the stepper motor record), there are configurable alarm limits. There are two limits for above normal operating range and two limits for the below-limit operating range. Each of these limits has an associated alarm severity, which is configured in the database. If the record's value drops below the low limit and an alarm severity of MAJOR was specified for that limit, then a MAJOR alarm is triggered. When the severity of a limit is set to NO_ALARM, none will be generated, even if the limit entered has been violated.

There are two limits at each end, two low values and two high values, so that a warning can be set off before the value goes into a dangerous condition.

Analog records also contain a hysteresis field, which is also used when determining limit violations. The hysteresis field is the deadband around the alarm limits. The deadband keeps a signal that is hovering at the limit from generating too many alarms. Let's take an example (*Figure 8*) where the range is -100 to 100 volts, the high alarm limit is 30 Volts, and the hysteresis is 10 Volts. If the value is normal and approaches the HIGH alarm limit, an alarm is generated when the value reaches 30 Volts. This will only go to normal if the value drops below the limit by more than the hysteresis. For instance, if the value changes from 30 to 28 this record will remain in HIGH alarm. Only when the value drops to 20 will this record return to normal state.

Figure 8


Power Supply
100 Volts
40
30
20
30 Volts is the HIGH alarm limit and the hysteresis is 10 volts
-100 Volts

State Alarms

For discrete values there are configurable state alarms. In this case a user may configure a certain state to be an alarm condition. Let's consider a cooling fan whose discrete states are high, low, and off. The off state can be configured to be an alarm condition so that whenever the fan is off the record is in a STATE alarm. The severity of this error is configured for each state. In this example, the low state could be a STATE alarm of MINOR severity, and the off state a STATE alarm of MAJOR severity.

Discrete records also have a field in which the user can specify the severity of an unknown state to NO_ALARM, MINOR or MAJOR. Thus, the unknown state alarm is not automatic.

Discrete records also have a field, which can specify an alarm when the record's state changes. Thus, an operator can know when the record's alarm state has changed. If this field specifies NO_ALARM, then a change of state will not trigger a change of state alarm. However, if it specifies either MINOR or MAJOR, a change of state will trigger an alarm with the corresponding severity.

Alarm Handling

A record handles alarms with the NSEV, NSTA, SEVR, and STAT fields. When a software component wants to raise an alarm, it first checks the new alarm state fields: NSTA, new alarm state, and NSEV, new alarm severity. If the severity in the NSEV field is higher than the severity in the current severity field (SEVR), then the software component sets the NSTA and NSEV fields to the severity and alarm state that corresponds to the outstanding alarm. When the record process routine next processes the record, it sets the current alarm state (STAT) and current severity

(SEVR) to the values in the NSEV and NSTA fields. This method of handling alarms ensures that the current severity (STAT) reflects the

highest severity of outstanding alarm conditions instead of simply the last raised alarm. This also means that the if multiple alarms of equal severity are present, the alarm status indicates the first one detected.

In addition, the get_alarm_double() routine can be called to format an alarm message and send it to an alarm handler. The alarm conditions may be monitored by the operator interface by explicitly monitoring the STAT and SEVR fields. All values monitored by the operator interface are returned from the database access with current status information.

1.1.7 Monitor Specification

EPICS provides the methods for clients to subscribe to be informed of changes in a PV; in EPICS vocabulary this method is called "monitor".

In Channel Access, as well as PVAccess clients connect to PVs to put, get, or monitor. There are fields in the EPICS records that help limit the monitors posted to these clients through the CA or PVA Server. These fields most typically apply when the client is monitoring the VAL field of a record. Most other fields post a monitor whenever they are changed. For instance, a put to an alarm limit, causes a monitor to be posted to any client that is monitoring that field. The client can select…

For more information about using monitors, see the Channel Access Reference Guide.

Rate Limits

The inherent rate limit is the rate at which the record is scanned. Monitors are only posted when the record is processed as a minimum. There are currently no mechanisms for the client to rate limit a monitor. If a record is being processed at a much higher rate than an application wants, either the database developer can make a second record at a lower rate and have the client connect to that version of the record or the client can disregard the monitors until the time stamp reflects the change.

Channel Access Deadband Selection

The Channel Access client can set a mask to indicate which alarm change it wants to monitor. There are three: value change, archive change, and alarm change.

Value Change Monitors

The value change monitors are typically sent whenever a field in the database changes. The VAL field is the exception. If the MDEL field is set, then the VAL field is sent when a monitor is set, and then only sent again, when the VAL field has changed by MDEL. Note that a MDEL of 0 sends a monitor whenever the VAL fields changes and an MDEL of -1 sends a monitor whenever the record is processed as the MDEL is applied to the absolute value of the difference between the previous scan and the current scan. An MDEL of -1 is useful for scalars that are triggered and a positive indication that the trigger occurred is required.

Archive Change Monitors

The archive change monitors are typically sent whenever a field in the database changes. The VAL field is the exception. If the ADEL field is set, then the VAL field is sent when a monitor is set, and then only sent again, when the VAL field has changed by ADEL.

Alarm Change Monitors

The alarm change monitors are only sent when the alarm severity or status change. As there are filters on the alarm condition checking, the change of alarm status or severity is already filtered through those mechanisms. These are described in *Alarm Specification*.

Metadata Changes

When a Channel Access Client connects to a field, it typically requests some metadata related to that field. One case is a connection from an operator interface typically requests metadata that includes: display limits, control limits, and display information such as precision and engineering units. If any of the fields in a record that are included in this metadata change after the connection is made, the client is not informed and therefore this is not reflected unless the client disconnects and reconnects. A new flag is being added to the Channel Access Client to support posting a monitor to the client whenever any of this metadata changes. Clients can then request the metadata and reflect the change.

Stay tuned for this improvement in the record support and channel access clients.

Client specific Filtering

Several situation have come up that would be useful. These include event filtering, rate guarantee, rate limit, and value change.

Event Filtering

There are several cases where a monitor was sent from a channel only when a specific event was true. For instance, there are diagnostics that are read at 1 kHz. A control program may only want this information when the machine is producing a particular beam such as a linac that has several injectors and beam lines. These are virtual machines that want to be notified when the machine is in their mode. These modes can be interleaved at 60 Hz in some cases. A fault analysis tool may only be interested in all of this data when a fault occurs and the beam is dumped.

There are two efforts here: one at LANL and one from ANL/BNL. These should be discussed in the near future.

Rate Guarantee

Some clients may want to receive a monitor at a given rate. Binary inputs that only notify on change of state may not post a monitor for a very long time. Some clients may prefer to have a notification at some rate even when the value is not changing.

Rate Limit

There is a limit to the rate that most clients care to be notified. Currently, only the SCAN period limits this. A user-imposed limit is needed in some cases such as a data archiver that would only want this channel at 1 Hz (all channels on the same 1 msec in this case).

Value Change

Different clients may have a need to set different deadbands among them. No specific case is cited.

1.1.8 Control Specification

A control loop is a set of database records used to maintain control autonomously. Each output record has two fields that are help implement this independent control: the desired output location field (DOL) and the output mode select field (OMSL). The OMSL field has two mode choices: closed_loop or supervisory. When the closed loop mode is chosen, the desired output is retrieved from the location specified by the DOL field and placed into the VAL field. When the supervisory mode is chosen, the desired output value is the VAL field. In supervisory mode the DOL link is not retrieved. In the supervisory mode, VAL is set typically by the operator through a Channel Access “Put”.

Closing an Analog Control Loop

In a simple control loop an analog input record reads the value of a process variable or PV. The operator sets the Setpoint in the PID record. Then, a PID record retrieves the value from the analog input record and computes the error - the difference between the readback and the setpoint. The PID record computes the new output setting to move the process variable toward the setpoint. The analog output record gets the value from the PID through the DOL when the OMSL is closed_loop. It sets the new output and on the next period repeats this process.

Configuring an Interlock

When certain conditions become true in the process, it may trip an interlock. The result of this interlock is to move something into a safe state or to mitigate damage by taking some action. One example is the closing of a vacuum valve to isolate a vacuum loss. When a vacuum reading in one region of a machine is not at the operating range, an interlock is used to either close a valve and prohibit it from being open. This can be implemented by reading several vacuum gauges in an area into a calculation record. The expression in the calculation record can express the condition that permits the valve to open. The result of the expression is then referenced to the DOL field of a binary output record that controls the valve. If the binary output has the OMSL field set to closed_loop it sets the valve to the value of the calculation record. If it is set to supervisory, the operator can override the interlock and control the valve directly.

1.2 Fields Common to All Record Types

This section contains a description of the fields that are common to all record types. These fields are defined in db-Common.dbd.

See also *Fields Common to Input Record Types* and *Fields Common to Output Record Types*.

1.2.1 Operator Display Parameters

The **NAME** field contains the record name which must be unique within an EPICS Channel Access name space. The name is supplied by the application developer and is the means of identifying a specific record. The name has a maximum length of 60 characters and should use only this limited set of characters:

```
a-z A-Z 0-9 _ - : [ ] < > ;
```

The **DESC** field may be set to provide a meaningful description of the record's purpose. Maximum length is 40 characters.

Field	Summary	Type	DCT	Default	Read	Write	CA PP
NAME	Record Name	STRING [61]	No		Yes	No	No
DESC	Descriptor	STRING [41]	Yes		Yes	Yes	No

1.2.2 Scan Fields

These fields contain information related to how and when a record processes. A few records have unique fields that also affect how they process. These fields, if any, will be listed and explained in the section for each record.

The **SCAN** field specifies the scanning period for periodic record scans or the scan type for non-periodic record scans. The default set of values for SCAN can be found in *menuScan.dbd*.

The choices provided by this menu are:

- `Passive` for the record scan to be triggered by other records or Channel Access
- `Event` for event-driven scan
- `I/O Intr` for interrupt-driven scan
- A set of periodic scan intervals

Additional periodic scan rates may be defined for individual IOCs by making a local copy of menuScan.dbd and adding more choices as required. Periodic scan rates should normally be defined in order following the other scan types, with the longest periods appearing first. Scan periods can be specified with a unit string of `second/seconds`, `minute/minutes`, `hour/hours` or `Hertz/Hz`. Seconds are used if no unit is included in the choice string. For example these rates are all valid:

```
1 hour
0.5 hours
15 minutes
3 seconds
1 second
2 Hertz
```

The **PINI** field specifies record processing at initialization. If it is set to YES during database configuration, the record is processed once at IOC initialization (before the normal scan tasks are started).

The **PHAS** field orders the records within a specific SCAN group. This is not meaningful for passive records. All records of a specified phase are processed before those with higher phase number. It is generally better practice to use linked passive records to enforce the order of processing rather than a phase number.

The **EVNT** field specifies an event number. This event number is used if the SCAN field is set to `Event`. All records with scan type `Event` and the same EVNT value will be processed when a call to post_event for EVNT is made. The call to post_event is: post_event(short event_number).

The **PRIO** field specifies the scheduling priority for processing records with SCAN=`I/O Event` and asynchronous record completion tasks.

The **DISV** field specifies a "disable value". Record processing cannot begin when the value of this field is equal to the value of the DISA field, meaning the record is disabled. Note that field values of a record can be changed by database or Channel Access puts, even if the record is disabled.

The **DISA** field contains the value that is compared with DISV to determine if the record is disabled. A value is obtained for the DISA field from the **SDIS** link field before the IOC tries to process the record. If SDIS is not set, DISA may be set by some other method to enable and disable the record.

The **DISS** field defines the record's "disable severity". If this field is not NO_ALARM and the record is disabled, the record will be put into alarm with this severity and a status of DISABLE_ALARM.

If the **PROC** field of a record is written to, the record is processed.

The **LSET** field contains the lock set to which this record belongs. All records linked in any way via input, output, or forward database links belong to the same lock set. Lock sets are determined at IOC initialization time, and are updated whenever a database link is added, removed or altered.

The **LCNT** field counts the number of times dbProcess finds the record active during successive scans, i.e. PACT is TRUE. If dbProcess finds the record active MAX_LOCK times (currently set to 10) it raises a SCAN_ALARM.

The **PACT** field is TRUE while the record is active (being processed). For asynchronous records PACT can be TRUE from the time record processing is started until the asynchronous completion occurs. As long as PACT is TRUE, dbProcess will not call the record processing routine. See Application Developers Guide for details on usage of PACT.

The **FLNK** field is a link pointing to another record (the "target" record). Processing a record with the FLNK field set will trigger processing of the target record towards the end of processing the first record (but before PACT is cleared), provided the target record's SCAN field is set to `Passive`. If the FLNK field is a Channel Access link it must point to the PROC field of the target record.

The **SPVT** field is for internal use by the scanning system.

Field	Summary	Type	DCT	Default	Read	Write	CA PP
SCAN	Scan Mechanism	MENU *menuScan*	Yes		Yes	Yes	No
PINI	Process at iocInit	MENU *menuPini*	Yes		Yes	Yes	No
PHAS	Scan Phase	SHORT	Yes		Yes	Yes	No
EVNT	Event Name	STRING [40]	Yes		Yes	Yes	No
PRIO	Scheduling Priority	MENU *menuPriority*	Yes		Yes	Yes	No
DISV	Disable Value	SHORT	Yes	1	Yes	Yes	No
DISA	Disable	SHORT	No		Yes	Yes	No
SDIS	Scanning Disable	INLINK	Yes		Yes	Yes	No
PROC	Force Processing	UCHAR	No		Yes	Yes	Yes
DISS	Disable Alarm Sevrty	MENU *menuAlarmSevr*	Yes		Yes	Yes	No
LCNT	Lock Count	UCHAR	No		Yes	No	No
PACT	Record active	UCHAR	No		Yes	No	No
FLNK	Forward Process Link	FWDLINK	Yes		Yes	Yes	No
SPVT	Scan Private	NOACCESS	No		No	No	No

1.2.3 Alarm Fields

Alarm fields indicate the status and severity of record alarms, or determine how and when alarms are triggered. Of course, many records have alarm-related fields not common to all records. Those fields are listed and explained in the appropriate section on each record.

The **STAT** field contains the current alarm status.

The **SEVR** field contains the current alarm severity.

The **AMSG** string field may contain more detailed information about the alarm.

The STAT, SEVR and AMSG fields hold alarm information as seen outside of the database. The **NSTA**, **NSEV** and **NAMSG** fields are used during record processing by the database access, record support, and device support routines to set new alarm status and severity values and message text. Whenever any software component discovers an alarm condition, it calls one of these routines to register the alarm:

```
recGblSetSevr(precord, new_status, new_severity);
recGblSetSevrMsg(precord, new_status, new_severity, "Message", ...);
```

These check the current alarm severity and update the NSTA, NSEV and NAMSG fields if appropriate so they always relate to the highest severity alarm seen so far during record processing. The file alarm.h defines the allowed alarm status and severity values. Towards the end of record processing these fields are copied into the STAT, SEVR and AMSG fields and alarm monitors triggered.

The **ACKS** field contains the highest unacknowledged alarm severity.

The **ACKT** field specifies whether it is necessary to acknowledge transient alarms.

The **UDF** indicates if the record's value is **U**n**D**e**F**ined. Typically this is caused by a failure in device support, the fact that the record has never been processed, or that the VAL field currently contains a NaN (not a number) or Inf (Infinite) value. UDF defaults to TRUE but can be set in a database file. Record and device support routines which write to the VAL field are generally responsible for setting and clearing UDF.

Field	Summary	Type	DCT	Default	Read	Write	CA PP
STAT	Alarm Status	MENU *menuAlarmStat*	No	UDF	Yes	No	No
SEVR	Alarm Severity	MENU *menuAlarmSevr*	No		Yes	No	No
AMSG	Alarm Message	STRING [40]	No		Yes	No	No
NSTA	New Alarm Status	MENU *menuAlarmStat*	No		Yes	No	No
NSEV	New Alarm Severity	MENU *menuAlarmSevr*	No		Yes	No	No
NAMSG	New Alarm Message	STRING [40]	No		Yes	No	No
ACKS	Alarm Ack Severity	MENU *menuAlarmSevr*	No		Yes	No	No
ACKT	Alarm Ack Transient	MENU *menuYesNo*	Yes	YES	Yes	No	No
UDF	Undefined	UCHAR	Yes	1	Yes	Yes	Yes

1.2.4 Device Fields

The **RSET** field contains the address of the Record Support Entry Table. See the Application Developers Guide for details on usage.

The **DSET** field contains the address of Device Support Entry Table. The value of this field is determined at IOC initialization time. Record support routines use this field to locate their device support routines.

The **DPVT** field is is for private use of the device support modules.

Field	Summary	Type	DCT	Default	Read	Write	CA PP
RSET	Address of RSET	NOACCESS	No		No	No	No
DSET	DSET address	NOACCESS	No		No	No	No
DPVT	Device Private	NOACCESS	No		No	No	No

1.2.5 Debugging Fields

The **TPRO** field can be used to trace record processing. When this field is non-zero and the record is processed, a trace message will be be printed for this record and any other record in the same lock-set that is triggered by a database link from this record. The trace message includes the name of the thread doing the processing, and the name of the record being processed.

The **BKPT** field indicates if there is a breakpoint set at this record. This supports setting a debug breakpoint in the record processing. STEP through database processing can be supported using this.

Field	Summary	Type	DCT	Default	Read	Write	CA PP
TPRO	Trace Processing	UCHAR	No		Yes	Yes	No
BKPT	Break Point	NOACCESS	No		No	No	No

1.2.6 Miscellaneous Fields

The **ASG** string field sets the name of the access security group used for this record. If left empty, the record is placed in group `DEFAULT`.

The **ASP** field is private for use by the access security system.

The **DISP** field can be set to a non-zero value to reject puts from outside of the IOC (i.e. via Channel Access or PV Access) to any field of the record other than to the DISP field itself. Field changes and record processing can still be instigated from inside the IOC using DB links and the IOC scan mechanisms.

The **DTYP** field specifies the device type for the record. Most record types have their own set of device types which are specified in the IOC's database definition file. If a record type does not call any device support routines, the DTYP and DSET fields are not used.

The **MLOK** field contains a mutex which is locked by the monitor routines in dbEvent.c whenever the monitor list for this record is accessed.

The **MLIS** field holds a linked list of client monitors connected to this record. Each record support module is responsible for triggering monitors for any fields that change as a result of record processing.

The **PPN** field contains the address of a putNotify callback.

The **PPNR** field contains the next record for PutNotify.

The **PUTF** field is set to TRUE if dbPutField caused the current record processing.

The **RDES** field contains the address of dbRecordType

The **RPRO** field specifies a reprocessing of the record when current processing completes.

The **TIME** field holds the time stamp when this record was last processed.

The **UTAG** field can be used to hold a site-specific 64-bit User Tag value that is associated with the record's time stamp.

The **TSE** field value indicates the mechanism to use to get the time stamp:

- `0` — Get the current time as normal
- `-1` — Ask the time stamp driver for its best source of the current time, if available.
- `-2` — Device support sets the time stamp and the optional User Tag from the hardware.
- Positive values (normally between 1-255) get the time of the last occurance of the numbered generalTime event.

The **TSEL** field contains an input link for obtaining the time stamp. If this link points to the TIME field of a record then the time stamp and User Tag of that record are copied directly into this record (Channel Access links can only copy the time stamp, not the User Tag). If the link points to any other field, that field's value is read and stored in the TSE field which is then used to provide the time stamp as described above.

Field	Summary	Type	DCT	Default	Read	Write	CA PP
ASG	Access Security Group	STRING [29]	Yes		Yes	Yes	No
ASP	Access Security Pvt	NOACCESS	No		No	No	No
DISP	Disable putField	UCHAR	Yes		Yes	Yes	No
DTYP	Device Type	DEVICE	Yes		Yes	Yes	No
MLOK	Monitor lock	NOACCESS	No		No	No	No
MLIS	Monitor List	NOACCESS	No		No	No	No
PPN	pprocessNotify	NOACCESS	No		No	No	No
PPNR	pprocessNotifyRecord	NOACCESS	No		No	No	No
PUTF	dbPutField process	UCHAR	No		Yes	No	No
RDES	Address of dbRecordType	NOACCESS	No		No	No	No
RPRO	Reprocess	UCHAR	No		Yes	No	No
TIME	Time	NOACCESS	No		No	No	No
UTAG	Time Tag	UINT64	No		Yes	No	No
TSE	Time Stamp Event	SHORT	Yes		Yes	Yes	No
TSEL	Time Stamp Link	INLINK	Yes		Yes	Yes	No

1.3 Fields Common to Input Record Types

This section describes fields that are found in many input record types. These fields usually have the same meaning whenever they are used.

See also Fields Common to All Record Types and Fields Common to Output Record Types.

1.3.1 Input and Value Fields

The **INP** field specifies an input link. It is used by the device support routines to obtain input. For soft analog records it can be a constant, a database link, or a channel access link.

The **DTYP** field specifies the name of the device support module that will input values. Each record type has its own set of device support routines. If a record type does not have any associated device support, DTYP is meaningless.

The **RVAL** field contains - whenever possible - the raw data value exactly as it is obtained from the hardware or from the associated device driver and before it undergoes any conversions. The Soft Channel device support module reads values directly into VAL, bypassing this field.

The **VAL** field contains the record's final value, after any needed conversions have been performed.

1.3.2 Device Input

A device input routine normally returns one of the following values to its associated record support routine:

- 0: Success and convert. The input value is in RVAL. The record support module will compute VAL from RVAL.
- 2: Success, but don't convert. The device support module can specify this value if it does not want any conversions. It might do this for two reasons:
  - A hardware error is detected (in this case, it should also raise an alarm condition).
  - The device support routine reads values directly into the VAL field and then sets UDF to FALSE. For some record types the device support routine may have to do other record-specific processing as well such as applying a smoothing filter to the engineering units value.

1.3.3 Device Support for Soft Records

In most cases, two soft output device support modules are provided: Soft Channel and Raw Soft Channel. Both allow INP to be a constant, a database link, or a channel access link. The Soft Channel device support module reads input directly into the VAL field and specifies that no value conversion should be performed. This allows the record to store values in the data type of its VAL field. Note that for Soft Channel input, the RVAL field is not used. The Raw Soft Channel support module reads input into RVAL and indicates that any specified unit conversions be performed.

The device support read routine normally calls `dbGetLink()` which fetches a value from the link.

If a value was returned by the link the UDF field is set to FALSE. The device support read routine normally returns the status from `dbGetLink()`.

1.3.4 Input Simulation Fields

The **SIMM** field controls simulation mode. By setting this field to YES or RAW, the record can be switched into simulation mode of operation. While in simulation mode, input will be obtained from SIOL instead of INP.

The **SIML** field specifies the simulation mode location. This field can be a constant, a database link, or a channel access link. If SIML is a database or channel access link, then SIMM is read from SIML. If SIML is a constant link then SIMM is initialized with the constant value, but can be changed via database or channel access puts.

The **SVAL** field contains the simulation value. This is the record's input value, in engineering units, when the record is switched into simulation mode, i.e., SIMM is set to YES or RAW. If the record type supports conversion, setting SIMM to RAW causes SVAL to be written to RVAL and the conversion to be done.

The **SIOL** field is a link that can be used to fetch the simulation value. The link can be a constant, a database link, or a channel access link. If SIOL is a database or channel access link, then SVAL is read from SIOL. If SIOL is a constant link then SVAL is initialized with the constant value but can be changed via database or channel access puts.

The **SIMS** field specifies the simulation mode alarm severity. When this field is set to a value other than NO_ALARM and the record is in simulation mode, it will be put into alarm with this severity and a status of SIMM_ALARM.

The **SDLY** field specifies a delay (in seconds) to implement asynchronous processing in simulation mode. A positive SDLY value will be used as delay between the first and second phase of processing in simulation mode. A negative value (default) specifies synchronous processing.

The **SSCN** field specifies the SCAN mechanism to be used in simulation mode. This is specifically useful for 'I/O Intr' scanned records, which would otherwise never be scanned in simulation mode.

1.3.5 Simulation Mode for Input Records

An input record can be switched into simulation mode of operation by setting the value of SIMM to YES or RAW. During simulation, the record will be put into alarm with a severity of SIMS and a status of SIMM_ALARM.

```
          --  (SIMM = NO?)
        /         (if supported and directed by device support,
       /              INP -> RVAL -- convert -> VAL),
                  (else INP -> VAL)
SIML -> SIMM

        \
          --  (SIMM = YES?) SIOL -> SVAL -> VAL
          \
            -- (SIMM = RAW?) SIOL -> SVAL -> RVAL -- convert -> VAL
```

If SIMM is set to YES, the input value, in engineering units, will be obtained from SIOL instead of INP and directly written to the VAL field. If SIMM is set to RAW, the value read through SIOL will be truncated and written to the RVAL field, followed by the regular raw value conversion. While the record is in simulation mode, there will be no calls to device support when the record is processed.

If SIOL contains a link, a TSE setting of "time from device" (-2) is honored in simulation mode by taking the time stamp from the record that SIOL points to.

Normally input records contain a private `readValue()` routine which performs the following steps:

- If PACT is TRUE, the device support read routine is called, status is set to its return code, and readValue returns.
- Call `dbGetLink()` to get a new value for SIMM from SIML.
- Check value of SIMM.

- If SIMM is NO, then call the device support read routine, set status to its return code, and return.
- If SIMM is YES or RAW, then
  - Set alarm status to SIMM_ALARM and severity to SIMS, if SIMS is greater than zero.
  - If the record simulation processing is synchronous (SDLY < 0) or the record is in the second phase of an asynchronous processing, call `dbGetLink()` to read the input value from SIOL into SVAL. Set status to the return code from `dbGetLink()`. If the call succeeded and SIMM is YES, write the value to VAL and set the status to 2 (don't convert), if SIMM is RAW and the record type supports conversion, cast the value to RVAL and leave the status as 0 (convert).

    Otherwise (record is in first phase of an asynchronous processing), set up a callback processing with the delay specified in SDLY.
- If SIMM is not YES, NO or RAW, a SOFT alarm with a severity of INVALID is raised, and return status is set to -1.

1.4 Fields Common to Output Record Types

This section describes fields that are found in many output record types. These fields usually have the same meaning whenever they are used.

See also Fields Common to All Record Types and Fields Common to Input Records.

1.4.1 Output and Value Fields

The **OUT** field specifies an output link. It is used by the device support routines to decide where to send output. For soft records, it can be a constant, a database link, or a channel access link. If the link is a constant, the result is no output.

The **DTYP** field specifies the name of the device support module that will input values. Each record type has its own set of device support routines. If a record type does not have any associated device support, DTYP is meaningless.

The **VAL** field contains the desired value before any conversions to raw output have been performed.

The **OVAL** field is used to decide when to invoke monitors. Archive and value change monitors are invoked if OVAL is not equal to VAL. If a record type needs to make adjustments, OVAL is used to enforce the maximum rate of change limit before converting the desired value to a raw value.

The **RVAL** field contains - whenever possible - the actual value sent to the hardware itself or to the associated device driver.

The **RBV** field contains - whenever possible - the actual read back value obtained from the hardware itself or from the associated device driver.

1.4.2 Device Support for Soft Records

Normally two soft output device support modules are provided, Soft Channel and and Raw Soft Channel. Both write a value through the output link OUT. The Soft Channel module writes output from the value associated with OVAL or VAL (if OVAL does not exist). The Raw Soft Channel support module writes the value associated with the RVAL field after conversion has been performed.

The device support write routine normally calls `dbPutLink()` which writes a value through the OUT link, and returns the status from that call.

1.4.3 Input and Mode Select Fields

The **DOL** field is a link from which the desired output value can be fetched. DOL can be a constant, a database link, or a channel access link. If DOL is a database or channel access link and OMSL is closed_loop, then VAL is obtained from DOL.

The **OMSL** field selects the output mode. This field has either the value `supervisory` or `closed_loop`. DOL is used to fetch VAL only if OMSL has the value `closed_loop`. By setting this field a record can be switched between supervisory and closed loop mode of operation. While in closed loop mode, the VAL field cannot be set via dbPuts.

1.4.4 Output Mode Selection

The fields DOL and OMSL are used to allow the output record to be part of a closed loop control algorithm. OMSL is meaningful only if DOL refers to a database or channel access link. It can have the values `supervisory` or `closed_loop`. If the mode is `supervisory`, then nothing is done to VAL. If the mode is `closed_loop` and the record type does not contain an OIF field, then each time the record is processed, VAL is set equal to the value obtained from the location referenced by DOL. If the mode is `closed_loop` in record types with an OIF field and OIF is Full, VAL is set equal to the value obtained from the location referenced by DOL; if OIF is Incremental VAL is incremented by the value obtained from DOL.

1.4.5 Invalid Output Action Fields

The **IVOA** field specifies the output action for the case that the record is put into an INVALID alarm severity. IVOA can be one of the following actions:

- `Continue normally`
- `Don't drive outputs`
- `Set output to IVOV`

The **IVOV** field contains the value for the IVOA action `Set output to IVOV` in engineering units. If a new severity has been set to INVALID and IVOA is `Set output to IVOV`, then VAL is set to IVOV and converted to RVAL before device support is called.

1.4.6 Invalid Alarm Output Action

Whenever an output record is put into INVALID alarm severity, IVOA specifies an action to take. The record support process routine for each output record contains code which performs the following steps.

- If new severity is less than INVALID, then call `writeValue()`:
- Else do the following:
  - If IVOA is `Continue normally` then call `writeValue()`.
  - If IVOA is `Don't drive outputs` then do not write output.
  - If IVOA is `Set output to IVOV` then set VAL to IVOV, call `convert()` if necessary, and then call `writeValue()`.
  - If IVOA not one of the above, an error message is generated.

1.4.7 Output Simulation Fields

The **SIMM** field controls simulation mode. It has either the value YES or NO. By setting this field to YES, the record can be switched into simulation mode of operation. While in simulation mode, output will be forwarded through SIOL instead of OUT.

The **SIML** field specifies the simulation mode location. This field can be a constant, a database link, or a channel access link. If SIML is a database or channel access link, then SIMM is read from SIML. If SIML is a constant link then SIMM is initialized with the constant value, but can be changed via database or channel access puts.

The **SIOL** field is a link that the output value is written to when the record is in simulation mode.

The **SIMS** field specifies the simulation mode alarm severity. When this field is set to a value other than NO_ALARM and the record is in simulation mode, it will be put into alarm with this severity and a status of SIMM_ALARM.

The **SDLY** field specifies a delay (in seconds) to implement asynchronous processing in simulation mode. A positive SDLY value will be used as delay between the first and second phase of processing in simulation mode. A negative value (default) specifies synchronous processing.

The **SSCN** field specifies the SCAN mechanism to be used in simulation mode. This is specifically useful for 'I/O Intr' scanned records, which would otherwise never be scanned in simulation mode.

1.4.8 Simulation Mode for Output Records

An output record can be switched into simulation mode of operation by setting the value of SIMM to YES. During simulation, the record will be put into alarm with a severity of SIMS and a status of SIMM_ALARM. While in simulation mode, output values, in engineering units, will be written to SIOL instead of OUT. However, the output values are never converted. Also, while the record is in simulation mode, there will be no calls to device support during record processing.

Normally output records contain a private `writeValue()` routine which performs the following steps:

- If PACT is TRUE, the device support write routine is called, status is set to its return code, and readValue returns.
- Call `dbGetLink()` to get a new value for SIMM if SIML is a DB_LINK or a CA_LINK.
- Check value of SIMM.
- If SIMM is NO, then call the device support write routine, set status to its return code, and return.
- If SIMM is YES, then
  - Set alarm status to SIMM_ALARM and severity to SIMS, if SIMS is greater than zero.
  - If the record simulation processing is synchronous (SDLY < 0) or the record is in the second phase of an asynchronous processing, call `dbPutLink()` to write the output value from VAL or OVAL to SIOL.

    Otherwise (record is in first phase of an asynchronous processing), set up a callback processing with the delay specified in SDLY.
  - Set status to the return code from `dbPutLink()` and return.
- If SIMM is not YES or NO, a SOFT alarm with a severity of INVALID is raised, and return status is set to -1.

1.5 EPICS Record Types

1.5.1 Analog Input Record (ai)

{#airec-usage} This record type is normally used to obtain an analog value from a hardware input and convert it to engineering units. The record supports linear and break-point conversion to engineering units, smoothing, alarm limits, alarm filtering, and graphics and control limits.

Parameter Fields

The record-specific fields are described below, grouped by functionality.

Input Specification

These fields control where the record will read data from when it is processed:

Field	Summary	Type	DCT	Default	Read	Write	CA PP
DTYP	Device Type	DEVICE	Yes		Yes	Yes	No
INP	Input Specification	INLINK	Yes		Yes	Yes	No

The DTYP field selects which device support layer should be responsible for providing input data to the record. The ai device support layers provided by EPICS Base are documented in the *Device Support* section. External support modules may provide additional device support for this record type. If not set explicitly, the DTYP value defaults to the first device support that is loaded for the record type, which will usually be the `Soft Channel` support that comes with Base.

The INP link field contains a database or channel access link or provides hardware address information that the device support uses to determine where the input data should come from. The format for the INP field value depends on the device support layer that is selected by the DTYP field. See Address Specification for a description of the various hardware address formats supported.

Units Conversion

These fields control if and how the raw input value gets converted into engineering units:

Field	Summary	Type	DCT	Default	Read	Write	CA PP
RVAL	Current Raw Value	LONG	No		Yes	Yes	Yes
ROFF	Raw Offset	ULONG	No		Yes	Yes	Yes
ASLO	Adjustment Slope	DOUBLE	Yes	1	Yes	Yes	Yes
AOFF	Adjustment Offset	DOUBLE	Yes		Yes	Yes	Yes
LINR	Linearization	MENU menuConvert	Yes		Yes	Yes	Yes
ESLO	Raw to EGU Slope	DOUBLE	Yes	1	Yes	Yes	Yes
EOFF	Raw to EGU Offset	DOUBLE	Yes		Yes	Yes	Yes
EGUL	Engineer Units Low	DOUBLE	Yes		Yes	Yes	Yes
EGUF	Engineer Units Full	DOUBLE	Yes		Yes	Yes	Yes

These fields are not used if the device support layer reads its value in engineering units and puts it directly into the VAL field. This applies to Soft Channel and Async Soft Channel device support, and is also fairly common for GPIB and similar high-level device interfaces.

If the device support sets the RVAL field, the LINR field controls how this gets converted into engineering units and placed in the VAL field as follows:

- 1.

RVAL is converted to a double and ROFF is added to it.

- 2.

If ASLO is non-zero the value is multiplied by ASLO.

- 3.

AOFF is added.

- 4.

If LINR is `NO CONVERSION` the units conversion is finished after the above steps.

- 5.

If LINR is `LINEAR` or `SLOPE`, the value from step 3 above is multiplied by ESLO and EOFF is added to complete the units conversion process.

- 6.

Any other value for LINR selects a particular breakpoint table to be used on the value from step 3 above.

The distinction between the `LINEAR` and `SLOPE` settings for the LINR field are in how the conversion parameters are calculated:

- With `LINEAR` conversion the user must set EGUL and EGUF to the lowest and highest possible engineering units values respectively that can be converted by the hardware. The device support knows the range of the raw data and calculates ESLO and EOFF from them.

- `SLOPE` conversion requires the user to calculate the appropriate scaling and offset factors and put them directly in ESLO and EOFF.

Smoothing Filter

This filter is usually only used if the device support sets the RVAL field and the Units Conversion process is used. Device support that directly sets the VAL field may implement the filter if desired.

The filter is controlled with a single parameter field:

Field	Summary	Type	DCT	Default	Read	Write	CA PP
SMOO	Smoothing	DOUBLE	Yes		Yes	Yes	No

The SMOO field should be set to a number between 0 and 1. If set to zero the filter is not used (no smoothing), while if set to one the result is infinite smoothing (the VAL field will never change). The calculation performed is:

VAL = VAL * SMOO + (1 - SMOO) * New Data

where `New Data` was the result from the Units Conversion above. This implements a first-order infinite impulse response (IIR) digital filter with z-plane pole at SMOO. The equivalent continuous-time filter time constant is given by

 = T / ln(SMOO)

where T is the time between record processing.

Undefined Check

If after applying the smoothing filter the VAL field contains a NaN (Not-a-Number) value, the UDF field is set to a non-zero value, indicating that the record value is undefined, which will trigger a `UDF_ALARM` with severity `INVALID_ALARM`.

Field	Summary	Type	DCT	Default	Read	Write	CA PP
UDF	Undefined	UCHAR	Yes	1	Yes	Yes	Yes

Operator Display Parameters

These parameters are used to present meaningful data to the operator. They do not affect the functioning of the record at all.

- NAME is the record's name, and can be useful when the PV name that a client knows is an alias for the record.
- DESC is a string that is usually used to briefly describe the record.
- EGU is a string of up to 16 characters naming the engineering units that the VAL field represents.
- The HOPR and LOPR fields set the upper and lower display limits for the VAL, HIHI, HIGH, LOW, and LOLO fields.
- The PREC field determines the floating point precision (i.e. the number of digits to show after the decimal point) with which to display VAL and the other DOUBLE fields.

See *Fields Common to All Record Types* for more about the record name (NAME) and description (DESC) fields.

Field	Summary	Type	DCT	Default	Read	Write	CA PP
NAME	Record Name	STRING [61]	No		Yes	No	No
DESC	Descriptor	STRING [41]	Yes		Yes	Yes	No
EGU	Engineering Units	STRING [16]	Yes		Yes	Yes	No
HOPR	High Operating Range	DOUBLE	Yes		Yes	Yes	No
LOPR	Low Operating Range	DOUBLE	Yes		Yes	Yes	No
PREC	Display Precision	SHORT	Yes		Yes	Yes	No

Alarm Limits

The user configures limit alarms by putting numerical values into the HIHI, HIGH, LOW and LOLO fields, and by setting the associated alarm severity in the corresponding HHSV, HSV, LSV and LLSV menu fields.

The HYST field controls hysteresis to prevent alarm chattering from an input signal that is close to one of the limits and suffers from significant readout noise.

The AFTC field sets the time constant on a low-pass filter that delays the reporting of limit alarms until the signal has been within the alarm range for that number of seconds (the default AFTC value of zero retains the previous behavior). The record must be scanned often enough for the filtering action to work effectively and the alarm severity can only change when the record is processed, but that processing does not have to be regular; the filter uses the time since the record last processed in its calculation. Setting AFTC to a positive number of seconds will delay the record going into or out of a minor alarm severity or from minor to major severity until the input signal has been in the alarm range for that number of seconds.

See Alarm Specification for a complete explanation of record alarms and of the standard fields. *Alarm Fields* lists other fields related to alarms that are common to all record types.

Field	Summary	Type	DCT	Default	Read	Write	CA PP
HIHI	Hihi Alarm Limit	DOUBLE	Yes		Yes	Yes	Yes
HIGH	High Alarm Limit	DOUBLE	Yes		Yes	Yes	Yes
LOW	Low Alarm Limit	DOUBLE	Yes		Yes	Yes	Yes
LOLO	Lolo Alarm Limit	DOUBLE	Yes		Yes	Yes	Yes
HHSV	Hihi Severity	MENU menuAlarmSevr	Yes		Yes	Yes	Yes
HSV	High Severity	MENU menuAlarmSevr	Yes		Yes	Yes	Yes
LSV	Low Severity	MENU menuAlarmSevr	Yes		Yes	Yes	Yes
LLSV	Lolo Severity	MENU menuAlarmSevr	Yes		Yes	Yes	Yes
HYST	Alarm Deadband	DOUBLE	Yes		Yes	Yes	No
AFTC	Alarm Filter Time Constant	DOUBLE	Yes		Yes	Yes	No
LALM	Last Value Alarmed	DOUBLE	No		Yes	No	No

Monitor Parameters

These parameters are used to determine when to send monitors placed on the VAL field. The monitors are sent when the current value exceeds the last transmitted value by the appropriate deadband. If these fields are set to zero, a monitor will be triggered every time the value changes; if set to -1, a monitor will be sent every time the record is processed.

The ADEL field sets the deadband for archive monitors (`DBE_LOG` events), while the MDEL field controls value monitors (`DBE_VALUE` events).

The remaining fields are used by the record at run-time to implement the record monitoring functionality.

Field	Summary	Type	DCT	Default	Read	Write	CA PP
ADEL	Archive Deadband	DOUBLE	Yes		Yes	Yes	No
MDEL	Monitor Deadband	DOUBLE	Yes		Yes	Yes	No
ALST	Last Value Archived	DOUBLE	No		Yes	No	No
MLST	Last Val Monitored	DOUBLE	No		Yes	No	No
ORAW	Previous Raw Value	LONG	No		Yes	No	No

Simulation Mode Parameters

The following fields are used to operate the record in simulation mode.

If SIMM (fetched through SIML) is YES or RAW, the record is put in SIMS severity and the value is fetched through SIOL (buffered in SVAL). If SIMM is YES, SVAL is written to VAL without conversion, if SIMM is RAW, SVAL is trancated to RVAL and converted. SSCN sets a different SCAN mechanism to use in simulation mode. SDLY sets a delay (in sec) that is used for asynchronous simulation processing.

See *Input Simulation Fields* for more information on simulation mode and its fields.

Field	Summary	Type	DCT	Default	Read	Write	CA PP
SIML	Simulation Mode Link	INLINK	Yes		Yes	Yes	No
SIMM	Simulation Mode	MENU menuSimm	No		Yes	Yes	No
SIOL	Simulation Input Link	INLINK	Yes		Yes	Yes	No
SVAL	Simulation Value	DOUBLE	No		Yes	Yes	No
SIMS	Simulation Mode Severity	MENU menuAlarmSevr	Yes		Yes	Yes	No
SDLY	Sim. Mode Async Delay	DOUBLE	Yes	-1.0	Yes	Yes	No
SSCN	Sim. Mode Scan	MENU menuScan	Yes	65535	Yes	Yes	No

Device Support Interface

The record requires device support to provide an entry table (dset) which defines the following members:

```
typedef struct {
    long number;
    long (*report)(int level);
    long (*init)(int after);
    long (*init_record)(aiRecord *prec);
    long (*get_ioint_info)(int cmd, aiRecord *prec, IOSCANPVT *piosl);
    long (*read_ai)(aiRecord *prec);
    long (*special_linconv)(aiRecord *prec, int after);
} aidset;
```

The module must set `number` to at least 6, and provide a pointer to its `read_ai()` routine; the other function pointers may be `NULL` if their associated functionality is not required for this support layer. Most device supports also provide an `init_record()` routine to configure the record instance and connect it to the hardware or driver support layer, and if using the record's *"Units Conversion"* features they set `special_linconv()` as well.

The individual routines are described below.

Device Support Routines

```
long report(int level)
```

This optional routine is called by the IOC command `dbior` and is passed the report level that was requested by the user. It should print a report on the state of the device support to stdout. The `level` parameter may be used to output increasingly more detailed information at higher levels, or to select different types of information with different levels. Level zero should print no more than a small summary.

```
long init(int after)
```

This optional routine is called twice at IOC initialization time. The first call happens before any of the `init_record()` calls are made, with the integer parameter `after` set to 0. The second call happens after all of the `init_record()` calls have been made, with `after` set to 1.

```
long init_record(aiRecord *prec)
```

This optional routine is called by the record initialization code for each ai record instance that has its DTYP field set to use this device support. It is normally used to check that the INP address is the expected type and that it points to a valid device; to allocate any record-specific buffer space and other memory; and to connect any communication channels needed for the `read_ai()` routine to work properly.

If the record type's unit conversion features are used, the `init_record()` routine should calculate appropriate values for the ESLO and EOFF fields from the EGUL and EGUF field values. This calculation only has to be performed if the record's LINR field is set to `LINEAR`, but it is not necessary to check that condition first. This same calculation takes place in the `special_linconv()` routine, so the implementation can usually just call that routine to perform the task.

```
long get_ioint_info(int cmd, aiRecord *prec, IOSCANPVT *piosl)
```

This optional routine is called whenever the record's SCAN field is being changed to or from the value `I/O Intr` to find out which I/O Interrupt Scan list the record should be added to or deleted from. If this routine is not provided, it will not be possible to set the SCAN field to the value `I/O Intr` at all.

The `cmd` parameter is zero when the record is being added to the scan list, and one when it is being removed from the list. The routine must determine which interrupt source the record should be connected to, which it indicates by the scan list that it points the location at `*piosl` to before returning. It can prevent the SCAN field from being changed at all by returning a non-zero value to its caller.

In most cases the device support will create the I/O Interrupt Scan lists that it returns for itself, by calling `void scanIoInit(IOSCANPVT *piosl)` once for each separate interrupt source. That routine allocates memory and inializes the list, then passes back a pointer to the new list in the location at `*piosl`.

When the device support receives notification that the interrupt has occurred, it announces that to the IOC by calling `void scanIoRequest(IOSCANPVT iosl)` which will arrange for the appropriate records to be processed in a suitable thread. The `scanIoRequest()` routine is safe to call from an interrupt service routine on embedded architectures (vxWorks and RTEMS).

```
long read_ai(aiRecord *prec)
```

This essential routine is called when the record wants a new value from the addressed device. It is responsible for performing (or at least initiating) a read operation, and (eventually) returning its value to the record.

… PACT and asynchronous processing …

… return value …

```
long special_linconv(aiRecord *prec, int after)
```

This optional routine should be provided if the record type's unit conversion features are used by the device support's `read_ai()` routine returning a status value of zero. It is called by the record code whenever any of the the fields LINR, EGUL or EGUF are modified and LINR has the value `LINEAR`. The routine must calculate and set the fields EOFF and ESLO appropriately based on the new values of EGUL and EGUF.

These calculations can be expressed in terms of the minimum and maximum raw values that the `read_ai()` routine can put in the RVAL field. When RVAL is set to *RVAL_max* the VAL field will be set to EGUF, and when RVAL is set to *RVAL_min* the VAL field will become EGUL.

The formulae to use are:

EOFF = (*RVAL_max* * EGUL *RVAL_min* * EGUF) / (*RVAL_max* *RVAL_min*)

ESLO = (EGUF EGUL) / (*RVAL_max* *RVAL_min*)

Note that the record support sets EOFF to EGUL before calling this routine, which is a very common case (when *RVAL_min* is zero).

Extended Device Support

. . .

1.5.2 Analog Output Record (ao)

This record type is normally used to send an analog value to an output device, converting it from engineering units into an integer value if necessary. The record supports alarm and drive limits, rate-of-change limiting, output value integration, linear and break-point conversion from engineering units, and graphics and control limits.

Record-specific Menus

Menu aoOIF

The OIF field which uses this menu controls whether the record acts as an integrator (`Incremental`) or not (`Full`).

Index	Identifier	Choice String
0	aoOIF_Full	Full
1	aoOIF_Incremental	Incremental

Parameter Fields

The record-specific fields are described below.

Output Value Determination

These fields control how the record determines the value to be output when it gets processed:

Field	Summary	Type	DCT	Default	Read	Write	CA PP
OMSL	Output Mode Select	MENU menuOmsl	Yes		Yes	Yes	No
DOL	Desired Output Link	INLINK	Yes		Yes	Yes	No
OIF	Out Full/Incremental	MENU *aoOIF*	Yes		Yes	Yes	No
PVAL	Previous value	DOUBLE	No		Yes	No	No
DRVH	Drive High Limit	DOUBLE	Yes		Yes	Yes	Yes
DRVL	Drive Low Limit	DOUBLE	Yes		Yes	Yes	Yes
VAL	Desired Output	DOUBLE	Yes		Yes	Yes	Yes
OROC	Output Rate of Change	DOUBLE	Yes		Yes	Yes	No
OVAL	Output Value	DOUBLE	No		Yes	Yes	No

The following steps are performed in order during record processing.

Fetch Value, Integrate

The OMSL menu field is used to determine whether the DOL link and OIF menu fields should be used during processing or not:

- If OMSL is `supervisory` the DOL and OIF fields are not used. The new output value is taken from the VAL field, which may have been set from elsewhere.
- If OMSL is `closed_loop` the DOL link field is read to obtain a value; if OIF is `Incremental` and the DOL link was read successfully, the record's previous output value PVAL is added to it.

Drive Limits

The output value is now clipped to the range DRVL to DRVH inclusive, provided that DRVH > DRVL. The result is copied into both the VAL and PVAL fields.

Limit Rate of Change

If the OROC field is not zero, the VAL field is now adjusted so it is no more than OROC different to the previous output value given in OVAL. OROC thus determines the maximum change in the output value that can occur each time the record gets processed. The result is copied into the OVAL field, which is used as the input to the following Units Conversion processing stage.

Units Conversion

…

For analog output records that do not use the Soft Channel device support routine, the specified conversions (if any) are performed on the OVAL field and the resulting value in the RVAL field is sent to the address contained in the output link after it is adjusted by the values in the AOFF and ASLO fields.

Field	Summary	Type	DCT	Default	Read	Write	CA PP
LINR	Linearization	MENU menuConvert	Yes		Yes	Yes	Yes
RVAL	Current Raw Value	LONG	No		Yes	Yes	Yes
ROFF	Raw Offset	ULONG	No		Yes	Yes	Yes
EGUF	Eng Units Full	DOUBLE	Yes		Yes	Yes	Yes
EGUL	Eng Units Low	DOUBLE	Yes		Yes	Yes	Yes
AOFF	Adjustment Offset	DOUBLE	Yes		Yes	Yes	Yes
ASLO	Adjustment Slope	DOUBLE	Yes		Yes	Yes	Yes
ESLO	EGU to Raw Slope	DOUBLE	Yes	1	Yes	Yes	Yes
EOFF	EGU to Raw Offset	DOUBLE	Yes		Yes	Yes	Yes

Conversion Related Fields and the Conversion Process

Except for analog outputs that use Soft Channel device support, the LINR field determines if a conversion is performed and which conversion algorithm is used to convert OVAL to RVAL.

The LINR field can specify `LINEAR` or `SLOPE` for linear conversions, `NO CONVERSION` for no conversions at all, or the name of a breakpoint table such as `typeKdegC` for breakpoint conversions.

The EGUF and EGUL fields should be set for `LINEAR` conversions, and the ESLO and EOFF fields for `SLOPE` conversion. Note that none of these fields have any significance for records that use the Soft Channel device support module.

- EGUF, EGUF

  The user must set these fields when configuring the database for records that use `LINEAR` conversions. They are used to calculate the values for ESLO and EOFF. See Conversion Specification for more information on how to calculate these fields.

- ESLO, EOFF

  Computed by device support from EGUF and EGUL when LINR specifies `LINEAR`. These values must be supplied by the user when LINR specifies `SLOPE`. Used only when LINR is `LINEAR` or `SLOPE`.

- AOFF, ASLO

  These fields are adjustment parameters for the raw output values. They are applied to the raw output value after conversion from engineering units.

- ROFF

  This field can be used to offset the raw value generated by the conversion process, which is needed for some kinds of hardware.

Conversion proceeds as follows:

- 1. If LINR==LINEAR or LINR==SLOPE, then X = (VAL - EOFF) / ESLO, else if LINR==NO_CONVERSION, then X = VAL, else X is obtained via breakpoint table.
- 2. X = (X - AOFF) / ASLO
- 3. RVAL = round(X) - ROFF

To see how the Raw Soft Channel device support routine uses these fields, see *"Device Support For Soft Records"* below for more information.

Output Specification

The analog output record sends its desired output to the address in the OUT field. For analog outputs that write their values to devices, the OUT field must specify the address of the I/O card. In addition, the DTYP field must contain the name of the device support module. Be aware that the address format differs according to the I/O bus used. See Address Specification for information on the format of hardware addresses.

For soft records the output link can be a database link, a channel access link, or a constant value. If the link is a constant, no output is sent.

Field	Summary	Type	DCT	Default	Read	Write	CA PP
DTYP	Device Type	DEVICE	Yes		Yes	Yes	No
OUT	Output Specification	OUTLINK	Yes		Yes	Yes	No

Operator Display Parameters

These parameters are used to present meaningful data to the operator. They display the value and other parameters of the analog output either textually or graphically.

EGU is a string of up to 16 characters describing the units that the analog output measures. It is retrieved by the get_units record support routine.

The HOPR and LOPR fields set the upper and lower display limits for the VAL, OVAL, PVAL, HIHI, HIGH, LOW, and LOLO fields. Both the get_graphic_double and get_control_double record support routines retrieve these fields. If these values are defined, they must be in the range: DRVL <= LOPR <= HOPR <= DRVH.

The PREC field determines the floating point precision with which to display VAL, OVAL and PVAL. It is used whenever the get_precision record support routine is called.

See *Fields Common to All Record Types* for more on the record name (NAME) and description (DESC) fields.

Field	Summary	Type	DCT	Default	Read	Write	CA PP
EGU	Engineering Units	STRING [16]	Yes		Yes	Yes	No
HOPR	High Operating Range	DOUBLE	Yes		Yes	Yes	No
LOPR	Low Operating Range	DOUBLE	Yes		Yes	Yes	No
PREC	Display Precision	SHORT	Yes		Yes	Yes	No
NAME	Record Name	STRING [61]	No		Yes	No	No
DESC	Descriptor	STRING [41]	Yes		Yes	Yes	No

Alarm Parameters

The possible alarm conditions for analog outputs are the SCAN, READ, INVALID and limit alarms. The SCAN, READ, and INVALID alarms are called by the record or device support routines.

The limit alarms are configured by the user in the HIHI, LOLO, HIGH, and LOW fields, which must be floating-point values. For each of these fields, there is a corresponding severity field which can be either NO_ALARM, MINOR, or MAJOR.

See *Invalid Output Action Fields* for more information on the IVOA and IVOV fields.

Alarm Fields lists other fields related to a alarms that are common to all record types.

Field	Summary	Type	DCT	Default	Read	Write	CA PP
HIHI	Hihi Alarm Limit	DOUBLE	Yes		Yes	Yes	Yes
HIGH	High Alarm Limit	DOUBLE	Yes		Yes	Yes	Yes
LOW	Low Alarm Limit	DOUBLE	Yes		Yes	Yes	Yes
LOLO	Lolo Alarm Limit	DOUBLE	Yes		Yes	Yes	Yes
HHSV	Hihi Severity	MENU menuAlarmSevr	Yes		Yes	Yes	Yes
HSV	High Severity	MENU menuAlarmSevr	Yes		Yes	Yes	Yes
LSV	Low Severity	MENU menuAlarmSevr	Yes		Yes	Yes	Yes
LLSV	Lolo Severity	MENU menuAlarmSevr	Yes		Yes	Yes	Yes
HYST	Alarm Deadband	DOUBLE	Yes		Yes	Yes	No
IVOA	INVALID output action	MENU menuIvoa	Yes		Yes	Yes	No
IVOV	INVALID output value	DOUBLE	Yes		Yes	Yes	No

Monitor Parameters

These parameters are used to specify deadbands for monitors on the VAL field. The monitors are sent when the value field exceeds the last monitored field by the specified deadband. If these fields have a value of zero, everytime the value changes, a monitor will be triggered; if they have a value of -1, everytime the record is processed, monitors are triggered. ADEL is the deadband for archive monitors, and MDEL the deadband for all other types of monitors. See Monitor Specification for a complete explanation of monitors.

Field	Summary	Type	DCT	Default	Read	Write	CA PP
ADEL	Archive Deadband	DOUBLE	Yes		Yes	Yes	No
MDEL	Monitor Deadband	DOUBLE	Yes		Yes	Yes	No

Run-time Parameters

These parameters are used by the run-time code for processing the analog output. They are not configurable. They represent the current state of the record. The record support routines use some of them for more efficient processing.

The ORAW field is used to decide if monitors should be triggered for RVAL when monitors are triggered for VAL. The RBV field is the actual read back value obtained from the hardware itself or from the associated device driver. It is the responsibility of the device support routine to give this field a value.

ORBV is used to decide if monitors should be triggered for RBV at the same time monitors are triggered for changes in VAL.

The LALM, MLST, and ALST fields are used to implement the hysteresis factors for monitor callbacks.

The INIT field is used to initialize the LBRK field and for smoothing.

The PBRK field contains a pointer to the current breakpoint table (if any), and LBRK contains a pointer to the last breakpoint table used.

The OMOD field indicates whether OVAL differs from VAL. It will be different if VAL or OVAL have changed since the last time the record was processed, or if VAL has been adjusted by OROC during the current processing.

Field	Summary	Type	DCT	Default	Read	Write	CA PP
ORAW	Previous Raw Value	LONG	No		Yes	No	No
RBV	Readback Value	LONG	No		Yes	No	No
ORBV	Prev Readback Value	LONG	No		Yes	No	No
LALM	Last Value Alarmed	DOUBLE	No		Yes	No	No
ALST	Last Value Archived	DOUBLE	No		Yes	No	No
MLST	Last Val Monitored	DOUBLE	No		Yes	No	No
INIT	Initialized?	SHORT	No		Yes	No	No
PBRK	Ptrto brkTable	NOACCESS	No		No	No	No
LBRK	LastBreak Point	SHORT	No		Yes	No	No
PVAL	Previous value	DOUBLE	No		Yes	No	No
OMOD	Was OVAL modified?	UCHAR	No		Yes	No	No

Simulation Mode Parameters

The following fields are used to operate the record in simulation mode.

If SIMM (fetched through SIML, if populated) is YES, the record is put in SIMS severity and the value is written through SIOL, without conversion. If SIMM is RAW, the value is converted and RVAL is written. SSCN sets a different SCAN mechanism to use in simulation mode. SDLY sets a delay (in sec) that is used for asynchronous simulation processing.

See *Output Simulation Fields* for more information on simulation mode and its fields.

Field	Summary	Type	DCT	Default	Read	Write	CA PP
SIML	Simulation Mode Link	INLINK	Yes		Yes	Yes	No
SIMM	Simulation Mode	MENU menuSimm	No		Yes	Yes	No
SIOL	Simulation Output Link	OUTLINK	Yes		Yes	Yes	No
SIMS	Simulation Mode Severity	MENU menuAlarmSevr	Yes		Yes	Yes	No
SDLY	Sim. Mode Async Delay	DOUBLE	Yes	-1.0	Yes	Yes	No
SSCN	Sim. Mode Scan	MENU menuScan	Yes	65535	Yes	Yes	No

Record Support

Record Support Routines

The following are the record support routines that would be of interest to an application developer. Other routines are the get_units, get_precision, get_graphic_double, and get_control_double routines.

- init_record

  `long init_record(aoRecord *prec, int pass);`

  This routine initializes SIMM if SIML is a constant or creates a channel access link if SIML is PV_LINK. If SIOL is PV_LINK a channel access link is created.

  This routine next checks to see that device support is available. If DOL is a constant, then VAL is initialized with its value and UDF is set to FALSE.

  The routine next checks to see if the device support write routine is defined. If either device support or the device support write routine does not exist, an error message is issued and processing is terminated.

  For compatibility with old device supports that don't know EOFF, if both EOFF and ESLO have their default value, EOFF is set to EGUL.

  If device support includes `init_record()`, it is called.

  INIT is set TRUE. This causes PBRK, LBRK, and smoothing to be re-initialized. If "backwards" linear conversion is requested, then VAL is computed from RVAL using the algorithm:

  ```
  VAL = ((RVAL+ROFF) * ASLO + AOFF) * ESLO + EOFF
  ```

  and UDF is set to FALSE.

  For breakpoint conversion, a call is made to cvtEngToRawBpt and UDF is then set to FALSE. PVAL is set to VAL.

- process

  `long process(aoRecord *prec);`

  See next section.

- special

  `long special(DBADDR *paddr, int after);`

  The only special processing for analog output records is SPC_LINCONV which is invoked whenever either of the fields LINR, EGUF, EGUL or ROFF is changed If the device support routine special_linconv exists it is called.

  INIT is set TRUE. This causes PBRK, LBRK, and smoothing to be re-initialized.

- get_alarm_double

  `long get_alarm_double(DBADDR *, struct dbr_alDouble *);`

  Sets the following values:

  ```
  upper_alarm_limit = HIHI
  upper_warning_limit = HIGH
  lower_warning_limit = LOW
  lower_alarm_limit = LOLO
  ```

Record Processing

Routine process implements the following algorithm:

- 1. Check to see that the appropriate device support module exists. If it doesn't, an error message is issued and processing is terminated with the PACT field set to TRUE. This ensures that processes will no longer be called for this record. Thus error storms will not occur.
- 2. Check PACT: If PACT is FALSE call fetch_values and convert which perform the following steps:
  - fetch_values:
    - if DOL is DB_LINK and OMSL is CLOSED_LOOP then get value from DOL
    - if OIF is INCREMENTAL then set value = value + VAL else value = VAL
  - convert:
    - If Drive limits are defined force value to be within limits
    - Set VAL equal to value
    - Set UDF to FALSE.
    - If OVAL is undefined set it equal to value
    - If OROC is defined and not 0 make |value-OVAL| <=OROC
    - Set OVAL equal to value
    - Compute RVAL from OVAL. using linear or break point table conversion. For linear conversions the algorithm is RVAL = (OVAL-EOFF)/ESLO.
    - For break point table conversion a call is made to cvtEngToRawBpt.
    - After that, for all conversion types AOFF, ASLO, and ROFF are calculated in, using the formula RVAL = (RVAL -AOFF) / ASLO - ROFF.
- 3. Check alarms: This routine checks to see if the new VAL causes the alarm status and severity to change. If so, NSEV, NSTA and y are set. It also honors the alarm hysteresis factor (HYST). Thus the value must change by at least HYST before the alarm status and severity is reduced.
- 4. Check severity and write the new value. See Invalid Alarm Output Action for details on how invalid alarms affect output records.
- 5. If PACT has been changed to TRUE, the device support write output routine has started but has not completed writing the new value. In this case, the processing routine merely returns, leaving PACT TRUE.
- 6. Check to see if monitors should be invoked:
  - Alarm monitors are invoked if the alarm status or severity has changed.
  - Archive and value change monitors are invoked if ADEL and MDEL conditions are met.
  - Monitors for RVAL and for RBV are checked whenever other monitors are invoked.
  - NSEV and NSTA are reset to 0.
- 7. Scan forward link if necessary, set PACT and INIT FALSE, and return.

Device Support

Fields Of Interest To Device Support

Each analog output record must have an associated set of device support routines. The primary responsibility of the device support routines is to output a new value whenever write_ao is called. The device support routines are primarily interested in the following fields:

- PACT — Process Active, used to indicate asynchronous completion
- DPVT — Device Private, reserved for device support to use
- OUT — Output Link, provides addressing information
- EGUF — Engineering Units Full
- EGUL — Engineering Units Low
- ESLO — Engineering Unit Slope
- EOFF — Engineering Unit Offset
- OVAL — Output Value, in Engineering units
- RVAL — Raw Output Value, after conversion

Device Support routines

Device support consists of the following routines:

- report

  `long report(int level);`

  This optional routine is called by the IOC command `dbior` and is passed the report level that was requested by the user. It should print a report on the state of the device support to stdout. The `level` parameter may be used to output increasingly more detailed information at higher levels, or to select different types of information with different levels. Level zero should print no more than a small summary.

- init

  `long init(int after);`

  This optional routine is called twice at IOC initialization time. The first call happens before any of the `init_record()` calls are made, with the integer parameter `after` set to 0. The second call happens after all of the `init_record()` calls have been made, with `after` set to 1.

- init_record

  `long init_record(aoRecord *prec);`

  This optional routine is called by the record initialization code for each ao record instance that has its DTYP field set to use this device support. It is normally used to check that the OUT address has the expected type and points to a valid device; to allocate any record-specific buffer space and other memory; and to connect any communication channels needed for the `write_ao()` routine to work properly.

  If the record type's unit conversion features are used, the `init_record()` routine should calculate appropriate values for the ESLO and EOFF fields from the EGUL and EGUF field values. This calculation only has to be performed if the record's LINR field is set to `LINEAR`, but it is not necessary to check that condition first. This same calculation takes place in the `special_linconv()` routine, so the implementation can usually just call that routine to perform the task.

If the the last output value can be read back from the hardware, this routine should also fetch that value and put it into the record's RVAL or VAL field. The return value should be zero if the RVAL field has been set, or 2 if either the VAL field has been set or if the last output value cannot be retrieved.

- get_ioint_info

  `long get_ioint_info(int cmd, aoRecord *prec, IOSCANPVT *piosl);`

  This optional routine is called whenever the record's SCAN field is being changed to or from the value `I/O Intr` to find out which I/O Interrupt Scan list the record should be added to or deleted from. If this routine is not provided, it will not be possible to set the SCAN field to the value `I/O Intr` at all.

  The `cmd` parameter is zero when the record is being added to the scan list, and one when it is being removed from the list. The routine must determine which interrupt source the record should be connected to, which it indicates by the scan list that it points the location at `*piosl` to before returning. It can prevent the SCAN field from being changed at all by returning a non-zero value to its caller.

  In most cases the device support will create the I/O Interrupt Scan lists that it returns for itself, by calling `void scanIoInit(IOSCANPVT *piosl)` once for each separate interrupt source. That API allocates memory and inializes the list, then passes back a pointer to the new list in the location at `*piosl`. When the device support receives notification that the interrupt has occurred, it announces that to the IOC by calling `void scanIoRequest(IOSCANPVT iosl)` which will arrange for the appropriate records to be processed in a suitable thread. The `scanIoRequest()` routine is safe to call from an interrupt service routine on embedded architectures (vxWorks and RTEMS).

- write_ao

  `long write_ao(aoRecord *prec);`

  This essential routine is called whenever the record has a new output value to send to the device. It is responsible for performing the write operation, using either the engineering units value found in the record's OVAL field, or the raw value from the record's RVAL field if the record type's unit conversion facilities are used. A return value of zero indicates success, any other value indicates that an error occurred.

  This routine must not block (busy-wait) if the device takes more than a few microseconds to accept the new value. In that case the routine must use asynchronous completion to tell the record when the write operation eventually completes. It signals that this is an asynchronous operation by setting the record's PACT field to TRUE before it returns, having arranged for the record's `process()` routine to be called later once the write operation is over. When that happens the `write_ao()` routine will be called again with PACT still set to TRUE; it should then set it to FALSE to indicate the write has completed, and return.

- special_linconv

  `long special_linconv(aoRecord *prec, int after);`

  This optional routine should be provided if the record type's unit conversion features are used by the device support's `write_ao()` routine utilizing the RVAL field rather than OVAL or VAL. It is called by the record code whenever any of the the fields LINR, EGUL or EGUF are modified and LINR has the value `LINEAR`. The routine must calculate and set the fields EOFF and ESLO appropriately based on the new values of EGUL and EGUF.

  These calculations can be expressed in terms of the minimum and maximum raw values that the `write_ao()` routine can accept in the RVAL field. When VAL is EGUF the RVAL field will be set to *RVAL_max*, and when VAL is EGUL the RVAL field will become *RVAL_min*. The fomulae to use are:

  ```
  EOFF = (_RVAL\_max_ \* EGUL  _RVAL\_min_ \* EGUF) /
  (_RVAL\_max_  _RVAL\_min_)

  ESLO = (EGUF  EGUL) / (_RVAL\_max_  _RVAL\_min_)
  ```

  Note that the record support sets EOFF to EGUL before calling this routine, which is a very common case (*RVAL_min* is zero).

Device Support For Soft Records

Two soft device support modules Soft Channel and Raw Soft Channel are provided for output records not related to actual hardware devices. The OUT link type must be either a CONSTANT, DB_LINK, or CA_LINK.

Soft Channel

This module writes the current value of OVAL.

If the OUT link type is PV_LINK, then dbCaAddInlink is called by `init_record()`. `init_record()` always returns a value of 2, which means that no conversion will ever be attempted.

write_ao calls recGblPutLinkValue to write the current value of VAL. See Soft Output for details.

Raw Soft Channel

This module is like the previous except that it writes the current value of RVAL.

1.5.3 Array Subroutine Record (aSub)

The aSub record is an advanced variant of the 'sub' (subroutine) record which has a number of additional features:

- It provides 20 different input and output fields which can hold array or scalar values. The types and array capacities of these are user configurable, and they all have an associated input or output link.
- The name of the C or C++ subroutine to be called when the record processes can be changed dynamically while the IOC is running. The name can either be fetched from another record using an input link, or written directly into the SNAM field.
- The user can choose whether monitor events should be posted for the output fields.
- The VAL field is set to the return value from the user subroutine, which is treated as a status value and controls whether the output links will be used or not. The record can also raise an alarm with a chosen severity if the status value is non-zero.

Record-specific Menus

Menu aSubLFLG

The LFLG menu field controls whether the SUBL link will be read to update the name of the subroutine to be called when the record processes.

Index	Identifier	Choice String
0	aSubLFLG_IGNORE	IGNORE
1	aSubLFLG_READ	READ

Menu aSubEFLG

The EFLG menu field indicates whether monitor events should be posted for the VALA..VALU output value fields.

Index	Identifier	Choice String
0	aSubEFLG_NEVER	NEVER
1	aSubEFLG_ON_CHANGE	ON CHANGE
2	aSubEFLG_ALWAYS	ALWAYS

Parameter Fields

The record-specific fields are described below.

Subroutine Fields

The VAL field is set to the value returned by the user subroutine. The value is treated as an error status value where zero mean success. The output links OUTA … OUTU will only be used to forward the associated output value fields when the subroutine has returned a zero status. If the return status was less than zero, the record will be put into `SOFT_ALARM` state with severity given by the BRSV field.

The INAM field may be used to name a subroutine that will be called once at IOC initialization time.

LFLG tells the record whether to read or ignore the SUBL link. If the value is `READ`, then the name of the subroutine to be called at process time is read from SUBL. If the value is `IGNORE`, the name of the subroutine is that currently held in SNAM.

A string is read from the SUBL link to fetch the name of the subroutine to be run during record processing.

SNAM holds the name of the subroutine to be called when the record processes. The value in this field can be overwritten by the SUBL link if LFLG is set to `READ`.

The SADR field is only accessible from C code; it points to the subroutine to be called.

The CADR field may be set by the user subroutine to point to another function that will be called immediately before setting the SADR field to some other routine. This allows the main user subroutine to allocate resources when it is first called and be able to release them again when they are no longer needed.

Field	Summary	Type	DCT	Default	Read	Write	CA PP
VAL	Subr. return value	LONG	No		Yes	Yes	No
OVAL	Old return value	LONG	No		Yes	No	No
INAM	Initialize Subr. Name	STRING [41]	Yes		Yes	No	No
LFLG	Subr. Input Enable	MENU *aSubLFLG*	Yes		Yes	Yes	No
SUBL	Subroutine Name Link	INLINK	Yes		Yes	No	No
SNAM	Process Subr. Name	STRING [41]	Yes		Yes	Yes	No
ONAM	Old Subr. Name	STRING [41]	Yes		Yes	No	No
SADR	Subroutine Address	NOACCESS	No		No	No	No
CADR	Subroutine Cleanup Address	NOACCESS	No		No	No	No
BRSV	Bad Return Severity	MENU menuAlarmSevr	Yes		Yes	Yes	Yes

Operator Display Parameters

The PREC field specifies the number of decimal places with which to display the values of the value fields A … U and VALA … VALU. Except when it doesn't.

Output Event Flag

This field tells the record when to post change events on the output fields VALA … VALU. If the value is `NEVER`, events are never posted. If the value is `ALWAYS`, events are posted every time the record processes. If the value is `ON CHANGE`, events are posted when any element of an array changes value. This flag controls value, log (archive) and alarm change events.

Field	Summary	Type	DCT	Default	Read	Write	CA PP
EFLG	Output Event Flag	MENU *aSubEFLG*	Yes	1	Yes	Yes	No

Input Link Fields

The input links from where the values of A,…,U are fetched during record processing.

Field	Summary	Type	DCT	Default	Read	Write	CA PP
INPA	Input Link A	INLINK	Yes		Yes	Yes	No
INPB	Input Link B	INLINK	Yes		Yes	Yes	No
INPC	Input Link C	INLINK	Yes		Yes	Yes	No
INPD	Input Link D	INLINK	Yes		Yes	Yes	No
INPE	Input Link E	INLINK	Yes		Yes	Yes	No
INPF	Input Link F	INLINK	Yes		Yes	Yes	No
INPG	Input Link G	INLINK	Yes		Yes	Yes	No
INPH	Input Link H	INLINK	Yes		Yes	Yes	No
INPI	Input Link I	INLINK	Yes		Yes	Yes	No
INPJ	Input Link J	INLINK	Yes		Yes	Yes	No
INPK	Input Link K	INLINK	Yes		Yes	Yes	No
INPL	Input Link L	INLINK	Yes		Yes	Yes	No
INPM	Input Link M	INLINK	Yes		Yes	Yes	No
INPN	Input Link N	INLINK	Yes		Yes	Yes	No
INPO	Input Link O	INLINK	Yes		Yes	Yes	No
INPP	Input Link P	INLINK	Yes		Yes	Yes	No
INPQ	Input Link Q	INLINK	Yes		Yes	Yes	No
INPR	Input Link R	INLINK	Yes		Yes	Yes	No
INPS	Input Link S	INLINK	Yes		Yes	Yes	No
INPT	Input Link T	INLINK	Yes		Yes	Yes	No
INPU	Input Link U	INLINK	Yes		Yes	Yes	No

Input Value Fields

Thse fields hold the scalar or array values fetched through the input links INPA,. . . ,INPU.

Field	Summary	Type	DCT	Default	Read	Write	CA PP
A	Input value A	Set by FTA	No		Yes	Yes	No
B	Input value B	Set by FTB	No		Yes	Yes	No
C	Input value C	Set by FTC	No		Yes	Yes	No
D	Input value D	Set by FTD	No		Yes	Yes	No
E	Input value E	Set by FTE	No		Yes	Yes	No
F	Input value F	Set by FTF	No		Yes	Yes	No
G	Input value G	Set by FTG	No		Yes	Yes	No
H	Input value H	Set by FTH	No		Yes	Yes	No
I	Input value I	Set by FTI	No		Yes	Yes	No
J	Input value J	Set by FTJ	No		Yes	Yes	No
K	Input value K	Set by FTK	No		Yes	Yes	No
L	Input value L	Set by FTL	No		Yes	Yes	No
M	Input value M	Set by FTM	No		Yes	Yes	No
N	Input value N	Set by FTN	No		Yes	Yes	No
O	Input value O	Set by FTO	No		Yes	Yes	No
P	Input value P	Set by FTP	No		Yes	Yes	No
Q	Input value Q	Set by FTQ	No		Yes	Yes	No
R	Input value R	Set by FTR	No		Yes	Yes	No
S	Input value S	Set by FTS	No		Yes	Yes	No
T	Input value T	Set by FTT	No		Yes	Yes	No
U	Input value U	Set by FTU	No		Yes	Yes	No

Input Value Data Types

Field types of the input value fields. The choices can be found by following the link to the menuFtype definition.

Field	Summary	Type	DCT	Default	Read	Write	CA PP
FTA	Type of A	MENU menuFtype	Yes	DOUBLE	Yes	No	No
FTB	Type of B	MENU menuFtype	Yes	DOUBLE	Yes	No	No
FTC	Type of C	MENU menuFtype	Yes	DOUBLE	Yes	No	No
FTD	Type of D	MENU menuFtype	Yes	DOUBLE	Yes	No	No
FTE	Type of E	MENU menuFtype	Yes	DOUBLE	Yes	No	No
FTF	Type of F	MENU menuFtype	Yes	DOUBLE	Yes	No	No
FTG	Type of G	MENU menuFtype	Yes	DOUBLE	Yes	No	No
FTH	Type of H	MENU menuFtype	Yes	DOUBLE	Yes	No	No
FTI	Type of I	MENU menuFtype	Yes	DOUBLE	Yes	No	No
FTJ	Type of J	MENU menuFtype	Yes	DOUBLE	Yes	No	No
FTK	Type of K	MENU menuFtype	Yes	DOUBLE	Yes	No	No
FTL	Type of L	MENU menuFtype	Yes	DOUBLE	Yes	No	No
FTM	Type of M	MENU menuFtype	Yes	DOUBLE	Yes	No	No
FTN	Type of N	MENU menuFtype	Yes	DOUBLE	Yes	No	No
FTO	Type of O	MENU menuFtype	Yes	DOUBLE	Yes	No	No
FTP	Type of P	MENU menuFtype	Yes	DOUBLE	Yes	No	No
FTQ	Type of Q	MENU menuFtype	Yes	DOUBLE	Yes	No	No
FTR	Type of R	MENU menuFtype	Yes	DOUBLE	Yes	No	No
FTS	Type of S	MENU menuFtype	Yes	DOUBLE	Yes	No	No
FTT	Type of T	MENU menuFtype	Yes	DOUBLE	Yes	No	No
FTU	Type of U	MENU menuFtype	Yes	DOUBLE	Yes	No	No

Input Value Array Capacity

These fields specify how many array elements the input value fields may hold.

Note that access to the `NOT` field from C code must use the field name in upper case, e.g. `prec->NOT` since the lower-case `not` is a reserved word in C++ and cannot be used as an identifier.

Field	Summary	Type	DCT	Default	Read	Write	CA PP
NOA	Max. elements in A	ULONG	Yes	1	Yes	No	No
NOB	Max. elements in B	ULONG	Yes	1	Yes	No	No
NOC	Max. elements in C	ULONG	Yes	1	Yes	No	No
NOD	Max. elements in D	ULONG	Yes	1	Yes	No	No
NOE	Max. elements in E	ULONG	Yes	1	Yes	No	No
NOF	Max. elements in F	ULONG	Yes	1	Yes	No	No
NOG	Max. elements in G	ULONG	Yes	1	Yes	No	No
NOH	Max. elements in H	ULONG	Yes	1	Yes	No	No
NOI	Max. elements in I	ULONG	Yes	1	Yes	No	No
NOJ	Max. elements in J	ULONG	Yes	1	Yes	No	No
NOK	Max. elements in K	ULONG	Yes	1	Yes	No	No
NOL	Max. elements in L	ULONG	Yes	1	Yes	No	No
NOM	Max. elements in M	ULONG	Yes	1	Yes	No	No
NON	Max. elements in N	ULONG	Yes	1	Yes	No	No
NOO	Max. elements in O	ULONG	Yes	1	Yes	No	No
NOP	Max. elements in P	ULONG	Yes	1	Yes	No	No
NOQ	Max. elements in Q	ULONG	Yes	1	Yes	No	No
NOR	Max. elements in R	ULONG	Yes	1	Yes	No	No
NOS	Max. elements in S	ULONG	Yes	1	Yes	No	No
NOT	Max. elements in T	ULONG	Yes	1	Yes	No	No
NOU	Max. elements in U	ULONG	Yes	1	Yes	No	No

Input Value Array Size

These fields specify how many array elements the input value fields currently contain.

Field	Summary	Type	DCT	Default	Read	Write	CA PP
NEA	Num. elements in A	ULONG	No	1	Yes	No	No
NEB	Num. elements in B	ULONG	No	1	Yes	No	No
NEC	Num. elements in C	ULONG	No	1	Yes	No	No
NED	Num. elements in D	ULONG	No	1	Yes	No	No
NEE	Num. elements in E	ULONG	No	1	Yes	No	No
NEF	Num. elements in F	ULONG	No	1	Yes	No	No
NEG	Num. elements in G	ULONG	No	1	Yes	No	No
NEH	Num. elements in H	ULONG	No	1	Yes	No	No
NEI	Num. elements in I	ULONG	No	1	Yes	No	No
NEJ	Num. elements in J	ULONG	No	1	Yes	No	No
NEK	Num. elements in K	ULONG	No	1	Yes	No	No
NEL	Num. elements in L	ULONG	No	1	Yes	No	No
NEM	Num. elements in M	ULONG	No	1	Yes	No	No
NEN	Num. elements in N	ULONG	No	1	Yes	No	No
NEO	Num. elements in O	ULONG	No	1	Yes	No	No
NEP	Num. elements in P	ULONG	No	1	Yes	No	No
NEQ	Num. elements in Q	ULONG	No	1	Yes	No	No
NER	Num. elements in R	ULONG	No	1	Yes	No	No
NES	Num. elements in S	ULONG	No	1	Yes	No	No
NET	Num. elements in T	ULONG	No	1	Yes	No	No
NEU	Num. elements in U	ULONG	No	1	Yes	No	No

Output Link Fields

The output links through which the VALA … VALU field values are sent during record processing, provided the subroutine returned 0.

Field	Summary	Type	DCT	Default	Read	Write	CA PP
OUTA	Output Link A	OUTLINK	Yes		Yes	Yes	No
OUTB	Output Link B	OUTLINK	Yes		Yes	Yes	No
OUTC	Output Link C	OUTLINK	Yes		Yes	Yes	No
OUTD	Output Link D	OUTLINK	Yes		Yes	Yes	No
OUTE	Output Link E	OUTLINK	Yes		Yes	Yes	No
OUTF	Output Link F	OUTLINK	Yes		Yes	Yes	No
OUTG	Output Link G	OUTLINK	Yes		Yes	Yes	No
OUTH	Output Link H	OUTLINK	Yes		Yes	Yes	No
OUTI	Output Link I	OUTLINK	Yes		Yes	Yes	No
OUTJ	Output Link J	OUTLINK	Yes		Yes	Yes	No
OUTK	Output Link K	OUTLINK	Yes		Yes	Yes	No
OUTL	Output Link L	OUTLINK	Yes		Yes	Yes	No
OUTM	Output Link M	OUTLINK	Yes		Yes	Yes	No
OUTN	Output Link N	OUTLINK	Yes		Yes	Yes	No
OUTO	Output Link O	OUTLINK	Yes		Yes	Yes	No
OUTP	Output Link P	OUTLINK	Yes		Yes	Yes	No
OUTQ	Output Link Q	OUTLINK	Yes		Yes	Yes	No
OUTR	Output Link R	OUTLINK	Yes		Yes	Yes	No
OUTS	Output Link S	OUTLINK	Yes		Yes	Yes	No
OUTT	Output Link T	OUTLINK	Yes		Yes	Yes	No
OUTU	Output Link U	OUTLINK	Yes		Yes	Yes	No

Output Value Fields

These fields hold scalar or array data generated by the subroutine which will be sent through the OUTA … OUTU links during record processing.

Field	Summary	Type	DCT	Default	Read	Write	CA PP
VALA	Output value A	Set by FTVA	No		Yes	Yes	No
VALB	Output value B	Set by FTVB	No		Yes	Yes	No
VALC	Output value C	Set by FTVC	No		Yes	Yes	No
VALD	Output value D	Set by FTVD	No		Yes	Yes	No
VALE	Output value E	Set by FTVE	No		Yes	Yes	No
VALF	Output value F	Set by FTVF	No		Yes	Yes	No
VALG	Output value G	Set by FTVG	No		Yes	Yes	No
VALH	Output value H	Set by FTVH	No		Yes	Yes	No
VALI	Output value I	Set by FTVI	No		Yes	Yes	No
VALJ	Output value J	Set by FTVJ	No		Yes	Yes	No
VALK	Output value K	Set by FTVK	No		Yes	Yes	No
VALL	Output value L	Set by FTVL	No		Yes	Yes	No
VALM	Output value M	Set by FTVM	No		Yes	Yes	No
VALN	Output value N	Set by FTVN	No		Yes	Yes	No
VALO	Output value O	Set by FTVO	No		Yes	Yes	No
VALP	Output value P	Set by FTVP	No		Yes	Yes	No
VALQ	Output value Q	Set by FTVQ	No		Yes	Yes	No
VALR	Output value R	Set by FTVR	No		Yes	Yes	No
VALS	Output value S	Set by FTVS	No		Yes	Yes	No
VALT	Output value T	Set by FTVT	No		Yes	Yes	No
VALU	Output value U	Set by FTVU	No		Yes	Yes	No

Old Value Fields

The previous values of the output fields. These are used to determine when to post events if EFLG is set to `ON CHANGE`.

Field	Summary	Type	DCT	Default	Read	Write	CA PP
OVLA	Old Output A	NOACCESS	No		No	No	No
OVLB	Old Output B	NOACCESS	No		No	No	No
OVLC	Old Output C	NOACCESS	No		No	No	No
OVLD	Old Output D	NOACCESS	No		No	No	No
OVLE	Old Output E	NOACCESS	No		No	No	No
OVLF	Old Output F	NOACCESS	No		No	No	No
OVLG	Old Output G	NOACCESS	No		No	No	No
OVLH	Old Output H	NOACCESS	No		No	No	No
OVLI	Old Output I	NOACCESS	No		No	No	No
OVLJ	Old Output J	NOACCESS	No		No	No	No
OVLK	Old Output K	NOACCESS	No		No	No	No
OVLL	Old Output L	NOACCESS	No		No	No	No
OVLM	Old Output M	NOACCESS	No		No	No	No
OVLN	Old Output N	NOACCESS	No		No	No	No
OVLO	Old Output O	NOACCESS	No		No	No	No
OVLP	Old Output P	NOACCESS	No		No	No	No
OVLQ	Old Output Q	NOACCESS	No		No	No	No
OVLR	Old Output R	NOACCESS	No		No	No	No
OVLS	Old Output S	NOACCESS	No		No	No	No
OVLT	Old Output T	NOACCESS	No		No	No	No
OVLU	Old Output U	NOACCESS	No		No	No	No

Output Value Data Types

Field types of the output value fields. The choices can be found by following a link to the menuFtype definition.

Field	Summary	Type	DCT	Default	Read	Write	CA PP
FTVA	Type of VALA	MENU menuFtype	Yes	DOUBLE	Yes	No	No
FTVB	Type of VALB	MENU menuFtype	Yes	DOUBLE	Yes	No	No
FTVC	Type of VALC	MENU menuFtype	Yes	DOUBLE	Yes	No	No
FTVD	Type of VALD	MENU menuFtype	Yes	DOUBLE	Yes	No	No
FTVE	Type of VALE	MENU menuFtype	Yes	DOUBLE	Yes	No	No
FTVF	Type of VALF	MENU menuFtype	Yes	DOUBLE	Yes	No	No
FTVG	Type of VALG	MENU menuFtype	Yes	DOUBLE	Yes	No	No
FTVH	Type of VALH	MENU menuFtype	Yes	DOUBLE	Yes	No	No
FTVI	Type of VALI	MENU menuFtype	Yes	DOUBLE	Yes	No	No
FTVJ	Type of VALJ	MENU menuFtype	Yes	DOUBLE	Yes	No	No
FTVK	Type of VALK	MENU menuFtype	Yes	DOUBLE	Yes	No	No
FTVL	Type of VALL	MENU menuFtype	Yes	DOUBLE	Yes	No	No
FTVM	Type of VALM	MENU menuFtype	Yes	DOUBLE	Yes	No	No
FTVN	Type of VALN	MENU menuFtype	Yes	DOUBLE	Yes	No	No
FTVO	Type of VALO	MENU menuFtype	Yes	DOUBLE	Yes	No	No
FTVP	Type of VALP	MENU menuFtype	Yes	DOUBLE	Yes	No	No
FTVQ	Type of VALQ	MENU menuFtype	Yes	DOUBLE	Yes	No	No
FTVR	Type of VALR	MENU menuFtype	Yes	DOUBLE	Yes	No	No
FTVS	Type of VALS	MENU menuFtype	Yes	DOUBLE	Yes	No	No
FTVT	Type of VALT	MENU menuFtype	Yes	DOUBLE	Yes	No	No
FTVU	Type of VALU	MENU menuFtype	Yes	DOUBLE	Yes	No	No

Output Value Array Capacity

These fields specify how many array elements the output value fields may hold.

Field	Summary	Type	DCT	Default	Read	Write	CA PP
NOVA	Max. elements in VALA	ULONG	Yes	1	Yes	No	No
NOVB	Max. elements in VALB	ULONG	Yes	1	Yes	No	No
NOVC	Max. elements in VALC	ULONG	Yes	1	Yes	No	No
NOVD	Max. elements in VALD	ULONG	Yes	1	Yes	No	No
NOVE	Max. elements in VALE	ULONG	Yes	1	Yes	No	No
NOVF	Max. elements in VALF	ULONG	Yes	1	Yes	No	No
NOVG	Max. elements in VALG	ULONG	Yes	1	Yes	No	No
NOVH	Max. elements in VAlH	ULONG	Yes	1	Yes	No	No
NOVI	Max. elements in VALI	ULONG	Yes	1	Yes	No	No
NOVJ	Max. elements in VALJ	ULONG	Yes	1	Yes	No	No
NOVK	Max. elements in VALK	ULONG	Yes	1	Yes	No	No
NOVL	Max. elements in VALL	ULONG	Yes	1	Yes	No	No
NOVM	Max. elements in VALM	ULONG	Yes	1	Yes	No	No
NOVN	Max. elements in VALN	ULONG	Yes	1	Yes	No	No
NOVO	Max. elements in VALO	ULONG	Yes	1	Yes	No	No
NOVP	Max. elements in VALP	ULONG	Yes	1	Yes	No	No
NOVQ	Max. elements in VALQ	ULONG	Yes	1	Yes	No	No
NOVR	Max. elements in VALR	ULONG	Yes	1	Yes	No	No
NOVS	Max. elements in VALS	ULONG	Yes	1	Yes	No	No
NOVT	Max. elements in VALT	ULONG	Yes	1	Yes	No	No
NOVU	Max. elements in VALU	ULONG	Yes	1	Yes	No	No

Output Value Array Size

These fields specify how many array elements the output value fields currently contain.

Field	Summary	Type	DCT	Default	Read	Write	CA PP
NEVA	Num. elements in VALA	ULONG	No	1	Yes	No	No
NEVB	Num. elements in VALB	ULONG	No	1	Yes	No	No
NEVC	Num. elements in VALC	ULONG	No	1	Yes	No	No
NEVD	Num. elements in VALD	ULONG	No	1	Yes	No	No
NEVE	Num. elements in VALE	ULONG	No	1	Yes	No	No
NEVF	Num. elements in VALF	ULONG	No	1	Yes	No	No
NEVG	Num. elements in VALG	ULONG	No	1	Yes	No	No
NEVH	Num. elements in VAlH	ULONG	No	1	Yes	No	No
NEVI	Num. elements in VALI	ULONG	No	1	Yes	No	No
NEVJ	Num. elements in VALJ	ULONG	No	1	Yes	No	No
NEVK	Num. elements in VALK	ULONG	No	1	Yes	No	No
NEVL	Num. elements in VALL	ULONG	No	1	Yes	No	No
NEVM	Num. elements in VALM	ULONG	No	1	Yes	No	No
NEVN	Num. elements in VALN	ULONG	No	1	Yes	No	No
NEVO	Num. elements in VALO	ULONG	No	1	Yes	No	No
NEVP	Num. elements in VALP	ULONG	No	1	Yes	No	No
NEVQ	Num. elements in VALQ	ULONG	No	1	Yes	No	No
NEVR	Num. elements in VALR	ULONG	No	1	Yes	No	No
NEVS	Num. elements in VALS	ULONG	No	1	Yes	No	No
NEVT	Num. elements in VALT	ULONG	No	1	Yes	No	No
NEVU	Num. elements in VALU	ULONG	No	1	Yes	No	No

Old Value Array Size

These fields specify how many array elements the old value fields currently contain.

Field	Summary	Type	DCT	Default	Read	Write	CA PP
ONVA	Num. elements in OVLA	ULONG	No	1	Yes	No	No
ONVB	Num. elements in OVLB	ULONG	No	1	Yes	No	No
ONVC	Num. elements in OVLC	ULONG	No	1	Yes	No	No
ONVD	Num. elements in OVLD	ULONG	No	1	Yes	No	No
ONVE	Num. elements in OVLE	ULONG	No	1	Yes	No	No
ONVF	Num. elements in OVLF	ULONG	No	1	Yes	No	No
ONVG	Num. elements in OVLG	ULONG	No	1	Yes	No	No
ONVH	Num. elements in VAlH	ULONG	No	1	Yes	No	No
ONVI	Num. elements in OVLI	ULONG	No	1	Yes	No	No
ONVJ	Num. elements in OVLJ	ULONG	No	1	Yes	No	No
ONVK	Num. elements in OVLK	ULONG	No	1	Yes	No	No
ONVL	Num. elements in OVLL	ULONG	No	1	Yes	No	No
ONVM	Num. elements in OVLM	ULONG	No	1	Yes	No	No
ONVN	Num. elements in OVLN	ULONG	No	1	Yes	No	No
ONVO	Num. elements in OVLO	ULONG	No	1	Yes	No	No
ONVP	Num. elements in OVLP	ULONG	No	1	Yes	No	No
ONVQ	Num. elements in OVLQ	ULONG	No	1	Yes	No	No
ONVR	Num. elements in OVLR	ULONG	No	1	Yes	No	No
ONVS	Num. elements in OVLS	ULONG	No	1	Yes	No	No
ONVT	Num. elements in OVLT	ULONG	No	1	Yes	No	No
ONVU	Num. elements in OVLU	ULONG	No	1	Yes	No	No

Record Support Routines

init_record

```
long (*init_record)(struct dbCommon *precord, int pass)
```

This routine is called twice at iocInit. On the first call it does the following:

- Calloc sufficient space to hold the number of input scalars and/or arrays defined by the settings of the fields FTA-FTU and NOA-NOU. Initialize fields NE* to the values of the associated NO* field values.
- Calloc sufficient space to hold the number of output scalars and/or arrays defined by the settings of the fields FTVA-FTVU and NOVA-NOVU. For the output fields, also calloc space to hold the previous value of a field. This is required when the decision is made on whether or not to post events.

On the second call, it does the following:

- Initializes SUBL if it is a constant link.
- Initializes each constant input link.
- If the field INAM is set, look-up the address of the routine and call it.
- If the field LFLG is set to IGNORE and SNAM is defined, look up the address of the process routine.

process

```
long (*process)(struct dbCommon *precord)
```

This routine implements the following algorithm:

- If PACT is FALSE, perform normal processing
- If PACT is TRUE, perform asynchronous-completion processing

Normal processing:

- Set PACT to TRUE.
- If the field LFLG is set to READ, get the subroutine name from the SUBL link. If the name is not NULL and it is not the same as the previous subroutine name, look up the subroutine address. Set the old subroutine name, ONAM, equal to the current name, SNAM.
- Fetch the values from the input links.
- Set PACT to FALSE
- If all input-link fetches succeeded, call the routine specified by SNAM.
- Set VAL equal to the return value from the routine specified by SNAM.
- If the SNAM routine set PACT to TRUE, then return. In this case, we presume the routine has arranged that process will be called at some later time for asynchronous completion.
- Set PACT to TRUE.
- If VAL is zero, write the output values using the output links.
- Get the time of processing and put it into the timestamp field.
- If VAL has changed, post a change-of value and log event for this field. If EFLG is set to ALWAYS, post change-of-value and log events for every output field. If EFLG is set to ON CHANGE, post change-of-value and log events for every output field which has changed. In the case of an array, an event will be posted if any single element of the array has changed. If EFLG is set to NEVER, no change-of-value or log events are posted for the output fields.
- Process the record on the end of the forward link, if one exists.
- Set PACT to FALSE.

Asynchronous-completion processing:

- Call the routine specified by SNAM (again).
- Set VAL equal to the return value from the routine specified by SNAM.
- Set PACT to TRUE.
- If VAL is zero, write the output values using the output links.
- Get the time of processing and put it into the timestamp field.
- If VAL has changed, post a change-of value and log event for this field. If EFLG is set to ALWAYS, post change-of-value and log events for every output field. If EFLG is set to ON CHANGE, post change-of-value and log events for every output field which has changed. In the case of an array, an event will be posted if any single element of the array has changed. If EFLG is set to NEVER, no change-of-value or log events are posted for the output fields.
- Process the record on the end of the forward link, if one exists.
- Set PACT to FALSE.

Use of the aSub Record

The aSub record has input-value fields (A-U) and output-value fields (VALA-VALU), which are completely independent. The input-value fields have associated input links (INPA-INPU), and the output-value fields have associated output links (OUTA-OUTU). Both inputs and outputs have type fields (FTA-FTU, FTVA-FTVU, which default to 'DOUBLE') and number-of-element fields (NOA-NOU, NOVA-NOVU, which default to '1'). The output links OUTA-OUTU will only be processed if the subroutine returns a zero (OK) status value.

Example database fragment

To use the A field to read an array from some other record, then, you would need a database fragment that might look something like this:

```
record(aSub,"my_asub_record") {
    field(SNAM,"my_asub_routine")
    ...
    field(FTA, "LONG")
    field(NOA, "100")
    field(INPA, "myWaveform_1 NPP NMS")
    ...
}
```

If you wanted some other record to be able to write to the A field, then you would delete the input link above. If you wanted the A field to hold a scalar value, you would either delete the NOA specification, or specify it as "1".

Example subroutine fragment

The associated subroutine code that uses the A field might look like this:

```
static long my_asub_routine(aSubRecord *prec) {
    long i, *a;
    double sum=0;
    ...
    a = (long *)prec->a;
    for (i=0; i<prec->noa; i++) {
        sum += a[i];
    }
    ...
    return 0; /* process output links */
}
```

Note that the subroutine code must always handle the value fields (A-U, VALA-VALU) as arrays, even if they contain only a single element.

Required export code

Aside from your own code, you must export and register your subroutines so the record can locate them. The simplest way is as follows:

```
#include <registryFunction.h>
#include <epicsExport.h>

static long my_asub_routine(aSubRecord *prec) {
    ...
}
epicsRegisterFunction(my_asub_routine);
```

Required database-definition code

The .dbd file loaded by the ioc must then contain the following line, which tells the linker to include your object file in the IOC binary:

```
function(my_asub_routine)
```

Device support, writing to hardware

The aSub record does not call any device support routines. If you want to write to hardware, you might use your output fields and links to write to some other record that can write to hardware.

Dynamically Changing the User Routine called during Record Processing

The aSub record allows the user to dynamically change which routine is called when the record processes. This can be done in two ways:

- The LFLG field can be set to READ so that the name of the routine is read from the SUBL link. Thus, whatever is feeding this link can change the name of the routine before the aSub record is processed. In this case, the record looks in the symbol table for the symbol name whenever the name of routine fetched from the link changes.
- The LFLG field can be set to IGNORE. In this case, the routine called during record processing is that specified in the SNAM field. Under these conditions, the SNAM field can be changed by a Channel Access write to that field. During development when trying several versions of the routine, it is not necessary to reboot the IOC and reload the database. A new routine can be loaded with the vxWorks ld command, and Channel Access or the dbpf command used to put the name of the routine into the record's SNAM field. The record will look up the symbol name in the symbol table whenever the SNAM field gets modified. The same routine name can even be used as the vxWorks symbol lookup returns the latest version of the code to have been loaded.

1.5.4 Array Analog Input (aai)

The array analog input record type is used to read array data. The array data can contain any of the supported data types. The record is in many ways similar to the waveform record. It allows, however, the device support to allocate the array storage.

Parameter Fields

The record-specific fields are described below, grouped by functionality.

Scan Parameters

The array analog input record has the standard fields for specifying under what circumstances the record will be processed. These fields are described in *Scan Fields*.

Read Parameters

These fields are configurable by the user to specify how and from where the record reads its data. The INP field determines from where the array analog input gets its input. It can be a hardware address, a channel access or database link, or a constant. Only in records that use soft device support can the INP field be a channel access link, a database link, or a constant. Otherwise, the INP field must be a hardware address.

Fields related to waveform reading

The DTYP field must contain the name of the appropriate device support module. The values retrieved from the input link are placed in an array referenced by VAL. (If the INP link is a constant, elements can be placed in the array via dbPuts.) NELM specifies the number of elements that the array will hold, while FTVL specifies the data type of the elements (follow the link in the table below for a list of the available choices).

Field	Summary	Type	DCT	Default	Read	Write	CA PP
DTYP	Device Type	DEVICE	Yes		Yes	Yes	No
INP	Input Specification	INLINK	Yes		Yes	Yes	No
NELM	Number of Elements	ULONG	Yes	1	Yes	No	No
FTVL	Field Type of Value	MENU menuFtype	Yes		Yes	No	No

Operator Display Parameters

These parameters are used to present meaningful data to the operator. They display the value and other parameters of the waveform either textually or graphically.

Fields related to *Operator Display*

EGU is a string of up to 16 characters describing the units that the array data measures. It is retrieved by the `get_units()` record support routine.

The HOPR and LOPR fields set the upper and lower display limits for array elements referenced by the VAL field. Both the `get_graphic_double()` and `get_control_double()` record support routines retrieve these fields.

The PREC field determines the floating point precision with which to display the array values. It is used whenever the `get_precision()` record support routine is called.

See *Fields Common to All Record Types* for more on the record name (NAME) and description (DESC) fields.

Field	Summary	Type	DCT	Default	Read	Write	CA PP
EGU	Engineering Units	STRING [16]	Yes		Yes	Yes	No
HOPR	High Operating Range	DOUBLE	Yes		Yes	Yes	No
LOPR	Low Operating Range	DOUBLE	Yes		Yes	Yes	No
PREC	Display Precision	SHORT	Yes		Yes	Yes	No
NAME	Record Name	STRING [61]	No		Yes	No	No
DESC	Descriptor	STRING [41]	Yes		Yes	Yes	No

Alarm Parameters

The array analog input record has the alarm parameters common to all record types.

Monitor Parameters

These parameters are used to determine when to send monitors placed on the VAL field. The APST and MPST fields are a menu with choices `Always` and `On Change`. The default is `Always`, thus monitors will normally be sent every time the record processes. Selecting `On Change` causes a 32-bit hash of the VAL field buffer to be calculated and compared with the previous hash value every time the record processes; the monitor will only be sent if the hash is different, indicating that the buffer has changed. Note that there is a small chance that two different value buffers might result in the same hash value, so for critical systems `Always` may be a better choice, even though it re-sends duplicate data.

Field	Summary	Type	DCT	Default	Read	Write	CA PP
APST	Post Archive Monitors	MENU *aaiPOST*	Yes		Yes	Yes	No
MPST	Post Value Monitors	MENU *aaiPOST*	Yes		Yes	Yes	No
HASH	Hash of OnChange data.	ULONG	No		Yes	Yes	No

Menu aaiPOST

These are the possible choices for the `APST` and `MPST` fields:

Index	Identifier	Choice String
0	aaiPOST_Always	Always
1	aaiPOST_OnChange	On Change

Run-time Parameters

These parameters are used by the run-time code for processing the array analog input record. They are not configured using a configuration tool. Only the VAL field is modifiable at run-time.

VAL references the array where the array analog input record stores its data. The BPTR field holds the address of the array.

The NORD field holds a counter of the number of elements that have been read into the array.

Field	Summary	Type	DCT	Default	Read	Write	CA PP
VAL	Value	DOUBLE[NELM]	No		Yes	Yes	Yes
BPTR	Buffer Pointer	NOACCESS	No		No	No	No
NORD	Number elements read	ULONG	No		Yes	No	No

Simulation Mode Parameters

The following fields are used to operate the record in simulation mode.

If SIMM (fetched through SIML) is YES, the record is put in SIMS severity and the value is fetched through SIOL. SSCN sets a different SCAN mechanism to use in simulation mode. SDLY sets a delay (in sec) that is used for asynchronous simulation processing.

See *Input Simulation Fields* for more information on simulation mode and its fields.

Field	Summary	Type	DCT	Default	Read	Write	CA PP
SIML	Simulation Mode Link	INLINK	Yes		Yes	Yes	No
SIMM	Simulation Mode	MENU menuYesNo	No		Yes	Yes	No
SIOL	Simulation Input Link	INLINK	Yes		Yes	Yes	No
SIMS	Simulation Mode Severity	MENU menuAlarmSevr	Yes		Yes	Yes	No
SDLY	Sim. Mode Async Delay	DOUBLE	Yes	-1.0	Yes	Yes	No
SSCN	Sim. Mode Scan	MENU menuScan	Yes	65535	Yes	Yes	No

Record Support

Record Support Routines

init_record

```
static long init_record(aaiRecord *prec, int pass)
```

If device support includes an `init_record()` routine it is called, but unlike most record types this occurs in pass 0, which allows the device support to allocate the array buffer itself.

Since EPICS 7.0.5 the device support may return `AAI_DEVINIT_PASS1` to request a second call to its `init_record()` routine in pass 1.

Checks if device support allocated array space. If not, space for the array is allocated using NELM and FTVL. The array address is stored in BPTR.

This routine initializes SIMM with the value of SIML if SIML type is CONSTANT link or creates a channel access link if SIML type is PV_LINK. VAL is likewise initialized if SIOL is CONSTANT or PV_LINK.

This routine next checks to see that device support is available and a device support read routine is defined. If either does not exist, an error message is issued and processing is terminated

process

```
static long process(aaiRecord *prec)
```

See *"Record Processing"* section below.

cvt_dbaddr

```
static long cvt_dbaddr(DBADDR *paddr)
```

This is called by dbNameToAddr. It makes the dbAddr structure refer to the actual buffer holding the result.

get_array_info

```
static long get_array_info(DBADDR *paddr, long *no_elements, long *offset)
```

Obtains values from the array referenced by VAL.

put_array_info

```
static long put_array_info(DBADDR *paddr, long nNew)
```

Writes values into the array referenced by VAL.

get_units

```
static long get_units(DBADDR *paddr, char *units)
```

Retrieves EGU.

get_prec

```
static long get_precision(DBADDR *paddr, long *precision)
```

Retrieves PREC if field is VAL field. Otherwise, calls `recGblGetPrec()`.

get_graphic_double

```
static long get_graphic_double(DBADDR *paddr, struct dbr_grDouble *pgd)
```

Sets the upper display and lower display limits for a field. If the field is VAL the limits are set to HOPR and LOPR, else if the field has upper and lower limits defined they will be used, else the upper and lower maximum values for the field type will be used.

Sets the following values:

```
upper_disp_limit = HOPR
lower_disp_limit = LOPR
```

get_control_double

```
static long get_control_double(DBADDR *paddr, struct dbr_ctrlDouble *pcd)
```

Sets the upper control and the lower control limits for a field. If the field is VAL the limits are set to HOPR and LOPR, else if the field has upper and lower limits defined they will be used, else the upper and lower maximum values for the field type will be used.

Sets the following values

```
upper_ctrl_limit = HOPR
lower_ctrl_limit = LOPR
```

Record Processing

Routine process implements the following algorithm:

1. Check to see that the appropriate device support module exists. If it doesn't, an error message is issued and processing is terminated with the PACT field still set to TRUE. This ensures that processes will no longer be called for this record. Thus error storms will not occur.
2. Call device support read routine `read_aai()`.
3. If PACT has been changed to TRUE, the device support read routine has started but has not completed writing the new value. In this case, the processing routine merely returns, leaving PACT TRUE.
4. Check to see if monitors should be invoked.
   - Alarm monitors are invoked if the alarm status or severity has changed.
   - Archive and value change monitors are invoked if APST or MPST are Always or if the result of the hash calculation is different.
   - NSEV and NSTA are reset to 0.
5. Scan forward link if necessary, set PACT FALSE, and return.

Device Support

Fields Of Interest To Device Support

Each array analog input record record must have an associated set of device support routines. The primary responsibility of the device support routines is to obtain a new array value whenever `read_aai()` is called. The device support routines are primarily interested in the following fields:

Field	Summary	Type	DCT	Default	Read	Write	CA PP
PACT	Record active	UCHAR	No		Yes	No	No
DPVT	Device Private	NOACCESS	No		No	No	No
NSEV	New Alarm Severity	MENU menuAlarmSevr	No		Yes	No	No
NSTA	New Alarm Status	MENU menuAlarmStat	No		Yes	No	No
INP	Input Specification	INLINK	Yes		Yes	Yes	No
NELM	Number of Elements	ULONG	Yes	1	Yes	No	No
FTVL	Field Type of Value	MENU menuFtype	Yes		Yes	No	No
BPTR	Buffer Pointer	NOACCESS	No		No	No	No
NORD	Number elements read	ULONG	No		Yes	No	No

Device Support Routines

Device support consists of the following routines:

report

```
long report(int level)
```

This optional routine is called by the IOC command `dbior` and is passed the report level that was requested by the user. It should print a report on the state of the device support to stdout. The `level` parameter may be used to output increasingly more detailed information at higher levels, or to select different types of information with different levels. Level zero should print no more than a small summary.

init

```
long init(int after)
```

This optional routine is called twice at IOC initialization time. The first call happens before any of the `init_record()` calls are made, with the integer parameter `after` set to 0. The second call happens after all of the `init_record()` calls have been made, with `after` set to 1.

init_record

```
long init_record(dbCommon *precord)
```

This routine is optional. If provided, it is called by the record support's `init_record()` routine in pass 0. The device support may allocate memory for the VAL field's array (enough space for NELM elements of type FTVA) from its own memory pool if desired, and store the pointer to this buffer in the BPTR field. The record will use `calloc()` for this memory allocation if BPTR has not been set by this routine. The routine must return 0 for success, -1 or a error status on failure.

Since EPICS 7.0.5 if this routine returns `AAI_DEVINIT_PASS1` in pass 0, it will be called again in pass 1 with the PACT field set to `AAI_DEVINIT_PASS1`. In pass 0 the PACT field is set to zero (FALSE).

get_ioint_info

```
long get_ioint_info(int cmd, dbCommon *precord, IOSCANPVT *ppvt)
```

This routine is called by the ioEventScan system each time the record is added or deleted from an I/O event scan list. `cmd` has the value (0,1) if the record is being (added to, deleted from) an I/O event list. It must be provided for any device type that can use the ioEvent scanner.

read_aai

```
long read_aai(dbCommon *precord)
```

This routine should provide a new input value. It returns the following values:

- 0: Success.
- Other: Error.

Device Support For Soft Records

The `Soft Channel` device support is provided to read values from other records via the INP link, or to hold array values that are written into it.

If INP is a constant link the array value gets loaded from the link constant by the `record_init()` routine, which also sets NORD. The `read_aai()` routine does nothing in this case.

If INP is a database or channel access link, the `read_aai()` routine gets a new array value from the link and sets NORD.

1.5.5 Array Analog Output (aao)

The array analog output record type is used to write array data. The array data can contain any of the supported data types. The record is in many ways similar to the waveform record but outputs arrays instead of reading them. It also allows the device support to allocate the array storage.

Parameter Fields

The record-specific fields are described below, grouped by functionality.

Scan Parameters

The array analog output record has the standard fields for specifying under what circumstances the record will be processed. These fields are described in *Scan Fields*.

Field	Summary	Type	DCT	Default	Read	Write	CA PP
SCAN	Scan Mechanism	MENU menuScan	Yes		Yes	Yes	No
PHAS	Scan Phase	SHORT	Yes		Yes	Yes	No
EVNT	Event Name	STRING [40]	Yes		Yes	Yes	No
PRIO	Scheduling Priority	MENU menuPriority	Yes		Yes	Yes	No
PINI	Process at iocInit	MENU menuPini	Yes		Yes	Yes	No

Write Parameters

These fields are configurable by the user to specify how and where to the record writes its data. The OUT field determines where the array analog output writes its output. It can be a hardware address, a channel access or database link, or a constant. Only in records that use soft device support can the OUT field be a channel access link, a database link, or a constant. Otherwise, the OUT field must be a hardware address. See Address Specification for information on the format of hardware addresses and database links.

Fields related to array writing

The DTYP field must contain the name of the appropriate device support module. The values in the array referenced by are written to the location specified in the OUT field. (If the OUT link is a constant, no data are written.) NELM specifies the maximum number of elements that the array can hold, while FTVL specifies the data type of the elements (follow the link in the table below for a list of the available choices).

Field	Summary	Type	DCT	Default	Read	Write	CA PP
DTYP	Device Type	DEVICE	Yes		Yes	Yes	No
OUT	Output Specification	OUTLINK	Yes		Yes	Yes	No
NELM	Number of Elements	ULONG	Yes	1	Yes	No	No
FTVL	Field Type of Value	MENU menuFtype	Yes		Yes	No	No

Operator Display Parameters

These parameters are used to present meaningful data to the operator. They display the value and other parameters of the waveform either textually or graphically.

Fields related to *Operator Display*

EGU is a string of up to 16 characters describing the units that the array data measures. It is retrieved by the `get_units` record support routine.

The HOPR and LOPR fields set the upper and lower display limits for array elements referenced by the VAL field. Both the `get_graphic_double` and `get_control_double` record support routines retrieve these fields.

The PREC field determines the floating point precision with which to display the array values. It is used whenever the `get_precision` record support routine is called.

See *Fields Common to All Record Types* for more on the record name (NAME) and description (DESC) fields.

Field	Summary	Type	DCT	Default	Read	Write	CA PP
EGU	Engineering Units	STRING [16]	Yes		Yes	Yes	No
HOPR	High Operating Range	DOUBLE	Yes		Yes	Yes	No
LOPR	Low Operating Range	DOUBLE	Yes		Yes	Yes	No
PREC	Display Precision	SHORT	Yes		Yes	Yes	No
NAME	Record Name	STRING [61]	No		Yes	No	No
DESC	Descriptor	STRING [41]	Yes		Yes	Yes	No

Alarm Parameters

The array analog output record has the alarm parameters common to all record types.

Monitor Parameters

These parameters are used to determine when to send monitors placed on the VAL field. The APST and MPST fields are a menu with choices "Always" and "On Change". The default is "Always", thus monitors will normally be sent every time the record processes. Selecting "On Change" causes a 32-bit hash of the VAL field buffer to be calculated and compared with the previous hash value every time the record processes; the monitor will only be sent if the hash is different, indicating that the buffer has changed. Note that there is a small chance that two different value buffers might result in the same hash value, so for critical systems "Always" may be a better choice, even though it re-sends duplicate data.

Record fields related to *Monitor Parameters*

Field	Summary	Type	DCT	Default	Read	Write	CA PP
APST	Post Archive Monitors	MENU *aaoPOST*	Yes		Yes	Yes	No
MPST	Post Value Monitors	MENU *aaoPOST*	Yes		Yes	Yes	No
HASH	Hash of OnChange data.	ULONG	No		Yes	Yes	No

Menu aaoPOST

These are the choices available for the `APST` and `MPST` fields

Index	Identifier	Choice String
0	aaoPOST_Always	Always
1	aaoPOST_OnChange	On Change

Run-time Parameters

These parameters are used by the run-time code for processing the array analog output record. They are not configured using a configuration tool. Only the VAL field is modifiable at run-time.

VAL references the array where the array analog output record stores its data. The BPTR field holds the address of the array.

The NORD field holds a counter of the number of elements that have been written to the output,

Field	Summary	Type	DCT	Default	Read	Write	CA PP
VAL	Value	DOUBLE[]	No		Yes	Yes	Yes
BPTR	Buffer Pointer	NOACCESS	No		No	No	No
NORD	Number elements read	ULONG	No		Yes	No	No
OMSL	Output Mode Select	MENU menuOmsl	Yes		Yes	Yes	No
DOL	Desired Output Link	INLINK	Yes		Yes	Yes	No

The following steps are performed in order during record processing.

Fetch Value

The OMSL menu field is used to determine whether the DOL link field should be used during processing or not:

- If OMSL is `supervisory` the DOL field are not used. The new output value is taken from the VAL field, which may have been set from elsewhere.
- If OMSL is `closed_loop` the DOL link field is read to obtain a value.

Note: The OMSL and DOL fields were added to the aaoRecord in Base 7.0.7.

Simulation Mode Parameters

The following fields are used to operate the record in simulation mode.

If SIMM (fetched through SIML) is YES, the record is put in SIMS severity and the value is written through SIOL. SSCN sets a different SCAN mechanism to use in simulation mode. SDLY sets a delay (in sec) that is used for asynchronous simulation processing.

See *Output Simulation Fields* for more information on simulation mode and its fields.

Field	Summary	Type	DCT	Default	Read	Write	CA PP
SIML	Simulation Mode Link	INLINK	Yes		Yes	Yes	No
SIMM	Simulation Mode	MENU menuYesNo	No		Yes	Yes	No
SIOL	Simulation Output Link	OUTLINK	Yes		Yes	Yes	No
SIMS	Simulation Mode Severity	MENU menuAlarmSevr	Yes		Yes	Yes	No
SDLY	Sim. Mode Async Delay	DOUBLE	Yes	-1.0	Yes	Yes	No
SSCN	Sim. Mode Scan	MENU menuScan	Yes	65535	Yes	Yes	No

Record Support

Record Support Routines

init_record

```
static long init_record(aaoRecord *prec, int pass)
```

If device support includes `init_record()`, it is called.

Checks if device support allocated array space. If not, space for the array is allocated using NELM and FTVL. The array address is stored in the record.

This routine initializes SIMM with the value of SIML if SIML type is CONSTANT link or creates a channel access link if SIML type is PV_LINK. VAL is likewise initialized if SIOL is CONSTANT or PV_LINK.

This routine next checks to see that device support is available and a device support write routine is defined. If either does not exist, an error message is issued and processing is terminated

process

```
static long process(aaoRecord *prec)
```

See *"Record Processing"* section below.

cvt_dbaddr

```
static long cvt_dbaddr(DBADDR *paddr)
```

This is called by dbNameToAddr. It makes the dbAddr structure refer to the actual buffer holding the result.

get_array_info

```
static long get_array_info(DBADDR *paddr, long *no_elements, long *offset)
```

Obtains values from the array referenced by VAL.

put_array_info

```
static long put_array_info(DBADDR *paddr, long nNew)
```

Writes values into the array referenced by VAL.

get_units

```
static long get_units(DBADDR *paddr, char *units)
```

Retrieves EGU.

get_prec

```
static long get_precision(DBADDR *paddr, long *precision)
```

Retrieves PREC if field is VAL field. Otherwise, calls `recGblGetPrec()`.

get_graphic_double

```
static long get_graphic_double(DBADDR *paddr, struct dbr_grDouble *pgd)
```

Sets the upper display and lower display limits for a field. If the field is VAL the limits are set to HOPR and LOPR, else if the field has upper and lower limits defined they will be used, else the upper and lower maximum values for the field type will be used.

Sets the following values:

```
upper_disp_limit = HOPR
lower_disp_limit = LOPR
```

get_control_double

```
static long get_control_double(DBADDR *paddr, struct dbr_ctrlDouble *pcd)
```

Sets the upper control and the lower control limits for a field. If the field is VAL the limits are set to HOPR and LOPR, else if the field has upper and lower limits defined they will be used, else the upper and lower maximum values for the field type will be used.

Sets the following values

```
upper_ctrl_limit = HOPR
lower_ctrl_limit = LOPR
```

Record Processing

Routine process implements the following algorithm:

1. Check to see that the appropriate device support module exists. If it doesn't, an error message is issued and processing is terminated with the PACT field still set to TRUE. This ensures that processes will no longer be called for this record. Thus error storms will not occur.
2. Call device support write routine `write_aao`.
3. If PACT has been changed to TRUE, the device support read routine has started but has not completed writing the new value. In this case, the processing routine merely returns, leaving PACT TRUE.
4. Check to see if monitors should be invoked.
   - Alarm monitors are invoked if the alarm status or severity has changed.
   - Archive and value change monitors are invoked if APST or MPST are Always or if the result of the hash calculation is different.
   - NSEV and NSTA are reset to 0.
5. Scan forward link if necessary, set PACT FALSE, and return.

Device Support

Fields Of Interest To Device Support

Each array analog output record record must have an associated set of device support routines. The primary responsibility of the device support routines is to write the array data value whenever `write_aao()` is called. The device support routines are primarily interested in the following fields:

Field	Summary	Type	DCT	Default	Read	Write	CA PP
PACT	Record active	UCHAR	No		Yes	No	No
DPVT	Device Private	NOACCESS	No		No	No	No
NSEV	New Alarm Severity	MENU menuAlarmSevr	No		Yes	No	No
NSTA	New Alarm Status	MENU menuAlarmStat	No		Yes	No	No
OUT	Output Specification	OUTLINK	Yes		Yes	Yes	No
NELM	Number of Elements	ULONG	Yes	1	Yes	No	No
FTVL	Field Type of Value	MENU menuFtype	Yes		Yes	No	No
BPTR	Buffer Pointer	NOACCESS	No		No	No	No
NORD	Number elements read	ULONG	No		Yes	No	No

Device Support Routines

Device support consists of the following routines:

report

```
long report(int level)
```

This optional routine is called by the IOC command `dbior` and is passed the report level that was requested by the user. It should print a report on the state of the device support to stdout. The `level` parameter may be used to output increasingly more detailed information at higher levels, or to select different types of information with different levels. Level zero should print no more than a small summary.

init

```
long init(int after)
```

This optional routine is called twice at IOC initialization time. The first call happens before any of the `init_record()` calls are made, with the integer parameter `after` set to 0. The second call happens after all of the `init_record()` calls have been made, with `after` set to 1.

init_record

```
long init_record(dbCommon *precord)
```

This routine is optional. If provided, it is called by the record support `init_record()` routine.

get_ioint_info

```
long get_ioint_info(int cmd, dbCommon *precord, IOSCANPVT *ppvt)
```

This routine is called by the ioEventScan system each time the record is added or deleted from an I/O event scan list. `cmd` has the value (0,1) if the record is being (added to, deleted from) an I/O event list. It must be provided for any device type that can use the ioEvent scanner.

write_aao

```
long write_aao(dbCommon *precord)
```

This routine must write the array data to output. It returns the following values:

- 0: Success.
- Other: Error.

Device Support For Soft Records

The `Soft Channel` device support module is provided to write values to other records and store them in arrays. If OUT is a constant link, then `write_aao()` does nothing. In this case, the record can be used to hold arrays written via dbPuts. If OUT is a database or channel access link, the array value is written to the link. NORD is set to the number of items in the array.

If the OUT link type is constant, then NORD is set to zero.

1.5.6 Binary Input Record (bi)

This record type is normally used to obtain a binary value of 0 or 1. Most device support modules obtain values from hardware and place the value in RVAL. For these devices, record processing sets VAL = (0,1) if RVAL is (0, not 0). Device support modules may optionally read a value directly from VAL.

Soft device modules are provided to obtain input via database or channel access links via dbPutField or dbPutLink requests. Two soft device support modules are provided: `Soft Channel` and `Raw Soft Channel`. The first allows VAL to be an arbitrary unsigned short integer. The second reads the value into RVAL just like normal hardware modules.

Parameter Fields

The record-specific fields are described below, grouped by functionality.

Scan Parameters

The binary input record has the standard fields for specifying under what circumstances the record will be processed. These fields are described in *Scan Fields*.

Field	Summary	Type	DCT	Default	Read	Write	CA PP
SCAN	Scan Mechanism	MENU menuScan	Yes		Yes	Yes	No
PHAS	Scan Phase	SHORT	Yes		Yes	Yes	No
EVNT	Event Name	STRING [40]	Yes		Yes	Yes	No
PRIO	Scheduling Priority	MENU menuPriority	Yes		Yes	Yes	No
PINI	Process at iocInit	MENU menuPini	Yes		Yes	Yes	No

Read and Convert Parameters

The read and convert fields determine where the binary input gets its input from and how to convert the raw signal to engineering units. The INP field contains the address from where device support retrieves the value. If the binary input record gets its value from hardware, the address of the card must be entered in the INP field, and the name of the device support module must be entered in the DTYP field. See Address Specification for information on the format of the hardware address.

For records that specify `Soft Channel` or `Raw Soft Channel` device support routines, the INP field can be a channel or a database link, or a constant. If a constant, VAL can be changed directly by dbPuts. See Address Specification for information on the format of database and channel access addresses. Also, see *"Device Support for Soft Records"* in this chapter for information on soft device support.

If the record gets its values from hardware or uses the `Raw Soft Channel` device support, the device support routines place the value in the RVAL field which is then converted using the process described in the next section.

Field	Summary	Type	DCT	Default	Read	Write	CA PP
INP	Input Specification	INLINK	Yes		Yes	Yes	No
DTYP	Device Type	DEVICE	Yes		Yes	Yes	No
ZNAM	Zero Name	STRING [26]	Yes		Yes	Yes	Yes
ONAM	One Name	STRING [26]	Yes		Yes	Yes	Yes
RVAL	Raw Value	ULONG	No		Yes	Yes	Yes
VAL	Current Value	ENUM	Yes		Yes	Yes	Yes

Conversion Fields

The VAL field is set equal to (0,1) if the RVAL field is (0, not 0), unless the device support module reads a value directly into VAL or the `Soft Channel` device support is used. The value can also be fetched as one of the strings specified in the ZNAM or ONAM fields. The ZNAM field has a string that corresponds to the 0 state, so when the value is fetched as this string, `put_enum_str()` will return a 0. The ONAM field hold the string that corresponds to the 1 state, so when the value is fetched as this string, `put_enum_str()` returns a 1.

Field	Summary	Type	DCT	Default	Read	Write	CA PP
ZNAM	Zero Name	STRING [26]	Yes		Yes	Yes	Yes
ONAM	One Name	STRING [26]	Yes		Yes	Yes	Yes

Operator Display Parameters

These parameters are used to present meaningful data to the operator. The `get_enum_str()` record support routine can retrieve the state string corresponding to the VAL's state. If the value is 1, `get_enum_str()` will return the string in the ONAM field; and if 0, `get_enum_str()` will return the ZNAM string.

See *Fields Common to All Record Types* for more on the record name (NAME) and description (DESC) fields.

Field	Summary	Type	DCT	Default	Read	Write	CA PP
ZNAM	Zero Name	STRING [26]	Yes		Yes	Yes	Yes
ONAM	One Name	STRING [26]	Yes		Yes	Yes	Yes
NAME	Record Name	STRING [61]	No		Yes	No	No
DESC	Descriptor	STRING [41]	Yes		Yes	Yes	No

Alarm Parameters

These parameters are used to determine if the binary input is in alarm condition and to determine the severity of that condition. The possible alarm conditions for binary inputs are the SCAN, READ state alarms, and the change of state alarm. The SCAN and READ alarms are called by the device supprt routines.

The user can choose the severity of each state in the ZSV and OSV fields. The possible values for these fields are `NO_ALARM`, `MINOR`, and `MAJOR`. The ZSV field holds the severity for the zero state; OSV, for the one state. COSV causes an alarm whenever the state changes between 0 and 1 and the severity is configured as MINOR or MAJOR.

See Alarm Specification for a complete explanation of record alarms and of the standard fields. *Alarm Fields* lists other fields related to alarms that are common to all record types.

Field	Summary	Type	DCT	Default	Read	Write	CA PP
ZSV	Zero Error Severity	MENU menuAlarmSevr	Yes		Yes	Yes	Yes
OSV	One Error Severity	MENU menuAlarmSevr	Yes		Yes	Yes	Yes
COSV	Change of State Svr	MENU menuAlarmSevr	Yes		Yes	Yes	Yes

Run-time Parameters

These parameters are used by the run-time code for processing the binary input. They are not configured using a database configuration tool.

ORAW is used to determine if monitors should be triggered for RVAL at the same time they are triggered for VAL.

MASK is given a value by ithe device support routines. This value is used to manipulate the record's value, but is only the concern of the hardware device support routines.

The LALM fields holds the value of the last occurence of the change of state alarm. It is used to implement the change of state alarm, and thus only has meaning if COSV is MAJOR or MINOR.

The MSLT field is used by the `process()` record support routine to determine if archive and value change monitors are invoked. They are if MSLT is not equal to VAL.

Field	Summary	Type	DCT	Default	Read	Write	CA PP
ORAW	prev Raw Value	ULONG	No		Yes	No	No
MASK	Hardware Mask	ULONG	No		Yes	No	No
LALM	Last Value Alarmed	USHORT	No		Yes	No	No
MLST	Last Value Monitored	USHORT	No		Yes	No	No

Simulation Mode Parameters

The following fields are used to operate the record in simulation mode.

If SIMM (fetched through SIML) is YES or RAW, the record is put in SIMS severity and the value is fetched through SIOL (buffered in SVAL). If SIMM is YES, SVAL is written to VAL without conversion, if SIMM is RAW, SVAL is trancated to RVAL and converted. SSCN sets a different SCAN mechanism to use in simulation mode. SDLY sets a delay (in sec) that is used for asynchronous simulation processing.

See *Input Simulation Fields* for more information on simulation mode and its fields.

Field	Summary	Type	DCT	Default	Read	Write	CA PP
SIML	Simulation Mode Link	INLINK	Yes		Yes	Yes	No
SIMM	Simulation Mode	MENU menuSimm	No		Yes	Yes	No
SIOL	Simulation Input Link	INLINK	Yes		Yes	Yes	No
SVAL	Simulation Value	ULONG	No		Yes	Yes	No
SIMS	Simulation Mode Severity	MENU menuAlarmSevr	Yes		Yes	Yes	No
SDLY	Sim. Mode Async Delay	DOUBLE	Yes	-1.0	Yes	Yes	No
SSCN	Sim. Mode Scan	MENU menuScan	Yes	65535	Yes	Yes	No

Record Support

Record Support Routines

```
long init_record(struct dbCommon *precord, int pass);
```

This routine initializes SIMM with the value of SIML if SIML type is a CONSTANT link or creates a channel access link if SIML type is PV_LINK. SVAL is likewise initialized if SIOL is a CONSTANT or PV_LINK.

This routine next checks to see that device support is available and a device support routine is defined. If neither exist, an error is issued and processing is terminated.

If device support includes `init_record()`, it is called.

```
long process(struct dbCommon *precord);
```

See *"Record Processing"* below.

```
long get_enum_str(const struct dbAddr *paddr, char *pbuffer);
```

Retrieves ASCII string corresponding to VAL.

```
long get_enum_strs(const struct dbAddr *paddr, struct dbr_enumStrs *p);
```

Retrieves ASCII strings for ZNAM and ONAM.

```
long put_enum_str(const struct dbAddr *paddr, const char *pbuffer);
```

Check if string matches ZNAM or ONAM, and if it does, sets VAL.

Record Processing

Routine process implements the following algorithm:

- 1.

Check to see that the appropriate device support module exists. If it doesn't, an error message is issued and processing is terminated with the PACT field still set to TRUE. This ensures that processes will no longer be called for this record. Thus error storms will not occur.

- 2.

`readValue()` is called. See *"Input Records"* for details.

- 3.

If PACT has been changed to TRUE, the device support read routine has started but has not completed reading a new input value. In this case, the processing routine merely returns, leaving PACT TRUE.

- 4.

Convert.

- status = read_bi
- PACT = TRUE
- `recGblGetTimeStamp()` is called.
- if status is 0, then set VAL=(0,1) if RVAL is (0, not 0) and UDF = False.
- if status is 2, set status = 0
- 5.

Check alarms: This routine checks to see if the new VAL causes the alarm status and severity to change. If so, NSEV, NSTA and LALM are set. Note that if VAL is greater than 1, no checking is performed.

- 6.

Check if monitors should be invoked:

- Alarm monitors are invoked if the alarm status or severity has changed.
- Archive and value change monitors are invoked if MSLT is not equal to VAL.
- Monitors for RVAL are checked whenever other monitors are invoked.
- NSEV and NSTA are reset to 0.
- 7.

Scan forward link if necessary, set PACT FALSE, and return.

Device Support

Fields of Interest to Device Support

Each binary input record must have an associated set of device support routines. The primary resposibility of the device support routines is to obtain a new raw input value whenever `read_bi()` is called. The device support routines are primarily interested in the following fields:

Field	Summary	Type	DCT	Default	Read	Write	CA PP
PACT	Record active	UCHAR	No		Yes	No	No
DPVT	Device Private	NOACCESS	No		No	No	No
UDF	Undefined	UCHAR	Yes	1	Yes	Yes	Yes
NSEV	New Alarm Severity	MENU menuAlarmSevr	No		Yes	No	No
NSTA	New Alarm Status	MENU menuAlarmStat	No		Yes	No	No
VAL	Current Value	ENUM	Yes		Yes	Yes	Yes
INP	Input Specification	INLINK	Yes		Yes	Yes	No
RVAL	Raw Value	ULONG	No		Yes	Yes	Yes
MASK	Hardware Mask	ULONG	No		Yes	No	No

Device Support routines

Device support consists of the following routines:

```
long report(int level);
```

This optional routine is called by the IOC command `dbior` and is passed the report level that was requested by the user. It should print a report on the state of the device support to stdout. The `level` parameter may be used to output increasingly more detailed information at higher levels, or to select different types of information with different levels. Level zero should print no more than a small summary.

```
long init(int after);
```

This optional routine is called twice at IOC initialization time. The first call happens before any of the `init_record()` calls are made, with the integer parameter `after` set to 0. The second call happens after all of the `init_record()` calls have been made, with `after` set to 1.

```
long init_record(struct dbCommon *precord);
```

This routine is optional. If provided, it is called by the record support `init_record()` routine.

```
long get_ioint_info(int cmd, struct dbCommon *precord, IOSCANPVT *ppvt);
```

This routine is called by the ioEventScan system each time the record is added or deleted from an I/O event scan list. `cmd` has the value (0,1) if the record is being (added to, deleted from) and I/O event list. It must be provided for any device type that can use the ioEvent scanner.

```
long read_bi(struct dbCommon *precord);
```

This routine must provide a new input value. It returns the following values:

- 0: Success. A new raw value is placed in RVAL. The record support module forces VAL to be (0,1) if RVAL is (0, not 0).
- 2: Success, but don't modify VAL.

- Other: Error.

Device Support for Soft Records

Two soft device support modules, Soft Channel and Raw Soft Channel, are provided for input records not related to actual hardware devices. The INP link type must be either CONSTANT, DB_LINK, or CA_LINK.

Soft Channel

`read_bi()` always returns a value of 2, which means that no conversion is performed.

If the INP link type is CONSTANT, then the constant value is stored in VAL by `init_record()`, and the UDF is set to FALSE. VAL can be changed via `dbPut()` requests. If the INP link type is PV_LINK, the `dbCaAddInlink()` is called by `init_record()`.

`read_bi()` calls `dbGetLinkValue` to read the current value of VAL. See *"Soft Input"* for details.

If the return status of `dbGetLinkValue()` is zero, then `read_bi()` sets UDF to FALSE. The status of `dbGetLinkValue()` is returned.

Raw Soft Channel

This module is like the previous except that values are read into RVAL.

`read_bi()` returns a value of 0. Thus the record processing routine will force VAL to be 0 or 1.

1.5.7 Binary Output Record (bo)

The normal use for this record type is to store a simple bit (0 or 1) value to be sent to a Digital Output module. It can also be used to write binary values into other records via database or channel access links. This record can implement both latched and momentary binary outputs depending on how the HIGH field is configured.

Scan Parameters

The binary output record has the standard fields for specifying under what circumstances the record will be processed. These fields are described in *Scan Fields*.

Field	Summary	Type	DCT	Default	Read	Write	CA PP
SCAN	Scan Mechanism	MENU menuScan	Yes		Yes	Yes	No
PHAS	Scan Phase	SHORT	Yes		Yes	Yes	No
EVNT	Event Name	STRING [40]	Yes		Yes	Yes	No
PRIO	Scheduling Priority	MENU menuPriority	Yes		Yes	Yes	No
PINI	Process at iocInit	MENU menuPini	Yes		Yes	Yes	No

Desired Output Parameters

The binary output record must specify where its desired output originates. The desired output needs to be in engineering units.

The first field that determines where the desired output originates is the output mode select (OMSL) field, which can have two possible values: `losed_loop` or `supervisory`. If `supervisory` is specified, the value in the VAL field can be set externally via dbPuts at run-time. If `closed_loop` is specified, the VAL field's value is obtained from the address specified in the Desired Output Link (DOL) field which can be a database link or a channel access link, but not a constant. To achieve continuous control, a database link to a control algorithm record should be entered in the DOL field.

See Address Specification for information on hardware addresses and links.

Field	Summary	Type	DCT	Default	Read	Write	CA PP
DOL	Desired Output Link	INLINK	Yes		Yes	Yes	No
OMSL	Output Mode Select	MENU menuOmsl	Yes		Yes	Yes	No

Convert and Write Parameters

These parameters are used to determine where the binary output writes to and how to convert the engineering units to a raw signal. After VAL is set and forced to be either 1 or 0, as the result of either a dbPut or a new value being retrieved from the link in the DOL field, then what happens next depends on which device support routine is used and how the HIGH field is configured.

If the `Soft Channel` device support routine is specified, then the device support routine writes the VAL field's value to the address specified in the OUT field. Otherwise, RVAL is the value written by the device support routines after being converted.

If VAL is equal to 0, then the record processing routine sets RVAL equal to zero. When VAL is not equal to 0, then RVAL is set equal to the value contained in the MASK field. (MASK is set by the device support routines and is of no concern to the user.) Also, when VAL is not 0 and after RVAL is set equal to MASK, the record processing routine checks to see if the HIGH field is greater than 0. If it is, then the routine will process the record again with VAL set to 0 after the number of seconds specified by HIGH. Thus, HIGH implements a momentary output which changes the state of the device back to 0 after *N* number of seconds.

Field	Summary	Type	DCT	Default	Read	Write	CA PP
DTYP	Device Type	DEVICE	Yes		Yes	Yes	No
OUT	Output Specification	OUTLINK	Yes		Yes	Yes	No
VAL	Current Value	ENUM	Yes		Yes	Yes	Yes
RVAL	Raw Value	ULONG	No		Yes	Yes	Yes
HIGH	Seconds to Hold High	DOUBLE	Yes		Yes	Yes	No
ZNAM	Zero Name	STRING [26]	Yes		Yes	Yes	Yes
ONAM	One Name	STRING [26]	Yes		Yes	Yes	Yes

Conversion Parameters

The ZNAM field has the string that corresponds to the 0 state, and the ONAM field holds the string that corresponds to the 1 state. These fields, other than being used to tell the operator what each state represents, are used to perform conversions if the value fetched by DOL is a string. If it is, VAL is set to the state which corresponds to that string. For instance, if the value fetched is the string "Off" and the ZNAM string is "Off," then VAL is set to 0.

After VAL is set, if VAL is equal to 0, then the record processing routine sets RVAL equal to zero. When VAL is not equal to 0, then RVAL is set equal to the value contained in the MASK field. (Mask is set by the device support routines and is of no concern to the user.) Also when VAL is equal to 1 and after RVAL is set equal to MASK, the record processing routine checks to see if the HIGH field is greater than 0. If it is, then the routine processes the record again with VAL=0 after the number of seconds specified by HIGH. Thus, HIGH implements a latched output which changes the state of the device or link to 1, then changes it back to 0 after *N* number of seconds.

Field	Summary	Type	DCT	Default	Read	Write	CA PP
ZNAM	Zero Name	STRING [26]	Yes		Yes	Yes	Yes
ONAM	One Name	STRING [26]	Yes		Yes	Yes	Yes
HIGH	Seconds to Hold High	DOUBLE	Yes		Yes	Yes	No

Output Specification

The OUT field specifies where the binary output record writes its output. It must specify the address of an I/O card if the record sends its output to hardware, and the DTYP field must contain the corresponding device support module. Be aware that the address format differs according to the I/O bus used. See Address Specification for information on the format of hardware addresses.

Otherwise, if the record is configured to use the soft device support modules, then it can be either a database link, a channel access link, or a constant. Be aware that nothing will be written when OUT is a constant. See Address Specification for information on the format of the database and channel access addresses. Also, see *"Device Support For Soft Records"* in this chapter for more on output to other records.

Operator Display Parameters

These parameters are used to present meaningful data to the operator, The `get_enum_str()` record support routine can retrieve the state string corresponding to the VAL's state. So, if the value is 1, `get_enum_str()` will return the string in the ONAM field: and if 0, `get_enum_str()` will return the ZNAM string.

See *Fields Common to All Record Types* for more on the record name (NAME) and description (DESC) fields.

Field	Summary	Type	DCT	Default	Read	Write	CA PP
ZNAM	Zero Name	STRING [26]	Yes		Yes	Yes	Yes
ONAM	One Name	STRING [26]	Yes		Yes	Yes	Yes
NAME	Record Name	STRING [61]	No		Yes	No	No
DESC	Descriptor	STRING [41]	Yes		Yes	Yes	No

Alarm Parameters

These parameters are used to determine the binary output's alarm condition and to determine the severity of that condition. The possible alarm conditions for binary outputs are the SCAN, READ, INVALID and state alarms. The user can configure the state alarm conditions using these fields.

The possible values for these fields are `NO_ALARM`, `MINOR`, and `MAJOR`. The ZSV holds the severity for the zero state; OSV for the one state. COSV is used to cause an alarm whenever the state changes between states (0-1, 1-0) and its severity is configured as MINOR or MAJOR.

See *Invalid Output Action Fields* for more information on the IVOA and IVOV fields. *Alarm Fields* lists other fields related to alarms that are common to all record types.

Field	Summary	Type	DCT	Default	Read	Write	CA PP
ZSV	Zero Error Severity	MENU menuAlarmSevr	Yes		Yes	Yes	Yes
OSV	One Error Severity	MENU menuAlarmSevr	Yes		Yes	Yes	Yes
COSV	Change of State Sevr	MENU menuAlarmSevr	Yes		Yes	Yes	Yes
IVOA	INVALID outpt action	MENU menuIvoa	Yes		Yes	Yes	No
IVOV	INVALID output value	USHORT	Yes		Yes	Yes	No

Run-Time Parameters

These parameters are used by the run-time code for processiong the binary output. They are not configurable using a configuration tool. They represent the current state of the binary output.

ORAW is used to determine if monitors should be triggered for RVAL at the same time they are triggered for VAL.

MASK is given a value by the device support routines and should not concern the user.

The RBV field is also set by device support. It is the actual read back value obtained from the hardware itself or from the associated device driver.

The ORBV field is used to decide if monitors should be triggered for RBV at the same time monitors are triggered for changes in VAL.

The LALM field holds the value of the last occurrence of the change of state alarm. It is used to implement the change of state alarm, and thus only has meaning if COSV is MINOR or MAJOR.

The MLST is used by the `process()` record support routine to determine if archive and value change monitors are invoked. They are if MLST is not equal to VAL.

The WPDT field is a private field for honoring seconds to hold HIGH.

Field	Summary	Type	DCT	Default	Read	Write	CA PP
ORAW	prev Raw Value	ULONG	No		Yes	No	No
MASK	Hardware Mask	ULONG	No		Yes	No	No
RBV	Readback Value	ULONG	No		Yes	No	No
ORBV	Prev Readback Value	ULONG	No		Yes	No	No
LALM	Last Value Alarmed	USHORT	No		Yes	No	No
MLST	Last Value Monitored	USHORT	No		Yes	No	No
RPVT	Record Private	NOACCESS	No		No	No	No
WDPT	Watch Dog Timer ID	NOACCESS	No		No	No	No

Simulation Mode Parameters

The following fields are used to operate the record in simulation mode.

If SIMM (fetched through SIML, if populated) is YES, the record is put in SIMS severity and the unconverted value is written through SIOL. If SIMM is RAW, the value is converted and RVAL is written. SSCN sets a different SCAN mechanism to use in simulation mode. SDLY sets a delay (in sec) that is used for asynchronous simulation processing.

See *Output Simulation Fields* for more information on simulation mode and its fields.

Field	Summary	Type	DCT	Default	Read	Write	CA PP
SIML	Simulation Mode Link	INLINK	Yes		Yes	Yes	No
SIMM	Simulation Mode	MENU menuSimm	No		Yes	Yes	No
SIOL	Simulation Output Link	OUTLINK	Yes		Yes	Yes	No
SIMS	Simulation Mode Severity	MENU menuAlarmSevr	Yes		Yes	Yes	No
SDLY	Sim. Mode Async Delay	DOUBLE	Yes	-1.0	Yes	Yes	No
SSCN	Sim. Mode Scan	MENU menuScan	Yes	65535	Yes	Yes	No

Record Support

Record Support Routines

`init_record`

This routine initializes SIMM if SIML is a constant or creates a channel access link if SIML is PV_LINK. If SIOL is a PV_LINK a channel access link is created.

This routine next checks to see that device support is available. The routine next checks to see if the device support write routine is defined.

If either device support or the device support write routine does not exist, and error message is issued and processing is terminated.

If DOL is a constant, then VAL is initialized to 1 if its value is nonzero or initialzed to 0 if DOL is zero, and UDF is set to FALSE.

If device support includes `init_record()`, it is called. VAL is set using RVAL, and UDF is set to FALSE.

`process`

See next section.

`get_enum_str`

Retrieves ASCII string corresponding to VAL.

`get_enum_strs`

Retrieves ASCII strings for ZNAM and ONAM.

`put_enum_str`

Checks if string matches ZNAM or ONAM, and if it does, sets VAL.

Record Processing

Routine process implements the following algorithm:

- 1.

Check to see that the appropriate device support module exists. If it doesn't, an error message is issued and processing is terminated with the PACT field still set to TRUE. This ensures that processes will no longer be called for this record. Thus error storms will not occur.

- 2.

If PACT is FALSE

- If DOL holds a link and OMSL is `closed_loop`
  - get values from DOL
  - check for link alarm
  - force VAL to be 0 or 1
  - if MASK is defined
    * if VAL is 0 set RVAL = 0
  - else set RVAL = MASK

- 3.

Check alarms: This routine checks to see if the new VAL causes the alarm status and severity to change. If so, NSEV, NSTA, and LALM are set.

- 4.

Check severity and write the new value. See *Invalid Output Action Fields* for more information on how INVALID alarms affect output.

- 5.

If PACT has been changed to TRUE, the device support write output routine has started but has not completed writing the new value. in this case, the processing routine merely returns, leaving PACT TRUE.

- 6.

Check WAIT. If VAL is 1 and WAIT is greater than 0, process again with a VAL=0 after WAIT seconds.

- 7.

Check to see if monitors should be invoked.

- Alarm monitors are invoked if the alarm status or severity has changed.
- Archive and value change monitors are invoked if MLST is not equal to VAL.
- Monitors for RVAL and for RBV are checked whenever other monitors are invoked.
- NSEV and NSTA are reset to 0.
- 8 Scan forward link if necessary, set PACT FALSE, and return

Device support

Fields Of Interest To Device Support

Each binary output record must have an associated set of device support routines. The primary responsibility of the device support routines is to write a new value whenever `write_bo()` is called. The device support routines are primarily interested in the following fields:

Field	Summary	Type	DCT	Default	Read	Write	CA PP
PACT	Record active	UCHAR	No		Yes	No	No
DPVT	Device Private	NOACCESS	No		No	No	No
NSEV	New Alarm Severity	MENU menuAlarmSevr	No		Yes	No	No
NSTA	New Alarm Status	MENU menuAlarmStat	No		Yes	No	No
VAL	Current Value	ENUM	Yes		Yes	Yes	Yes
OUT	Output Specification	OUTLINK	Yes		Yes	Yes	No
RVAL	Raw Value	ULONG	No		Yes	Yes	Yes
MASK	Hardware Mask	ULONG	No		Yes	No	No
RBV	Readback Value	ULONG	No		Yes	No	No

Device Support Routines

Device support consists of the following routines:

long report(int level)

This optional routine is called by the IOC command `dbior` and is passed the report level that was requested by the user. It should print a report on the state of the device support to stdout. The `level` parameter may be used to output increasingly more detailed information at higher levels, or to select different types of information with different levels. Level zero should print no more than a small summary.

long init(int after)

This optional routine is called twice at IOC initialization time. The first call happens before any of the `init_record()` calls are made, with the integer parameter `after` set to 0. The second call happens after all of the `init_record()` calls have been made, with `after` set to 1.

`init_record(precord)`

This routine is optional. If provided, it is called by record support `init_record()` routine. It should determine MASK if it is needed.

- 0: Success. RVAL modified (VAL will be set accordingly)
- 2: Success. VAL modified
- other: Error

`get_ioint_info(int cmd, struct dbCommon *precord, IOSCANPVT *ppvt)`

This routine is called by the ioEventScan system each time the record is added or deleted from an I/O event scan list. `cmd` has the value (0,1) if the record is being (added to, deleted from) an I/O event list. It must be provided for any device type that can use the ioEvent scanner.

`write_bo(precord)`

This routine must output a new value. It returns the following values:

- 0: Success
- other: Error.

Device Support For Soft Records

Two soft device support modules `Soft Channel` and `Raw Soft Channel` are provided for output records not related to actual hardware devices. The OUT link type must be either CONSTANT, DB_LINK, or CA_LINK.

Soft Channel

This module writes the current value of VAL.

If the OUT link type is PV_LINK, then `dbCaAddInlink()` is called by `init_record()`. `init_record()` always returns a value of 2, which means that no conversion will ever be attempted. `write_bo()` calls `recGblPutLinkValue()` to write the current value of VAL. See *"Soft Output"* for details.

Raw Soft Channel

This module is like the previous except that it writes the current value of RVAL

1.5.8 Calculation Record (calc)

The calculation or "Calc" record is used to perform algebraic, relational, and logical operations on values retrieved from other records. The result of its operations can then be accessed by another record so that it can then be used.

Parameter Fields

The record-specific fields are described below, grouped by functionality.

Scan Parameters

The Calc record has the standard fields for specifying under what circumstances the record will be processed. These fields are described in *Scan Fields*.

Field	Summary	Type	DCT	Default	Read	Write	CA PP
SCAN	Scan Mechanism	MENU menuScan	Yes		Yes	Yes	No
PHAS	Scan Phase	SHORT	Yes		Yes	Yes	No
EVNT	Event Name	STRING [40]	Yes		Yes	Yes	No
PRIO	Scheduling Priority	MENU menuPriority	Yes		Yes	Yes	No
PINI	Process at iocInit	MENU menuPini	Yes		Yes	Yes	No

Read Parameters

The read parameters for the Calc record consist of 12 input links INPA, INPB, … INPL. The fields can be database links, channel access links, or constants. If they are links, they must specify another record's field or a channel access link. If they are constants, they will be initialized with the value they are configured with and can be changed via `dbPuts`. They cannot be hardware addresses.

See Address Specification for information on how to specify database links.

Field	Summary	Type	DCT	Default	Read	Write	CA PP
INPA	Input A	INLINK	Yes		Yes	Yes	No
INPB	Input B	INLINK	Yes		Yes	Yes	No
INPC	Input C	INLINK	Yes		Yes	Yes	No
INPD	Input D	INLINK	Yes		Yes	Yes	No
INPE	Input E	INLINK	Yes		Yes	Yes	No
INPF	Input F	INLINK	Yes		Yes	Yes	No
INPG	Input G	INLINK	Yes		Yes	Yes	No
INPH	Input H	INLINK	Yes		Yes	Yes	No
INPI	Input I	INLINK	Yes		Yes	Yes	No
INPJ	Input J	INLINK	Yes		Yes	Yes	No
INPK	Input K	INLINK	Yes		Yes	Yes	No
INPL	Input L	INLINK	Yes		Yes	Yes	No

Expression

At the core of the Calc record lies the CALC and RPCL fields. The CALC field contains the infix expresion which the record routine will use when it processes the record. The resulting value is placed in the VAL field and can be accessed from there. The CALC expression is actually converted to opcode and stored as Reverse Polish Notation in the RPCL field. It is this expression which is actually used to calculate VAL. The Reverse Polish expression is evaluated more efficiently during run-time than an infix expression. CALC can be changed at run-time, and a special record routine calls a function to convert it to Reverse Polish Notation.

The infix expressions that can be used are very similar to the C expression syntax, but with some additions and subtle differences in operator meaning and precedence. The string may contain a series of expressions separated by a semi-colon character ";" any one of which may actually provide the calculation result; however, all of the other expressions included must assign their result to a variable. All alphabetic elements described below are case independent, so upper and lower case letters may be used and mixed in the variable and function names as desired. Spaces may be used anywhere within an expression except between characters that make up a single expression element.

The range of expressions supported by the calculation record are separated into literals, constants, operands, algebraic operators, trigonometric operators, relational operators, logical operators, the assignment operator, parentheses and commas, and the question mark or '?:' operator.

Field	Summary	Type	DCT	Default	Read	Write	CA PP
CALC	Calculation	STRING [80]	Yes		Yes	Yes	Yes
RPCL	Reverse Polish Calc	NOACCESS	No		No	No	No

Literals

- Standard double precision floating point numbers
- Inf: Infinity
- Nan: Not a Number

Constants

- PI: returns the mathematical constant
- D2R: evaluates to /180 which, when used as a multiplier, converts an angle from degrees to radians
- R2D: evaluates to 180/ which as a multiplier converts an angle from radians to degrees

Operands

The expression uses the values retrieved from the INPx links as operands, though constants can be used as operands too. These values retrieved from the input links are stored in the A-L fields. The values to be used in the expression are simply referenced by the field letter. For instance, the value obtained from INPA link is stored in the field A, and the value obtained from INPB is stored in field B. The field names can be included in the expression which will operate on their respective values, as in A+B. Also, the RNDM nullary function can be included as an operand in the expression in order to generate a random number between 0 and 1.

Field	Summary	Type	DCT	Default	Read	Write	CA PP
A	Value of Input A	DOUBLE	No		Yes	Yes	Yes
B	Value of Input B	DOUBLE	No		Yes	Yes	Yes
C	Value of Input C	DOUBLE	No		Yes	Yes	Yes
D	Value of Input D	DOUBLE	No		Yes	Yes	Yes
E	Value of Input E	DOUBLE	No		Yes	Yes	Yes
F	Value of Input F	DOUBLE	No		Yes	Yes	Yes
G	Value of Input G	DOUBLE	No		Yes	Yes	Yes
H	Value of Input H	DOUBLE	No		Yes	Yes	Yes
I	Value of Input I	DOUBLE	No		Yes	Yes	Yes
J	Value of Input J	DOUBLE	No		Yes	Yes	Yes
K	Value of Input K	DOUBLE	No		Yes	Yes	Yes
L	Value of Input L	DOUBLE	No		Yes	Yes	Yes

The keyword VAL returns the current contents of the VAL field (which can be written to by a CA put, so it might *not* be the result from the last time the expression was evaluated).

Algebraic Operators

- ABS: Absolute value (unary)
- SQR: Square root (unary)
- MIN: Minimum (any number of args)
- MAX: Maximum (any number of args)
- FINITE: returns non-zero if none of the arguments are NaN or Inf (any number of args)
- ISNAN: returns non-zero if any of the arguments is NaN or Inf (any number of args)
- CEIL: Ceiling (unary)
- FLOOR: Floor (unary)
- LOG: Log base 10 (unary)
- LOGE: Natural log (unary)
- LN: Natural log (unary)
- EXP: Exponential function (unary)
- ^ : Exponential (binary)
- ** : Exponential (binary)

- – : Addition (binary)
- – : Subtraction (binary)
- * : Multiplication (binary)
- / : Division (binary)
-
- NOT: Negate (unary)

Trigonometric Operators

- SIN: Sine
- SINH: Hyperbolic sine
- ASIN: Arc sine
- COS: Cosine
- COSH: Hyperbolic cosine
- ACOS: Arc cosine
- TAN: Tangent
- TANH: Hyperbolic tangent
- ATAN: Arc tangent

Relational Operators

- >= : Greater than or equal to
- > : Greater than
- <= : Less than or equal to
- < : Less than
- # : Not equal to
- = : Equal to

Logical Operators

- && : And
- || : Or
- ! : Not

Bitwise Operators

- `|` : Bitwise Or
- `&` : Bitwise And
- OR : Bitwise Or
- AND : Bitwise And
- XOR : Bitwise Exclusive Or
- ~ : One's Complement
- << : Arithmetic Left Shift
- >> : Arithmetic Right Shift
- >>> : Logical Right Shift

Assignment Operator

- `:=` : assigns a value (right hand side) to a variable (i.e. field)

Parantheses, Comma, and Semicolon

The open and close parentheses are supported. Nested parentheses are supported.

The comma is supported when used to separate the arguments of a binary function.

The semicolon is used to separate expressions. Although only one traditional calculation expression is allowed, multiple assignment expressions are allowed.

Conditional Expression

The C language's question mark operator is supported. The format is: `condition ? True result : False result`

Expression Examples

Algebraic

`A + B + 10`

- Result is `A + B + 10`

Relational

`(A + B) < (C + D)`

- Result is 1 if `(A + B) < (C + D)`
- Result is 0 if `(A + B) >= (C + D)`

Question Mark

`(A + B) < (C + D) ? E : F + L + 10`

- Result is E if `(A + B) < (C + D)`
- Result is `F + L + 10` if `(A + B) >= (C + D)`

Prior to Base 3.14.9 it was legal to omit the : and the second (else) part of the conditional, like this:

`(A + B)<(C + D) ? E`

-

Result is E if (A + B)<(C + D)

Result is unchanged if (A + B)>=(C + D)

```
From 3.14.9 onwards, this expresion must be written as
`(A + B) < (C + D) ? E : VAL`
```

Logical

`A & B`

- Causes the following to occur:
  - Convert A to integer
  - Convert B to integer
  - Bitwise And A and B
  - Convert result to floating point

Assignment

`sin(a); a:=a+D2R`

- Causes the Calc record to output the successive values of a sine curve in 1 degree intervals.

Operator Display Parameters

These parameters are used to present meaningful data to the operator. These fields are used to display VAL and other parameters of the calculation record either textually or graphically.

The EGU field contains a string of up to 16 characters which is supplied by the user and which describes the values being operated upon. The string is retrieved whenever the routine `get_units` is called. The EGU string is solely for an operator's sake and does not have to be used.

The HOPR and LOPR fields only refer to the limits of the VAL, HIHI, HIGH, LOW and LOLO fields. PREC controls the precision of the VAL field.

See *Fields Common to All Record Types* for more on the record name (NAME) and description (DESC) fields.

Field	Summary	Type	DCT	Default	Read	Write	CA PP
EGU	Engineering Units	STRING [16]	Yes		Yes	Yes	No
PREC	Display Precision	SHORT	Yes		Yes	Yes	No
HOPR	High Operating Rng	DOUBLE	Yes		Yes	Yes	No
LOPR	Low Operating Range	DOUBLE	Yes		Yes	Yes	No
NAME	Record Name	STRING [61]	No		Yes	No	No
DESC	Descriptor	STRING [41]	Yes		Yes	Yes	No

Alarm Parameters

The possible alarm conditions for the Calc record are the SCAN, READ, Calculation, and limit alarms. The SCAN and READ alarms are called by the record support routines. The Calculation alarm is called by the record processing routine when the CALC expression is an invalid one, upon which an error message is generated.

The following alarm parameters which are configured by the user, define the limit alarms for the VAL field and the severity corresponding to those conditions.

The HYST field defines an alarm deadband for each limit.

See Alarm Specification for a complete explanation of record alarms and of the standard fields. *Alarm Fields* lists other fields related to alarms that are common to all record types.

Field	Summary	Type	DCT	Default	Read	Write	CA PP
HIHI	Hihi Alarm Limit	DOUBLE	Yes		Yes	Yes	Yes
HIGH	High Alarm Limit	DOUBLE	Yes		Yes	Yes	Yes
LOW	Low Alarm Limit	DOUBLE	Yes		Yes	Yes	Yes
LOLO	Lolo Alarm Limit	DOUBLE	Yes		Yes	Yes	Yes
HHSV	Hihi Severity	MENU menuAlarmSevr	Yes		Yes	Yes	Yes
HSV	High Severity	MENU menuAlarmSevr	Yes		Yes	Yes	Yes
LSV	Low Severity	MENU menuAlarmSevr	Yes		Yes	Yes	Yes
LLSV	Lolo Severity	MENU menuAlarmSevr	Yes		Yes	Yes	Yes
HYST	Alarm Deadband	DOUBLE	Yes		Yes	Yes	No

Monitor Parameters

These paramaeters are used to determine when to send monitors for the value fields. These monitors are sent when the value field exceeds the last monitored field by the appropriate deadband, the ADEL for archiver monitors and the MDEL field for all other types of monitors. If these fields have a value of zero, everytime the value changes, monitors are triggered; if they have a value of -1, everytime the record is scanned, monitors are triggered. See *"Monitor Specification"* for a complete explanation of monitors.

Field	Summary	Type	DCT	Default	Read	Write	CA PP
ADEL	Archive Deadband	DOUBLE	Yes		Yes	Yes	No
MDEL	Monitor Deadband	DOUBLE	Yes		Yes	Yes	No

Run-time Parameters

These fields are not configurable using a configuration tool and none are modifiable at run-time. They are used to process the record.

The LALM field is used to implement the hysteresis factor for the alarm limits.

The LA-LL fields are used to decide when to trigger monitors for the corresponding fields. For instance, if LA does not equal the value A, monitors for A are triggered. The MLST and ALST fields are used in the same manner for the VAL field.

Field	Summary	Type	DCT	Default	Read	Write	CA PP
LALM	Last Value Alarmed	DOUBLE	No		Yes	No	No
ALST	Last Value Archived	DOUBLE	No		Yes	No	No
MLST	Last Val Monitored	DOUBLE	No		Yes	No	No
LA	Prev Value of A	DOUBLE	No		Yes	No	No
LB	Prev Value of B	DOUBLE	No		Yes	No	No
LC	Prev Value of C	DOUBLE	No		Yes	No	No
LD	Prev Value of D	DOUBLE	No		Yes	No	No
LE	Prev Value of E	DOUBLE	No		Yes	No	No
LF	Prev Value of F	DOUBLE	No		Yes	No	No
LG	Prev Value of G	DOUBLE	No		Yes	No	No
LH	Prev Value of H	DOUBLE	No		Yes	No	No
LI	Prev Value of I	DOUBLE	No		Yes	No	No
LJ	Prev Value of J	DOUBLE	No		Yes	No	No
LK	Prev Value of K	DOUBLE	No		Yes	No	No
LL	Prev Value of L	DOUBLE	No		Yes	No	No

Record Support

Record Support Routines

`init_record`

For each constant input link, the corresponding value field is initialized with the constant value if the input link is CONSTANT or a channel access link is created if the input link is a PV_LINK.

A routine postfix is called to convert the infix expression in CALC to Reverse Polish Notation. The result is stored in RPCL.

`process`

See next section.

`special`

This is called if CALC is changed. `special` calls postfix.

`get_units`

Retrieves EGU.

`get_precision`

Retrieves PREC.

`get_graphic_double`

Sets the upper display and lower display limits for a field. If the field is VAL, HIHI, HIGH, LOW, or LOLO, the limits are set to HOPR and LOPR, else if the field has upper and lower limits defined they will be used, else the upper and lower maximum values for the field will be used.

`get_control_double`

Sets the upper control and the lower control limits for a field. If the field is VAL, HIHI, HIGH, LOW, or LOLO, the limits are set to HOPR and LOPR, else if the field has upper and lower limits defined they will be used, else the upper and lower maximum values for the field type will be used.

`get_alarm_double`

Sets the following values:

upper_alarm_limit = HIHI

upper_warning_limit = HIGH

lower_warning_limit = LOW

lower_alarm_limit = LOLO

Record Processing

Routine process implements the following algorithm:

- 1.

Fetch all arguments.

- 2.

Call routine `calcPerform`, which calculates VAL from the postfix version of the expression given in CALC. If `calcPerform` returns success UDF is set to FALSE.

- 3.

Check alarms. This routine checks to see if the new VAL causes the alarm status and severity to change. If so, NSEV, NSTA, and LALM are set. It also honors the alarm hysteresis factor (HYST). Thus the value must change by at least HYST before the alarm status and severity changes.

- 4.

Check to see if monitors should be invoked.

- Alarm monitors are invoked if the alarm status or severity has changed.
- Archive and values change monitors are invoked if ADEL and MDEL conditions are met.
- Monitors for A-L are checked whenever other monitors are invoked.
- NSEV and NSTA are reset to 0.
- 5.

Scan forward link if necessary, set PACT FALSE, and return.

1.5.9 Calculation Output Record (calcout)

The Calculation Output or "Calcout" record is similar to the Calc record with the added feature of having outputs (an "output link" and an "output event") which are conditionally executed based on the result of the calculation. This feature allows conditional branching to be implemented within an EPICS database (e.g. process Record_A only if Record_B has a value of 0). The Calcout record is also similar to the Wait record (with additional features) but uses EPICS standard INLINK and OUTLINK fields rather than the DBF_STRING fields used in the Wait record. For new databases, it is recommended that the Calcout record be used instead of the Wait record.

Parameter Fields

The record-specific fields are described below, grouped by functionality.

Scan Parameters

The Calcout record has the standard fields for specifying under what circumstances the record will be processed. These fields are listed in *Scan Fields*.

Read Parameters

The read parameters for the Calcout record consists of 12 input links INPA, INPB, … INPL. The fields can be database links, channel access links, or constants. If they are links, they must specify another record's field. If they are constants, they will be initialized with the value they are configured with and can be changed via `dbPuts`. These fields cannot be hardware addresses. In addition, the Calcout record contains the INAV, INBV, … INLV fields which indicate the status of the link fields, for example, whether or not the specified PV was found and a link to it established. See *"Operator Display Parameters"* for an explanation of these fields.

Field	Summary	Type	DCT	Default	Read	Write	CA PP
INPA	Input A	INLINK	Yes		Yes	Yes	No
INPB	Input B	INLINK	Yes		Yes	Yes	No
INPC	Input C	INLINK	Yes		Yes	Yes	No
INPD	Input D	INLINK	Yes		Yes	Yes	No
INPE	Input E	INLINK	Yes		Yes	Yes	No
INPF	Input F	INLINK	Yes		Yes	Yes	No
INPG	Input G	INLINK	Yes		Yes	Yes	No
INPH	Input H	INLINK	Yes		Yes	Yes	No
INPI	Input I	INLINK	Yes		Yes	Yes	No
INPJ	Input J	INLINK	Yes		Yes	Yes	No
INPK	Input K	INLINK	Yes		Yes	Yes	No
INPL	Input L	INLINK	Yes		Yes	Yes	No

Expression

Like the Calc record, the Calcout record has a CALC field in which the developer can enter an infix expression which the record routine will evaluate when it processes the record. The resulting value is placed in the VAL field. This value can then be used by the OOPT field (see *"Output Parameters"*) to determine whether or not to write to the output link or post an output event. It can also be the value that is written to the output link. The CALC expression is actually converted to opcode and stored in Reverse Polish Notation in the RPCL field. It is this expression which is actually used to calculate VAL. The Reverse Polish expression is evaluated more efficiently during run-time than an infix expression. CALC can be changes at run-time, and a special record routine will call a function to convert it to Reverse Polish Notation.

The infix expressions that can be used are very similar to the C expression syntax, but with some additions and subtle differences in operator meaning and precedence. The string may contain a series of expressions separated by a semi-colon character ';' any one of which may actually provide the calculation result; however all of the other expressions included must assign their result to a variable. All alphabetic elements described below are case independent, so upper and lower case letters may be used and mixed in the variable and function names as desired. Spaces may be used anywhere within an expression except between the characters that make up a single expression element.

The range of expressions supported by the calculation record are separated into literals, constants, operands, algebraic operators, trigonometric operators, relational operators, logical operator, the assignment operator, parentheses and commas, and the question mark or '?:' operator.

Field	Summary	Type	DCT	Default	Read	Write	CA PP
CALC	Calculation	STRING [80]	Yes		Yes	Yes	Yes
VAL	Result	DOUBLE	Yes		Yes	Yes	No
RPCL	Reverse Polish Calc	NOACCESS	No		No	No	No

Literals

- Standard double precision floating point numbers
- Inf: Infinity
- Nan: Not a Number

Constants

- PI: returns the mathematical constant
- D2R: evaluates to /180 which, when used as a multiplier, converts an angle from degrees to radians
- R2D: evaluates to 180/ which, when used as a multiplier, converts an angle from radians to degrees

Operands

The expression can use the values retrieved from the INPx links as operands, though constants can be used as operands too. These values retrieved from the input links are stored in the A-L fields. The values to be used in the expression are simple references by the field letter. For instance, the value obtained from the INPA link is stored in field A, and the values obtained from the INPB link is stored in the field B. The names can be included in the expression will operate on their respective values, as in A+B.

Field	Summary	Type	DCT	Default	Read	Write	CA PP
A	Value of Input A	DOUBLE	No		Yes	Yes	Yes
B	Value of Input B	DOUBLE	No		Yes	Yes	Yes
C	Value of Input C	DOUBLE	No		Yes	Yes	Yes
D	Value of Input D	DOUBLE	No		Yes	Yes	Yes
E	Value of Input E	DOUBLE	No		Yes	Yes	Yes
F	Value of Input F	DOUBLE	No		Yes	Yes	Yes
G	Value of Input G	DOUBLE	No		Yes	Yes	Yes
H	Value of Input H	DOUBLE	No		Yes	Yes	Yes
I	Value of Input I	DOUBLE	No		Yes	Yes	Yes
J	Value of Input J	DOUBLE	No		Yes	Yes	Yes
K	Value of Input K	DOUBLE	No		Yes	Yes	Yes
L	Value of Input L	DOUBLE	No		Yes	Yes	Yes

The keyword VAL returns the current contents of the expression's result field, i.e. the VAL field for the CALC expression and the OVAL field for the OCAL expression. (These fields can be written to by CA put, so it might *not* be the result from the last time the expression was evaluated).

Algebraic Operations

- ABS: Absolute value (unary)
- SQR: Square root (unary)
- MIN: Minimum (any number of args)
- MAX: Maximum (any number of args)
- FINITE: returns non-zero if none of the arguments are NaN or Inf (any number of args)
- ISNAN: returns non-zero if any of the arguments is NaN or Inf (any number of args)
- CEIL: Ceiling (unary)
- FLOOR: Floor (unary)
- LOG: Log base 10 (unary)
- LOGE: Natural log (unary)
- LN: Natural log (unary)
- EXP: Exponential function (unary)
- ^ : Exponential (binary)

- ** : Exponential (binary)
- – : Addition (binary)
- – : Subtraction (binary)
- * : Multiplication (binary)
- / : Division (binary)
-
- NOT: Negate (unary)

Trigonometric Operators

- SIN: Sine
- SINH: Hyperbolic sine
- ASIN: Arc sine
- COS: Cosine
- COSH: Hyperbolic cosine
- ACOS: Arc cosine
- TAN: Tangent
- TANH: Hyperbolic tangent
- ATAN: Arc tangent

Relational Operators

- >= : Greater than or equal to
- > : Greater than
- <= : Less than or equal to
- < : Less than
- # : Not equal to
- = : Equal to

Logical Operators

- && : And
- || : Or
- ! : Not

Bitwise Operators

- `|` : Bitwise Or
- `&` : Bitwise And
- OR : Bitwise Or
- AND : Bitwise And
- XOR : Bitwise Exclusive Or
- ~ : One's Complement
- << : Arithmetic Left Shift
- >> : Arithmetic Right Shift
- >>> : Logical Right Shift

Assignment Operator

- `:=` : assigns a value (right hand side) to a variable (i.e. field)

Parentheses, Comma, and Semicolon

The open and close parentheses are supported. Nested parentheses are supported.

The comma is supported when used to separate the arguments of a binary function.

The semicolon is used to separate expressions. Although only one traditional calculation expression is allowed, multiple assignment expressions are allowed.

Conditional Expression

The C language's question mark operator is supported. The format is: `condition ? True result : False result`

Expression Examples

Algebraic

`A + B + 10`

- Result is `A + B + 10`

Relational

`(A + B) < (C + D)`

- Result is 1 if `(A + B) < (C + D)`
- Result is 0 if `(A + B) >= (C + D)`

Question Mark

`(A + B) < (C + D) ? E : F + L + 10`

- Result is E if `(A + B) < (C + D)`
- Result is `F + L + 10` if `(A + B) >= (C + D)`

Prior to Base 3.14.9 it was legal to omit the : and the second (else) part of the conditional, like this:

`(A + B)<(C + D) ? E`

-

Result is E if (A + B)<(C + D)

Result is unchanged if (A + B)>=(C + D)

```
From 3.14.9 onwards, this expression must be written as
`(A + B) < (C + D) ? E : VAL`
```

Logical

`A & B`

- Causes the following to occur:
  - Convert A to integer
  - Convert B to integer
  - Bitwise And A and B
  - Convert result to floating point

Assignment

`sin(a); a:=a+D2R`

- Causes the Calc record to output the successive values of a sine curve in 1 degree intervals.

Output Parameters

These parameters specify and control the output capabilities of the Calcout record. They determine when to write the output, where to write it, and what the output will be. The OUT link specifies the Process Variable to which the result will be written.

Menu calcoutOOPT

The OOPT field determines the condition that causes the output link to be written to. It's a menu field that has six choices:

Index	Identifier	Choice String
0	calcoutOOPT_Every_Time	Every Time
1	calcoutOOPT_On_Change	On Change
2	calcoutOOPT_When_Zero	When Zero
3	calcoutOOPT_When_Non_zero	When Non-zero
4	calcoutOOPT_Transition_To_Zero	Transition To Zero
5	calcoutOOPT_Transition_To_Non_zero	Transition To Non-zero

- `Every Time` – write output every time record is processed.
- `On Change` – write output every time VAL changes, i.e., every time the result of the expression changes.
- `When Zero` – when record is processed, write output if VAL is zero.
- `When Non-zero` – when record is processed, write output if VAL is non-zero.
- `Transition To Zero` – when record is processed, write output only if VAL is zero and the last value was non-zero.
- `Transition To Non-zero` – when record is processed, write output only if VAL is non-zero and last value was zero.

Menu calcoutDOPT

The DOPT field determines what data is written to the output link when the output is executed. The field is a menu field with two options:

Index	Identifier	Choice String
0	calcoutDOPT_Use_VAL	Use CALC
1	calcoutDOPT_Use_OVAL	Use OCAL

If `Use CALC` is specified, when the record writes its output it will write the result of the expression in the CALC field, that is, it will write the value of the VAL field. If `Use OCAL` is specified, the record will instead write the result of the expression in the OCAL field, which is contained in the OVAL field. The OCAL field is exactly like the CALC field and has the same functionality it can contain the string representation of an expression which is evaluated at run-time. Thus, if necessary, the record can use the result of the CALC expression to determine if data should be written and can use the result of the OCAL expression as the data to write.

If the OEVT field specifies a non-zero integer and the condition in the OOPT field is met, the record will post a corresponding event. If the ODLY field is non-zero, the record pauses for the specified number of seconds before executing the OUT link or posting the output event. During this waiting period the record is "active" and will not be processed again until the wait is over. The field DLYA is equal to 1 during the delay period. The resolution of the delay entry system dependent.

The IVOA field specifies what action to take with the OUT link if the Calcout record enters an INVALID alarm status. The options are `Continue normally`, `Don't drive outputs`, and `Set output to IVOV`. If the IVOA field is `Set output to IVOV`, the data entered into the IVOV field is written to the OUT link if the record alarm severity is INVALID.

Field	Summary	Type	DCT	Default	Read	Write	CA PP
OUT	Output Specification	OUTLINK	Yes		Yes	Yes	No
OOPT	Output Execute Opt	MENU *calcoutOOPT*	Yes		Yes	Yes	No
DOPT	Output Data Opt	MENU *calcoutDOPT*	Yes		Yes	Yes	No
OCAL	Output Calculation	STRING [80]	Yes		Yes	Yes	Yes
OVAL	Output Value	DOUBLE	No		Yes	Yes	No
OEVT	Event To Issue	STRING [40]	Yes		Yes	Yes	No
ODLY	Output Execute Delay	DOUBLE	Yes		Yes	Yes	No
IVOA	INVALID output action	MENU menuIvoa	Yes		Yes	Yes	No
IVOV	INVALID output value	DOUBLE	Yes		Yes	Yes	No

Operator Display Parameter

These parameters are used to present meaningful data to the operator. Some are also meant to represent the status of the record at run-time.

The EGU field contains a string of up to 16 characters which is supplied by the user and which describes the values being operated upon. The string is retrieved whenever the routine `get_units()` is called. The EGU string is solely for an operator's sake and does not have to be used.

The HOPR and LOPR fields only refer to the limits of the VAL, HIHI, HIGH, LOW, and LOLO fields. PREC controls the precision of the VAL field.

Menu calcoutINAV

The INAV-INLV fields indicate the status of the link to the PVs specified in the INPA-INPL fields respectively. These fields can have four possible values:

Index	Identifier	Choice String
0	calcoutINAV_EXT_NC	Ext PV NC
1	calcoutINAV_EXT	Ext PV OK
2	calcoutINAV_LOC	Local PV
3	calcoutINAV_CON	Constant

- `Ext PV NC` – the PV wasn't found on this IOC and a Channel Access link hasn't been established.
- `Ext PV OK` – the PV wasn't found on this IOC and a Channel Access link has been established.
- `Local PV` – the PV was found on this IOC.

- `Constant` – the corresponding link field is a constant.

The OUTV field indicates the status of the OUT link. If has the same possible values as the INAV-INLV fields.

The CLCV and OLCV fields indicate the validity of the expression in the CALC and OCAL fields respectively. If the expression in invalid, the field is set to one.

The DLYA field is set to one during the delay specified in ODLY.

See *Fields Common to All Record Types* for more on the record name (NAME) and description (DESC) fields.

Field	Summary	Type	DCT	Default	Read	Write	CA PP
EGU	Engineering Units	STRING [16]	Yes		Yes	Yes	No
PREC	Display Precision	SHORT	Yes		Yes	Yes	No
HOPR	High Operating Rng	DOUBLE	Yes		Yes	Yes	No
LOPR	Low Operating Range	DOUBLE	Yes		Yes	Yes	No
INAV	INPA PV Status	MENU *calcoutINAV*	No	1	Yes	No	No
INBV	INPB PV Status	MENU *calcoutINAV*	No	1	Yes	No	No
INCV	INPC PV Status	MENU *calcoutINAV*	No	1	Yes	No	No
INDV	INPD PV Status	MENU *calcoutINAV*	No	1	Yes	No	No
INEV	INPE PV Status	MENU *calcoutINAV*	No	1	Yes	No	No
INFV	INPF PV Status	MENU *calcoutINAV*	No	1	Yes	No	No
INGV	INPG PV Status	MENU *calcoutINAV*	No	1	Yes	No	No
INHV	INPH PV Status	MENU *calcoutINAV*	No	1	Yes	No	No
INIV	INPI PV Status	MENU *calcoutINAV*	No	1	Yes	No	No
INJV	INPJ PV Status	MENU *calcoutINAV*	No	1	Yes	No	No
INKV	INPK PV Status	MENU *calcoutINAV*	No	1	Yes	No	No
INLV	INPL PV Status	MENU *calcoutINAV*	No	1	Yes	No	No
OUTV	OUT PV Status	MENU *calcoutINAV*	No		Yes	No	No
CLCV	CALC Valid	LONG	No		Yes	Yes	No
OCLV	OCAL Valid	LONG	No		Yes	Yes	No
DLYA	Output Delay Active	USHORT	No		Yes	No	No
NAME	Record Name	STRING [61]	No		Yes	No	No
DESC	Descriptor	STRING [41]	Yes		Yes	Yes	No

Alarm Parameters

The possible alarm conditions for the Calcout record are the SCAN, READ, Calculation, and limit alarms. The SCAN and READ alarms are called by the record support routines. The Calculation alarm is called by the record processing routine when the CALC expression is an invalid one, upon which an error message is generated.

The following alarm parameters, which are configured by the user, define the limit alarms for the VAL field and the severity corresponding to those conditions.

The HYST field defines an alarm deadband for each limit.

See Alarm Specification for a complete explanation of record alarms and of the standard fields. *Alarm Fields* lists other fields related to alarms that are common to all record types.

Field	Summary	Type	DCT	Default	Read	Write	CA PP
HIHI	Hihi Alarm Limit	DOUBLE	Yes		Yes	Yes	Yes
HIGH	High Alarm Limit	DOUBLE	Yes		Yes	Yes	Yes
LOW	Low Alarm Limit	DOUBLE	Yes		Yes	Yes	Yes
LOLO	Lolo Alarm Limit	DOUBLE	Yes		Yes	Yes	Yes
HHSV	Hihi Severity	MENU menuAlarmSevr	Yes		Yes	Yes	Yes
HSV	High Severity	MENU menuAlarmSevr	Yes		Yes	Yes	Yes
LSV	Low Severity	MENU menuAlarmSevr	Yes		Yes	Yes	Yes
LLSV	Lolo Severity	MENU menuAlarmSevr	Yes		Yes	Yes	Yes
HYST	Alarm Deadband	DOUBLE	Yes		Yes	Yes	No

Monitor Parameters

These parameters are used to determine when to send monitors for the value fields. These monitors are sent when the value field exceeds the last monitored field by the appropriate deadband, the ADEL for archiver monitors and the MDEL field for all other types of monitors. If these fields have a value of zero, every time the value changes, monitors are triggered; if they have a value of -1, every time the record is scanned, monitors are triggered. See *"Monitor Specification"* for a complete explanation of monitors.

Field	Summary	Type	DCT	Default	Read	Write	CA PP
ADEL	Archive Deadband	DOUBLE	Yes		Yes	Yes	No
MDEL	Monitor Deadband	DOUBLE	Yes		Yes	Yes	No

Run-time Parameters

These fields are not configurable using a configuration tool and none are modifiable at run-time. They are used to process the record.

The LALM field is used to implement the hysteresis factor for the alarm limits.

The LA-LL fields are used to decide when to trigger monitors for the corresponding fields. For instance, if LA does not equal the value for A, monitors for A are triggered. The MLST and ALST fields are used in the same manner for the VAL field.

Field	Summary	Type	DCT	Default	Read	Write	CA PP
LALM	Last Value Alarmed	DOUBLE	No		Yes	No	No
ALST	Last Value Archived	DOUBLE	No		Yes	No	No
MLST	Last Val Monitored	DOUBLE	No		Yes	No	No
LA	Prev Value of A	DOUBLE	No		Yes	No	No
LB	Prev Value of B	DOUBLE	No		Yes	No	No
LC	Prev Value of C	DOUBLE	No		Yes	No	No
LD	Prev Value of D	DOUBLE	No		Yes	No	No
LE	Prev Value of E	DOUBLE	No		Yes	No	No
LF	Prev Value of F	DOUBLE	No		Yes	No	No
LG	Prev Value of G	DOUBLE	No		Yes	No	No
LH	Prev Value of H	DOUBLE	No		Yes	No	No
LI	Prev Value of I	DOUBLE	No		Yes	No	No
LJ	Prev Value of J	DOUBLE	No		Yes	No	No
LK	Prev Value of K	DOUBLE	No		Yes	No	No
LL	Prev Value of L	DOUBLE	No		Yes	No	No

Record Support

Record Support Routines

`init_record`

For each constant input link, the corresponding value field is initialized with the constant value if the input link is CONSTANT or a channel access link is created if the input link is PV_LINK.

A routine postfix is called to convert the infix expression in CALC and OCAL to Reverse Polish Notation. The result is stored in RPCL and ORPC, respectively.

`process`

See next section.

`special`

This is called id CALC or OCAL is changed. `special` calls postfix.

`get_units`

Retrieves EGU.

`get_precision`

Retrieves PREC.

`get_graphic_double`

Sets the upper display and lower display limits for a field. If the field is VAL, HIHI, HIGH, LOW, or LOLO, the limits are set to HOPR and LOPR, else if the field has upper and lower limits defined they will be used, else the upper and lower maximum values for the field type will be used.

`get_control_double`

Sets the upper control and lower control limits for a field. If the VAL, HIHI, HIGH, LOW, or LOLO, the limits are set to HOPR and LOPR, else if the field has upper and lower limits defined they will be used, else the upper and lower maximum values for the field will be used.

`get_alarm_double`

Sets the following values:

> upper_alarm_limit = HIHI
>
> upper_warning_limit = HIGH
>
> lower warning_limit = LOW
>
> lower_alarm_limit = LOLO

Record Processing

`process()`

The `process()` routine implements the following algorithm:

- 1.

Fetch all arguments.

- 2.

Call routine `calcPerform()`, which calculates VAL from the prefix version of the expression given in CALC. If `calcPerform()` returns success, UDF is set to FALSE.

- 3.

Check alarms. This routine checks to see if the new VAL causes the alarm status and severity to change. If so, NSEV, NSTA and LALM are set. If also honors the alarm hysteresis factor (HYST). Thus the value must change by at least HYST before the alarm status and severity changes.

- 4.

Determine if the Output Execution Option (OOPT) is met. If met, either execute the output link (and output event) immediately (if ODLY = 0), or schedule a callback after the specified interval. See the explanation for the `execOutput()` routine below.

- 5.

Check to see if monitors should be invoked. - Alarm monitors are invoked if the alarm status or severity has changed. - Archive and value change monitors are invoked if ADEL and MDEL conditions are met. - Monitors for A-L are checked whenever other monitors are invoked. - NSEV and NSTA are reset to 0

- 6.

If no output delay was specified, scan forward link if necessary, set PACT FALSE, and return.

`execOutput()`

- 1.

If DOPT field specifies the use of OCAL, call the routine `calcPerform()` for the postfix version of the expression in OCAL. Otherwise, use VAL.

- 2.

If the Alarm Severity is INVALID, follow the option as designated by the field IVOA.

- 3.

The Alarm Severity is not INVALID or IVOA specifies "Continue Normally", put the value of OVAL to the OUT link and post the event in OEVT (if non-zero).

- 4.

If an output delay was implemented, process the forward link.

1.5.10 Compression Record (compress)

The data compression record is used to collect and compress data from arrays. When the INP field references a data array field, it immediately compresses the entire array into an element of an array using one of several algorithms, overwriting the previous element. If the INP field obtains its value from a scalar-value field, the compression record will collect a new sample each time the record is processed and add it to the compressed data array as a circular buffer.

The INP link can also specify a constant; however, if this is the case, the compression algorithms are ignored, and the record support routines merely return after checking the FLNK field.

Record-specific Menus

Menu compressALG

The ALG field which uses this menu controls the compression algorithm used by the record.

Index	Identifier	Choice String
0	compressALG_N_to_1_Low_Value	N to 1 Low Value
1	compressALG_N_to_1_High_Value	N to 1 High Value
2	compressALG_N_to_1_Average	N to 1 Average
3	compressALG_Average	Average
4	compressALG_Circular_Buffer	Circular Buffer
5	compressALG_N_to_1_Median	N to 1 Median

Menu bufferingALG

The BALG field which uses this menu controls whether new values are inserted at the beginning or the end of the VAL array.

Index	Identifier	Choice String
0	bufferingALG_FIFO	FIFO Buffer
1	bufferingALG_LIFO	LIFO Buffer

Parameter Fields

The record-specific fields are described below.

Parameter Fields

The record-specific fields are described below, grouped by functionality.

Scanning Parameters

The compression record has the standard fields for specifying under what circumstances the record will be processed. Since the compression record supports no direct interfaces to hardware, its SCAN field cannot be set to `I/O Intr`. These fields are described in *Scan Fields*.

Field	Summary	Type	DCT	Default	Read	Write	CA PP
SCAN	Scan Mechanism	MENU menuScan	Yes		Yes	Yes	No
PHAS	Scan Phase	SHORT	Yes		Yes	Yes	No
EVNT	Event Name	STRING [40]	Yes		Yes	Yes	No
PRIO	Scheduling Priority	MENU menuPriority	Yes		Yes	Yes	No
PINI	Process at iocInit	MENU menuPini	Yes		Yes	Yes	No

Algorithms and Related Parameters

The user specifies the algorithm to be used in the ALG field. There are six possible algorithms which can be specified as follows:

Menu compressALG

Index	Identifier	Choice String
0	compressALG_N_to_1_Low_Value	N to 1 Low Value
1	compressALG_N_to_1_High_Value	N to 1 High Value
2	compressALG_N_to_1_Average	N to 1 Average
3	compressALG_Average	Average
4	compressALG_Circular_Buffer	Circular Buffer
5	compressALG_N_to_1_Median	N to 1 Median

The following fields determine what channel to read and how to compress the data:

Field	Summary	Type	DCT	Default	Read	Write	CA PP
ALG	Compression Algorithm	MENU *compressALG*	Yes		Yes	Yes	No
INP	Input Specification	INLINK	Yes		Yes	Yes	No
NSAM	Number of Values	ULONG	Yes	1	Yes	No	No
N	N to 1 Compression	ULONG	Yes	1	Yes	Yes	No
ILIL	Init Low Interest Lim	DOUBLE	Yes		Yes	Yes	No
IHIL	Init High Interest Lim	DOUBLE	Yes		Yes	Yes	No
OFF	Offset	ULONG	No		Yes	No	No
RES	Reset	SHORT	No		Yes	Yes	No

As stated above, the ALG field specifies which algorithm to be performed on the data.

The INP should be a database or channel access link. Though INP can be a constant, the data compression algorithms are supported only when INP is a database link. See Address Specification for information on specifying links.

IHIL and ILIL can be set to provide an initial value filter on the input array. If ILIL < IHIL, the input elements will be skipped until a value is found that is in the range of ILIL to IHIL. Note that ILIL and IHIL are used only in `N to 1` algorithms.

OFF provides the offset to the current beginning of the array data. Note that OFF is used only in `N to 1` algorithms.

The RES field can be accessed at run time to cause the algorithm to reset itself before the maximum number of samples are reached.

Algorithms

Circular Buffer algorithm keeps a circular buffer of length NSAM. Each time the record is processed, it gets the data referenced by INP and puts it into the circular buffer referenced by VAL. The INP can refer to both scalar or array data and VAL is just a time ordered circular buffer of values obtained from INP. Note that N, ILIL, IHIL and OFF are not used in `Circular Buffer` algorithm.

Average takes an average of every element of the array obtained from INP over time; that is, the entire array referenced by INP is retrieved, and for each element, the new average is calculated and placed in the corresponding element of the value buffer. The retrieved array is truncated to be of length NSAM. N successive arrays are averaged and placed in the buffer. Thus, VAL[0] holds the average of the first element of INP over N samples, VAL[1] holds the average of the next element of INP over N samples, and so on. The following shows the equation:

N to 1 If any of the `N to 1` algorithms are chosen, then VAL is a circular buffer of NSAM samples. The actual algorithm depends on whether INP references a scalar or an array.

If INP refers to a scalar, then N successive time ordered samples of INP are taken. After the Nth sample is obtained, a new value determined by the algorithm (Lowest, Highest, or Average), is written to the circular buffer referenced by VAL. If `Low Value` the lowest value of all the samples is written; if `High Value` the highest value is written; and if `Average`, the average of all the samples are written. The `Median` setting behaves like `Average` with scalar input data.

If INP refers to an array, then the following applies:

- `N to 1 Low Value`

  Compress N to 1 samples, keeping the lowest value.

- `N to 1 High Value`

  Compress N to 1 samples, keeping the highest value.

- `N to 1 Average`

  Compress N to 1 samples, taking the average value.

- `N to 1 Median`

  Compress N to 1 samples, taking the median value.

The compression record keeps NSAM data samples.

The field N determines the number of elements to compress into each result.

Thus, if NSAM was 3, and N was also equal to 3, then the algorithms would work as in the following diagram:

Operator Display Parameters

These parameters are used to present meaningful data to the operator. They display the value and other parameters of the record either textually or graphically.

Field	Summary	Type	DCT	Default	Read	Write	CA PP
EGU	Engineering Units	STRING [16]	Yes		Yes	Yes	No
HOPR	High Operating Range	DOUBLE	Yes		Yes	Yes	No
LOPR	Low Operating Range	DOUBLE	Yes		Yes	Yes	No
PREC	Display Precision	SHORT	Yes		Yes	Yes	No
NAME	Record Name	STRING [61]	No		Yes	No	No
DESC	Descriptor	STRING [41]	Yes		Yes	Yes	No

The EGU field should be given a string that describes the value of VAL, but is used whenever the `get_units` record support routine is called.

The HOPR and LOPR fields only specify the upper and lower display limits for VAL, HIHI, HIGH, LOLO and LOW fields.

PREC controls the floating-point precision whenever `get_precision` is called, and the field being referenced is the VAL field (i.e., one of the values contained in the circular buffer).

See *Fields Common to All Record Types* for more on the record name (NAME) and description (DESC) fields.

Alarm Parameters

The compression record has the alarm parameters common to all record types described in *Alarm Fields*.

Run-time Parameters

These parameters are used by the run-time code for processing the data compression algorithm. They are not configurable by the user, though some are accessible at run-time. They can represent the current state of the algorithm or of the record whose field is referenced by the INP field.

Field	Summary	Type	DCT	Default	Read	Write	CA PP
NUSE	Number Used	ULONG	No		Yes	No	No
OUSE	Old Number Used	ULONG	No		Yes	No	No
BPTR	Buffer Pointer	NOACCESS	No		No	No	No
SPTR	Summing Buffer Ptr	NOACCESS	No		No	No	No
WPTR	Working Buffer Ptr	NOACCESS	No		No	No	No
CVB	Compress Value Buffer	DOUBLE	No		Yes	No	No
INPN	Number of elements in Working Buffer	LONG	No		Yes	No	No
INX	Current number of readings	ULONG	No		Yes	No	No

NUSE and OUSE hold the current and previous number of elements stored in VAL.

BPTR points to the buffer referenced by VAL.

SPTR points to an array that is used for array averages.

WPTR points to the buffer containing data referenced by INP.

CVB stores the current compressed value for `N to 1` algorithms when INP references a scalar.

INPN is updated when the record processes; if INP references an array and the size changes, the WPTR buffer is reallocated.

INX counts the number of readings collected.

Record Support

Record Support Routines

```
long init_record(struct dbCommon *precord, int pass)
```

Space for all necessary arrays is allocated. The addresses are stored in the appropriate fields in the record.

```
long process(struct dbCommon *precord)
```

See *"Record Processing"* below.

```
long special(struct dbAddr *paddr, int after)
```

This routine is called when RSET, ALG, or N are set. It performs a reset.

```
long cvt_dbaddr(struct dbAddr *paddr)
```

This is called by dbNameToAddr. It makes the dbAddr structure refer to the actual buffer holding the result.

```
long get_array_info(struct dbAddr *paddr, long *no_elements, long *offset)
```

Obtains values from the circular buffer referenced by VAL.

```
long put_array_info(struct dbAddr *paddr, long nNew);
```

Writes values into the circular buffer referenced by VAL.

```
long get_units(struct dbAddr *paddr, char *units);
```

Retrieves EGU.

```
long get_precision(const struct dbAddr *paddr, long *precision);
```

Retrieves PREC.

```
long get_graphic_double(struct dbAddr *paddr, struct dbr_grDouble *p);
```

Sets the upper display and lower display limits for a field. If the field is VAL, the limits are set to HOPR and LOPR, else if the field has upper and lower limits defined they will be used, else the upper and lower maximum values for the field type will be used.

```
long get_control_double(struct dbAddr *paddr, struct dbr_ctrlDouble *p);
```

Sets the upper control and the lower control limits for a field. If the field is VAL, the limits are set to HOPR and LOPR, else if the field has upper and lower limits defined they will be used, else the upper and lower maximum values for the field type will be used.

Record Processing

Routine process implements the following algorithm:

1. If INP is not a database link, check monitors and the forward link and return.
2. Get the current data referenced by INP.
3. Perform the appropriate algorithm:
    - Average: Read N successive instances of INP and perform an element by element average. Until N instances have been obtained it just return without checking monitors or the forward link. When N instances have been obtained complete the algorithm, store the result in the VAL array, check monitors and the forward link, and return.
    - Circular Buffer: Write the values obtained from INP into the VAL array as a circular buffer, check monitors and the forward link, and return.
    - N to 1 xxx when INP refers to a scalar: Obtain N successive values from INP and apply the N to 1 xxx algorithm to these values. Until N values are obtained monitors and forward links are not triggered. When N successive values have been obtained, complete the algorithm, check monitors and trigger the forward link, and return.

   - N to 1 xxx when INP refers to an array: The ILIL and IHIL are honored if ILIL < IHIL. The input array is divided into subarrays of length N. The specified N to 1 xxx compression algorithm is applied to each sub-array and the result stored in the array referenced by VAL. The monitors and forward link are checked.

4. If success, set UDF to FALSE.
5. Check to see if monitors should be invoked:
   - Alarm monitors are invoked if the alarm status or severity has changed.
   - NSEV and NSTA are reset to 0.
6. Scan forward link if necessary, set PACT FALSE, and return.

1.5.11 Data Fanout Record (dfanout)

The Data Fanout or "dfanout" record is used to forward data to up to eight other records. It's similar to the fanout record except that the capability to forward data has been added to it. If has no associated device support.

Parameter Fields

The record-specific fields are described below, grouped by functionality.

Scan Parameters

The data fanout record has the standard fields for specifying under what circumstances it will be processed. These fields are listed in *Scan Fields*.

Desired Output Parameters

The data fanout record must specify where the desired output value originates, i.e., the data which is to be fowarded to the records in its output links. The output mode select (OMSL) field determines whether the output originates from another record or from run-time database access. When set to `closed_loop`, the desired output is retrieved from the link specified in the Desired Output Link (DOL) field, which can specify either a database or a channel access link, and placed into the VAL field. When set to `supervisory`, the desired output can be written to the VAL field via dbPuts at run-time.

The DOL field can also be a constant in which case the VAL field is initialized to the constant value.

Note that there are no conversion parameters, so the desired output value undergoes no conversions before it is sent out to the output links.

Field	Summary	Type	DCT	Default	Read	Write	CA PP
DOL	Desired Output Link	INLINK	Yes		Yes	Yes	No
OMSL	Output Mode Select	MENU menuOmsl	Yes		Yes	Yes	No
VAL	Desired Output	DOUBLE	Yes		Yes	Yes	Yes

Write Parameters

The OUTA-OUTH fields specify where VAL is to be sent. Each field that is to forward data must specify an address to another record. See Address Specification for information on specifying links.

The SELL, SELM, and SELN fields specify which output links are to be used.

Menu dfanoutSELM

SELM is a menu, with three choices:

Index	Identifier	Choice String
0	dfanoutSELM_All	All
1	dfanoutSELM_Specified	Specified
2	dfanoutSELM_Mask	Mask

If SELM is `All`, then all output links are used, and the values of SELL and SELN are ignored.

If SELM is `Specified`, then the value of SELN is used to specify a single link which will be used. If SELN==0, then no link will be used; if SELN==1, then OUTA will be used, and so on.

SELN can either have its value set directly, or have it retrieved from another EPICS PV. If SELL is a valid PV link, then SELN will be read from the linked PV.

If SELM is `Mask`, then SELN will be treated as a bit mask. If bit zero (the LSB) of SELN is set, then OUTA will be written to; if bit one is set, OUTB will be written to, and so on. Thus when SELN==5, both OUTC and OUTA will be written to.

Field	Summary	Type	DCT	Default	Read	Write	CA PP
SELL	Link Selection Loc	INLINK	Yes		Yes	Yes	No
SELM	Select Mechanism	MENU *dfanoutSELM*	Yes		Yes	Yes	No
SELN	Link Selection	USHORT	No	1	Yes	Yes	No
OUTA	Output Spec A	OUTLINK	Yes		Yes	Yes	No
OUTB	Output Spec B	OUTLINK	Yes		Yes	Yes	No
OUTC	Output Spec C	OUTLINK	Yes		Yes	Yes	No
OUTD	Output Spec D	OUTLINK	Yes		Yes	Yes	No
OUTE	Output Spec E	OUTLINK	Yes		Yes	Yes	No
OUTF	Output Spec F	OUTLINK	Yes		Yes	Yes	No
OUTG	Output Spec G	OUTLINK	Yes		Yes	Yes	No
OUTH	Output Spec H	OUTLINK	Yes		Yes	Yes	No

Operator Display Parameters

These parameters are used to present meaningful data to the operator. They do not affect the functioning of the record at all.

- NAME is the record's name, and can be useful when the PV name that a client knows is an alias for the record.
- DESC is a string that is usually used to briefly describe the record.
- EGU is a string of up to 16 characters naming the engineering units that the VAL field represents.
- The HOPR and LOPR fields set the upper and lower display limits for the VAL, HIHI, HIGH, LOW, and LOLO fields.
- The PREC field determines the floating point precision (i.e. the number of digits to show after the decimal point) with which to display VAL and the other DOUBLE fields.

See *Fields Common to All Record Types* for more about the record name (NAME) and description (DESC) fields.

Field	Summary	Type	DCT	Default	Read	Write	CA PP
NAME	Record Name	STRING [61]	No		Yes	No	No
DESC	Descriptor	STRING [41]	Yes		Yes	Yes	No
EGU	Engineering Units	STRING [16]	Yes		Yes	Yes	No
HOPR	High Operating Range	DOUBLE	Yes		Yes	Yes	No
LOPR	Low Operating Range	DOUBLE	Yes		Yes	Yes	No
PREC	Display Precision	SHORT	Yes		Yes	Yes	No

Alarm Parameters

The possible alarm conditions for data fanouts are the SCAN, READ, INVALID, and limit alarms. The SCAN and READ alarms are called by the record routines. The limit alarms are configured by the user in the HIHI, LOLO, HIGH, and LOW fields using floating point values. The limit alarms apply only to the VAL field. The severity for each of these limits is specified in the corresponding field (HHSV, LLSV, HSV, LSV) and can be either NO_ALARM, MINOR, or MAJOR. In the hysteresis field (HYST) can be entered a number which serves as the deadband on the limit alarms.

See Alarm Specification for a complete explanation of record alarms and of the standard fields. *Alarm Fields* lists other fields related to alarms that are common to all record types.

Field	Summary	Type	DCT	Default	Read	Write	CA PP
HIHI	Hihi Alarm Limit	DOUBLE	Yes		Yes	Yes	Yes
HIGH	High Alarm Limit	DOUBLE	Yes		Yes	Yes	Yes
LOW	Low Alarm Limit	DOUBLE	Yes		Yes	Yes	Yes
LOLO	Lolo Alarm Limit	DOUBLE	Yes		Yes	Yes	Yes
HHSV	Hihi Severity	MENU menuAlarmSevr	Yes		Yes	Yes	Yes
HSV	High Severity	MENU menuAlarmSevr	Yes		Yes	Yes	Yes
LSV	Low Severity	MENU menuAlarmSevr	Yes		Yes	Yes	Yes
LLSV	Lolo Severity	MENU menuAlarmSevr	Yes		Yes	Yes	Yes
HYST	Alarm Deadband	DOUBLE	Yes		Yes	Yes	No

Monitor Parameters

These parameters are used to determine when to send monitors placed on the VAL field. These monitors are sent when the value field exceeds the last monitored fields by the specified deadband, ADEL for archivers monitors and MDEL for all other types of monitors. If these fields have a value of zero, everytime the value changes, a monitor will be triggered; if they have a value of -1, everytime the record is scanned, monitors are triggered. See *"Monitor Specification"* for a complete explanation of monitors.

Field	Summary	Type	DCT	Default	Read	Write	CA PP
ADEL	Archive Deadband	DOUBLE	Yes		Yes	Yes	No
MDEL	Monitor Deadband	DOUBLE	Yes		Yes	Yes	No

Run-Time Parameters and Simulation Mode Parameters

These parameters are used by the run-time code for processing the data fanout record. Ther are not configurable. They are used to implement the hysteresis factors for monitor callbacks.

Field	Summary	Type	DCT	Default	Read	Write	CA PP
LALM	Last Value Alarmed	DOUBLE	No		Yes	No	No
ALST	Last Value Archived	DOUBLE	No		Yes	No	No
MLST	Last Val Monitored	DOUBLE	No		Yes	No	No

Record Support

Record Support Routines

`init_record()`

This routine initializes all output links that are defined. Then it initializes DOL if DOL is a constant or a PV_LINK. When initializing the output links and the DOL link, a non-zero value is returned if an error occurs.

`process()`

See next section.

`get_units()`

The routine copies the string specified in the EGU field to the location specified by a pointer which is passed to the routine.

`get_graphic_double()`

If the referenced field is VAL, HIHI, HIGH, LOW, or LOLO, this routine sets the `upper_disp_limit` member of the `dbr_grDouble` structure to the HOPR and the `lower_disp_limit` member to the LOPR. If the referenced field is not one of the above fields, then `recGblGetControlDouble()` routine is called.

`get_control_double()`

Same as the `get_graphic_double()` routine except that it uses the `dbr_ctrlDouble` structure.

`get_alarm_double()`

This sets the members of the `dbr_alDouble` structure to the specified alarm limits when the referenced field is VAL:

> upper_alarm_limit = HIHI
>
> upper_warning_limit = HIGH
>
> lower_warning_limit = LOW
>
> lower_alarm_limit = LOLO

If the referenced field is not VAL, the `recGblGetAlarmDouble()` routine is called.

Record Processing

- 1.

The `process()` routine first checks that DOL is not a constant link and that OMSL is set to “closed_loop”. If so, it retrieves a value through DOL and places it into VAL. If no errors occur, UDF is set to FALSE.

- 2.

PACT is set TRUE, and the record’s timestamp is set.

- 3.

A value is fetched from SELL and placed into SELN.

- 4.

Alarms ranges are checked against the contents of the VAL field.

- 5.

VAL is then sent through the OUTA-OUTH links by calling `dbPutLink()` for each link, conditional on the setting of SELM and the value in SELN.

- 6.

Value and archive monitors are posted on the VAL field if appropriate based on the settings of MDEL and ADEL respectively.

- 7.

The data fanout's forward link FLNK is processed.

- 6.

PACT is set FALSE, and the `process()` routine returns.

1.5.12 Event Record (event)

The normal use for this record type is to post an event and/or process a forward link. Device support for this record can provide a hardware interrupt handler routine for I/O Event-scanned records.

Parameter Fields

The record-specific fields are described below, grouped by functionality.

Scan Parameters

The event record has the standard fields for specifying under what circumstances it will be processed. These fields are described in *Scan Fields*.

Field	Summary	Type	DCT	Default	Read	Write	CA PP
SCAN	Scan Mechanism	MENU menuScan	Yes		Yes	Yes	No
PHAS	Scan Phase	SHORT	Yes		Yes	Yes	No
EVNT	Event Name	STRING [40]	Yes		Yes	Yes	No
PRIO	Scheduling Priority	MENU menuPriority	Yes		Yes	Yes	No
PINI	Process at iocInit	MENU menuPini	Yes		Yes	Yes	No

Event Number Parameters

The VAL field contains the event number read by the device support routines. It is this number which is posted. For records that use `Soft Channel` device support, it can be configured before run-time or set via dbPuts.

Field	Summary	Type	DCT	Default	Read	Write	CA PP
VAL	Event Name To Post	STRING [40]	Yes		Yes	Yes	No

Input Specification

The device support routines use the address in this record to obtain input. For records that provide an interrupt handler, the INP field should specify the address of the I/O card, and the DTYP field should specify a valid device support module. Be aware that the address format differs according to the card type used. See Address Specification for information on the format of hardware addresses and specifying links.

For soft records, the INP field can be a constant, a database link, or a channel access link. For soft records, the DTYP field should specify `Soft Channel`.

Field	Summary	Type	DCT	Default	Read	Write	CA PP
INP	Input Specification	INLINK	Yes		Yes	Yes	No
DTYP	Device Type	DEVICE	Yes		Yes	Yes	No

Operator Display Parameters

See *Fields Common to All Record Types* for more on the record name (NAME) and description (DESC) fields.

Field	Summary	Type	DCT	Default	Read	Write	CA PP
NAME	Record Name	STRING [61]	No		Yes	No	No
DESC	Descriptor	STRING [41]	Yes		Yes	Yes	No

Alarm Parameters

The Event record has the alarm parameters common to all record types. *Alarm Fields* lists other fields related to alarms that are common to all record types.

Simulation Mode Parameters

The following fields are used to operate the event record in the simulation mode. See *"Fields Common to Many Record Types"* for more information on these fields.

Field	Summary	Type	DCT	Default	Read	Write	CA PP
SIOL	Sim Input Specifctn	INLINK	Yes		Yes	Yes	No
SVAL	Simulation Value	STRING [40]	No		Yes	Yes	No
SIML	Sim Mode Location	INLINK	Yes		Yes	Yes	No
SIMM	Simulation Mode	MENU menuYesNo	No		Yes	Yes	No
SIMS	Sim mode Alarm Svrty	MENU menuAlarmSevr	Yes		Yes	Yes	No

Record Support

Record Support Routines

init_record

This routine initializes SIMM with the value of SIML if SIML type is a CONSTANT link or creates a channel access link if SIML type is PV_LINK. SVAL is likewise initialized if SIOL is CONSTANT or PV_LINK.

If device support includes `init_record()`, it is called.

process

See next section.

Record Processing

Routine process implements the following algorithm:

1. readValue is called. See *"Input Records"* for more information.
2. If PACT has been changed to TRUE, the device support read routine has started but has not completed reading a new input value. In this case, the processing routine merely returns, leaving PACT TRUE.
3. If VAL > 0, post event number VAL.
4. Check to see if monitors should be invoked. Alarm monitors are invoked if the alarm status or severity has chanet to 0.
5. Scan forward link if necessary, set PACT FALSE, and return.

Device Support

Fields of Interest To Device Support

Each record must have an associated set of device support routines. The device support routines are primarily interested in the following fields:

Field	Summary	Type	DCT	Default	Read	Write	CA PP
PACT	Record active	UCHAR	No		Yes	No	No
DPVT	Device Private	NOACCESS	No		No	No	No
UDF	Undefined	UCHAR	Yes	1	Yes	Yes	Yes
NSEV	New Alarm Severity	MENU menuAlarmSevr	No		Yes	No	No
NSTA	New Alarm Status	MENU menuAlarmStat	No		Yes	No	No
INP	Input Specification	INLINK	Yes		Yes	Yes	No
PRIO	Scheduling Priority	MENU menuPriority	Yes		Yes	Yes	No

Device Support Routines

Device support consists of the following routines:

long report(int level)

This optional routine is called by the IOC command `dbior` and is passed the report level that was requested by the user. It should print a report on the state of the device support to stdout. The `level` parameter may be used to output increasingly more detailed information at higher levels, or to select different types of information with different levels. Level zero should print no more than a small summary.

long init(int after)

This optional routine is called twice at IOC initialization time. The first call happens before any of the `init_record()` calls are made, with the integer parameter `after` set to 0. The second call happens after all of the `init_record()` calls have been made, with `after` set to 1.

init_record

```
init_record(precord)
```

This routine is optional. If provided, it is called by the record support `init_record()` routine.

get_ioint_info

```
get_ioint_info(int cmd, struct dbCommon *precord, IOSCANPVT *ppvt)
```

This routine is called by the ioEventScan system each time the record is added or deleted from an I/O event scan list. `cmd` has the value (0,1) if the record is being (added to, deleted from) an I/O event list. It must be provided for any device type that can use the ioEvent scanner.

read_event

```
read_event(precord)
```

This routine returns the following values:

- 0: Success.
- Other: Error.

Device Support For Soft Records

The `Soft Channel` device support module is available. The INP link type must be either CONSTANT, DB_LINK, or CA_LINK.

If the INP link type is CONSTANT, then the constant value is stored into VAL by `init_record()`, and UDF is set to FALSE. If the INP link type is PV_LINK, then dbCaAddInlink is called by `init_record()`.

`read_event` calls recGblGetLinkValue to read the current value of VAL. See *"Input Records"* for details on soft input.

1.5.13 Fanout Record (fanout)

The fanout record uses several forward processing links to force multiple passive records to scan. When more than one record needs to be scanned as the result of a record being processed, the forward link of that record can specify a fanout record. The fanout record can specify up to sixteen other records to process. If more than sixteen are needed, one of the forward links in the fanout record (or its FLNK field) can point to another fanout record.

NOTE: Fanout records only propagate processing, not data. The dfanout or Data Fanout record can, on the other hand, send data to other records.

Parameter Fields

The record-specific fields are described below, grouped by functionality.

Scan Parameters

The forward link fields of the fanout record (LNK0-LNK9, LNKA-LNKF) specify records to be scanned. The records to be processed must specify `Passive` in their SCAN fields; otherwise the forward link will not cause them to process. Also when specifying database links for the fanout record, the user needs only to specify the record name. As no value is being sent or retrieved, a field name is only required when the link will be over Channel Access, in which case the field PROC must be named.

The SELM, SELN, and SELL fields specify the order of processing for the forward links. The select mechanism menu field (SELM) has three choices:

Index	Identifier	Choice String
0	fanoutSELM_All	All
1	fanoutSELM_Specified	Specified
2	fanoutSELM_Mask	Mask

How the SELM value affects which links to process and in which order is as follows:

- **All** Links are processed in numerical order - LNK0, LNK1, etc.
- **Specified** The sum of the values in the SELN and OFFS fields is used as the specifier of which link to process. For instance, with OFFS=0 and SELN=1, the record targeted by LNK1 will be processed.
- **Mask** The individual bits in SELN are shifted by SHFT bits (negative means shift left) and the result used to select which links to process as follows:
  - If bit 0 (LSB) is set, LNK0 is processed.
  - If bit 1 is set, LNK2 is processed.
  - If bit 2 is set, LNK3 is processed, etc.

SELN reads its value from SELL. SELL can be a constant, a database link, or a channel access link. If a constant, SELN is initialized with the constant value and can be changed via dbPuts. For database/channel access links, SELN is retrieved from SELL each time the record is processed and can also be changed via dbPuts.

The Fanout record also has the standard scanning fields common to all records. These fields are listed in *Scan Fields*.

Field	Summary	Type	DCT	Default	Read	Write	CA PP
SELM	Select Mechanism	MENU *fanoutSELM*	Yes		Yes	Yes	No
SELN	Link Selection	USHORT	No	1	Yes	Yes	No
SELL	Link Selection Loc	INLINK	Yes		Yes	Yes	No
OFFS	Offset for Specified	SHORT	Yes		Yes	Yes	No
SHFT	Shift for Mask mode	SHORT	Yes	-1	Yes	Yes	No
LNK0	Forward Link 0	FWDLINK	Yes		Yes	Yes	No
LNK1	Forward Link 1	FWDLINK	Yes		Yes	Yes	No
LNK2	Forward Link 2	FWDLINK	Yes		Yes	Yes	No
LNK3	Forward Link 3	FWDLINK	Yes		Yes	Yes	No
LNK4	Forward Link 4	FWDLINK	Yes		Yes	Yes	No
LNK5	Forward Link 5	FWDLINK	Yes		Yes	Yes	No
LNK6	Forward Link 6	FWDLINK	Yes		Yes	Yes	No
LNK7	Forward Link 7	FWDLINK	Yes		Yes	Yes	No
LNK8	Forward Link 8	FWDLINK	Yes		Yes	Yes	No
LNK9	Forward Link 9	FWDLINK	Yes		Yes	Yes	No
LNKA	Forward Link 10	FWDLINK	Yes		Yes	Yes	No
LNKB	Forward Link 11	FWDLINK	Yes		Yes	Yes	No
LNKC	Forward Link 12	FWDLINK	Yes		Yes	Yes	No
LNKD	Forward Link 13	FWDLINK	Yes		Yes	Yes	No
LNKE	Forward Link 14	FWDLINK	Yes		Yes	Yes	No
LNKF	Forward Link 15	FWDLINK	Yes		Yes	Yes	No

Operator Display Parameters

These parameters are used to present meaningful data to the operator. See *Fields Common to All Record Types* for more on these fields.

Field	Summary	Type	DCT	Default	Read	Write	CA PP
NAME	Record Name	STRING [61]	No		Yes	No	No
DESC	Descriptor	STRING [41]	Yes		Yes	Yes	No

Alarm Parameters

The Fanout record has the alarm parameters common to all record types. *Alarm Fields* lists the fields related to alarms that are common to all record types.

Run-time Parameters

The VAL field performs no specific function, but a Channel Access put to it will cause the record to process.

Field	Summary	Type	DCT	Default	Read	Write	CA PP
VAL	Used to trigger	LONG	No		Yes	Yes	Yes

Record Support

Record Support Routines

init_record

This routine initializes SELN with the value of SELL, if SELL type is CONSTANT link, or creates a channel access link if SELL type is PV_LINK.

process

See next section.

Record Processing

Routine process implements the following algorithm:

1. PACT is set to TRUE.
2. The link selection SELN is fetched.
3. Depending on the selection mechanism, the link selection forward links are processed, and UDF is set to FALSE.
4. Check to see if monitors should be invoked:
   - Alarm monitors are invoked if the alarm status or severity has changed.

- NSEV and NSTA are reset to 0.

5. Scan forward link field FLNK if used, set PACT FALSE, and return.

1.5.14 Histogram Record (histogram)

The histogram record is used to store frequency counts of a signal into an array of arbitrary length. The user can configure the range of the signal value that the array will store. Anything outside this range will be ignored.

Parameter Fields

The record-specific fields are described below.

Read Parameters

The SVL is the input link where the record reads its value. It can be a constant, a database link, or a channel access link. If SVL is a database or channel access link, then SGNL is read from SVL. If SVL is a constant, then SGNL is initialized with the constant value but can be changed via dbPuts. The `Soft Channel` device support module can be specified in the DTYP field.

The ULIM and LLIM fields determine the usable range of signal values. Any value of SGNL below LLIM or above ULIM is outside the range and will not be stored in the array. In the NELM field the user must specify the array size, e.g., the number of array elements. Each element in the NELM field holds the counts for an interval of the range of signal counts, the range specified by ULIM and LLIM. These intervals are determined by dividing the range by NELM:

```
(ULIM - LLIM) / NELM.
```

Field	Summary	Type	DCT	Default	Read	Write	CA PP
SVL	Signal Value Location	INLINK	Yes		Yes	Yes	No
SGNL	Signal Value	DOUBLE	No		Yes	Yes	No
DTYP	Device Type	DEVICE	Yes		Yes	Yes	No
NELM	Num of Array Elements	USHORT	Yes	1	Yes	No	No
ULIM	Upper Signal Limit	DOUBLE	Yes		Yes	Yes	No
LLIM	Lower Signal Limit	DOUBLE	Yes		Yes	Yes	No

Operator Display Parameters

These parameters are used to present meaningful data to the operator. These fields are used to display the value and other parameters of the histogram either textually or graphically. See *Fields Common to All Record Types* for more on the record name (NAME) and description (DESC) fields.

Field	Summary	Type	DCT	Default	Read	Write	CA PP
NAME	Record Name	STRING [61]	No		Yes	No	No
DESC	Descriptor	STRING [41]	Yes		Yes	Yes	No

Alarm Parameters

The Histogram record has the alarm parameters common to all record types. *Alarm Fields* lists the fields related to alarms that are common to all record types.

Monitor Parameters

The MDEL field implements the monitor count deadband. Only when MCNT is greater than the value given to MDEL are monitors triggered, MCNT being the number of counts since the last time the record was processed. If MDEL is -1, everytime the record is processed, a monitor is triggered regardless.

If SDEL is greater than 0, it causes a callback routine to be called. The number specified in SDEL is the callback routines interval. The callback routine is called every SDEL seconds. The callback routine posts an event if MCNT is greater than 0.

Field	Summary	Type	DCT	Default	Read	Write	CA PP
MDEL	Monitor Count Deadband	SHORT	Yes		Yes	Yes	No
SDEL	Monitor Seconds Dband	DOUBLE	Yes		Yes	Yes	No

Run-time and Simulation Mode Parameters

These parameters are used by the run-time code for processing the histogram. They are not configurable by the user prior to run-time. They represent the current state of the record. Many of them are used to process the histogram more efficiently.

The BPTR field contains a pointer to the unsigned long array of frequency values. The VAL field references this array as well. However, the BPTR field is not accessible at run-time.

The MCNT field keeps counts the number of signal counts since the last monitor was invoked.

The collections controls field (CMD) is a menu field with five choices:

Index	Identifier	Choice String
0	histogramCMD_Read	Read
1	histogramCMD_Clear	Clear
2	histogramCMD_Start	Start
3	histogramCMD_Stop	Stop

When CMD is `Read`, the record retrieves its values and adds them to the signal array. This command will first clear the signal counts which have already been read when it is first invoked.

The `Clear` command erases the signal counts, setting the elements in the array back to zero. Afterwards, the CMD field is set back to `Read`.

The `Start` command simply causes the record to read signal values into the array. Unlike `Read`, it doesn't clear the array first.

The `Stop` command disables the reading of signal values into the array.

The `Setup` command waits until the `start` or `read` command has been issued to start counting.

The CSTA or collections status field implements the CMD field choices by enabling or disabling the reading of values into the histogram array. While FALSE, no signals are added to the array. While TRUE, signals are read and added to the array. The field is initialized to TRUE. The `Stop` command is the only command that sets CSTA to FALSE. On the other hand, the `Start` command is the only command that sets it to TRUE. Thus, `Start` must be invoked after each `Stop` command in order to enable counting; invoking `Read` will not enable signal counting after `Stop` has been invoked.

A typical use of these fields would be to initialize the CMD field to `Read` (it is initialized to this command by default), to use the `Stop` command to disable counting when necessary, after which the `Start` command can be invoked to re-start the signal count.

The WDTH field is a private field that holds the signal width of the array elements. For instance, if the LLIM was configured to be 4.0 and ULIM was configured to be 12.0 and the NELM was set to 4, then the WDTH for each array would be 2. Thus, it is (ULIM - LLIM) / NELM.

Field	Summary	Type	DCT	Default	Read	Write	CA PP
BPTR	Buffer Pointer	NOACCESS	No		No	No	No
VAL	Value	ULONG[]	No		Yes	Yes	No
MCNT	Counts Since Monitor	SHORT	No		Yes	No	No
CMD	Collection Control	MENU *histogramCMD*	No		Yes	Yes	No
CSTA	Collection Status	SHORT	No	1	Yes	No	No
WDTH	Element Width	DOUBLE	No		Yes	No	No

The following fields are used to operate the histogram record in simulation mode. See *"Fields Common to Many Record Types"* for more information on the simulation mode fields.

Field	Summary	Type	DCT	Default	Read	Write	CA PP
SIOL	Simulation Input Link	INLINK	Yes		Yes	Yes	No
SVAL	Simulation Value	DOUBLE	No		Yes	Yes	No
SIML	Simulation Mode Link	INLINK	Yes		Yes	Yes	No
SIMM	Simulation Mode	MENU menuYesNo	No		Yes	Yes	No
SIMS	Simulation Mode Severity	MENU menuAlarmSevr	Yes		Yes	Yes	No

Record Support

Record Support Routines

init_record

Using NELM, space for the unsigned long array is allocated and the width WDTH of the array is calculated.

This routine initializes SIMM with the value of SIML if SIML type is CONSTANT link or creates a channel access link if SIML type is PV_LINK. SVAL is likewise initialized if SIOL is CONSTANT or PV_LINK.

This routine next checks to see that device support and a device support read routine are available. If device support includes `init_record()`, it is called.

process

See next section.

special

Special is invoked whenever the fields CMD, SGNL, ULIM, or LLIM are changed.

If SGNL is changed, add_count is called.

If ULIM or LLIM are changed, WDTH is recalculated and clear_histogram is called.

If CMD is less or equal to 1, clear_histogram is called and CMD is reset to 0. If CMD is 2, CSTA is set to TRUE and CMD is reset to 0. If CMD is 3, CSTA is set to FALSE and CMD is reset to 0.

clear_histogram zeros out the histogram array. add_count increments the frequency in the histogram array.

cvt_dbaddr

This is called by dbNameToAddr. It makes the dbAddr structure refer to the actual buffer holding the array.

get_array_info

Obtains values from the array referenced by VAL.

put_array_info

Writes values into the array referenced by VAL.

Record Processing

Routine process implements the following algorithm:

1. Check to see that the appropriate device support module exists. If it doesn't, an error message is issued and processing is terminated with the PACT field set to TRUE. This ensures that processes will no longer be called for this record. Thus error storms will not occur.
2. readValue is called. See *"Input Records"* for more information
3. If PACT has been changed to TRUE, the device support read routine has started but has not completed writing the new value. In this case, the processing routine merely returns, leaving PACT TRUE.
4. Add count to histogram array.
5. Check to see if monitors should be invoked. Alarm monitors are invoked if the alarm status or severity has changed. Archive and value change monitors are invoked if MDEL conditions are met. NSEV and NSTA are reset to 0.
6. Scan forward link if necessary, set PACT and INIT to FALSE, and return.

Device Support

Fields Of Interest To Device Support

The device support routines are primarily interested in the following fields:

Field	Summary	Type	DCT	Default	Read	Write	CA PP
PACT	Record active	UCHAR	No		Yes	No	No
DPVT	Device Private	NOACCESS	No		No	No	No
UDF	Undefined	UCHAR	Yes	1	Yes	Yes	Yes
NSEV	New Alarm Severity	MENU menuAlarmSevr	No		Yes	No	No
NSTA	New Alarm Status	MENU menuAlarmStat	No		Yes	No	No
SVL	Signal Value Location	INLINK	Yes		Yes	Yes	No
SGNL	Signal Value	DOUBLE	No		Yes	Yes	No

Device Support Routines

Device support consists of the following routines:

long report(int level)

This optional routine is called by the IOC command `dbior` and is passed the report level that was requested by the user. It should print a report on the state of the device support to stdout. The `level` parameter may be used to output increasingly more detailed information at higher levels, or to select different types of information with different levels. Level zero should print no more than a small summary.

long init(int after)

This optional routine is called twice at IOC initialization time. The first call happens before any of the `init_record()` calls are made, with the integer parameter `after` set to 0. The second call happens after all of the `init_record()` calls have been made, with `after` set to 1.

init_record

```
init_record(precord)
```

This routine is called by the record support `init_record()` routine. It makes sure that SGNL is a CONSTANT, PV_LINK, DB_LINK, or CA_LINK. It also retrieves a value for SVL from SGNL. If SGNL is none of the above, an error is generated.

read_histogram

```
read_histogram(*precord)
```

This routine is called by the record support routines. It retrieves a value for SVL from SGNL.

Device Support For Soft Records

Only the device support module `Soft Channel` is currently provided, though other device support modules may be provided at the user's site.

Soft Channel

The `Soft Channel` device support routine retrieves a value from SGNL. SGNL must be CONSTANT, PV_LINK, DB_LINK, or CA_LINK.

1.5.15 64bit Integer Input Record (int64in)

This record type is normally used to obtain an integer value of up to 64 bits from a hardware input. The record supports alarm limits, alarm filtering, graphics and control limits.

Parameter Fields

The record-specific fields are described below.

Input Specification

These fields control where the record will read data from when it is processed:

Field	Summary	Type	DCT	Default	Read	Write	CA PP
DTYP	Device Type	DEVICE	Yes		Yes	Yes	No
INP	Input Specification	INLINK	Yes		Yes	Yes	No

The DTYP field selects which device support layer should be responsible for providing input data to the record. The int64in device support layers provided by EPICS Base are documented in the *"Device Support"* section. External support modules may provide additional device support for this record type. If not set explicitly, the DTYP value defaults to the first device support that is loaded for the record type, which will usually be the `Soft Channel` support that comes with Base.

The INP link field contains a database or channel access link or provides hardware address information that the device support uses to determine where the input data should come from.

Operator Display Parameters

These parameters are used to present meaningful data to the operator. They do not affect the functioning of the record.

- DESC is a string that is usually used to briefly describe the record.
- EGU is a string of up to 16 characters naming the engineering units that the VAL field represents.
- The HOPR and LOPR fields set the upper and lower display limits for the VAL, HIHI, HIGH, LOW, and LOLO fields.

Field	Summary	Type	DCT	Default	Read	Write	CA PP
DESC	Descriptor	STRING [41]	Yes		Yes	Yes	No
EGU	Units name	STRING [16]	Yes		Yes	Yes	No
HOPR	High Operating Range	INT64	Yes		Yes	Yes	No
LOPR	Low Operating Range	INT64	Yes		Yes	Yes	No

Alarm Limits

The user configures limit alarms by putting numerical values into the HIHI, HIGH, LOW and LOLO fields, and by setting the associated alarm severity in the corresponding HHSV, HSV, LSV and LLSV menu fields.

The HYST field controls hysteresis to prevent alarm chattering from an input signal that is close to one of the limits and suffers from significant readout noise.

The AFTC field sets the time constant on a low-pass filter that delays the reporting of limit alarms until the signal has been within the alarm range for that number of seconds (the default AFTC value of zero retains the previous behavior).

The LALM field is used by the record at run-time to implement the alarm limit functionality.

Field	Summary	Type	DCT	Default	Read	Write	CA PP
HIHI	Hihi Alarm Limit	INT64	Yes		Yes	Yes	Yes
HIGH	High Alarm Limit	INT64	Yes		Yes	Yes	Yes
LOW	Low Alarm Limit	INT64	Yes		Yes	Yes	Yes
LOLO	Lolo Alarm Limit	INT64	Yes		Yes	Yes	Yes
HHSV	Hihi Severity	MENU menuAlarmSevr	Yes		Yes	Yes	Yes
HSV	High Severity	MENU menuAlarmSevr	Yes		Yes	Yes	Yes
LSV	Low Severity	MENU menuAlarmSevr	Yes		Yes	Yes	Yes
LLSV	Lolo Severity	MENU menuAlarmSevr	Yes		Yes	Yes	Yes
HYST	Alarm Deadband	INT64	Yes		Yes	Yes	No
AFTC	Alarm Filter Time Constant	DOUBLE	Yes		Yes	Yes	No
LALM	Last Value Alarmed	INT64	No		Yes	No	No

Monitor Parameters

These parameters are used to determine when to send monitors placed on the VAL field. The monitors are sent when the current value exceeds the last transmitted value by the appropriate deadband. If these fields are set to zero, a monitor will be triggered every time the value changes; if set to -1, a monitor will be sent every time the record is processed.

The ADEL field sets the deadband for archive monitors (`DBE_LOG` events), while the MDEL field controls value monitors (`DBE_VALUE` events).

The remaining fields are used by the record at run-time to implement the record monitoring deadband functionality.

Field	Summary	Type	DCT	Default	Read	Write	CA PP
ADEL	Archive Deadband	INT64	Yes		Yes	Yes	No
MDEL	Monitor Deadband	INT64	Yes		Yes	Yes	No
ALST	Last Value Archived	INT64	No		Yes	No	No
MLST	Last Val Monitored	INT64	No		Yes	No	No

Simulation Mode Parameters

The following fields are used to operate the record in simulation mode.

If SIMM (fetched through SIML) is YES, the record is put in SIMS severity and the value is fetched through SIOL (buffered in SVAL). SSCN sets a different SCAN mechanism to use in simulation mode. SDLY sets a delay (in sec) that is used for asynchronous simulation processing.

See *Input Simulation Fields* for more information on simulation mode and its fields.

Field	Summary	Type	DCT	Default	Read	Write	CA PP
SIML	Simulation Mode Link	INLINK	Yes		Yes	Yes	No
SIMM	Simulation Mode	MENU menuYesNo	No		Yes	Yes	No
SIOL	Simulation Input Link	INLINK	Yes		Yes	Yes	No
SVAL	Simulation Value	INT64	No		Yes	Yes	No
SIMS	Simulation Mode Severity	MENU menuAlarmSevr	Yes		Yes	Yes	No
SDLY	Sim. Mode Async Delay	DOUBLE	Yes	-1.0	Yes	Yes	No
SSCN	Sim. Mode Scan	MENU menuScan	Yes	65535	Yes	Yes	No

Record Support

Record Support Routines

The following are the record support routines that would be of interest to an application developer. Other routines are the `get_units`, `get_graphic_double`, `get_alarm_double` and `get_control_double` routines, which are used to collect properties from the record for the complex DBR data structures.

init_record

This routine first initializes the simulation mode mechanism by setting SIMM if SIML is a constant, and setting SVAL if SIOL is a constant.

It then checks if the device support and the device support's `read_int64in` routine are defined. If either one does not exist, an error message is issued and processing is terminated.

If device support includes `init_record`, it is called.

Finally, the deadband mechanisms for monitors and level alarms are initialized.

process

See next section.

Record Processing

Routine `process` implements the following algorithm:

1. Check to see that the appropriate device support module and its `read_int64in` routine are defined. If either one does not exist, an error message is issued and processing is terminated with the PACT field set to TRUE, effectively blocking the record to avoid error storms.
2. Determine the value:

   If PACT is TRUE, call the device support `read_int64in` routine and return.

   If PACT is FALSE, read the value, honoring simulation mode:

   - Get SIMM by reading the SIML link.
   - If SIMM is `NO`, call the device support `read_int64in` routine and return.
   - If SIMM is `YES`, then
     - Set alarm status to SIMM_ALARM and severity to SIMS, if SIMS is greater than zero.
     - If the record simulation processing is synchronous (SDLY < 0) or the record is in the second phase of an asynchronous processing, call `dbGetLink()` to read the input value from SIOL into SVAL. Set status to the return code from `dbGetLink()`. If the call succeeded, write the value to VAL and set UDF to 0.

       Otherwise (record is in first phase of an asynchronous processing), set up a callback processing with the delay specified in SDLY.
   - Raise an alarm for other values of SIMM.
3. If PACT has been changed to TRUE, the device support signals asynchronous processing: its `read_int64in` output routine has started, but not completed reading the new value. In this case, the processing routine merely returns, leaving PACT TRUE.
4. Set PACT to TRUE. Get the processing time stamp. Set UDF to 0 if reading the value was successful.
5. Check UDF and level alarms: This routine checks to see if the record is undefined (UDF is TRUE) or if the new VAL causes the alarm status and severity to change. In the latter case, NSEV, NSTA and LALM are set. It also honors the alarm hysteresis factor (HYST): the value must change by at least HYST between level alarm status and severity changes. If AFTC is set, alarm level filtering is applied.
6. Check to see if monitors should be invoked:

   - Alarm monitors are posted if the alarm status or severity have changed.
   - Archive and value change monitors are posted if ADEL and MDEL conditions (see *"Monitor Parameters"*) are met.

7. Scan (process) forward link if necessary, set PACT to FALSE, and return.

Device Support

Device Support Interface

The record requires device support to provide an entry table (dset) which defines the following members:

```
typedef struct {
    long number;
    long (*report)(int level);
    long (*init)(int after);
    long (*init_record)(int64inRecord *prec);
    long (*get_ioint_info)(int cmd, int64inRecord *prec, IOSCANPVT *piosl);
    long (*read_int64in)(int64inRecord *prec);
} int64indset;
```

The module must set `number` to at least 5, and provide a pointer to its `read_int64in()` routine; the other function pointers may be `NULL` if their associated functionality is not required for this support layer. Most device supports also provide an `init_record()` routine to configure the record instance and connect it to the hardware or driver support layer.

The individual routines are described below.

Device Support Routines

long report(int level)

This optional routine is called by the IOC command `dbior` and is passed the report level that was requested by the user. It should print a report on the state of the device support to stdout. The `level` parameter may be used to output increasingly more detailed information at higher levels, or to select different types of information with different levels. Level zero should print no more than a small summary.

long init(int after)

This optional routine is called twice at IOC initialization time. The first call happens before any of the `init_record()` calls are made, with the integer parameter `after` set to 0. The second call happens after all of the `init_record()` calls have been made, with `after` set to 1.

long init_record(int64inRecord *prec)

This optional routine is called by the record initialization code for each int64in record instance that has its DTYP field set to use this device support. It is normally used to check that the INP address is the expected type and that it points to a valid device; to allocate any record-specific buffer space and other memory; and to connect any communication channels needed for the `read_int64in()` routine to work properly.

long get_ioint_info(int cmd, int64inRecord *prec, IOSCANPVT *piosl)

This optional routine is called whenever the record's SCAN field is being changed to or from the value `I/O Intr` to find out which I/O Interrupt Scan list the record should be added to or deleted from. If this routine is not provided, it will not be possible to set the SCAN field to the value `I/O Intr` at all.

The `cmd` parameter is zero when the record is being added to the scan list, and one when it is being removed from the list. The routine must determine which interrupt source the record should be connected to, which it indicates by the scan list that it points the location at `*piosl` to before returning. It can prevent the SCAN field from being changed at all by returning a non-zero value to its caller.

In most cases the device support will create the I/O Interrupt Scan lists that it returns for itself, by calling `void scanIoInit(IOSCANPVT *piosl)` once for each separate interrupt source. That routine allocates memory and inializes the list, then passes back a pointer to the new list in the location at `*piosl`.

When the device support receives notification that the interrupt has occurred, it announces that to the IOC by calling `void scanIoRequest(IOSCANPVT iosl)` which will arrange for the appropriate records to be processed in a suitable thread. The `scanIoRequest()` routine is safe to call from an interrupt service routine on embedded architectures (vxWorks and RTEMS).

long read_int64in(int64inRecord *prec)

This essential routine is called when the record wants a new value from the addressed device. It is responsible for performing (or at least initiating) a read operation, and (eventually) returning its value to the record.

If the device may take more than a few microseconds to return the new value, this routine must never block (busy-wait), but use the asynchronous processing mechanism. In that case it signals the asynchronous operation by setting the record's PACT field to TRUE before it returns, having arranged for the record's `process()` routine to be called later once the read operation is finished. When that happens, the `read_int64in()` routine will be called again with PACT still set to TRUE; it should then set it to FALSE to indicate the read has completed, and return.

A return value of zero indicates success, any other value indicates that an error occurred.

Extended Device Support

…

Device Support For Soft Records

Two soft device support modules, Soft Channel and Soft Callback Channel, are provided for input records not related to actual hardware devices. The INP link type must be either a CONSTANT, DB_LINK, or CA_LINK.

Soft Channel

This module reads the value using the record's INP link.

`read_int64in` calls `dbGetLink` to read the value.

Soft Callback Channel

This module is like the previous except that it reads the value using asynchronous processing that will not complete until an asynchronous processing of the INP target record has completed.

1.5.16 64bit Integer Output Record (int64out)

This record type is normally used to send an integer value of up to 64 bits to an output device. The record supports alarm, drive, graphics and control limits.

Parameter Fields

The record-specific fields are described below.

Output Value Determination

These fields control how the record determines the value to be output when it gets processed:

Field	Summary	Type	DCT	Default	Read	Write	CA PP
OMSL	Output Mode Select	MENU menuOmsl	Yes		Yes	Yes	No
DOL	Desired Output Link	INLINK	Yes		Yes	Yes	No
DRVH	Drive High Limit	INT64	Yes		Yes	Yes	Yes
DRVL	Drive Low Limit	INT64	Yes		Yes	Yes	Yes
VAL	Desired Output	INT64	Yes		Yes	Yes	Yes

The following steps are performed in order during record processing.

Fetch Value

The OMSL menu field is used to determine whether the DOL link field should be used during processing or not:

- If OMSL is `supervisory` the DOL link field is not used. The new output value is taken from the VAL field, which may have been set from elsewhere.
- If OMSL is `closed_loop` the DOL link field is used to obtain a value.

Drive Limits

The output value is clipped to the range DRVL to DRVH inclusive, provided that DRVH > DRVL. The result is copied into the VAL field.

Output Specification

These fields control where the record will read data from when it is processed:

Field	Summary	Type	DCT	Default	Read	Write	CA PP
DTYP	Device Type	DEVICE	Yes		Yes	Yes	No
OUT	Output Specification	OUTLINK	Yes		Yes	Yes	No

The DTYP field selects which device support layer should be responsible for writing output data. The int64out device support layers provided by EPICS Base are documented in the *"Device Support"* section. External support modules may provide additional device support for this record type. If not set explicitly, the DTYP value defaults to the first device support that is loaded for the record type, which will usually be the `Soft Channel` support that comes with Base.

The OUT link field contains a database or channel access link or provides hardware address information that the device support uses to determine where the output data should be sent to.

Operator Display Parameters

These parameters are used to present meaningful data to the operator. They do not affect the functioning of the record.

- DESC is a string that is usually used to briefly describe the record.
- EGU is a string of up to 16 characters naming the engineering units that the VAL field represents.
- The HOPR and LOPR fields set the upper and lower display limits for the VAL, HIHI, HIGH, LOW, and LOLO fields.

Field	Summary	Type	DCT	Default	Read	Write	CA PP
DESC	Descriptor	STRING [41]	Yes		Yes	Yes	No
EGU	Units name	STRING [16]	Yes		Yes	Yes	No
HOPR	High Operating Range	INT64	Yes		Yes	Yes	No
LOPR	Low Operating Range	INT64	Yes		Yes	Yes	No

Alarm Limits

The user configures limit alarms by putting numerical values into the HIHI, HIGH, LOW and LOLO fields, and by setting the associated alarm severities in the corresponding HHSV, HSV, LSV and LLSV menu fields.

The HYST field controls hysteresis to prevent alarm chattering from an input signal that is close to one of the limits and suffers from significant readout noise.

The LALM field is used by the record at run-time to implement the alarm limit hysteresis functionality.

Field	Summary	Type	DCT	Default	Read	Write	CA PP
HIHI	Hihi Alarm Limit	INT64	Yes		Yes	Yes	Yes
HIGH	High Alarm Limit	INT64	Yes		Yes	Yes	Yes
LOW	Low Alarm Limit	INT64	Yes		Yes	Yes	Yes
LOLO	Lolo Alarm Limit	INT64	Yes		Yes	Yes	Yes
HHSV	Hihi Severity	MENU menuAlarmSevr	Yes		Yes	Yes	Yes
HSV	High Severity	MENU menuAlarmSevr	Yes		Yes	Yes	Yes
LSV	Low Severity	MENU menuAlarmSevr	Yes		Yes	Yes	Yes
LLSV	Lolo Severity	MENU menuAlarmSevr	Yes		Yes	Yes	Yes
HYST	Alarm Deadband	INT64	Yes		Yes	Yes	No
LALM	Last Value Alarmed	INT64	No		Yes	No	No

Monitor Parameters

These parameters are used to determine when to send monitors placed on the VAL field. The monitors are sent when the current value exceeds the last transmitted value by the appropriate deadband. If these fields are set to zero, a monitor will be triggered every time the value changes; if set to -1, a monitor will be sent every time the record is processed.

The ADEL field sets the deadband for archive monitors (`DBE_LOG` events), while the MDEL field controls value monitors (`DBE_VALUE` events).

The remaining fields are used by the record at run-time to implement the record monitoring deadband functionality.

Field	Summary	Type	DCT	Default	Read	Write	CA PP
ADEL	Archive Deadband	INT64	Yes		Yes	Yes	No
MDEL	Monitor Deadband	INT64	Yes		Yes	Yes	No
ALST	Last Value Archived	INT64	No		Yes	No	No
MLST	Last Val Monitored	INT64	No		Yes	No	No

Simulation Mode Parameters

The following fields are used to operate the record in simulation mode.

If SIMM (fetched through SIML) is YES, the record is put in SIMS severity and the value is written through SIOL. SSCN sets a different SCAN mechanism to use in simulation mode. SDLY sets a delay (in sec) that is used for asynchronous simulation processing.

See *Output Simulation Fields* for more information on simulation mode and its fields.

Field	Summary	Type	DCT	Default	Read	Write	CA PP
SIML	Simulation Mode Link	INLINK	Yes		Yes	Yes	No
SIMM	Simulation Mode	MENU menuYesNo	No		Yes	Yes	No
SIOL	Simulation Output Link	OUTLINK	Yes		Yes	Yes	No
SIMS	Simulation Mode Severity	MENU menuAlarmSevr	Yes		Yes	Yes	No
SDLY	Sim. Mode Async Delay	DOUBLE	Yes	-1.0	Yes	Yes	No
SSCN	Sim. Mode Scan	MENU menuScan	Yes	65535	Yes	Yes	No

Invalid Alarm Output Action

Whenever an output record is put into INVALID alarm severity, IVOA specifies the action to take.

- `Continue normally` (default)

  Write the value. Same as if severity is lower than INVALID.

- `Don't drive outputs`

  Do not write value.

- `Set output to IVOV`

  Set VAL to IVOV, then write the value.

Field	Summary	Type	DCT	Default	Read	Write	CA PP
IVOA	INVALID output action	MENU menuIvoa	Yes		Yes	Yes	No
IVOV	INVALID output value	INT64	Yes		Yes	Yes	No

Record Support

Record Support Routines

The following are the record support routines that would be of interest to an application developer. Other routines are the `get_units`, `get_graphic_double`, `get_alarm_double` and `get_control_double` routines, which are used to collect properties from the record for the complex DBR data structures.

init_record

This routine first initializes the simulation mode mechanism by setting SIMM if SIML is a constant.

It then checks if the device support and the device support's `write_int64out` routine are defined. If either one does not exist, an error message is issued and processing is terminated.

If DOL is a constant, then VAL is initialized with its value and UDF is set to FALSE.

If device support includes `init_record`, it is called.

Finally, the deadband mechanisms for monitors and level alarms are initialized.

process

See next section.

Record Processing

Routine `process` implements the following algorithm:

1. Check to see that the appropriate device support module and its `write_int64out` routine are defined. If either one does not exist, an error message is issued and processing is terminated with the PACT field set to TRUE, effectively blocking the record to avoid error storms.
2. Check PACT. If PACT is FALSE, do the following:
    - Determine value, honoring closed loop mode: if DOL is not a CONSTANT and OMSL is CLOSED_LOOP then get value from DOL setting UDF to FALSE in case of success, else use the VAL field.
    - Call `convert`: if drive limits are defined then force value to be within those limits.
3. Check UDF and level alarms: This routine checks to see if the record is undefined (UDF is TRUE) or if the new VAL causes the alarm status and severity to change. In the latter case, NSEV, NSTA and LALM are set. It also honors the alarm hysteresis factor (HYST): the value must change by at least HYST between level alarm status and severity changes.
4. Check severity and write the new value. See *Invalid Output Action Fields* for details on how invalid alarms affect output records.
5. If PACT has been changed to TRUE, the device support signals asynchronous processing: its `write_int64out` output routine has started, but not completed writing the new value. In this case, the processing routine merely returns, leaving PACT TRUE.
6. Check to see if monitors should be invoked:
    - Alarm monitors are posted if the alarm status or severity have changed.
    - Archive and value change monitors are posted if ADEL and MDEL conditions (see *"Monitor Parameters"*) are met.
    - NSEV and NSTA are reset to 0.
7. Scan (process) forward link if necessary, set PACT to FALSE, and return.

Device Support

Device Support Interface

The record requires device support to provide an entry table (dset) which defines the following members:

```
typedef struct {
    long number;
    long (*report)(int level);
    long (*init)(int after);
    long (*init_record)(int64outRecord *prec);
    long (*get_ioint_info)(int cmd, int64outRecord *prec, IOSCANPVT *piosl);
    long (*write_int64out)(int64outRecord *prec);
} int64outdset;
```

The module must set `number` to at least 5, and provide a pointer to its `write_int64out()` routine; the other function pointers may be `NULL` if their associated functionality is not required for this support layer. Most device supports also provide an `init_record()` routine to configure the record instance and connect it to the hardware or driver support layer.

The individual routines are described below.

Device Support Routines

long report(int level)

This optional routine is called by the IOC command `dbior` and is passed the report level that was requested by the user. It should print a report on the state of the device support to stdout. The `level` parameter may be used to output increasingly more detailed information at higher levels, or to select different types of information with different levels. Level zero should print no more than a small summary.

long init(int after)

This optional routine is called twice at IOC initialization time. The first call happens before any of the `init_record()` calls are made, with the integer parameter `after` set to 0. The second call happens after all of the `init_record()` calls have been made, with `after` set to 1.

long init_record(int64outRecord *prec)

This optional routine is called by the record initialization code for each int64out record instance that has its DTYP field set to use this device support. It is normally used to check that the OUT address is the expected type and that it points to a valid device, to allocate any record-specific buffer space and other memory, and to connect any communication channels needed for the `write_int64out()` routine to work properly.

long get_ioint_info(int cmd, int64outRecord *prec, IOSCANPVT *piosl)

This optional routine is called whenever the record's SCAN field is being changed to or from the value `I/O Intr` to find out which I/O Interrupt Scan list the record should be added to or deleted from. If this routine is not provided, it will not be possible to set the SCAN field to the value `I/O Intr` at all.

The `cmd` parameter is zero when the record is being added to the scan list, and one when it is being removed from the list. The routine must determine which interrupt source the record should be connected to, which it indicates by the scan list that it points the location at `*piosl` to before returning. It can prevent the SCAN field from being changed at all by returning a non-zero value to its caller.

In most cases the device support will create the I/O Interrupt Scan lists that it returns for itself, by calling `void scanIoInit(IOSCANPVT *piosl)` once for each separate interrupt source. That routine allocates memory and inializes the list, then passes back a pointer to the new list in the location at `*piosl`.

When the device support receives notification that the interrupt has occurred, it announces that to the IOC by calling `void scanIoRequest(IOSCANPVT iosl)` which will arrange for the appropriate records to be processed in a suitable thread. The `scanIoRequest()` routine is safe to call from an interrupt service routine on embedded architectures (vxWorks and RTEMS).

long write_int64out(int64outRecord *prec)

This essential routine is called when the record wants to write a new value to the addressed device. It is responsible for performing (or at least initiating) a write operation, using the value from the record's VAL field.

If the device may take more than a few microseconds to accept the new value, this routine must never block (busy-wait), but use the asynchronous processing mechanism. In that case it signals the asynchronous operation by setting the record's PACT field to TRUE before it returns, having arranged for the record's `process()` routine to be called later once the write operation is over. When that happens, the `write_int64out()` routine will be called again with PACT still set to TRUE; it should then set it to FALSE to indicate the write has completed, and return.

A return value of zero indicates success, any other value indicates that an error occurred.

Extended Device Support

…

Device Support For Soft Records

Two soft device support modules, Soft Channel and Soft Callback Channel, are provided for output records not related to actual hardware devices. The OUT link type must be either a CONSTANT, DB_LINK, or CA_LINK.

Soft Channel

This module writes the current value using the record's VAL field.

`write_int64out` calls `dbPutLink` to write the current value.

Soft Callback Channel

This module is like the previous except that it writes the current value using asynchronous processing that will not complete until an asynchronous processing of the target record has completed.

1.5.17 Long Input Record (longin)

The normal use for the long input record or "longin" record is to retrieve a long integer value of up to 32 bits. Device support routines are provided to support direct interfaces to hardware. In addition, the `Soft Channel` device module is provided to obtain input via database or channel access links or via dbPutField or dbPutLink requests.

Parameter Fields

The record-specific fields are described below, grouped by functionality.

Scan Parameters

The long input record has the standard fields for specifying under what circumstances the record will be processed. These fields are listed in *Scan Fields*.

Read Parameters

The device support routines use the INP field to obtain the record's input. For records that obtain their input from devices, the INP field must contain the address of the I/O card, and the DTYP field must specify the proper device support module. Be aware that the address format differs according to the I/O bus used.

For soft records, the INP can be a constant, a database link, or a channel access link. The value is read directly into VAL. The `Soft Channel` device support module is available for longin records.

Field	Summary	Type	DCT	Default	Read	Write	CA PP
VAL	Current value	LONG	Yes		Yes	Yes	Yes
INP	Input Specification	INLINK	Yes		Yes	Yes	No
DTYP	Device Type	DEVICE	Yes		Yes	Yes	No

Operator Display Parameters

These parameters are used to present meaningful data to the operator. These fields are used to display the value and other parameters of the long input either textually or graphically.

EGU is a string of up to 16 characters describing the units that the long input measures. It is retrieved by the `get_units` record support routine.

The HOPR and LOPR fields set the upper and lower display limits for the VAL, HIHI, HIGH, LOW, and LOLO fields. Both the `get_graphic_double` and `get_control_double` record support routines retrieve these fields.

See *Fields Common to All Record Types* for more on the record name (NAME) and description (DESC) fields.

Field	Summary	Type	DCT	Default	Read	Write	CA PP
EGU	Engineering Units	STRING [16]	Yes		Yes	Yes	No
HOPR	High Operating Range	LONG	Yes		Yes	Yes	No
LOPR	Low Operating Range	LONG	Yes		Yes	Yes	No
NAME	Record Name	STRING [61]	No		Yes	No	No
DESC	Descriptor	STRING [41]	Yes		Yes	Yes	No

Alarm Parameters

The possible alarm conditions for long inputs are the SCAN, READ, and limit alarms. The SCAN and READ alarms are called by the record or device support routines.

The limit alarms are configured by the user in the HIHI, LOLO, HIGH, and LOW fields using numerical values. For each of these fields, there is a corresponding severity field which can be either NO_ALARM, MINOR, or MAJOR. The HYST field can be used to specify a deadband around each limit. *Alarm Fields* lists the fields related to alarms that are common to all record types.

Field	Summary	Type	DCT	Default	Read	Write	CA PP
HIHI	Hihi Alarm Limit	LONG	Yes		Yes	Yes	Yes
HIGH	High Alarm Limit	LONG	Yes		Yes	Yes	Yes
LOW	Low Alarm Limit	LONG	Yes		Yes	Yes	Yes
LOLO	Lolo Alarm Limit	LONG	Yes		Yes	Yes	Yes
HHSV	Hihi Severity	MENU menuAlarmSevr	Yes		Yes	Yes	Yes
HSV	High Severity	MENU menuAlarmSevr	Yes		Yes	Yes	Yes
LSV	Low Severity	MENU menuAlarmSevr	Yes		Yes	Yes	Yes
LLSV	Lolo Severity	MENU menuAlarmSevr	Yes		Yes	Yes	Yes
HYST	Alarm Deadband	LONG	Yes		Yes	Yes	No

Monitor Parameters

These parameters are used to determine when to send monitors placed on the value field. The monitors are sent when the value field exceeds the last monitored field (see the next section) by the appropriate deadband. If these fields have a value of zero, everytime the value changes, a monitor will be triggered; if they have a value of -1, everytime the record is scanned, monitors are triggered. The ADEL field is used by archive monitors and the MDEL field for all other types of monitors.

Field	Summary	Type	DCT	Default	Read	Write	CA PP
ADEL	Archive Deadband	LONG	Yes		Yes	Yes	No
MDEL	Monitor Deadband	LONG	Yes		Yes	Yes	No

Run-time Parameters

The LALM, MLST, and ALST fields are used to implement the hysteresis factors for monitor callbacks. Only if the difference between these fields and the corresponding value field is greater than the appropriate delta (MDEL, ADEL, HYST) will monitors be triggered. For instance, only if the difference between VAL and MLST is greater than MDEL are the monitors triggered for VAL.

Field	Summary	Type	DCT	Default	Read	Write	CA PP
LALM	Last Value Alarmed	LONG	No		Yes	No	No
ALST	Last Value Archived	LONG	No		Yes	No	No
MLST	Last Val Monitored	LONG	No		Yes	No	No

Simulation Mode Parameters

The following fields are used to operate the record in simulation mode.

If SIMM (fetched through SIML) is YES, the record is put in SIMS severity and the value is fetched through SIOL (buffered in SVAL). SSCN sets a different SCAN mechanism to use in simulation mode. SDLY sets a delay (in sec) that is used for asynchronous simulation processing.

See *Input Simulation Fields* for more information on simulation mode and its fields.

Field	Summary	Type	DCT	Default	Read	Write	CA PP
SIML	Sim Mode Location	INLINK	Yes		Yes	Yes	No
SIMM	Simulation Mode	MENU menuYesNo	No		Yes	Yes	No
SIOL	Sim Input Specifctn	INLINK	Yes		Yes	Yes	No
SVAL	Simulation Value	LONG	No		Yes	Yes	No
SIMS	Sim mode Alarm Svrty	MENU menuAlarmSevr	Yes		Yes	Yes	No
SDLY	Sim. Mode Async Delay	DOUBLE	Yes	-1.0	Yes	Yes	No
SSCN	Sim. Mode Scan	MENU menuScan	Yes	65535	Yes	Yes	No

Record Support

Record Support Routines

init_record

This routine initializes SIMM with the value of SIML if SIML type is CONSTANT link or creates a channel access link if SIML type is PV_LINK. SVAL is likewise initialized if SIOL is CONSTANT or PV_LINK.

This routine next checks to see that device support is available and a device support read routine is defined. If either does not exist, an error message is issued and processing is terminated.

If device support includes `init_record()`, it is called.

process

See next section.

get_units

Retrieves EGU.

get_graphic_double

Sets the upper display and lower display limits for a field. If the field is VAL, HIHI, HIGH, LOW, or LOLO, the limits are set to HOPR and LOPR, else if the field has upper and lower limits defined they will be used, else the upper and lower maximum values for the field type will be used.

get_control_double

Sets the upper control and the lower control limits for a field. If the field is VAL, HIHI, HIGH, LOW, or LOLO, the limits are set to HOPR and LOPR, else if the field has upper and lower limits defined they will be used, else the upper and lower maximum values for the field type will be used.

get_alarm_double

Sets the following values:

```
upper_alarm_limit = HIHI
upper_warning_limit = HIGH
lower_warning_limit = LOW
lower_alarm_limit = LOLO
```

Record Processing

Routine process implements the following algorithm:

1. Check to see that the appropriate device support module exists. If it doesn't, an error message is issued and processing is terminated with the PACT field still set to TRUE. This ensures that processes will no longer be called for this record. Thus error storms will not occur.
2. readValue is called. See *"Input Records"* for more information.
3. If PACT has been changed to TRUE, the device support read routine has started but has not completed reading a new input value. In this case, the processing routine merely returns, leaving PACT TRUE.
4. Check alarms. This routine checks to see if the new VAL causes the alarm status and severity to change. If so, NSEV, NSTA and LALM are set. It also honors the alarm hysteresis factor (HYST). Thus the value must change by more than HYST before the alarm status and severity is lowered.
5. Check to see if monitors should be invoked:
   - Alarm monitors are invoked if the alarm status or severity has changed.
   - Archive and value change monitors are invoked if ADEL and MDEL conditions are met.
   - NSEV and NSTA are reset to 0.
6. Scan forward link if necessary, set PACT FALSE, and return.

Device Support

Fields Of Interest To Device Support

Each long input record must have an associated set of device support routines. The primary responsibility of the device support routines is to obtain a new input value whenever read_longin is called. The device support routines are primarily interested in the following fields:

Field	Summary	Type	DCT	Default	Read	Write	CA PP
PACT	Record active	UCHAR	No		Yes	No	No
DPVT	Device Private	NOACCESS	No		No	No	No
UDF	Undefined	UCHAR	Yes	1	Yes	Yes	Yes
NSEV	New Alarm Severity	MENU menuAlarmSevr	No		Yes	No	No
NSTA	New Alarm Status	MENU menuAlarmStat	No		Yes	No	No
VAL	Current value	LONG	Yes		Yes	Yes	Yes
INP	Input Specification	INLINK	Yes		Yes	Yes	No

Device Support Routines

Device support consists of the following routines:

long report(int level)

This optional routine is called by the IOC command `dbior` and is passed the report level that was requested by the user. It should print a report on the state of the device support to stdout. The `level` parameter may be used to output increasingly more detailed information at higher levels, or to select different types of information with different levels. Level zero should print no more than a small summary.

long init(int after)

This optional routine is called twice at IOC initialization time. The first call happens before any of the `init_record()` calls are made, with the integer parameter `after` set to 0. The second call happens after all of the `init_record()` calls have been made, with `after` set to 1.

init_record

```
init_record(precord)
```

This routine is optional. If provided, it is called by the record support `init_record()` routine.

get_ioint_info

```
get_ioint_info(int cmd,struct dbCommon *precord,IOSCANPVT *ppvt)
```

This routine is called by the ioEventScan system each time the record is added or deleted from an I/O event scan list. `cmd` has the value (0,1) if the record is being (added to, deleted from) an I/O event list. It must be provided for any device type that can use the ioEvent scanner.

read_longin

```
read_longin(precord)
```

This routine must provide a new input value. It returns the following values:

- 0: Success. A new value is placed in VAL.
- Other: Error.

Device Support For Soft Records

The `Soft Channel` device support module places a value directly in VAL.

If the INP link type is constant, then the constant value is stored into VAL by `init_record()`, and UDF is set to FALSE. If the INP link type is PV_LINK, then dbCaAddInlink is called by `init_record()`.

`read_longin` calls recGblGetLinkValue to read the current value of VAL. See *"Soft Input"* for more information

If the return status of `recGblGetLinkValue` is zero then read_longin sets UDF to FALSE. read_longin returns the status of `recGblGetLinkValue`.

1.5.18 Long Output Record (longout)

The normal use for the long output or "longout" record type is to store long integer values of up to 32 bits and write them to hardware devices. The `Soft Channel` device support layer can also be used to write values to other records via database or channel access links. The OUT field determines how the record is used. The record supports alarm limits and graphics and control limits.

Parameter Fields

The record-specific fields are described below, grouped by functionality.

Scan Parameters

The longout record has the standard fields for specifying under what circumstances it will be processed. These fields are listed in *Scan Fields*.

Desired Output Parameters

The record must specify where the desired output originates, i.e., the 32 bit integer value it is to write. The output mode select (OMSL) field determines whether the output originates from another record or from database access. When set to `closed_loop`, the desired output is retrieved from the link specified in the Desired Output Link (DOL) field (which can specify either a database or channel access link) and placed into the VAL field. When set to `supervisory`, the desired output can be written into the VAL field via dpPuts at run-time.

A third type of value for the DOL field is a constant in which case, when the record is initialized, the VAL field will be initialized with this constant value.

The VAL field's value will be clipped within limits specified in the fields DRVH and DRVL if these have been configured by the database designer:

```
DRVL <= VAL <= DRVH
```

Note: These limits are only enforced as long as DRVH > DRVL. If they are not set or DRVH <= DRVL they will not be used.

Field	Summary	Type	DCT	Default	Read	Write	CA PP
DOL	Desired Output Link	INLINK	Yes		Yes	Yes	No
OMSL	Output Mode Select	MENU menuOmsl	Yes		Yes	Yes	No
DRVH	Drive High Limit	LONG	Yes		Yes	Yes	Yes
DRVL	Drive Low Limit	LONG	Yes		Yes	Yes	Yes
VAL	Desired Output	LONG	Yes		Yes	Yes	Yes

Write Parameters

The OUT link field determines where the record is to send its output. For records that write values to hardware devices, the OUT output link field must specify the address of the I/O card, and the DTYP field must specify the name of the corresponding device support module.

For soft records, the OUT output link can be a constant, a database link, or a channel access link. If the link is a constant, the result is no output. The DTYP field must then specify the `Soft Channel` device support routine.

See Address Specification for information on the format of hardware addresses and database links.

Field	Summary	Type	DCT	Default	Read	Write	CA PP
OUT	Output Specification	OUTLINK	Yes		Yes	Yes	No
DTYP	Device Type	DEVICE	Yes		Yes	Yes	No

Operator Display Parameters

These parameters are used to present meaningful data to the operator. They display the value and other parameters of the long output either textually or graphically.

EGU is a string of up to 16 characters describing the units that the long output measures. It is retrieved by the `get_units` record support routine.

The HOPR and LOPR fields set the upper and lower display limits for the VAL, HIHI, HIGH, LOW, and LOLO fields. Both the `get_graphic_double` and `get_control_double` record support routines retrieve these fields.

See *Fields Common to All Record Types* for more on the record name (NAME) and description (DESC) fields.

Field	Summary	Type	DCT	Default	Read	Write	CA PP
EGU	Engineering Units	STRING [16]	Yes		Yes	Yes	No
HOPR	High Operating Range	LONG	Yes		Yes	Yes	No
LOPR	Low Operating Range	LONG	Yes		Yes	Yes	No
NAME	Record Name	STRING [61]	No		Yes	No	No
DESC	Descriptor	STRING [41]	Yes		Yes	Yes	No

Alarm Parameters

The possible alarm conditions for long inputs are the SCAN, READ, INVALID, and limit alarms. The SCAN and READ alarms are not configurable by the user because their severity is always MAJOR. The INVALID alarm is called by the record support routine when the record or device support routines cannot write the record's output. The IVOA field specifies the action to take in this case.

The limit alarms are configured by the user in the HIHI, LOLO, HIGH, and LOW fields using floating-point values. For each of these fields, there is a corresponding severity field which can be either NO_ALARM, MINOR, or MAJOR.

The HYST field sets an alarm deadband around each limit alarm.

For an explanation of the IVOA and IVOV fields, see *Invalid Output Action Fields*.

See Alarm Specification for a complete explanation of record alarms and of the standard fields. *Alarm Fields* lists other fields related to alarms that are common to all record types.

Field	Summary	Type	DCT	Default	Read	Write	CA PP
HIHI	Hihi Alarm Limit	LONG	Yes		Yes	Yes	Yes
HIGH	High Alarm Limit	LONG	Yes		Yes	Yes	Yes
LOW	Low Alarm Limit	LONG	Yes		Yes	Yes	Yes
LOLO	Lolo Alarm Limit	LONG	Yes		Yes	Yes	Yes
HHSV	Hihi Severity	MENU menuAlarmSevr	Yes		Yes	Yes	Yes
HSV	High Severity	MENU menuAlarmSevr	Yes		Yes	Yes	Yes
LSV	Low Severity	MENU menuAlarmSevr	Yes		Yes	Yes	Yes
LLSV	Lolo Severity	MENU menuAlarmSevr	Yes		Yes	Yes	Yes
HYST	Alarm Deadband	LONG	Yes		Yes	Yes	No
IVOA	INVALID output action	MENU menuIvoa	Yes		Yes	Yes	No
IVOV	INVALID output value	LONG	Yes		Yes	Yes	No

Monitor Parameters

These parameters are used to determine when to send monitors placed on the value field. The monitors are sent when the value field exceeds the last monitored field by the appropriate delta. If these fields have a value of zero, everytime the value changes, a monitor will be triggered; if they have a value of -1, everytime the record is scanned, monitors are triggered. The ADEL field is the delta for archive monitors, and the MDEL field is the delta for all other types of monitors. See *"Monitor Specification"* for a complete explanation of monitors.

Field	Summary	Type	DCT	Default	Read	Write	CA PP
ADEL	Archive Deadband	LONG	Yes		Yes	Yes	No
MDEL	Monitor Deadband	LONG	Yes		Yes	Yes	No

Run-time Parameters

The LALM, MLST, and ALST fields are used to implement the hysteresis factors for monitor callbacks. Only if the difference between these fields and the corresponding value field is greater than the appropriate delta (MDEL, ADEL, HYST)–only then are monitors triggered. For instance, only if the difference between VAL and MLST is greater than MDEL are the monitors triggered for VAL.

Field	Summary	Type	DCT	Default	Read	Write	CA PP
LALM	Last Value Alarmed	LONG	No		Yes	No	No
ALST	Last Value Archived	LONG	No		Yes	No	No
MLST	Last Val Monitored	LONG	No		Yes	No	No

Simulation Mode Parameters

The following fields are used to operate the record in simulation mode.

If SIMM (fetched through SIML) is YES, the record is put in SIMS severity and the value is written through SIOL. SSCN sets a different SCAN mechanism to use in simulation mode. SDLY sets a delay (in sec) that is used for asynchronous simulation processing.

See *Output Simulation Fields* for more information on simulation mode and its fields.

Field	Summary	Type	DCT	Default	Read	Write	CA PP
SIML	Sim Mode Location	INLINK	Yes		Yes	Yes	No
SIMM	Simulation Mode	MENU menuYesNo	No		Yes	Yes	No
SIOL	Sim Output Specifctn	OUTLINK	Yes		Yes	Yes	No
SIMS	Sim mode Alarm Svrty	MENU menuAlarmSevr	Yes		Yes	Yes	No
SDLY	Sim. Mode Async Delay	DOUBLE	Yes	-1.0	Yes	Yes	No
SSCN	Sim. Mode Scan	MENU menuScan	Yes	65535	Yes	Yes	No

Record Support

Record Support Routines

init_record

This routine initializes SIMM if SIML is a constant or creates a channel access link if SIML is PV_LINK. If SIOL is PV_LINK a channel access link is created.

This routine next checks to see that device support is available. The routine next checks to see if the device support write routine is defined.

If either device support or the device support write routine does not exist, an error message is issued and processing is terminated.

If DOL is a constant, then VAL is initialized to its value and UDF is set to FALSE. If DOL type is a PV_LINK then dbCaAddInlink is called to create a channel access link.

If device support includes `init_record()`, it is called.

process

See next section.

get_units

Retrieves EGU.

get_graphic_double

Sets the upper display and lower display limits for a field. If the field is VAL, HIHI, HIGH, LOW, or LOLO, the limits are set to HOPR and LOPR, else if the field has upper and lower limits defined they will be used, else the upper and lower maximum values for the field type will be used.

get_control_double

Sets the upper control and the lower control limits for a field. If the field is VAL, HIHI, HIGH, LOW, or LOLO, the limits are set to HOPR and LOPR, else if the field has upper and lower limits defined they will be used, else the upper and lower maximum values for the field type will be used.

get_alarm_double

Sets the following values:

```
upper_alarm_limit = HIHI
upper_warning_limit = HIGH
lower_warning_limit = LOW
lower_alarm_limit = LOLO
```

Record Processing

Routine process implements the following algorithm:

1. Check to see that the appropriate device support module exists. If it doesn't, an error message is issued and processing is terminated with the PACT field still set to TRUE. This ensures that processes will no longer be called for this record. Thus error storms will not occur.
2. If PACT is FALSE and OMSL is CLOSED_LOOP recGblGetLinkValue is called to read the current value of VAL. See *"Output Records"* for more information. If the return status of recGblGetLinkValue is zero then UDF is set to FALSE.
3. Check alarms. This routine checks to see if the new VAL causes the alarm status and severity to change. If so, NSEV, NSTA and LALM are set. It also honors the alarm hysteresis factor (HYST). Thus the value must change by more than HYST before the alarm status and severity is lowered.
4. Check severity and write the new value. See *Invalid Output Action Fields* for information on how INVALID alarms affect output records.
5. If PACT has been changed to TRUE, the device support write output routine has started but has not completed writing the new value. In this case, the processing routine merely returns, leaving PACT TRUE.
6. Check to see if monitors should be invoked:
   - Alarm monitors are invoked if the alarm status or severity has changed.
   - Archive and value change monitors are invoked if ADEL and MDEL conditions are met.
   - NSEV and NSTA are reset to 0.
7. Scan forward link if necessary, set PACT FALSE, and return.

Device Support

Fields Of Interest To Device Support

Each long output record must have an associated set of device support routines. The primary responsibility of the device support routines is to output a new value whenever write_longout is called. The device support routines are primarily interested in the following fields:

Field	Summary	Type	DCT	Default	Read	Write	CA PP
PACT	Record active	UCHAR	No		Yes	No	No
DPVT	Device Private	NOACCESS	No		No	No	No
NSEV	New Alarm Severity	MENU menuAlarmSevr	No		Yes	No	No
NSTA	New Alarm Status	MENU menuAlarmStat	No		Yes	No	No
OUT	Output Specification	OUTLINK	Yes		Yes	Yes	No

Device Support Routines

Device support consists of the following routines:

long report(int level)

This optional routine is called by the IOC command `dbior` and is passed the report level that was requested by the user. It should print a report on the state of the device support to stdout. The `level` parameter may be used to output increasingly more detailed information at higher levels, or to select different types of information with different levels. Level zero should print no more than a small summary.

long init(int after)

This optional routine is called twice at IOC initialization time. The first call happens before any of the `init_record()` calls are made, with the integer parameter `after` set to 0. The second call happens after all of the `init_record()` calls have been made, with `after` set to 1.

init_record

```
init_record(precord)
```

This routine is optional. If provided, it is called by the record support `init_record()` routine.

get_ioint_info

```
get_ioint_info(int cmd,struct dbCommon *precord,IOSCANPVT *ppvt)
```

This routine is called by the ioEventScan system each time the record is added or deleted from an I/O event scan list. `cmd` has the value (0,1) if the record is being (added to, deleted from) an I/O event list. It must be provided for any device type that can use the ioEvent scanner.

write_longout

```
write_longout(precord)
```

This routine must output a new value. It returns the following values:

- 0: Success.
- Other: Error.

Device Support For Soft Records

The `Soft Channel` module writes the current value of VAL.

If the OUT link type is PV_LINK, then dbCaAddInlink is called by `init_record()`.

write_longout calls recGblPutLinkValue to write the current value of VAL.

See *"Soft Output"* for a further explanation.

1.5.19 Long String Input Record (lsi)

The long string input record is used to retrieve an arbitrary ASCII string with a maximum length of 65535 characters.

Parameter Fields

The record-specific fields are described below, grouped by functionality.

Scan Parameters

The long string input record has the standard fields for specifying under what circumstances it will be processed. These fields are listed in *Scan Fields*.

Input Specification

The INP field determines where the long string input record obtains its string from. It can be a database or channel access link, or a constant. If constant, the VAL field is initialized with the constant and can be changed via dbPuts. Otherwise, the string is read from the specified location each time the record is processed and placed in the VAL field. The maximum number of characters in VAL is given by SIZV, and cannot be larger than 65535. In addition, the appropriate device support module must be entered into the DTYP field.

Field	Summary	Type	DCT	Default	Read	Write	CA PP
VAL	Current Value	STRING or CHAR[SIZV]	No		Yes	Yes	Yes
OVAL	Old Value	STRING or [SIZV]	No		Yes	No	No
SIZV	Size of buffers	USHORT	Yes	41	Yes	No	No
INP	Input Specification	INLINK	Yes		Yes	Yes	No
DTYP	Device Type	DEVICE	Yes		Yes	Yes	No

Monitor Parameters

These parameters are used to specify when the monitor post should be sent by the `monitor()` routine. There are two possible choices:

APST is used for archiver monitors and MPST for all other type of monitors.

Field	Summary	Type	DCT	Default	Read	Write	CA PP
MPST	Post Value Monitors	MENU menuPost	Yes		Yes	Yes	No
APST	Post Archive Monitors	MENU menuPost	Yes		Yes	Yes	No

Operator Display Parameters

See *Fields Common to All Record Types* for more on the record name (NAME) and description (DESC) fields.

Field	Summary	Type	DCT	Default	Read	Write	CA PP
NAME	Record Name	STRING [61]	No		Yes	No	No
DESC	Descriptor	STRING [41]	Yes		Yes	Yes	No

Alarm Parameters

The long string input record has the alarm parameters common to all record types. *Alarm Fields* lists the fields related to alarms that are common to all record types.

Run-time Parameters

The old value field (OVAL) of the long string input record is used to implement value change monitors for VAL. If VAL is not equal to OVAL, then monitors are triggered. LEN contains the length of the string in VAL, OLEN contains the length of the string in OVAL.

Field	Summary	Type	DCT	Default	Read	Write	CA PP
OVAL	Old Value	STRING or [SIZV]	No		Yes	No	No
LEN	Length of VAL	ULONG	No		Yes	No	No
OLEN	Length of OVAL	ULONG	No		Yes	No	No

Simulation Mode Parameters

The following fields are used to operate the record in simulation mode.

If SIMM (fetched through SIML) is YES, the record is put in SIMS severity and the value is fetched through SIOL. SSCN sets a different SCAN mechanism to use in simulation mode. SDLY sets a delay (in sec) that is used for asynchronous simulation processing.

See *Input Simulation Fields* for more information on simulation mode and its fields.

Field	Summary	Type	DCT	Default	Read	Write	CA PP
SIML	Simulation Mode Link	INLINK	Yes		Yes	Yes	No
SIMM	Simulation Mode	MENU menuYesNo	No		Yes	Yes	No
SIOL	Simulation Input Link	INLINK	Yes		Yes	Yes	No
SIMS	Simulation Mode Severity	MENU menuAlarmSevr	Yes		Yes	Yes	No
SDLY	Sim. Mode Async Delay	DOUBLE	Yes	-1.0	Yes	Yes	No
SSCN	Sim. Mode Scan	MENU menuScan	Yes	65535	Yes	Yes	No

Device Support Interface

The record requires device support to provide an entry table (dset) which defines the following members:

```
typedef struct {
    long number;
    long (*report)(int level);
    long (*init)(int after);
    long (*init_record)(lsiRecord *prec);
    long (*get_ioint_info)(int cmd, lsiRecord *prec, IOSCANPVT *piosl);
    long (*read_string)(lsiRecord *prec);
} lsidset;
```

The module must set `number` to at least 5, and provide a pointer to its `read_string()` routine; the other function pointers may be `NULL` if their associated functionality is not required for this support layer. Most device supports also provide an `init_record()` routine to configure the record instance and connect it to the hardware or driver support layer.

Device Support for Soft Records

A device support module for DTYP `Soft Channel` is provided for retrieving values from other records or other software components.

Device support for DTYP `getenv` is provided for retrieving strings from environment variables. `INST_IO` addressing `@<environment variable>` is used on the `INP` link field to select the desired environment variable.

1.5.20 Long String Output Record (lso)

The long string output record is used to write an arbitrary ASCII string with a maximum length of 65535 characters.

Parameter Fields

The record-specific fields are described below, grouped by functionality.

Scan Parameters

The long string output record has the standard fields for specifying under what circumstances it will be processed. These fields are listed in *Scan Fields*.

Desired Output Parameters

The long string output record must specify from where it gets its desired output string. The first field that determines where the desired output originates is the output mode select (OMSL) field, which can have two possible values: `closed_loop` or `supervisory`. If `closed_loop` is specified, the VAL field's value is fetched from the address specified in the Desired Output Link field (DOL) which can be either a database link or a channel access link. If `supervisory` is specified, DOL is ignored, the current value of VAL is written, and VAL can be changed externally via dbPuts at run-time.

The maximum number of characters in VAL is given by SIZV, and cannot be larger than 65535.

DOL can also be a constant instead of a link, in which case VAL is initialized to the constant value. Most simple string constants are likely to be interpreted as a CA link name though. To initialize a string output record it is simplest to set the VAL field directly; alternatively use a JSON constant link type in the DOL field.

Field	Summary	Type	DCT	Default	Read	Write	CA PP
VAL	Current Value	STRING or CHAR[SIZV]	No		Yes	Yes	Yes
SIZV	Size of buffers	USHORT	Yes	41	Yes	No	No
DOL	Desired Output Link	INLINK	Yes		Yes	Yes	No
OMSL	Output Mode Select	MENU menuOmsl	Yes		Yes	Yes	No

Output Specification

The output link specified in the OUT field specifies where the long string output record is to write its string. The link can be a database or channel access link. If the OUT field is a constant, no output will be written.

In addition, the appropriate device support module must be entered into the DTYP field.

Field	Summary	Type	DCT	Default	Read	Write	CA PP
OUT	Output Specification	OUTLINK	Yes		Yes	Yes	No
DTYP	Device Type	DEVICE	Yes		Yes	Yes	No

Monitor Parameters

These parameters are used to specify when the monitor post should be sent by the `monitor()` routine. There are two possible choices:

APST is used for archiver monitors and MPST for all other type of monitors.

Field	Summary	Type	DCT	Default	Read	Write	CA PP
MPST	Post Value Monitors	MENU menuPost	Yes		Yes	Yes	No
APST	Post Archive Monitors	MENU menuPost	Yes		Yes	Yes	No

Operator Display Parameters

See *Fields Common to All Record Types* for more on the record name (NAME) and description (DESC) fields.

Field	Summary	Type	DCT	Default	Read	Write	CA PP
NAME	Record Name	STRING [61]	No		Yes	No	No
DESC	Descriptor	STRING [41]	Yes		Yes	Yes	No

Alarm Parameters

The long string input record has the same alarm parameters common to all record types. *Alarm Fields* lists the fields related to alarms that are common to all record types.

The IVOA field specifies an action to take when the INVALID alarm is triggered. When `Set output to IVOV`, the value contained in the IVOV field is written to the output link during an alarm condition. See *Invalid Output Action Fields* for more information on the IVOA and IVOV fields.

Field	Summary	Type	DCT	Default	Read	Write	CA PP
IVOA	INVALID Output Action	MENU menuIvoa	Yes		Yes	Yes	No
IVOV	INVALID Output Value	STRING [40]	Yes		Yes	Yes	No

Run-time Parameters

The old value field (OVAL) of the long string input record is used to implement value change monitors for VAL. If VAL is not equal to OVAL, then monitors are triggered. LEN contains the length of the string in VAL, OLEN contains the length of the string in OVAL.

Field	Summary	Type	DCT	Default	Read	Write	CA PP
OVAL	Previous Value	STRING or [SIZV]	No		Yes	No	No
LEN	Length of VAL	ULONG	No		Yes	No	No
OLEN	Length of OVAL	ULONG	No		Yes	No	No

Simulation Mode Parameters

The following fields are used to operate the record in simulation mode.

If SIMM (fetched through SIML) is YES, the record is put in SIMS severity and the value is written through SIOL. SSCN sets a different SCAN mechanism to use in simulation mode. SDLY sets a delay (in sec) that is used for asynchronous simulation processing.

See *Output Simulation Fields* for more information on simulation mode and its fields.

Field	Summary	Type	DCT	Default	Read	Write	CA PP
SIML	Simulation Mode link	INLINK	Yes		Yes	Yes	No
SIMM	Simulation Mode	MENU menuYesNo	No		Yes	Yes	No
SIOL	Simulation Output Link	OUTLINK	Yes		Yes	Yes	No
SIMS	Simulation Mode Severity	MENU menuAlarmSevr	Yes		Yes	Yes	No
SDLY	Sim. Mode Async Delay	DOUBLE	Yes	-1.0	Yes	Yes	No
SSCN	Sim. Mode Scan	MENU menuScan	Yes	65535	Yes	Yes	No

Device Support Interface

The record defines a device support entry table type `lsodset` in the generated lsoRecord.h file as follows:

```
typedef struct lsodset {
    dset common;
    long (*write_string)(struct lsoRecord *prec);
} lsodset;
#define HAS_lsodset
```

The support module must set `common.number` to at least 5, and provide a pointer to its `write_string()` routine; the other function pointers may be `NULL` if their associated functionality is not required for this support layer. Most device supports also provide a `common.init_record()` routine to configure the record instance and connect it to the hardware or driver support layer.

Device Support for Soft Records

Device support for DTYP `Soft Channel` is provided for writing values to other records or other software components.

Device support for DTYP `stdio` is provided for writing values to the stdout, stderr, or errlog streams. `INST_IO` addressing `@stdout`, `@stderr` or `@errlog` is used on the OUT link field to select the desired stream.

1.5.21 Multi-Bit Binary Input Direct Record (mbbiDirect)

The mbbiDirect record retrieves a 32-bit hardware value and converts it to an array of 32 unsigned characters, each representing a bit of the word. These fields (B0-B9, BA-BF, B10-B19, B1A-B1F) are set to 1 if the corresponding bit is set, and 0 if not.

This record's operation is similar to that of the *multi-bit binary input record*, and it has many fields in common with it. This record also has two available soft device support modules: `Soft Channel` and `Raw Soft Channel`.

Parameter Fields

The record-specific fields are described below, grouped by functionality.

Scan Parameters

The mbbiDirect record has the standard fields for specifying under what circumstances the record will be processed. These fields are listed in *Scan Fields*.

Read and Convert Parameters

The device support routines obtain the record's input from the device or link specified in the INP field. For records that obtain their input from devices, the INP field must contain the address of the I/O card, and the DTYP field must specify the proper device support module. Be aware that the address format differs according to the I/O bus used.

Two soft device support modules can be specified in DTYP `Soft Channel` and `Raw Soft Channel`.

`Raw Soft Channel` reads the value into RVAL, upon which the normal conversion process is undergone. `Soft Channel` reads any unsigned integer directly into VAL. For a soft mbbiDirect record, the INP field can be a constant, a database, or a channel access link. If INP is a constant, then the VAL is initialized to the INP value but can be changed at run-time via dbPutField or dbPutLink.

For records that don't use `Soft Channel` device support, RVAL is used to determine VAL as follows:

- 1. RVAL is assigned to a temporary variable *rval* = RVAL
- 2. *rval* is shifted right SHFT number of bits.
- 3. VAL is set equal to *rval*.

Each of the fields, B0-BF and B10-B1F, represents one bit of the word.

Field	Summary	Type	DCT	Default	Read	Write	CA PP
VAL	Current Value	LONG	Yes		Yes	Yes	Yes
INP	Input Specification	INLINK	Yes		Yes	Yes	No
RVAL	Raw Value	ULONG	No		Yes	Yes	Yes
SHFT	Shift	USHORT	Yes		Yes	Yes	No

continues on next page

Table 1 – continued from previous page

Field	Summary	Type	DCT	Default	Read	Write	CA PP
B0	Bit 0	UCHAR	No		Yes	Yes	Yes
B1	Bit 1	UCHAR	No		Yes	Yes	Yes
B2	Bit 2	UCHAR	No		Yes	Yes	Yes
B3	Bit 3	UCHAR	No		Yes	Yes	Yes
B4	Bit 4	UCHAR	No		Yes	Yes	Yes
B5	Bit 5	UCHAR	No		Yes	Yes	Yes
B6	Bit 6	UCHAR	No		Yes	Yes	Yes
B7	Bit 7	UCHAR	No		Yes	Yes	Yes
B8	Bit 8	UCHAR	No		Yes	Yes	Yes
B9	Bit 9	UCHAR	No		Yes	Yes	Yes
BA	Bit 10	UCHAR	No		Yes	Yes	Yes
BB	Bit 11	UCHAR	No		Yes	Yes	Yes
BC	Bit 12	UCHAR	No		Yes	Yes	Yes
BD	Bit 13	UCHAR	No		Yes	Yes	Yes
BE	Bit 14	UCHAR	No		Yes	Yes	Yes
BF	Bit 15	UCHAR	No		Yes	Yes	Yes
B10	Bit 16	UCHAR	No		Yes	Yes	Yes
B11	Bit 17	UCHAR	No		Yes	Yes	Yes
B12	Bit 18	UCHAR	No		Yes	Yes	Yes
B13	Bit 19	UCHAR	No		Yes	Yes	Yes
B14	Bit 20	UCHAR	No		Yes	Yes	Yes
B15	Bit 21	UCHAR	No		Yes	Yes	Yes
B16	Bit 22	UCHAR	No		Yes	Yes	Yes
B17	Bit 23	UCHAR	No		Yes	Yes	Yes
B18	Bit 24	UCHAR	No		Yes	Yes	Yes
B19	Bit 25	UCHAR	No		Yes	Yes	Yes
B1A	Bit 26	UCHAR	No		Yes	Yes	Yes
B1B	Bit 27	UCHAR	No		Yes	Yes	Yes
B1C	Bit 28	UCHAR	No		Yes	Yes	Yes
B1D	Bit 29	UCHAR	No		Yes	Yes	Yes
B1E	Bit 30	UCHAR	No		Yes	Yes	Yes
B1F	Bit 31	UCHAR	No		Yes	Yes	Yes

Operator Display Parameters

These parameters are used to present meaningful data to the operator.

See *Fields Common to All Record Types* for more on the record name (NAME) and description (DESC) fields.

Field	Summary	Type	DCT	Default	Read	Write	CA PP
NAME	Record Name	STRING [61]	No		Yes	No	No
DESC	Descriptor	STRING [41]	Yes		Yes	Yes	No

Run-time Parameters

These parameters are used by the run-time code for processing the mbbi direct record. They are not configurable prior to run-time.

MASK is used by device support routine to read hardware register. Record support sets low order NOBT bits in MASK. Device support can shift this value.

MLST holds the value when the last monitor for value change was triggered.

Field	Summary	Type	DCT	Default	Read	Write	CA PP
NOBT	Number of Bits	SHORT	Yes		Yes	No	No
ORAW	Prev Raw Value	ULONG	No		Yes	No	No
MASK	Hardware Mask	ULONG	No		Yes	No	No
MLST	Last Value Monitored	LONG	No		Yes	No	No

Simulation Mode Parameters

The following fields are used to operate the record in simulation mode.

If SIMM (fetched through SIML) is YES or RAW, the record is put in SIMS severity and the value is fetched through SIOL (buffered in SVAL). If SIMM is YES, SVAL is written to VAL without conversion, if SIMM is RAW, SVAL is trancated to RVAL and converted. SSCN sets a different SCAN mechanism to use in simulation mode. SDLY sets a delay (in sec) that is used for asynchronous simulation processing.

See *Input Simulation Fields* for more information on simulation mode and its fields.

Field	Summary	Type	DCT	Default	Read	Write	CA PP
SIML	Simulation Mode Link	INLINK	Yes		Yes	Yes	No
SIMM	Simulation Mode	MENU menuSimm	No		Yes	Yes	No
SIOL	Simulation Input Link	INLINK	Yes		Yes	Yes	No
SVAL	Simulation Value	LONG	No		Yes	Yes	No
SIMS	Simulation Mode Severity	MENU menuAlarmSevr	Yes		Yes	Yes	No
SDLY	Sim. Mode Async Delay	DOUBLE	Yes	-1.0	Yes	Yes	No
SSCN	Sim. Mode Scan	MENU menuScan	Yes	65535	Yes	Yes	No

Alarm Parameters

The possible alarm conditions for multi-bit binary input direct records are the SCAN and READ alarms. These alarms are not configurable by the user since they are always of MAJOR severity. No fields exist for the mbbi direct record to have state alarms.

Alarm Fields lists the fields related to alarms that are common to all record types.

Record Support

Record Support Routines

init_record

This routine initializes SIMM with the value of SIML if SIML type is CONSTANT link or creates a channel access link if SIML type is PV_LINK. SVAL is likewise initialized if SIOL is CONSTANT or PV_LINK.

This routine next checks to see that device support is available and a device support read routine is defined. If either does not exist, an error message is issued and processing is terminated.

Clears MASK and then sets the NOBT low order bits.

If device support includes `init_record()`, it is called.

refresh_bits is then called to refresh all the bit fields based on a hardware value.

process

See next section.

Record Processing

Routine process implements the following algorithm:

1. Check to see that the appropriate device support module exists. If it doesn't, an error message is issued and processing is terminated with the PACT field still set to TRUE. This ensures that processes will no longer be called for this record. Thus error storms will not occur.
2. readValue is called. See *"Output Records"* for information.
3. If PACT has been changed to TRUE, the device support read routine has started but has not completed reading a new input value. In this case, the processing routine merely returns, leaving PACT TRUE.
4. Convert.
    - status = read_mbbiDirect
    - PACT = TRUE
    - `recGblGetTimeStamp()` is called.
    - If status is 0, then determine VAL
        - Set rval = RVAL
        - Shift rval right SHFT bits
        - Set VAL = RVAL

   - If status is 1, return 0
   - If status is 2, set status = 0

5. Check to see if monitors should be invoked.
   - Alarm monitors are invoked if the alarm status or severity has changed.
   - Archive and value change monitors are invoked if MLST is not equal to VAL.
   - Monitors for RVAL are checked whenever other monitors are invoked.
   - NSEV and NSTA are reset to 0.
6. Scan forward link if necessary, set PACT FALSE, and return.

Device Support

Fields Of Interest To Device Support

Each input record must have an associated set of device support routines.

The primary responsibility of the device support routines is to obtain a new raw input value whenever read_mbbiDirect is called. The device support routines are primarily interested in the following fields:

Field	Summary	Type	DCT	Default	Read	Write	CA PP
PACT	Record active	UCHAR	No		Yes	No	No
DPVT	Device Private	NOACCESS	No		No	No	No
UDF	Undefined	UCHAR	Yes	1	Yes	Yes	Yes
NSEV	New Alarm Severity	MENU menuAlarmSevr	No		Yes	No	No
NSTA	New Alarm Status	MENU menuAlarmStat	No		Yes	No	No
NOBT	Number of Bits	SHORT	Yes		Yes	No	No
VAL	Current Value	LONG	Yes		Yes	Yes	Yes
INP	Input Specification	INLINK	Yes		Yes	Yes	No
RVAL	Raw Value	ULONG	No		Yes	Yes	Yes
MASK	Hardware Mask	ULONG	No		Yes	No	No
SHFT	Shift	USHORT	Yes		Yes	Yes	No

Device Support Routines

Device support consists of the following routines:

long report(int level)

This optional routine is called by the IOC command `dbior` and is passed the report level that was requested by the user. It should print a report on the state of the device support to stdout. The `level` parameter may be used to output increasingly more detailed information at higher levels, or to select different types of information with different levels. Level zero should print no more than a small summary.

long init(int after)

This optional routine is called twice at IOC initialization time. The first call happens before any of the `init_record()` calls are made, with the integer parameter `after` set to 0. The second call happens after all of the `init_record()` calls have been made, with `after` set to 1.

init_record

```
init_record(precord)
```

This routine is optional. If provided, it is called by the record support `init_record()` routine. If it uses MASK, it should shift it as necessary and also give SHFT a value.

get_ioint_info

```
get_ioint_info(int cmd,struct dbCommon *precord,IOSCANPVT *ppvt)
```

This routine is called by the ioEventScan system each time the record is added or deleted from an I/O event scan list. `cmd` has the value (0,1) if the record is being (added to, deleted from) an I/O event list. It must be provided for any device type that can use the ioEvent scanner.

read_mbbiDirect

```
read_mbbiDirect(precord)
```

This routine must provide a new input value. It returns the following values:

- 0: Success. A new raw value is placed in RVAL. The record support module determines VAL from RVAL and SHFT.
- 2: Success, but don't modify VAL.
- Other: Error.

Device Support For Soft Records

Two soft device support modules, `Soft Channel` and `Raw Soft Channel`, are provided for multi-bit binary input direct records not related to actual hardware devices. The INP link type must be either CONSTANT, DB_LINK, or CA_LINK.

Soft Channel

For this module, read_mbbiDirect always returns a value of 2, which means that no conversion is performed.

If the INP link type is constant, then the constant value is stored into VAL by `init_record()`, and UDF is set to FALSE. VAL can be changed via dbPut requests. If the INP link type is PV_LINK, then dbCaAddInlink is called by `init_record()`.

read_mbbiDirect calls recGblGetLinkValue to read the current value of VAL.

See *"Input Records"* for a further explanation.

If the return status of recGblGetLinkValue is zero, then read_mbbi sets UDF to FALSE. The status of recGblGetLinkValue is returned.

Raw Soft Channel

This module is like the previous except that values are read into RVAL, VAL is computed from RVAL, and read_mbbiDirect returns a value of 0. Thus the record processing routine will determine VAL in the normal way.

1.5.22 Multi-Bit Binary Input Record (mbbi)

The normal use for the multi-bit binary input record is to read contiguous, multiple bit inputs from hardware. The binary value represents a state from a range of up to 16 states. The multi-bit input record interfaces with devices that use more than one bit.

Most device support modules obtain values from hardware and place the value in RVAL. For these device support modules record processing uses RVAL to determine the current state (VAL is given a value between 0 and 15). Device support modules may optionally read a value directly into VAL.

Soft device modules are provided to obtain input via database or channel access links or via dbPutField or dbPutLink requests. Two soft device support modules are provided: `Soft Channel` allows VAL to be an arbitrary unsigned short integer. `Raw Soft Channel` reads the value into RVAL just like normal device support modules.

Parameter Fields

The record-specific fields are described below, grouped by functionality.

Scan Parameters

The multi-bit binary input record has the standard fields for specifying under what circumstances it will be processed. These fields are listed in *Scan Fields*.

Read and Convert Parameters

The device support routines obtain the record's input from the device or link specified in the INP field. For records that obtain their input from devices, the INP field must contain the address of the I/O card, and the DTYP field must specify the proper device support module. Be aware that the address format differs according to the I/O bus used.

Two soft device support modules can be specified in DTYP `Soft Channel` and `Raw Soft Channel`.

`Raw Soft Channel` reads the value into RVAL, upon which the normal conversion process is undergone. `Soft Channel` reads any unsigned integer directly into VAL. For a soft mbbi record, the INP field can be a constant, a database, or a channel access link. If INP is a constant, then the VAL is initialized to the constant value but can be changed at run-time via dbPutField or dbPutLink.

MASK is used by the raw soft channel read routine, and by typical device support read routines, to select only the desired bits when reading the hardware register. It is initialized to ((1 << NOBT) - 1) by record initialization. The user can configure the NOBT field, but the device support routines may set it, in which case the value given to it by the user is simply overridden. The device support routines may also override MASK or shift it left by SHFT bits. If MASK is non-zero, only the bits specified by MASK will appear in RVAL.

Unless the device support routine specifies no conversion, RVAL is used to determine VAL as follows:

1. RVAL is assigned to a temporary variable – rval = RVAL
2. rval is shifted right SHFT number of bits.
3. A match is sought between rval and one of the state value fields, ZRVL-FFVL.

Each of the fields, ZRVL-FFVL, represents one of the possible sixteen states (not all sixteen have to be used).

Alternatively, the input value can be read as a string, in which case, a match is sought with one of the strings specified in the ZRST-FFST fields. Then RVAL is set equal to the corresponding value for that string, and the conversion process occurs.

Field	Summary	Type	DCT	Default	Read	Write	CA PP
VAL	Current Value	ENUM	Yes		Yes	Yes	Yes
INP	Input Specification	INLINK	Yes		Yes	Yes	No
MASK	Hardware Mask	ULONG	No		Yes	No	No
NOBT	Number of Bits	USHORT	Yes		Yes	No	No
RVAL	Raw Value	ULONG	No		Yes	Yes	Yes
SHFT	Shift	USHORT	Yes		Yes	Yes	No
ZRVL	Zero Value	ULONG	Yes		Yes	Yes	Yes
ONVL	One Value	ULONG	Yes		Yes	Yes	Yes
TWVL	Two Value	ULONG	Yes		Yes	Yes	Yes
THVL	Three Value	ULONG	Yes		Yes	Yes	Yes
FRVL	Four Value	ULONG	Yes		Yes	Yes	Yes
FVVL	Five Value	ULONG	Yes		Yes	Yes	Yes
SXVL	Six Value	ULONG	Yes		Yes	Yes	Yes
SVVL	Seven Value	ULONG	Yes		Yes	Yes	Yes
EIVL	Eight Value	ULONG	Yes		Yes	Yes	Yes
NIVL	Nine Value	ULONG	Yes		Yes	Yes	Yes
TEVL	Ten Value	ULONG	Yes		Yes	Yes	Yes
ELVL	Eleven Value	ULONG	Yes		Yes	Yes	Yes
TVVL	Twelve Value	ULONG	Yes		Yes	Yes	Yes
TTVL	Thirteen Value	ULONG	Yes		Yes	Yes	Yes
FTVL	Fourteen Value	ULONG	Yes		Yes	Yes	Yes
FFVL	Fifteen Value	ULONG	Yes		Yes	Yes	Yes

Operator Display Parameters

These parameters are used to present meaningful data to the operator. They display the value and other parameters of the mbbi record either textually or graphically. The ZRST-FFST fields contain strings describing one of the possible states of the record. The `get_enum_str` and `get_enum_strs` record routines retrieve these strings for the operator. `Get_enum_str` gets the string corresponding to the value set in VAL, and `get_enum_strs` retrieves all the strings.

See *Fields Common to All Record Types* for more on the record name (NAME) and description (DESC) fields.

Field	Summary	Type	DCT	Default	Read	Write	CA PP
NAME	Record Name	STRING [61]	No		Yes	No	No
DESC	Descriptor	STRING [41]	Yes		Yes	Yes	No
ZRST	Zero String	STRING [26]	Yes		Yes	Yes	Yes
ONST	One String	STRING [26]	Yes		Yes	Yes	Yes
TWST	Two String	STRING [26]	Yes		Yes	Yes	Yes
THST	Three String	STRING [26]	Yes		Yes	Yes	Yes
FRST	Four String	STRING [26]	Yes		Yes	Yes	Yes
FVST	Five String	STRING [26]	Yes		Yes	Yes	Yes
SXST	Six String	STRING [26]	Yes		Yes	Yes	Yes
SVST	Seven String	STRING [26]	Yes		Yes	Yes	Yes
EIST	Eight String	STRING [26]	Yes		Yes	Yes	Yes
NIST	Nine String	STRING [26]	Yes		Yes	Yes	Yes
TEST	Ten String	STRING [26]	Yes		Yes	Yes	Yes
ELST	Eleven String	STRING [26]	Yes		Yes	Yes	Yes
TVST	Twelve String	STRING [26]	Yes		Yes	Yes	Yes
TTST	Thirteen String	STRING [26]	Yes		Yes	Yes	Yes
FTST	Fourteen String	STRING [26]	Yes		Yes	Yes	Yes
FFST	Fifteen String	STRING [26]	Yes		Yes	Yes	Yes

Alarm Parameters

The possible alarm conditions for multi-bit binary inputs are the SCAN, READ, and state alarms. The state alarms are configured in the below severity fields. These fields have the usual possible values for severity fields: NO_ALARM, MINOR, and MAJOR.

The unknown state severity (UNSV) field, if set to MINOR or MAJOR, triggers an alarm when the record support routine cannot find a matching value in the state value fields for `rval`.

The change of state severity (COSV) field triggers an alarm when any change of state occurs, if set to MAJOR or MINOR.

The other fields, when set to MAJOR or MINOR, trigger an alarm when VAL equals the corresponding state.

See Alarm Specification for a complete explanation of record alarms and of the standard fields. *Alarm Fields* lists other fields related to alarms that are common to all record types.

Field	Summary	Type	DCT	Default	Read	Write	CA PP
UNSV	Unknown State Severity	MENU menuAlarmSevr	Yes		Yes	Yes	Yes
COSV	Change of State Svr	MENU menuAlarmSevr	Yes		Yes	Yes	Yes
ZRSV	State Zero Severity	MENU menuAlarmSevr	Yes		Yes	Yes	Yes
ONSV	State One Severity	MENU menuAlarmSevr	Yes		Yes	Yes	Yes
TWSV	State Two Severity	MENU menuAlarmSevr	Yes		Yes	Yes	Yes
THSV	State Three Severity	MENU menuAlarmSevr	Yes		Yes	Yes	Yes
FRSV	State Four Severity	MENU menuAlarmSevr	Yes		Yes	Yes	Yes
FVSV	State Five Severity	MENU menuAlarmSevr	Yes		Yes	Yes	Yes
SXSV	State Six Severity	MENU menuAlarmSevr	Yes		Yes	Yes	Yes
SVSV	State Seven Severity	MENU menuAlarmSevr	Yes		Yes	Yes	Yes
EISV	State Eight Severity	MENU menuAlarmSevr	Yes		Yes	Yes	Yes
NISV	State Nine Severity	MENU menuAlarmSevr	Yes		Yes	Yes	Yes
TESV	State Ten Severity	MENU menuAlarmSevr	Yes		Yes	Yes	Yes
ELSV	State Eleven Severity	MENU menuAlarmSevr	Yes		Yes	Yes	Yes
TVSV	State Twelve Severity	MENU menuAlarmSevr	Yes		Yes	Yes	Yes
TTSV	State Thirteen Sevr	MENU menuAlarmSevr	Yes		Yes	Yes	Yes
FTSV	State Fourteen Sevr	MENU menuAlarmSevr	Yes		Yes	Yes	Yes
FFSV	State Fifteen Severity	MENU menuAlarmSevr	Yes		Yes	Yes	Yes

Run-time Parameters

These parameters are used by the run-time code for processing the multi-bit binary input.

ORAW is used by record processing to hold the prior RVAL for use in determining when to post a monitor event for the RVAL field.

The LALM field implements the change of state alarm severity by holding the value of VAL when the previous change of state alarm was issued.

MLST holds the value when the last monitor for value change was triggered.

SDEF is used by record support to save time if no states are defined.

Field	Summary	Type	DCT	Default	Read	Write	CA PP
ORAW	Prev Raw Value	ULONG	No		Yes	No	No
LALM	Last Value Alarmed	USHORT	No		Yes	No	No
MLST	Last Value Monitored	USHORT	No		Yes	No	No
SDEF	States Defined	SHORT	No		Yes	No	No

Simulation Mode Parameters

The following fields are used to operate the record in simulation mode.

If SIMM (fetched through SIML) is YES or RAW, the record is put in SIMS severity and the value is fetched through SIOL (buffered in SVAL). If SIMM is YES, SVAL is written to VAL without conversion, if SIMM is RAW, SVAL is trancated to RVAL and converted. SSCN sets a different SCAN mechanism to use in simulation mode. SDLY sets a delay (in sec) that is used for asynchronous simulation processing.

See *Input Simulation Fields* for more information on simulation mode and its fields.

Field	Summary	Type	DCT	Default	Read	Write	CA PP
SIML	Simulation Mode Link	INLINK	Yes		Yes	Yes	No
SIMM	Simulation Mode	MENU menuSimm	No		Yes	Yes	No
SIOL	Simulation Input Link	INLINK	Yes		Yes	Yes	No
SVAL	Simulation Value	ULONG	No		Yes	Yes	No
SIMS	Simulation Mode Severity	MENU menuAlarmSevr	Yes		Yes	Yes	No
SDLY	Sim. Mode Async Delay	DOUBLE	Yes	-1.0	Yes	Yes	No
SSCN	Sim. Mode Scan	MENU menuScan	Yes	65535	Yes	Yes	No

Record Support

Record Support Routines

init_record

This routine initializes SIMM with the value of SIML if SIML type is CONSTANT link or creates a channel access link if SIML type is PV_LINK. SVAL is likewise initialized if SIOL is CONSTANT or PV_LINK.

This routine next checks to see that device support is available and a device support read routine is defined. If either does not exist, an error message is issued and processing is terminated.

Clears MASK and then sets the NOBT low order bits.

If device support includes `init_record()`, it is called.

init_common is then called to determine if any states are defined. If states are defined, SDEF is set to TRUE.

process

See next section.

special

Calls init_common to compute SDEF when any of the fields ZRVL, … FFVL change value.

get_enum_str

Retrieves ASCII string corresponding to VAL.

get_enum_strs

Retrieves ASCII strings for ZRST,…FFST.

put_enum_str

Checks if string matches ZRST,…FFST and if it does, sets VAL.

Record Processing

Routine process implements the following algorithm:

1. Check to see that the appropriate device support module exists. If it doesn't, an error message is issued and processing is terminated with the PACT field still set to TRUE. This ensures that processes will no longer be called for this record. Thus error storms will not occur.
2. readValue is called. See *"Input Records"* for more information.
3. If PACT has been changed to TRUE, the device support read routine has started but has not completed reading a new input value. In this case, the processing routine merely returns, leaving PACT TRUE.

4. Convert:
    - status=read_mbbi
    - PACT = TRUE
    - `recGblGetTimeStamp()` is called.
    - If status is 0, then determine VAL
        - Set rval = RVAL
        - Shift rval right SHFT bits
    - If at least one state value is defined
        - Set UDF to TRUE
    - If RVAL is ZRVL,…,FFVL then set
        - VAL equals index of state
        - UDF set to FALSE
    - Else set VAL = undefined
        - Else set VAL = RVAL
    - Set UDF to FALSE
        - If status is 1, return 0
        - If status is 2, set status = 0
5. Check alarms. This routine checks to see if the new VAL causes the alarm status and severity to change. If so, NSEV, NSTA and LALM are set.
6. Check to see if monitors should be invoked.
    - Alarm monitors are invoked if the alarm status or severity has changed.
    - Archive and value change monitors are invoked if MLST is not equal to VAL.
    - Monitors for RVAL are checked whenever other monitors are invoked.
    - NSEV and NSTA are reset to 0.
7. Scan forward link if necessary, set PACT FALSE, and return.

Device Support

Fields Of Interest To Device Support

Each input record must have an associated set of device support routines.

The primary responsibility of the device support routines is to obtain a new raw input value whenever read_mbbi is called. The device support routines are primarily interested in the following fields:

Field	Summary	Type	DCT	Default	Read	Write	CA PP
PACT	Record active	UCHAR	No		Yes	No	No
DPVT	Device Private	NOACCESS	No		No	No	No
UDF	Undefined	UCHAR	Yes	1	Yes	Yes	Yes
NSEV	New Alarm Severity	MENU menuAlarmSevr	No		Yes	No	No
NSTA	New Alarm Status	MENU menuAlarmStat	No		Yes	No	No
NOBT	Number of Bits	USHORT	Yes		Yes	No	No
VAL	Current Value	ENUM	Yes		Yes	Yes	Yes
INP	Input Specification	INLINK	Yes		Yes	Yes	No
RVAL	Raw Value	ULONG	No		Yes	Yes	Yes
MASK	Hardware Mask	ULONG	No		Yes	No	No
SHFT	Shift	USHORT	Yes		Yes	Yes	No

Device Support Routines

Device support consists of the following routines:

long report(int level)

This optional routine is called by the IOC command `dbior` and is passed the report level that was requested by the user. It should print a report on the state of the device support to stdout. The `level` parameter may be used to output increasingly more detailed information at higher levels, or to select different types of information with different levels. Level zero should print no more than a small summary.

long init(int after)

This optional routine is called twice at IOC initialization time. The first call happens before any of the `init_record()` calls are made, with the integer parameter `after` set to 0. The second call happens after all of the `init_record()` calls have been made, with `after` set to 1.

init_record

```
init_record(precord)
```

This routine is optional. If provided, it is called by the record support `init_record()` routine. If it uses MASK, it should shift it as necessary and also give SHFT a value.

get_ioint_info

```
get_ioint_info(int cmd,struct dbCommon *precord,IOSCANPVT *ppvt)
```

This routine is called by the ioEventScan system each time the record is added or deleted from an I/O event scan list. `cmd` has the value (0,1) if the record is being (added to, deleted from) an I/O event list. It must be provided for any device type that can use the I/O Event scanner.

read_mbbi

```
read_mbbi(precord)
```

This routine must provide a new input value. It returns the following values:

- 0: Success. A new raw value is placed in RVAL. The record support module determines VAL from RVAL, SHFT, and ZEVL … FFVL.
- 2: Success, but don't modify VAL.
- Other: Error.

Device Support For Soft Records

Two soft device support modules `Soft Channel` and `Raw Soft Channel` are provided for multi-bit binary input records not related to actual hardware devices. The INP link type must be either CONSTANT, DB_LINK, or CA_LINK.

Soft Channel

read_mbbi always returns a value of 2, which means that no conversion is performed.

If the INP link type is constant, then the constant value is stored into VAL by `init_record()`, and UDF is set to FALSE. VAL can be changed via dbPut requests. If the INP link type is PV_LINK, then dbCaAddInlink is called by `init_record()`.

read_mbbi calls recGblGetLinkValue to read the current value of VAL. See *Soft Input*.

If the return status of recGblGetLinkValue is zero, then read_mbbi sets UDF to FALSE. The status of recGblGetLinkValue is returned.

Raw Soft Channel

This module is like the previous except that values are read into RVAL, VAL is computed from RVAL, and read_mbbi returns a value of 0. Thus the record processing routine will determine VAL in the normal way.

1.5.23 Multi-Bit Binary Output Direct Record (mbboDirect)

The mbboDirect record performs roughly the opposite function to that of the *mbbiDirect record*.

It can accept boolean values in its 32 bit fields (B0-B9, BA-BF, B10-B19 and B1A-B1F), and converts them to a 32-bit signed integer in VAL which is provided to the device support. A zero value in a bit field becomes a zero bit in VAL, a non-zero value in a bit field becomes a one bit in VAL, with B0 being the least signficant bit and B1F the MSB/sign bit.

Parameter Fields

The record-specific fields are described below, grouped by functionality.

Scan Parameters

The mbboDirect record has the standard fields for specifying under what circumstances it will be processed. These fields are listed in *Scan Fields*.

Field	Summary	Type	DCT	Default	Read	Write	CA PP
SCAN	Scan Mechanism	MENU menuScan	Yes		Yes	Yes	No
PHAS	Scan Phase	SHORT	Yes		Yes	Yes	No
EVNT	Event Name	STRING [40]	Yes		Yes	Yes	No
PRIO	Scheduling Priority	MENU menuPriority	Yes		Yes	Yes	No
PINI	Process at iocInit	MENU menuPini	Yes		Yes	Yes	No

Desired Output Parameters

Like all output records, the mbboDirect record must specify where its output should originate when it gets processed. The Output Mode SeLect field (OMSL) determines whether the output value should be read from another record or not. When set to `closed_loop`, a 32-bit integer value (the “desired output”) will be read from a link specified in the Desired Output Link (DOL) field and placed into the VAL field.

When OMSL is set to `supervisory`, the DOL field is ignored during processing and the contents of VAL are used. A value to be output may thus be written direcly into the VAL field from elsewhere as long as the record is in `supervisory` mode.

Field	Summary	Type	DCT	Default	Read	Write	CA PP
OMSL	Output Mode Select	MENU menuOmsl	Yes		Yes	Yes	Yes
DOL	Desired Output Link	INLINK	Yes		Yes	Yes	No
VAL	Word	LONG	Yes		Yes	Yes	Yes

Bit Fields

The fields B0 through BF and B10 through B1F provide an alternative way to set the individual bits of the VAL field when the record is in `supervisory` mode. Writing to one of these fields will then modify the corresponding bit in VAL, and writing to VAL will update these bit fields from that value.

The VAL field is signed so it can be accessed through Channel Access as an integer; if it were made unsigned (a `DBF_ULONG`) its representation through Channel Access would become a `double`, which could cause problems with some client programs.

Prior to the EPICS 7.0.6.1 release the individual bit fields were not updated while the record was in `closed_loop` mode with VAL being set from the DOL link, and writing to the bit fields in that mode could cause the record to process but the actual field values would not affect VAL at all. Changing the OMSL field from `closed_loop` to `supervisory` would set the bit fields from VAL at that time and trigger a monitor event for the bits that changed at that time. At record initialization if VAL is defined and the OMSL field is `supervisory` the bit fields would be set from VAL.

From EPICS 7.0.6.1 the bit fields get updated from VAL during record processing and monitors are triggered on them in either mode. Attempts to write to the bit fields while in `closed_loop` mode will be rejected by the `special()` routine which may trigger an error from the client that wrote to them. During initialization if the record is still undefined (UDF) after DOL has been read and the device support initialized but at least one of the B0-B1F fields is non-zero, the VAL field will be set from those fields and UDF will be cleared.

Field	Summary	Type	DCT	Default	Read	Write	CA PP
B0	Bit 0	UCHAR	Yes		Yes	Yes	Yes
B1	Bit 1	UCHAR	Yes		Yes	Yes	Yes
B2	Bit 2	UCHAR	Yes		Yes	Yes	Yes
B3	Bit 3	UCHAR	Yes		Yes	Yes	Yes
B4	Bit 4	UCHAR	Yes		Yes	Yes	Yes
B5	Bit 5	UCHAR	Yes		Yes	Yes	Yes
B6	Bit 6	UCHAR	Yes		Yes	Yes	Yes
B7	Bit 7	UCHAR	Yes		Yes	Yes	Yes
B8	Bit 8	UCHAR	Yes		Yes	Yes	Yes
B9	Bit 9	UCHAR	Yes		Yes	Yes	Yes
BA	Bit 10	UCHAR	Yes		Yes	Yes	Yes
BB	Bit 11	UCHAR	Yes		Yes	Yes	Yes
BC	Bit 12	UCHAR	Yes		Yes	Yes	Yes
BD	Bit 13	UCHAR	Yes		Yes	Yes	Yes
BE	Bit 14	UCHAR	Yes		Yes	Yes	Yes
BF	Bit 15	UCHAR	Yes		Yes	Yes	Yes
B10	Bit 16	UCHAR	Yes		Yes	Yes	Yes
B11	Bit 17	UCHAR	Yes		Yes	Yes	Yes
B12	Bit 18	UCHAR	Yes		Yes	Yes	Yes
B13	Bit 19	UCHAR	Yes		Yes	Yes	Yes

continues on next page

Table 2 – continued from previous page

Field	Summary	Type	DCT	Default	Read	Write	CA PP
B14	Bit 20	UCHAR	Yes		Yes	Yes	Yes
B15	Bit 21	UCHAR	Yes		Yes	Yes	Yes
B16	Bit 22	UCHAR	Yes		Yes	Yes	Yes
B17	Bit 23	UCHAR	Yes		Yes	Yes	Yes
B18	Bit 24	UCHAR	Yes		Yes	Yes	Yes
B19	Bit 25	UCHAR	Yes		Yes	Yes	Yes
B1A	Bit 26	UCHAR	Yes		Yes	Yes	Yes
B1B	Bit 27	UCHAR	Yes		Yes	Yes	Yes
B1C	Bit 28	UCHAR	Yes		Yes	Yes	Yes
B1D	Bit 29	UCHAR	Yes		Yes	Yes	Yes
B1E	Bit 30	UCHAR	Yes		Yes	Yes	Yes
B1F	Bit 31	UCHAR	Yes		Yes	Yes	Yes

Convert and Write Parameters

For records that are to write values to hardware devices, the OUT output link must contain the address of the I/O card, and the DTYP field must specify the proper device support module. Be aware that the address format differs according to the I/O bus used. See Address Specification for information on the format of hardware addresses.

During record processing VAL is converted into RVAL, which is the actual 32-bit word to be sent out. RVAL is set to VAL shifted left by the number of bits specified in the SHFT field (SHFT is normally set by device support). RVAL is then sent out to the location specified in the OUT field.

The fields NOBT and MASK can be used by device support to force some of the output bits written by that support to be zero. By default all 32 bits can be sent, but the NOBT field can be set to specify a smaller number of contiguous bits, or MASK can specify a non-contiguous set of bits. When setting MASK it is often necessary to set NOBT to a non-zero value as well, although in this case the actual value of NOBT may be ignored by the device support. If a device support sets the SHFT field it will also left-shift the value of MASK at the same time.

For mbboDirect records writing to a link instead of to hardware, the DTYP field must select one of the soft device support routines `Soft Channel` or `Raw Soft Channel`. The `Soft Channel` support writes the contents of the VAL field to the output link. The `Raw Soft Channel` support allows SHFT to be set in the DB file, and sends the result of ANDing the shifted MASK with the RVAL field's value.

Field	Summary	Type	DCT	Default	Read	Write	CA PP
OUT	Output Specification	OUTLINK	Yes		Yes	Yes	No
RVAL	Raw Value	ULONG	No		Yes	No	Yes
SHFT	Shift	USHORT	Yes		Yes	Yes	No
MASK	Hardware Mask	ULONG	No		Yes	No	No
NOBT	Number of Bits	SHORT	Yes		Yes	No	No

Operator Display Parameters

See *Fields Common to All Record Types* for more on the record name (NAME) and description (DESC) fields.

Field	Summary	Type	DCT	Default	Read	Write	CA PP
NAME	Record Name	STRING [61]	No		Yes	No	No
DESC	Descriptor	STRING [41]	Yes		Yes	Yes	No

Run-time Parameters

These parameters are used by the run-time code for processing the mbbo Direct record.

MASK is used by device support routine to read the hardware register. Record support sets the low order NOBT bits of MASK at initialization, and device support is allowed to shift this value.

MLST holds the value when the last monitor for value change was triggered. OBIT has a similar role for bits held in the B0-B1F fields.

Field	Summary	Type	DCT	Default	Read	Write	CA PP
NOBT	Number of Bits	SHORT	Yes		Yes	No	No
ORAW	Prev Raw Value	ULONG	No		Yes	No	No
MASK	Hardware Mask	ULONG	No		Yes	No	No
MLST	Last Value Monitored	LONG	No		Yes	No	No
OBIT	Last Bit mask Monitored	LONG	No		Yes	No	No

Simulation Mode Parameters

The following fields are used to operate the record in simulation mode.

If SIMM (fetched through SIML) is YES, the record is put in SIMS severity and the value is written through SIOL, without conversion. If SIMM is RAW, the value is converted and RVAL is written. SSCN sets a different SCAN mechanism to use in simulation mode. SDLY sets a delay (in sec) that is used for asynchronous simulation processing.

See *Output Simulation Fields* for more information on simulation mode and its fields.

Field	Summary	Type	DCT	Default	Read	Write	CA PP
SIML	Simulation Mode Link	INLINK	Yes		Yes	Yes	No
SIMM	Simulation Mode	MENU menuSimm	No		Yes	Yes	No
SIOL	Simulation Output Link	OUTLINK	Yes		Yes	Yes	No
SIMS	Simulation Mode Severity	MENU menuAlarmSevr	Yes		Yes	Yes	No
SDLY	Sim. Mode Async Delay	DOUBLE	Yes	-1.0	Yes	Yes	No
SSCN	Sim. Mode Scan	MENU menuScan	Yes	65535	Yes	Yes	No

Alarm Parameters

The possible alarm conditions for mbboDirect records are the SCAN, READ, and INVALID alarms.

The IVOA field specifies an action to take when an INVALID alarm is triggered. There are three possible actions: `Continue normally`, `Don't drive outputs`, or `Set output to IVOV`. When `Set output to IVOV` is specified and a INVALID alarm is triggered, the record will write the value in the IVOV field to the output.

See *Invalid Output Action Fields* for more information about IVOA and IVOV.

See Alarm Specification for a complete explanation of record alarms and of the standard fields. *Alarm Fields* lists other fields related to alarms that are common to all record types.

Field	Summary	Type	DCT	Default	Read	Write	CA PP
IVOA	INVALID outpt action	MENU menuIvoa	Yes		Yes	Yes	No
IVOV	INVALID output value	LONG	Yes		Yes	Yes	No

Record Support

Record Support Routines

init_record

This routine initializes SIMM if SIML is a constant or creates a channel access link if SIML is PV_LINK. If SIOL is PV_LINK a channel access link is created.

This routine next checks to see that device support is available.The routine next checks to see if the device support write routine is defined. If either device support or the device support write routine does not exist, an error message is issued and processing is terminated.

If DOL is a constant, then VAL is initialized to its value and UDF is set to FALSE.

MASK is cleared and then the NOBT low order bits are set.

If device support includes `init_record()`, it is called.

If device support returns success, VAL is then set from RVAL and UDF is set to FALSE.

Process

See next section.

Record Processing

Routine process implements the following algorithm:

1. Check to see that the appropriate device support module exists. If it doesn't, an error message is issued and processing is terminated with the PACT field still set to TRUE. This ensures that processes will no longer be called for this record. Thus error storms will not occur.
2. If PACT is FALSE
    - If DOL is DB_LINK and OMSL is CLOSED_LOOP
        - Get value from DOL
        - Set PACT to FALSE
3. Convert
    - If PACT is FALSE, compute RVAL
        - Set RVAL = VAL
        - Shift RVAL left SHFT bits
    - Status=write_mbboDirect
4. If PACT has been changed to TRUE, the device support write output routine has started but has not completed writing the new value. In this case, the processing routine merely returns, leaving PACT TRUE.
5. Check to see if monitors should be invoked.
    - Alarm monitors are invoked if the alarm status or severity has changed.
    - Archive and value change monitors are invoked if MLST is not equal to VAL.
    - Monitors for RVAL and RBV are checked whenever other monitors are invoked.
    - NSEV and NSTA are reset to 0.
6. Scan forward link if necessary, set PACT FALSE, and return.

Device Support

Fields Of Interest To Device Support

Each mbboDirect record must have an associated set of device support routines. The primary responsibility of the device support routines is to obtain a new raw mbbo value whenever write_mbboDirect is called. The device support routines are primarily interested in the following fields:

Field	Summary	Type	DCT	Default	Read	Write	CA PP
PACT	Record active	UCHAR	No		Yes	No	No
DPVT	Device Private	NOACCESS	No		No	No	No
UDF	Undefined	UCHAR	Yes	1	Yes	Yes	Yes
NSEV	New Alarm Severity	MENU menuAlarmSevr	No		Yes	No	No
NSTA	New Alarm Status	MENU menuAlarmStat	No		Yes	No	No
NOBT	Number of Bits	SHORT	Yes		Yes	No	No
OUT	Output Specification	OUTLINK	Yes		Yes	Yes	No
RVAL	Raw Value	ULONG	No		Yes	No	Yes
RBV	Readback Value	ULONG	No		Yes	No	No
MASK	Hardware Mask	ULONG	No		Yes	No	No
SHFT	Shift	USHORT	Yes		Yes	Yes	No

Device Support Routines

Device support consists of the following routines:

long report(int level)

This optional routine is called by the IOC command `dbior` and is passed the report level that was requested by the user. It should print a report on the state of the device support to stdout. The `level` parameter may be used to output increasingly more detailed information at higher levels, or to select different types of information with different levels. Level zero should print no more than a small summary.

long init(int after)

This optional routine is called twice at IOC initialization time. The first call happens before any of the `init_record()` calls are made, with the integer parameter `after` set to 0. The second call happens after all of the `init_record()` calls have been made, with `after` set to 1.

init_record

```
init_record(precord)
```

This routine is optional. If provided, it is called by the record support `init_record()` routine. If MASK is used, it should be shifted if necessary and SHFT given a value.

get_ioint_info

```
get_ioint_info(int cmd,struct dbCommon *precord,IOSCANPVT *ppvt)
```

This routine is called by the ioEventScan system each time the record is added or deleted from an I/O event scan list. `cmd` has the value (0,1) if the record is being (added to, deleted from) an I/O event list. It must be provided for any device type that can use the ioEvent scanner.

write_mbboDirect

```
write_mbboDirect(precord)
```

This routine must output a new value. It returns the following values:

- 0: Success.
- Other: Error.

Device Support For Soft Records

This `SOft Channel` module writes the current value of VAL.

If the OUT link type is PV_LINK, then dbCaAddInlink is called by `init_record()`.

write_mbboDirect calls recGblPutLinkValue to write the current value of VAL.

See *Soft Output*.

1.5.24 Multi-Bit Binary Output Record (mbbo)

The normal use for the mbbo record type is to send a binary value (representing one of up to 16 states) to a Digital Output module. It is used for any device that uses more than one contiguous bit to control it. The mbbo record can also be used to write discrete values to other records via database or channel access links.

Parameter Fields

The record-specific fields are described below, grouped by functionality.

Scan Parameters

The mbbo record has the standard fields for specifying under what circumstances it will be processed. These fields are listed in *Scan Fields*.

Desired Output Parameters

The multi-bit binary output record, like all output records, must specify where its output originates. The output mode select (OMSL) field determines whether the output originates from another record or from database access (i.e., the operator). When set to `closed_loop`, the desired output is retrieved from the link specified in the desired output (DOL) field–which can specify either a database or channel access link–and placed into the VAL field. When set to `supervisory`, the DOL field is ignored and the current value of VAL is simply written. VAL can be changed via dpPuts at run-time when OMSL is `supervisory`. The DOL field can also be a constant, in which case the VAL field is initialized to the constant value. If DOL is a constant, OMSL cannot be set to `closed_loop`.

The VAL field itself usually consists of an index that specifies one of the states. The actual output written is the value of RVAL, which is converted from VAL following the routine explained in the next section. However, records that use the `Soft Channel` device support module write the VAL field's value without any conversion.

Field	Summary	Type	DCT	Default	Read	Write	CA PP
OMSL	Output Mode Select	MENU menuOmsl	Yes		Yes	Yes	No
DOL	Desired Output Link	INLINK	Yes		Yes	Yes	No
VAL	Desired Value	ENUM	Yes		Yes	Yes	Yes

Convert and Write Parameters

The device support routines write the desired output to the location specified in the OUT field. If the record uses soft device support, OUT can contain a constant, a database link, or a channel access link; however, if OUT is a constant, no value will be written.

For records that write their values to hardware devices, the OUT output link must specify the address of the I/O card, and the DTYP field must specify the corresponding device support module. Be aware that the address format differs according to the I/O bus used.

For mbbo records that write to hardware, the value written to the output location is the value contained in RVAL, which is converted from VAL, VAL containing an index of one of the 16 states (0-15). RVAL is then set to the corresponding state value, the value in one of the fields ZRVL through FFVL. Then this value is shifted left according to the number in the SHFT field so that the value is in the correct position for the bits being used (the SHFT value is set by device support and is not configurable by the user).

The state value fields ZRVL through FFVL must be configured by the user before run-time. When the state values are not defined, the states defined (SDEF) field is set to FALSE at initialization time by the record routines. When SDEF is FALSE, then the record processing routine does not try to find a match, RVAL is set equal to VAL, the bits are shifted using the number in SHFT, and the value is written thus.

If the OUT output link specifies a database link, channel access link, or constant, then the DTYP field must specify either one of the two soft device support modules– `Soft Channel` or `Raw Soft Channel`. `Soft Channel` writes the value of VAL to the output link, without any conversion, while `Raw Soft Channel` writes the value from RVAL after it has undergone the above conversion.

Note also that when a string is retrieved as the desired output, a record support routine is provided (`put_enum_str()`) that will check to see if the string matches one of the strings in the ZRST through FFST fields. If a match is found, RVAL is set equal to the corresponding state value of that string.

Field	Summary	Type	DCT	Default	Read	Write	CA PP
OUT	Output Specification	OUTLINK	Yes		Yes	Yes	No
DTYP	Device Type	DEVICE	Yes		Yes	Yes	No
RVAL	Raw Value	ULONG	No		Yes	Yes	Yes
SHFT	Shift	USHORT	Yes		Yes	Yes	No
SDEF	States Defined	SHORT	No		Yes	No	No
ZRVL	Zero Value	ULONG	Yes		Yes	Yes	Yes
ONVL	One Value	ULONG	Yes		Yes	Yes	Yes
TWVL	Two Value	ULONG	Yes		Yes	Yes	Yes
THVL	Three Value	ULONG	Yes		Yes	Yes	Yes
FRVL	Four Value	ULONG	Yes		Yes	Yes	Yes
FVVL	Five Value	ULONG	Yes		Yes	Yes	Yes
SXVL	Six Value	ULONG	Yes		Yes	Yes	Yes
SVVL	Seven Value	ULONG	Yes		Yes	Yes	Yes
EIVL	Eight Value	ULONG	Yes		Yes	Yes	Yes
NIVL	Nine Value	ULONG	Yes		Yes	Yes	Yes
TEVL	Ten Value	ULONG	Yes		Yes	Yes	Yes
ELVL	Eleven Value	ULONG	Yes		Yes	Yes	Yes
TVVL	Twelve Value	ULONG	Yes		Yes	Yes	Yes
TTVL	Thirteen Value	ULONG	Yes		Yes	Yes	Yes
FTVL	Fourteen Value	ULONG	Yes		Yes	Yes	Yes
FFVL	Fifteen Value	ULONG	Yes		Yes	Yes	Yes

Operator Display Parameters

These parameters are used to present meaningful data to the operator. These fields are used to display the value and other parameters of the mbbo record either textually or graphically. The ZRST-FFST fields contain strings describing each of the corresponding states. The `get_enum_str()` and `get_enum_strs()` record routines retrieve these strings for the operator. `get_enum_str()` gets the string corresponding to the value in VAL, and `get_enum_strs()` retrieves all the strings.

See *Fields Common to All Record Types* for more on the record name (NAME) and description (DESC) fields.

Field	Summary	Type	DCT	Default	Read	Write	CA PP
NAME	Record Name	STRING [61]	No		Yes	No	No
DESC	Descriptor	STRING [41]	Yes		Yes	Yes	No
ZRST	Zero String	STRING [26]	Yes		Yes	Yes	Yes
ONST	One String	STRING [26]	Yes		Yes	Yes	Yes
TWST	Two String	STRING [26]	Yes		Yes	Yes	Yes
THST	Three String	STRING [26]	Yes		Yes	Yes	Yes
FRST	Four String	STRING [26]	Yes		Yes	Yes	Yes
FVST	Five String	STRING [26]	Yes		Yes	Yes	Yes
SXST	Six String	STRING [26]	Yes		Yes	Yes	Yes
SVST	Seven String	STRING [26]	Yes		Yes	Yes	Yes
EIST	Eight String	STRING [26]	Yes		Yes	Yes	Yes
NIST	Nine String	STRING [26]	Yes		Yes	Yes	Yes
TEST	Ten String	STRING [26]	Yes		Yes	Yes	Yes
ELST	Eleven String	STRING [26]	Yes		Yes	Yes	Yes
TVST	Twelve String	STRING [26]	Yes		Yes	Yes	Yes
TTST	Thirteen String	STRING [26]	Yes		Yes	Yes	Yes
FTST	Fourteen String	STRING [26]	Yes		Yes	Yes	Yes
FFST	Fifteen String	STRING [26]	Yes		Yes	Yes	Yes

Alarm Parameters

The possible alarm conditions for multi-bit binary outputs are the SCAN, READ, INVALID, and state alarms. The SCAN and READ alarms are called by the support modules and are not configurable by the user, as their severity is always MAJOR.

The IVOA field specifies an action to take from a number of possible choices when the INVALID alarm is triggered. The IVOV field contains a value to be written once the INVALID alarm has been triggered if `Set output to IVOV` has been chosen in the IVOA field. The severity of the INVALID alarm is not configurable by the user.

The state alarms are configured in the below severity fields. These fields have the usual possible values for severity fields: NO_ALARM, MINOR, and MAJOR.

The unknown state severity field (UNSV), if set to MINOR or MAJOR, triggers an alarm when the record support routine cannot find a matching value in the state value fields for VAL or when VAL is out of range.

The change of state severity field (COSV) triggers an alarm when the record's state changes, if set to MAJOR or MINOR.

The state severity (ZRSV-FFSV) fields, when set to MAJOR or MINOR, trigger an alarm when VAL equals the corresponding field.

See *Invalid Output Action Fields* for an explanation of the IVOA and IVOV fields. *Alarm Fields* lists the fields related to alarms that are common to all record types.

Field	Summary	Type	DCT	Default	Read	Write	CA PP
UNSV	Unknown State Sevr	MENU menuAlarmSevr	Yes		Yes	Yes	Yes
COSV	Change of State Sevr	MENU menuAlarmSevr	Yes		Yes	Yes	Yes
IVOA	INVALID outpt action	MENU menuIvoa	Yes		Yes	Yes	No
IVOV	INVALID output value	USHORT	Yes		Yes	Yes	No
ZRSV	State Zero Severity	MENU menuAlarmSevr	Yes		Yes	Yes	Yes
ONSV	State One Severity	MENU menuAlarmSevr	Yes		Yes	Yes	Yes
TWSV	State Two Severity	MENU menuAlarmSevr	Yes		Yes	Yes	Yes
THSV	State Three Severity	MENU menuAlarmSevr	Yes		Yes	Yes	Yes
FRSV	State Four Severity	MENU menuAlarmSevr	Yes		Yes	Yes	Yes
FVSV	State Five Severity	MENU menuAlarmSevr	Yes		Yes	Yes	Yes
SXSV	State Six Severity	MENU menuAlarmSevr	Yes		Yes	Yes	Yes
SVSV	State Seven Severity	MENU menuAlarmSevr	Yes		Yes	Yes	Yes
EISV	State Eight Severity	MENU menuAlarmSevr	Yes		Yes	Yes	Yes
NISV	State Nine Severity	MENU menuAlarmSevr	Yes		Yes	Yes	Yes
TESV	State Ten Severity	MENU menuAlarmSevr	Yes		Yes	Yes	Yes
ELSV	State Eleven Severity	MENU menuAlarmSevr	Yes		Yes	Yes	Yes
TVSV	State Twelve Severity	MENU menuAlarmSevr	Yes		Yes	Yes	Yes
TTSV	State Thirteen Sevr	MENU menuAlarmSevr	Yes		Yes	Yes	Yes
FTSV	State Fourteen Sevr	MENU menuAlarmSevr	Yes		Yes	Yes	Yes
FFSV	State Fifteen Sevr	MENU menuAlarmSevr	Yes		Yes	Yes	Yes

Run-Time Parameters

These parameters are used by the run-time code for processing the multi-bit binary output.

MASK is used by device support routine to read the hardware register. Record support sets low order of MASK the number of bits specified in NOBT. Device support can shift this value.

The LALM field implements the change of state alarm severity by holding the value of VAL when the previous change of state alarm was issued.

MLST holds the value when the last monitor for value change was triggered.

SDEF is used by record support to save time if no states are defined; it is used for converting VAL to RVAL.

Field	Summary	Type	DCT	Default	Read	Write	CA PP
NOBT	Number of Bits	USHORT	Yes		Yes	No	No
ORAW	Prev Raw Value	ULONG	No		Yes	No	No
MASK	Hardware Mask	ULONG	No		Yes	No	No
LALM	Last Value Alarmed	USHORT	No		Yes	No	No
MLST	Last Value Monitored	USHORT	No		Yes	No	No
SDEF	States Defined	SHORT	No		Yes	No	No

Simulation Mode Parameters

The following fields are used to operate the record in simulation mode.

If SIMM (fetched through SIML, if populated) is YES, the record is put in SIMS severity and the value is written through SIOL, without conversion. If SIMM is RAW, the value is converted and RVAL is written. SSCN sets a different SCAN mechanism to use in simulation mode. SDLY sets a delay (in sec) that is used for asynchronous simulation processing.

See *Output Simulation Fields* for more information on simulation mode and its fields.

Field	Summary	Type	DCT	Default	Read	Write	CA PP
SIML	Simulation Mode Link	INLINK	Yes		Yes	Yes	No
SIMM	Simulation Mode	MENU menuSimm	No		Yes	Yes	No
SIOL	Simulation Output Link	OUTLINK	Yes		Yes	Yes	No
SIMS	Simulation Mode Severity	MENU menuAlarmSevr	Yes		Yes	Yes	No
SDLY	Sim. Mode Async Delay	DOUBLE	Yes	-1.0	Yes	Yes	No
SSCN	Sim. Mode Scan	MENU menuScan	Yes	65535	Yes	Yes	No

Record Support

Record Support Routines

init_record

This routine initializes SIMM if SIML is a constant or creates a channel access link if SIML is PV_LINK. If SIOL is PV_LINK a channel access link is created.

This routine next checks to see that device support is available. The routine next checks to see if the device support write routine is defined. If either device support or the device support write routine does not exist, an error message is issued and processing is terminated.

If DOL is a constant, then VAL is initialized to its value and UDF is set to FALSE.

MASK is cleared and then the NOBT low order bits are set.

If device support includes `init_record()`, it is called.

init_common is then called to determine if any states are defined. If states are defined, SDEF is set to TRUE.

If device support returns success, VAL is then set from RVAL and UDF is set to FALSE.

process

See next section.

special

Computes SDEF when any of the fields ZRVL,…FFVL change value.

get_value

Fills in the values of struct valueDes so that they refer to VAL.

get_enum_str

Retrieves ASCII string corresponding to VAL.

get_enum_strs

Retrieves ASCII strings for ZRST,…FFST.

put_enum_str

Checks if string matches ZRST,…FFST and if it does, sets VAL.

Record Processing

Routine process implements the following algorithm:

1. Check to see that the appropriate device support module exists. If it doesn't, an error message is issued and processing is terminated with the PACT field still set to TRUE. This ensures that processes will not longer be called for this record. Thus error storms will not occur.
2. If PACT is FALSE
   - If DOL is DB_LINK and OMSL is CLOSED_LOOP
     - Get value from DOL
     - Set UDF to FALSE
     - Check for link alarm
   - If any state values are defined
     - If VAL > 15, then raise alarm and go to 4
     - Else using VAL as index set RVAL = one of ZRVL,…FFVL
   - Else set RVAL = VAL
   - Shift RVAL left SHFT bits
3. Convert
   - If PACT is FALSE, compute RVAL
     - If VAL is 0,…,15, set RVAL from ZRVL,…,FFVL
     - If VAL out of range, set RVAL = undefined
   - Status = write_mbbo
4. Check alarms. This routine checks to see if the new VAL causes the alarm status and severity to change. If so, NSEV, NSTA and LALM are set.
5. Check severity and write the new value. See *Output Simulation Fields* and *Invalid Output Action Fields* for more information.
6. If PACT has been changed to TRUE, the device support write output routine has started but has not completed writing the new value. In this case, the processing routine merely returns, leaving PACT TRUE.
7. Check to see if monitors should be invoked.
   - Alarm monitors are invoked if the alarm status or severity has changed.
   - Archive and value change monitors are invoked if MLST is not equal to VAL.
   - Monitors for RVAL and RBV are checked whenever other monitors are invoked.
   - NSEV and NSTA are reset to 0.
8. Scan forward link if necessary, set PACT FALSE, and return.

Device Support

Fields Of Interest To Device Support

Each mbbo record must have an associated set of device support routines. The primary responsibility of the device support routines is to obtain a new raw mbbo value whenever write_mbbo is called. The device support routines are primarily interested in the following fields:

Field	Summary	Type	DCT	Default	Read	Write	CA PP
PACT	Record active	UCHAR	No		Yes	No	No
DPVT	Device Private	NOACCESS	No		No	No	No
NSEV	New Alarm Severity	MENU menuAlarmSevr	No		Yes	No	No
NSTA	New Alarm Status	MENU menuAlarmStat	No		Yes	No	No
NOBT	Number of Bits	USHORT	Yes		Yes	No	No
OUT	Output Specification	OUTLINK	Yes		Yes	Yes	No
RVAL	Raw Value	ULONG	No		Yes	Yes	Yes
RBV	Readback Value	ULONG	No		Yes	No	No
MASK	Hardware Mask	ULONG	No		Yes	No	No
SHFT	Shift	USHORT	Yes		Yes	Yes	No

Device Support Routines

Device support consists of the following routines:

long report(int level)

This optional routine is called by the IOC command `dbior` and is passed the report level that was requested by the user. It should print a report on the state of the device support to stdout. The `level` parameter may be used to output increasingly more detailed information at higher levels, or to select different types of information with different levels. Level zero should print no more than a small summary.

long init(int after)

This optional routine is called twice at IOC initialization time. The first call happens before any of the `init_record()` calls are made, with the integer parameter `after` set to 0. The second call happens after all of the `init_record()` calls have been made, with `after` set to 1.

init_record

```
init_record(precord)
```

This routine is optional. If provided, it is called by the record support's `init_record()` routine. If MASK is used, it should be shifted if necessary and SHFT given a value.

get_ioint_info

```
get_ioint_info(int cmd,struct dbCommon *precord,IOSCANPVT *ppvt)
```

This routine is called by the ioEventScan system each time the record is added or deleted from an I/O event scan list. `cmd` has the value (0,1) if the record is being (added to, deleted from) an I/O event list. It must be provided for any device type that can use the ioEvent scanner.

write_mbbo

```
write_mbbo(precord)
```

This routine must output a new value. It returns the following values:

- 0: Success.
- Other: Error.

Device Support For Soft Records

Soft Channel

The `Soft Channel` module writes the current value of VAL.

If the OUT link type is PV_LINK, then dbCaAddInlink is called by `init_record()`.

`write_mbbo()` calls `dbPutLink()` to write the current value of VAL. See *"Soft Output"* for more information.

Raw Soft Channel

This module writes RVAL to the location specified in the output link. It returns a 0.

1.5.25 Permissive Record (permissive)

The permissive record is for communication between a server and a client. An example would be a sequence program server and an operator interface client. By using multiple permissive records a sequence program can communicate its current state to the client.

Note this record is deprecated and may be removed in a future EPICS release.

Parameter Fields

The record-specific fields are described below, grouped by functionality.

Scan Parameters

The permissive record has the standard fields for specifying under what circumstances the record will be processed. These fields are listed in *Scan Fields*.

Client-server Parameters

The client and server communicate through the VAL and watchdog flag (WFLG) fields. At initialization, both fields are set equal to 0, which means OFF. The server sets WFLG equal to ON when it is ready to accept a request. The client monitors WFLG and when WFLG equals 1, the client-server action is performed (a private matter between server and client).

When WFLG is off–when the server is busy–the client program may turn the VAL field from OFF to ON. After the server finishes its task, it will notice that VAL is ON and will turn both WFLG and VAL OFF and performs the requested service.

Note that when WFLG is ON, the client program ''must not" turn VAL to on.

Field	Summary	Type	DCT	Default	Read	Write	CA PP
VAL	Status	USHORT	Yes		Yes	Yes	Yes
WFLG	Wait Flag	USHORT	No		Yes	Yes	Yes

Operator Display Parameters

The label field (LABL) contains a string given to it that should describe the record in further detail. In addition to the DESC field. See *"Fields Common to All Record Types"* for more on the record name (NAME) and description (DESC) fields.

Field	Summary	Type	DCT	Default	Read	Write	CA PP
LABL	Button Label	STRING [20]	Yes		Yes	Yes	Yes
NAME	Record Name	STRING [61]	No		Yes	No	No
DESC	Descriptor	STRING [41]	Yes		Yes	Yes	No

Alarm Parameters

The Permissive record has the alarm parameters common to all record types. *Alarm Fields* lists the fields related to alarms that are common to all record types.

Run-time Parameters

These fields are used to trigger monitors for each field. Monitors for the VAL field are triggered when OVAL, the old value field, does not equal VAL. Likewise, OFLG causes monitors to be invoked for WFLG when WFLG does not equal OLFG.

Field	Summary	Type	DCT	Default	Read	Write	CA PP
OVAL	Old Status	USHORT	No		Yes	No	No
OFLG	Old Flag	USHORT	No		Yes	No	No

Record Support

Record Support Routines

process

```
long (*process)(struct dbCommon *precord)
```

`process()` sets UDF to FALSE, triggers monitors on VAL and WFLG when they change, and scans the forward link if necessary.

1.5.26 Printf Record (printf)

The printf record is used to generate and write a string using a format specification and parameters, analogous to the C `printf()` function.

Parameter Fields

The record-specific fields are described below, grouped by functionality.

Scan Parameters

The printf record has the standard fields for specifying under what circumstances it will be processed. These fields are described in *Scan Fields*.

Field	Summary	Type	DCT	Default	Read	Write	CA PP
SCAN	Scan Mechanism	MENU menuScan	Yes		Yes	Yes	No
PHAS	Scan Phase	SHORT	Yes		Yes	Yes	No
EVNT	Event Name	STRING [40]	Yes		Yes	Yes	No
PRIO	Scheduling Priority	MENU menuPriority	Yes		Yes	Yes	No
PINI	Process at iocInit	MENU menuPini	Yes		Yes	Yes	No

String Generation Parameters

The printf record must specify the desired output string with embedded format specifiers in the FMT field. Plain characters are copied directly to the output string. A pair of percent characters '`%%`' are converted into a single percent character in the output string. A single precent character '`%`' introduces a format specifier and is followed by zero or more of the standard `printf()` format flags and modifiers:

- Plus ('`+`')
- Minus ('`-`')
- Space ('` `')
- Hash ('`#`')
- Minimum Field Width (decimal digits or '`*`')
- Precision ('`.`' followed by decimal digits or '`*`')
- Length Modifier '`hh`' – Reads link as DBR_CHAR or DBR_UCHAR
- Length Modifier '`h`' – Reads link as DBR_SHORT or DBR_USHORT for integer conversions, DBR_FLOAT for floating-point conversions.
- Length Modifier '`l`' – Reads link as DBR_LONG or DBR_ULONG for integer conversions, array of DBR_CHAR for string conversion.
- Length Modifier '`ll`' – Reads link as DBR_INT64 or DBR_UINT64 for integer conversions.

The following character specifies the conversion to perform, see your operating system's `printf()` documentation for more details. These conversions ultimately call the `snprintf()` routine for the actual string conversion process, so are subject to the behaviour of that routine.

- '`c`' – Convert to a character. Only single byte characters are permitted.
- '`d`' or '`i`' – Convert to a decimal integer.

- ‘`o`’ – Convert to an unsigned octal integer.
- ‘`u`’ – Convert to an unsigned decimal integer.
- ‘`x`’ – Convert to an unsigned hexadecimal integer, using `abcdef`.
- ‘`X`’ – Convert to an unsigned hexadecimal integer, using `ABCDEF`.
- ‘`e`’ or ‘`E`’ – Convert to floating-point in exponent style, reading the link as DBR_DOUBLE or DBR_FLOAT.
- ‘`f`’ or ‘`F`’ – Convert to floating-point in fixed-point style, reading the link as DBR_DOUBLE or DBR_FLOAT.
- ‘`g`’ or ‘`G`’ – Convert to floating-point in general style, reading the link as DBR_DOUBLE or DBR_FLOAT.
- ‘`s`’ – Insert string, reading the link as DBR_STRING or array of DBR_CHAR.

The fields INP0 ... INP9 are input links that provide the parameter values to be formatted into the output. The format specifiers in the FMT string determine which type of the data is requested through the appropriate input link. As with `printf()` a `*` character may be used in the format to specify width and/or precision instead of numeric literals, in which case additional input links are used to provide the necessary integer parameter or parameters. See Address Specification for information on specifying links.

The formatted string is written to the VAL field. The maximum number of characters in VAL is given by SIZV, and cannot be larger than 65535. The LEN field contains the length of the formatted string in the VAL field.

Field	Summary	Type	DCT	Default	Read	Write	CA PP
FMT	Format String	STRING [81]	Yes		Yes	Yes	Yes
INP0	Input 0	INLINK	Yes		Yes	Yes	No
INP1	Input 1	INLINK	Yes		Yes	Yes	No
INP2	Input 2	INLINK	Yes		Yes	Yes	No
INP3	Input 3	INLINK	Yes		Yes	Yes	No
INP4	Input 4	INLINK	Yes		Yes	Yes	No
INP5	Input 5	INLINK	Yes		Yes	Yes	No
INP6	Input 6	INLINK	Yes		Yes	Yes	No
INP7	Input 7	INLINK	Yes		Yes	Yes	No
INP8	Input 8	INLINK	Yes		Yes	Yes	No
INP9	Input 9	INLINK	Yes		Yes	Yes	No
VAL	Result	STRING[SIZV]	No		Yes	Yes	Yes
SIZV	Size of VAL buffer	USHORT	Yes	41	Yes	No	No
LEN	Length of VAL	ULONG	No		Yes	No	No

Output Specification

The output link specified in the OUT field specifies where the printf record is to write the contents of its VAL field. The link can be a database or channel access link. If the OUT field is a constant, no output will be written.

In addition, the appropriate device support module must be entered into the DTYP field.

Field	Summary	Type	DCT	Default	Read	Write	CA PP
OUT	Output Specification	OUTLINK	Yes		Yes	Yes	No
DTYP	Device Type	DEVICE	Yes		Yes	Yes	No

Operator Display Parameters

See *Fields Common to All Record Types* for more on the record name (NAME) and description (DESC) fields.

Field	Summary	Type	DCT	Default	Read	Write	CA PP
NAME	Record Name	STRING [61]	No		Yes	No	No
DESC	Descriptor	STRING [41]	Yes		Yes	Yes	No

Alarm Parameters

The printf record has the alarm parameters common to all record types. *Alarm Fields* lists the fields related to alarms that are common to all record types.

The IVLS field specifies a string which is sent to the OUT link if if input link data are invalid.

Field	Summary	Type	DCT	Default	Read	Write	CA PP
IVLS	Invalid Link String	STRING [16]	Yes	LNK	Yes	Yes	No

Device Support Interface

The record requires device support to provide an entry table (dset) which defines the following members:

```
typedef struct {
    long number;
    long (*report)(int level);
    long (*init)(int after);
    long (*init_record)(printfRecord *prec);
    long (*get_ioint_info)(int cmd, printfRecord *prec, IOSCANPVT *piosl);
    long (*write_string)(printfRecord *prec);
} printfdset;
```

The module must set `number` to at least 5, and provide a pointer to its `write_string()` routine; the other function pointers may be `NULL` if their associated functionality is not required for this support layer. Most device supports also provide an `init_record()` routine to configure the record instance and connect it to the hardware or driver support layer.

Device Support for Soft Records

A soft device support module Soft Channel is provided for writing values to other records or other software components.

Device support for DTYP `stdio` is provided for writing values to the stdout, stderr, or errlog streams. `INST_IO` addressing `@stdout`, `@stderr` or `@errlog` is used on the OUT link field to select the desired stream.

1.5.27 Select Record (sel)

The select record computes a value based on input obtained from up to 12 locations. The selection algorithm can be one of the following: `Specified`, `High Signal`, `Low Signal`, `Median Signal`. Each input can be a constant, a database link, or a channel access link.

Parameter Fields

The record-specific fields are described below, grouped by functionality.

Scan Parameters

The select record has the standard fields for specifying under what circumstances the record will be processed. These fields are listed in *Scan Fields*.

Read Parameters

The INPA-L links determine where the selection record retrieves the values from which it is to select or compute its final value. The INPA-L links are input links configured by the user to be either constants, channel access links, or database links. If channel access or database links, a value is retrieved for each link and placed in the corresponding value field, A-L. If any input link is a constant, the value field for that link will be initialized with the constant value given to it and can be modified via dbPuts.

Any links not defined are ignored by the selection record and its algorithm. An undefined link is any constant link whose value is 0. At initialization time, the corresponding value links for such fields are set to NaN, which means MISSING. The value field of an undefined link can be changed at run-time from NaN to another value in order to define the link and its field. Note that all undefined links must be recognized as such if the selection algorithm is to work as expected.

Field	Summary	Type	DCT	Default	Read	Write	CA PP
INPA	Input A	INLINK	Yes		Yes	Yes	No
INPB	Input B	INLINK	Yes		Yes	Yes	No
INPC	Input C	INLINK	Yes		Yes	Yes	No
INPD	Input D	INLINK	Yes		Yes	Yes	No
INPE	Input E	INLINK	Yes		Yes	Yes	No
INPF	Input F	INLINK	Yes		Yes	Yes	No
INPG	Input G	INLINK	Yes		Yes	Yes	No
INPH	Input H	INLINK	Yes		Yes	Yes	No
INPI	Input I	INLINK	Yes		Yes	Yes	No
INPJ	Input J	INLINK	Yes		Yes	Yes	No
INPK	Input K	INLINK	Yes		Yes	Yes	No
INPL	Input L	INLINK	Yes		Yes	Yes	No
A	Value of Input A	DOUBLE	No		Yes	Yes	Yes
B	Value of Input B	DOUBLE	No		Yes	Yes	Yes
C	Value of Input C	DOUBLE	No		Yes	Yes	Yes
D	Value of Input D	DOUBLE	No		Yes	Yes	Yes
E	Value of Input E	DOUBLE	No		Yes	Yes	Yes
F	Value of Input F	DOUBLE	No		Yes	Yes	Yes
G	Value of Input G	DOUBLE	No		Yes	Yes	Yes
H	Value of Input H	DOUBLE	No		Yes	Yes	Yes
I	Value of Input I	DOUBLE	No		Yes	Yes	Yes
J	Value of Input J	DOUBLE	No		Yes	Yes	Yes
K	Value of Input K	DOUBLE	No		Yes	Yes	Yes
L	Value of Input L	DOUBLE	No		Yes	Yes	Yes

Select Parameters

The selection algorithm is determined by three fields configurable by the user: the select mechanism (SELM) field, the select number (SELN) field, and the index value location (NVL) field.

The SELM field has four choices, i.e., four algorithms as follows:

Menu selSELM

Index	Identifier	Choice String
0	selSELM_Specified	Specified
1	selSELM_High_Signal	High Signal
2	selSELM_Low_Signal	Low Signal
3	selSELM_Median_Signal	Median Signal

The selection record's VAL field is determined differently for each algorithm. For `Specified`, the VAL field is set equal to the value field (A, B, C, D, E, F, G, H, I, J, K, or L) specified by the SELN field. The SELN field contains a number from 0-11 which corresponds to the value field to be used (0 means use A; 1 means use B, etc.). How the NVL field is configured determines, in turn, SELN's value. NVL is an input link from which a value for SELN can be retrieved, Like most other input links NVL can be a constant, or a channel access or database link. If NVL is a link, SELN is retrieved from the location in NVL. If a constant, SELN is initialized to the value given to the constant and can be changed via dbPuts.

The `High Signal`, `Low Signal`, and `Median Signal` algorithms do not use SELN or NVL. If `High Signal` is chosen, VAL is set equal to the highest value out of all the defined value fields (A-L). If `Low Signal` is chosen, VAL is set equal to lowest value of all the defined fields (A-L). And if `Median Signal` is chosen, VAL is set equal to the median value of the defined value fields (A-L). (Note that these algorithms select from the value fields; they do not select from the value field index. For instance, `Low Signal` will not select the A field's value unless the value itself is the lowest of all the defined values.)

Field	Summary	Type	DCT	Default	Read	Write	CA PP
SELM	Select Mechanism	MENU *selSELM*	Yes		Yes	Yes	No
SELN	Index value	USHORT	No		Yes	Yes	No
NVL	Index Value Location	INLINK	Yes		Yes	Yes	No

Operator Display Parameters

These parameters are used to present meaningful data to the operator. They display the value and other parameters of the select record either textually or graphically.

EGU is a string of up to 16 characters describing the units that the selection record manipulates. It is retrieved by the `get_units` record support routine.

The HOPR and LOPR fields set the upper and lower display limits for the VAL, HIHI, HIGH, LOW, and LOLO fields. Both the `get_graphic_double` and `get_control_double` record support routines retrieve these fields.

The PREC field determines the floating point precision with which to display VAL. It is used whenever the `get_precision` record support routine is called.

See *Fields Common to All Record Types* for more on the record name (NAME) and description (DESC) fields.

Field	Summary	Type	DCT	Default	Read	Write	CA PP
EGU	Engineering Units	STRING [16]	Yes		Yes	Yes	No
HOPR	High Operating Rng	DOUBLE	Yes		Yes	Yes	No
LOPR	Low Operating Range	DOUBLE	Yes		Yes	Yes	No
PREC	Display Precision	SHORT	Yes		Yes	Yes	No
NAME	Record Name	STRING [61]	No		Yes	No	No
DESC	Descriptor	STRING [41]	Yes		Yes	Yes	No

Alarm Parameters

The possible alarm conditions for select records are the SCAN, READ, and limit alarms. The SCAN and READ alarms are called by the record or device support routines. The limit alarms are configured by the user in the HIHI, LOLO, HIGH, and LOW fields using numerical values. They specify conditions for the VAL field. For each of these fields, there is a corresponding severity field which can be either NO_ALARM, MINOR, or MAJOR. *Alarm Fields* lists the fields related to alarms that are common to all record types.

Field	Summary	Type	DCT	Default	Read	Write	CA PP
HIHI	Hihi Alarm Limit	DOUBLE	Yes		Yes	Yes	Yes
HIGH	High Alarm Limit	DOUBLE	Yes		Yes	Yes	Yes
LOW	Low Alarm Limit	DOUBLE	Yes		Yes	Yes	Yes
LOLO	Lolo Alarm Limit	DOUBLE	Yes		Yes	Yes	Yes
HHSV	Hihi Severity	MENU menuAlarmSevr	Yes		Yes	Yes	Yes
HSV	High Severity	MENU menuAlarmSevr	Yes		Yes	Yes	Yes
LSV	Low Severity	MENU menuAlarmSevr	Yes		Yes	Yes	Yes
LLSV	Lolo Severity	MENU menuAlarmSevr	Yes		Yes	Yes	Yes
HYST	Alarm Deadband	DOUBLE	Yes		Yes	Yes	No

Monitor Parameters

These fields are configurable by the user. They are used as deadbands for the archiver and monitor calls for the VAL field. Unless, VAL changes by more than the value specified by each, then the respective monitors will not be called. If these fields have a value of zero, everytime the VAL changes, monitors are triggered; if they have a value of -1, everytime the record is processed, monitors are triggered.

Field	Summary	Type	DCT	Default	Read	Write	CA PP
ADEL	Archive Deadband	DOUBLE	Yes		Yes	Yes	No
MDEL	Monitor Deadband	DOUBLE	Yes		Yes	Yes	No

Run-time Parameters

These parameters are used by the run-time code for processing the selection record. They are not configurable prior to run-time, nor are they modifiable at run-time. They represent the current state of the record. The record support routines use some of them for more efficient processing.

The VAL field is the result of the selection record's processing. It can be accessed in the normal way by another record or through database access, but is not modifiable except by the record itself. The LALM, ALST, and the MLST are used to implement the HYST, ADEL, and MDEL hysteresis factors for the alarms, archiver, and monitors, respectively.

The LA-LL fields are used to implement the monitors for each of the value fields, A-L. They represent previous input values. For example, unless LA is not equal to A, no monitor is invoked for A.

Field	Summary	Type	DCT	Default	Read	Write	CA PP
VAL	Result	DOUBLE	Yes		Yes	No	No
LALM	Last Value Alarmed	DOUBLE	No		Yes	No	No
ALST	Last Value Archived	DOUBLE	No		Yes	No	No
MLST	Last Val Monitored	DOUBLE	No		Yes	No	No
LA	Prev Value of A	DOUBLE	No		Yes	No	No
LB	Prev Value of B	DOUBLE	No		Yes	No	No
LC	Prev Value of C	DOUBLE	No		Yes	No	No
LD	Prev Value of D	DOUBLE	No		Yes	No	No
LE	Prev Value of E	DOUBLE	No		Yes	No	No
LF	Prev Value of F	DOUBLE	No		Yes	No	No
LG	Prev Value of G	DOUBLE	No		Yes	No	No
LH	Prev Value of H	DOUBLE	No		Yes	No	No
LI	Prev Value of I	DOUBLE	No		Yes	No	No
LJ	Prev Value of J	DOUBLE	No		Yes	No	No
LK	Prev Value of K	DOUBLE	No		Yes	No	No
LL	Prev Value of L	DOUBLE	No		Yes	No	No

Record Support

Record Support Routines

init_record

```
long (*init_record)(struct dbCommon *precord, int pass)
```

IF NVL is a constant, SELN is set to its value. If NVL is a PV_LINK a channel access link is created.

For each constant input link, the corresponding value field is initialized with the constant value (or NaN if the constant has the value 0).

For each input link that is of type PV_LINK, a database or channel access link is created.

process

```
long (*process)(struct dbCommon *precord)
```

See *"Record Processing"*.

get_units

```
long (*get_units)(struct dbAddr *paddr, char *units)
```

Retrieves EGU.

get_precision

```
long (*get_precision)(const struct dbAddr *paddr, long *precision)
```

Retrieves PREC.

get_graphic_double

```
long (*get_graphic_double)(struct dbAddr *paddr, struct dbr_grDouble *p)
```

Sets the upper display and lower display limits for a field. If the field is VAL, HIHI, HIGH, LOW, or LOLO, the limits are set to HOPR and LOPR, else if the field has upper and lower limits defined they will be used, else the upper and lower maximum values for the field type will be used.

get_control_double

```
long (*get_control_double)(struct dbAddr *paddr, struct dbr_ctrlDouble *p)
```

Sets the upper control and the lower control limits for a field. If the field is VAL, HIHI, HIGH, LOW, or LOLO, the limits are set to HOPR and LOPR, else if the field has upper and lower limits defined they will be used, else the upper and lower maximum values for the field type will be used.

get_alarm_double

```
long (*get_alarm_double)(struct dbAddr *paddr, struct dbr_alDouble *p)
```

Sets the following values:

```
upper_alarm_limit = HIHI
upper_warning_limit = HIGH
lower_warning_limit = LOW
lower_alarm_limit = LOLO
```

Record Processing

Routine process implements the following algorithm:

1. If NVL is a database or channel access link, SELN is obtained from NVL. Fetch all values if database or channel access links. If SELM is SELECTED, then only the selected link is fetched.
2. Implement the appropriate selection algorithm. For SELECT_HIGH, SELECT_LOW, and SELECT_MEDIAN, input fields are ignored if they are undefined. If success, UDF is set to FALSE.
3. Check alarms. This routine checks to see if the new VAL causes the alarm status and severity to change. If so, NSEV, NSTA, and LALM are set. It also honors the alarm hysteresis factor (HYST). Thus the value must change by more than HYST before the alarm status and severity is lowered.
4. Check to see if monitors should be invoked.
   - Alarm monitors are invoked if the alarm status or severity has changed.
   - Archive and value change monitors are invoked if ADEL and MDEL conditions are met
   - Monitors for A-L are checked whenever other monitors are invoked
   - NSEV and NSTA are reset to 0.
5. Scan forward link if necessary, set PACT FALSE, and return.

1.5.28 Sequence Record (seq)

The Sequence record is used to trigger the processing of up to ten other records and send values to those records. It is similar to the fanout record, except that it will fetch an input value and write an output value instead of simply processing a collection of forward links. It can also specify one of several selection algorithms that determine which values to write. It has no associated device support.

Parameter Fields

The record-specific fields are described below, grouped by functionality.

Scan Parameters

The sequence record has the standard fields for specifying under what circumstances it will be processed. These fields are listed in *Scan Fields*.

Desired Output Parameters

These fields determine where the record retrieves the values it is to write to other records. All of these values are not necessarily used, depending on the selection algorithm.

The sequence record can retrieve up to 16 values from 16 locations. The user specifies the locations in the Desired Output Link fields (DOL0-DOLF), which can be either constants, database links, or channel access links. If a Desired Output Link is a constant, the corresponding value field for that link is initialized to the constant value. Otherwise, if the Desired Output Link is a database or channel access link, a value is fetched from the link each time the record is processed.

The value fetched from the Desired Output Links are stored in the corresponding Desired Output Value fields (DO0-DOF). These fields can be initialized to a constant value, and may subsequently be changed via dbPuts.

Desired Output Link Fields

Field	Summary	Type	DCT	Default	Read	Write	CA PP
DOL0	Input link 0	INLINK	Yes		Yes	Yes	No
DOL1	Input link1	INLINK	Yes		Yes	Yes	No
DOL2	Input link 2	INLINK	Yes		Yes	Yes	No
DOL3	Input link 3	INLINK	Yes		Yes	Yes	No
DOL4	Input link 4	INLINK	Yes		Yes	Yes	No
DOL5	Input link 5	INLINK	Yes		Yes	Yes	No
DOL6	Input link 6	INLINK	Yes		Yes	Yes	No
DOL7	Input link 7	INLINK	Yes		Yes	Yes	No
DOL8	Input link 8	INLINK	Yes		Yes	Yes	No
DOL9	Input link 9	INLINK	Yes		Yes	Yes	No
DOLA	Input link 10	INLINK	Yes		Yes	Yes	No
DOLB	Input link 11	INLINK	Yes		Yes	Yes	No
DOLC	Input link 12	INLINK	Yes		Yes	Yes	No
DOLD	Input link 13	INLINK	Yes		Yes	Yes	No
DOLE	Input link 14	INLINK	Yes		Yes	Yes	No
DOLF	Input link 15	INLINK	Yes		Yes	Yes	No

Desired Output Value Fields

Field	Summary	Type	DCT	Default	Read	Write	CA PP
DO0	Value 0	DOUBLE	No		Yes	Yes	No
DO1	Value 1	DOUBLE	No		Yes	Yes	No
DO2	Value 2	DOUBLE	No		Yes	Yes	No
DO3	Value 3	DOUBLE	No		Yes	Yes	No
DO4	Value 4	DOUBLE	No		Yes	Yes	No
DO5	Value 5	DOUBLE	No		Yes	Yes	No
DO6	Value 6	DOUBLE	No		Yes	Yes	No
DO7	Value 7	DOUBLE	No		Yes	Yes	No
DO8	Value 8	DOUBLE	No		Yes	Yes	No
DO9	Value 9	DOUBLE	No		Yes	Yes	No
DOA	Value 10	DOUBLE	No		Yes	Yes	No
DOB	Value 11	DOUBLE	No		Yes	Yes	No
DOC	Value 12	DOUBLE	No		Yes	Yes	No
DOD	Value 13	DOUBLE	No		Yes	Yes	No
DOE	Value 14	DOUBLE	No		Yes	Yes	No
DOF	Value 15	DOUBLE	No		Yes	Yes	No

Output Parameters

When the record is processed, the desired output values are retrieved for the links in the record's selection algorithm and are written to the corresponding output link (LNK0-LNKF). These output links can be database links or channel access links; they cannot be device addresses. There are sixteen output links, one for each desired output link. Only those that are defined are used.

Field	Summary	Type	DCT	Default	Read	Write	CA PP
LNK0	Output Link 0	OUTLINK	Yes		Yes	Yes	No
LNK1	Output Link 1	OUTLINK	Yes		Yes	Yes	No
LNK2	Output Link 2	OUTLINK	Yes		Yes	Yes	No
LNK3	Output Link 3	OUTLINK	Yes		Yes	Yes	No
LNK4	Output Link 4	OUTLINK	Yes		Yes	Yes	No
LNK5	Output Link 5	OUTLINK	Yes		Yes	Yes	No
LNK6	Output Link 6	OUTLINK	Yes		Yes	Yes	No
LNK7	Output Link 7	OUTLINK	Yes		Yes	Yes	No
LNK8	Output Link 8	OUTLINK	Yes		Yes	Yes	No
LNK9	Output Link 9	OUTLINK	Yes		Yes	Yes	No
LNKA	Output Link 10	OUTLINK	Yes		Yes	Yes	No
LNKB	Output Link 11	OUTLINK	Yes		Yes	Yes	No
LNKC	Output Link 12	OUTLINK	Yes		Yes	Yes	No
LNKD	Output Link 13	OUTLINK	Yes		Yes	Yes	No
LNKE	Output Link 14	OUTLINK	Yes		Yes	Yes	No
LNKF	Output Link 15	OUTLINK	Yes		Yes	Yes	No

Selection Algorithm Parameters

When the sequence record is processed, it uses a selection algorithm similar to that of the selection record to decide which links to process.The select mechanism field (SELM) has three algorithms to choose from: `All`, `Specified` or `Mask`.

Record fields related to the Selection Algorithm

Field	Summary	Type	DCT	Default	Read	Write	CA PP
SELM	Select Mechanism	MENU *seqSELM*	Yes		Yes	Yes	No
SELN	Link Selection	USHORT	No	1	Yes	Yes	No
SELL	Link Selection Loc	INLINK	Yes		Yes	Yes	No
SHFT	Shift for Mask mode	SHORT	Yes	-1	Yes	Yes	No
OFFS	Offset for Specified	SHORT	Yes		Yes	Yes	No

Fields Description

SELM - Selection Mode

Index	Identifier	Choice String
0	seqSELM_All	All
1	seqSELM_Specified	Specified
2	seqSELM_Mask	Mask

See *"Selection Algorithms Description"* below;

SELL - Link Selection Location

This field can be initialized as a CONSTANT or as a LINK to any other record. SELN will fetch its value from this field when the seq record is processed. Thus, when using `Mask` or `Specified` modes, the links that seq will process can be dynamically changed by the record pointed by SELL.

SELN - Link Selection

When **SELM** has the value `Specified` the **SELN** field sets the index number of the link that will be processed, after adding the **OFFS** field:

LNK_n_ where *n* = `SELN + OFFS`

(If not set, the OFFS field is ZERO)

When **SELM** has the value `Mask` the **SELN** field provides the bitmask that determines which links will be processed, after shifting by **SHFT** bits:

```
if (SHFT >= 0)
  bits = SELN >> SHFT
else
  bits = SELN << -SHFT
```

(If not set, the SHFT field is -1 so bits from SELN are shifted left by 1)

Note about SHFT and OFFS fields

The first versions of seq record had DO, DOL, LNK and DLY fields starting with index ONE (DO1, DOL1, LNK1 and DLY1). Since EPICS 7 the seq record now supports 16 links, starting from index ZERO (DO0, DOL0, LNK0 and DLY0). The SHFT and OFFS fields were introduced to keep compatibility of old databases that used seq records with links indexed from one.

To use the DO0, DOL0, LNK0, DLY0 fields when SELM = Mask, the SHFT field must be explicitly set to ZERO

Selection Algorithms Description

All

The `All` algorithm causes the record to process each input and output link each time the record is processed, in order from 0 to 15. So when SELM is `All`, the desired output value from DOL0 will fetched and sent to LNK0, then the desired output value from DOL1 will be fetched and sent to the location in LNK1, and so on until the last input and output link DOF and LNKF. (Note that undefined links are not used.) If DOL_x_ is a constant, the current value field is simply used and the desired output link is ignored. The SELN field is not used when `All` is the algorithm.

Specified

When the `Specified` algorithm is chosen, each time the record is processed it gets the integer value in the Link Selection (SELN) field and uses that as the index of the link to process. For instance, if SELN is 4, the desired output value from DO4 will be retrieved and sent to LNK4. If DOL_x_ is a constant, DO_x_ is simply used without the value being fetched from the input link.

Mask

When `Mask` is chosen, the record uses the individual bits of the SELN field to determine the links to process. When bit 0 of SELN is set, the value from DO0 will be written to the location in LNK0; when bit 1 is set, the valud from DO1 will be written to the location in LNK1 etc. Thus for example if SELN is 3, the record will retrieve the values from DO0 and DO1 and write them to the locations in LNK0 and LNK1, respectively. If SELN is 63, DO0…DO5 will be written to LNK0…LNK5.

Delay Parameters

The delay parameters consist of 16 fields, one for each I/O link discussed above. These fields can be configured to cause the record to delay processing the link. For instance, if the user gives the DLY1 field a value of 3.0, each time the record is processed at run-time, the record will delay processing the DOL1, DOV1, and LNK1 fields for three seconds. That is, the desired output value will not be fetched and written to the output link until three seconds have lapsed.

Field	Summary	Type	DCT	Default	Read	Write	CA PP
DLY0	Delay 0	DOUBLE	Yes		Yes	Yes	No
DLY1	Delay 1	DOUBLE	Yes		Yes	Yes	No
DLY2	Delay 2	DOUBLE	Yes		Yes	Yes	No
DLY3	Delay 3	DOUBLE	Yes		Yes	Yes	No
DLY4	Delay 4	DOUBLE	Yes		Yes	Yes	No
DLY5	Delay 5	DOUBLE	Yes		Yes	Yes	No
DLY6	Delay 6	DOUBLE	Yes		Yes	Yes	No
DLY7	Delay 7	DOUBLE	Yes		Yes	Yes	No
DLY8	Delay 8	DOUBLE	Yes		Yes	Yes	No
DLY9	Delay 9	DOUBLE	Yes		Yes	Yes	No
DLYA	Delay 10	DOUBLE	Yes		Yes	Yes	No
DLYB	Delay 11	DOUBLE	Yes		Yes	Yes	No
DLYC	Delay 12	DOUBLE	Yes		Yes	Yes	No
DLYD	Delay 13	DOUBLE	Yes		Yes	Yes	No
DLYE	Delay 14	DOUBLE	Yes		Yes	Yes	No
DLYF	Delay 15	DOUBLE	Yes		Yes	Yes	No

Operator Display Parameters

These parameters are used to present meaningful data to the operator. The Precision field (PREC) determines the decimal precision for the VAL field when it is displayed. It is used when the `get_precision` record routine is called.

See *Fields Common to All Record Types* for more on the record name (NAME) and description (DESC) fields.

Field	Summary	Type	DCT	Default	Read	Write	CA PP
PREC	Display Precision	SHORT	Yes		Yes	Yes	No
NAME	Record Name	STRING [61]	No		Yes	No	No
DESC	Descriptor	STRING [41]	Yes		Yes	Yes	No

Alarm Parameters

The sequence record has the alarm parameters common to all record types. *Alarm Fields* lists the fields related to alarms that are common to all record types.

Record Support

Record Processing

Routine process implements the following algorithm:

1. First, PACT is set to TRUE, and the link selection is fetched. Depending on the selection mechanism chosen, the appropriate set of link groups will be processed. If multiple link groups need to be processed they are done in increasing numerical order, from LNK0 to LNKF.
2. When LNK_x_ is to be processed, the corresponding DLY_x_ value is first used to generate the requested time delay, using the IOC's Callback subsystem to perform subsequent operations. This means that although PACT remains TRUE, the lockset that the sequence record belongs to will be unlocked for the duration of the delay time (an unlock occurs even when the delay is zero).
3. After DLY_x_ seconds have expired, the value in DO_x_ is saved locally and a new value is read into DO_x_ through the link DOL_x_ (if the link is valid). Next the record's timestamp is set, and the value in DO_x_ is written through the LNK_x_ output link. If the value of DO_x_ was changed when it was read in a monitor event is triggered on that field.
4. If any link groups remain to be processed, the next group is selected and the operations for that group are executed again from step 2 above.

   If the last link group has been processed, UDF is set to FALSE and the record's timestamp is set.
5. Monitors are posted on VAL and SELN.
6. The forward link is scanned, PACT is set FALSE, and the process routine returns.

For the delay mechanism to operate properly, the record is normally processed asynchronously. The only time the record will not be processed asynchronously is if it has nothing to do, because no link groups or only empty link groups are selected for processing (groups where both DOL_x_ and LNK_x_ are unset or contain only a constant value).

1.5.29 State Record (state)

The state record is a means for a state program to communicate with the operator interface. Its only function is to provide a place in the database through which the state program can inform the operator interface of its state by storing an arbitrary ASCII string in its VAL field.

Note this record is deprecated and may be removed in a future EPICS release.

Parameter Fields

The record-specific fields are described below, grouped by functionality.

Scan Parameters

The state record has the standard fields for specifying under what circumstances it will be processed. These fields are listed in *Scan Fields*.

Operator Display Parameters

See *Fields Common to All Record Types* for more on the record name (NAME) and description (DESC) fields.

Field	Summary	Type	DCT	Default	Read	Write	CA PP
NAME	Record Name	STRING [61]	No		Yes	No	No
DESC	Descriptor	STRING [41]	Yes		Yes	Yes	No

Alarm Parameters

The state record has the alarm parameters common to all record types. *Alarm Fields* lists the fields related to alarms that are common to all record types.

Run-time Parameters

These parameters are used by the application code to convey the state of the program to the operator interface. The VAL field holds the string retrieved from the state program. The OVAL is used to implement monitors for the VAL field. When the string in OVAL differs from the one in VAL, monitors are triggered. They represent the current state of the sequence program.

Field	Summary	Type	DCT	Default	Read	Write	CA PP
VAL	Value	STRING [20]	Yes		Yes	Yes	Yes
OVAL	Prev Value	STRING [20]	No		Yes	No	No

Record Support

Record Support Routines

process

```
long (*process)(struct dbCommon *precord)
```

`process()` triggers monitors on VAL when it changes and scans the forward link if necessary.

1.5.30 String Input Record (stringin)

The string input record retrieves an arbitrary ASCII string of up to 40 characters. Several device support routines are available, all of which are soft device support for retrieving values from other records or other software components.

Parameter Fields

The record-specific fields are described below, grouped by functionality.

Scan Parameters

The string input record has the standard fields for specifying under what circumstances it will be processed. These fields are listed in *Scan Fields*.

Input Specification

The INP field determines where the string input record gets its string. It can be a database or channel access link, or a constant. If constant, the VAL field is initialized with the constant and can be changed via dbPuts. Otherwise, the string is read from the specified location each time the record is processed and placed in the VAL field. The maximum number of characters that the string in VAL can be is 40. In addition, the appropriate device support module must be entered into the DTYP field.

Field	Summary	Type	DCT	Default	Read	Write	CA PP
VAL	Current Value	STRING [40]	Yes		Yes	Yes	Yes
INP	Input Specification	INLINK	Yes		Yes	Yes	No
DTYP	Device Type	DEVICE	Yes		Yes	Yes	No

Monitor Parameters

These parameters are used to specify when the monitor post should be sent by `monitor()` routine. There are two possible choices:

Menu stringinPOST

Index	Identifier	Choice String
0	stringinPOST_OnChange	On Change
1	stringinPOST_Always	Always

APST is used for archiver monitors and MPST is for all other type of monitors.

Field	Summary	Type	DCT	Default	Read	Write	CA PP
MPST	Post Value Monitors	MENU *stringinPOST*	Yes		Yes	Yes	No
APST	Post Archive Monitors	MENU *stringinPOST*	Yes		Yes	Yes	No

Operator Display Parameters

See *Fields Common to All Record Types* for more on the record name (NAME) and description (DESC) fields.

Field	Summary	Type	DCT	Default	Read	Write	CA PP
NAME	Record Name	STRING [61]	No		Yes	No	No
DESC	Descriptor	STRING [41]	Yes		Yes	Yes	No

Alarm Parameters

The string input record has the alarm parameters common to all record types. *Alarm Fields* lists the fields related to alarms that are common to all record types.

Run-time Parameters

The old value field (OVAL) of the string input is used to implement value change monitors for VAL. If VAL is not equal to OVAL, then monitors are triggered.

Field	Summary	Type	DCT	Default	Read	Write	CA PP
OVAL	Previous Value	STRING [40]	No		Yes	No	No

Simulation Mode Parameters

The following fields are used to operate the record in simulation mode.

If SIMM (fetched through SIML) is YES, the record is put in SIMS severity and the value is fetched through SIOL (buffered in SVAL). SSCN sets a different SCAN mechanism to use in simulation mode. SDLY sets a delay (in sec) that is used for asynchronous simulation processing.

See *Input Simulation Fields* for more information on simulation mode and its fields.

Field	Summary	Type	DCT	Default	Read	Write	CA PP
SIML	Simulation Mode Link	INLINK	Yes		Yes	Yes	No
SIMM	Simulation Mode	MENU menuYesNo	No		Yes	Yes	No
SIOL	Simulation Input Link	INLINK	Yes		Yes	Yes	No
SVAL	Simulation Value	STRING [40]	No		Yes	Yes	Yes
SIMS	Simulation Mode Severity	MENU menuAlarmSevr	Yes		Yes	Yes	No
SDLY	Sim. Mode Async Delay	DOUBLE	Yes	-1.0	Yes	Yes	No
SSCN	Sim. Mode Scan	MENU menuScan	Yes	65535	Yes	Yes	No

Record Support

Record Support Routines

init_record

```
long (*init_record)(struct dbCommon *precord, int pass)
```

This routine initializes SIMM with the value of SIML if SIML type is CONSTANT link or creates a channel access link if SIML type is PV_LINK. SVAL is likewise initialized if SIOL is CONSTANT or PV_LINK.

This routine next checks to see that device support is available and a record support read routine is defined. If either does not exist, an error message is issued and processing is terminated.

If device support includes an `init_record()` routine it is called.

process

```
long (*process)(struct dbCommon *precord)
```

See *"Record Processing"*.

Record Processing

Routine process implements the following algorithm:

1. Check to see that the appropriate device support module exists. If it doesn't, an error message is issued and processing is terminated with the PACT field still set to TRUE. This ensures that processes will no longer be called for this record. Thus error storms will not occur.
2. readValue is called. See *"Simulation Mode"* for more information on simulation mode fields and how they affect input.
3. If PACT has been changed to TRUE, the device support read routine has started but has not completed reading a new input value. In this case, the processing routine merely returns, leaving PACT TRUE.
4. `recGblGetTimeStamp()` is called.
5. Check to see if monitors should be invoked.
   - Alarm monitors are invoked if the alarm status or severity has changed.
   - Archive and value change monitors are invoked if OVAL is not equal to VAL.
   - NSEV and NSTA are reset to 0.
6. Scan forward link if necessary, set PACT FALSE, and return.

Device Support

Fields Of Interest To Device Support

Each stringin input record must have an associated set of device support routines. The primary responsibility of the device support routines is to obtain a new ASCII string value whenever read_stringin is called. The device support routines are primarily interested in the following fields:

Field	Summary	Type	DCT	Default	Read	Write	CA PP
PACT	Record active	UCHAR	No		Yes	No	No
DPVT	Device Private	NOACCESS	No		No	No	No
UDF	Undefined	UCHAR	Yes	1	Yes	Yes	Yes
VAL	Current Value	STRING [40]	Yes		Yes	Yes	Yes
INP	Input Specification	INLINK	Yes		Yes	Yes	No

Device Support Routines

Device support consists of the following routines:

report

```
long report(int level)
```

This optional routine is called by the IOC command `dbior` and is passed the report level that was requested by the user. It should print a report on the state of the device support to stdout. The `level` parameter may be used to output increasingly more detailed information at higher levels, or to select different types of information with different levels. Level zero should print no more than a small summary.

init

```
long init(int after)
```

This optional routine is called twice at IOC initialization time. The first call happens before any of the `init_record()` calls are made, with the integer parameter `after` set to 0. The second call happens after all of the `init_record()` calls have been made, with `after` set to 1.

init_record

```
long init_record(dbCommon *prec)
```

This routine is optional. If provided, it is called by the record support `init_record()` routine.

get_ioint_info

```
long get_ioint_info(int cmd, dbCommon *precord, IOSCANPVT *ppvt)
```

This routine is called by the ioEventScan system each time the record is added or deleted from an I/O event scan list. `cmd` has the value (0,1) if the record is being (added to, deleted from) an I/O event list. It must be provided for any device type that can use the ioEvent scanner.

read_stringin

```
long read_stringin(stringinRecord *prec)
```

This routine must provide a new input value. It returns the following values:

- 0: Success. A new ASCII string is stored into VAL.
- Other: Error.

Device Support for Soft Records

The `Soft Channel` module reads a value directly into VAL.

Device support for DTYP `getenv` is provided for retrieving strings from environment variables. `INST_IO` addressing `@<environment variable>` is used in the INP link field to select the desired environment variable.

1.5.31 String Output Record (stringout)

The stringout record is used to write an arbitrary ASCII string of up to 40 characters to other records or software variables.

Parameter Fields

The record-specific fields are described below, grouped by functionality.

Scan Parameters

The string output record has the standard fields for specifying under what circumstances it will be processed. These fields are listed in *Scan Fields*.

Desired Output Parameters

The string output record must specify from where it gets its desired output string. The first field that determines where the desired output originates is the output mode select (OMSL) field, which can have two possible value: `closed_loop` or `supervisory`. If `supervisory` is specified, DOL is ignored, the current value of VAL is written, and the VAL can be changed externally via dbPuts at run-time. If `closed_loop` is specified, the VAL field's value is obtained from the address specified in the Desired Output Link field (DOL) which can be either a database link or a channel access link.

DOL can also be a constant, in which case VAL will be initialized to the constant value. However to be interpreted as a constant instead of a CA link the constant can only be numeric, so string output records are best initialized by dirctly setting the VAL field. Note that if DOL is a constant, OMSL cannot be `closed_loop`.

Field	Summary	Type	DCT	Default	Read	Write	CA PP
VAL	Current Value	STRING [40]	Yes		Yes	Yes	Yes
DOL	Desired Output Link	INLINK	Yes		Yes	Yes	No
OMSL	Output Mode Select	MENU menuOmsl	Yes		Yes	Yes	No

Output Specification

The output link specified in the OUT field specifies where the string output record is to write its string. The link can be a database or channel access link. If the OUT field is a constant, no output will be written.

In addition, the appropriate device support module must be entered into the DTYP field.

Field	Summary	Type	DCT	Default	Read	Write	CA PP
OUT	Output Specification	OUTLINK	Yes		Yes	Yes	No
DTYP	Device Type	DEVICE	Yes		Yes	Yes	No

Monitor Parameters

These parameters are used to specify when the monitor post should be sent by `monitor()` routine. There are two possible choices:

Menu stringoutPOST

Index	Identifier	Choice String
0	stringoutPOST_OnChange	On Change
1	stringoutPOST_Always	Always

APST is used for archiver monitors and MPST is for all other type of monitors.

Field	Summary	Type	DCT	Default	Read	Write	CA PP
MPST	Post Value Monitors	MENU *stringoutPOST*	Yes		Yes	Yes	No
APST	Post Archive Monitors	MENU *stringoutPOST*	Yes		Yes	Yes	No

Operator Display Parameters

These parameters are used to present meaningful data to the operator. These fields are used to display the value and other parameters of the string output either textually or graphically.

See *Fields Common to All Record Types* for more on the record name (NAME) and description (DESC) fields.

Field	Summary	Type	DCT	Default	Read	Write	CA PP
NAME	Record Name	STRING [61]	No		Yes	No	No
DESC	Descriptor	STRING [41]	Yes		Yes	Yes	No

Run-time Parameters

The old value field (OVAL) of the string input is used to implement value change monitors for VAL. If VAL is not equal to OVAL, then monitors are triggered.

Field	Summary	Type	DCT	Default	Read	Write	CA PP
OVAL	Previous Value	STRING [40]	No		Yes	No	No

Simulation Mode Parameters

The following fields are used to operate the record in simulation mode.

If SIMM (fetched through SIML) is YES, the record is put in SIMS severity and the value is written through SIOL. SSCN sets a different SCAN mechanism to use in simulation mode. SDLY sets a delay (in sec) that is used for asynchronous simulation processing.

See *Output Simulation Fields* for more information on simulation mode and its fields.

Field	Summary	Type	DCT	Default	Read	Write	CA PP
SIML	Simulation Mode Link	INLINK	Yes		Yes	Yes	No
SIMM	Simulation Mode	MENU menuYesNo	No		Yes	Yes	No
SIOL	Simulation Output Link	OUTLINK	Yes		Yes	Yes	No
SIMS	Simulation Mode Severity	MENU menuAlarmSevr	Yes		Yes	Yes	No
SDLY	Sim. Mode Async Delay	DOUBLE	Yes	-1.0	Yes	Yes	No
SSCN	Sim. Mode Scan	MENU menuScan	Yes	65535	Yes	Yes	No

Alarm Parameters

The possible alarm conditions for the string output record are the SCAN, READ, and INVALID alarms. The severity of the first two is always MAJOR and not configurable.

The IVOA field specifies an action to take when the INVALID alarm is triggered. When `Set output to IVOV`, the value contained in the IVOV field is written to the output link during an alarm condition. See *Invalid Output Action Fields* for more information on the IVOA and IVOV fields.

Alarm Fields lists the fields related to alarms that are common to all record types.

Field	Summary	Type	DCT	Default	Read	Write	CA PP
IVOA	INVALID output action	MENU menuIvoa	Yes		Yes	Yes	No
IVOV	INVALID output value	STRING [40]	Yes		Yes	Yes	No

Record Support

Record Support Routines

init_record

```
long (*init_record)(struct dbCommon *precord, int pass)
```

This routine initializes SIMM if SIML is a constant or creates a channel access link if SIML is PV_LINK. If SIOL is PV_LINK a channel access link is created.

This routine next checks to see that device support is available. The routine next checks to see if the device support write routine is defined. If either device support or the device support write routine does not exist, an error message is issued and processing is terminated.

If DOL is a constant, then the type double constant, if non-zero, is converted to a string and stored into VAL and UDF is set to FALSE. If DOL type is a PV_LINK then dbCaAddInlink is called to create a channel access link.

If device support includes `init_record()`, it is called.

process

```
long (*process)(struct dbCommon *precord)
```

See *"Record Processing"*.

Record Processing

Routine process implements the following algorithm:

1. Check to see that the appropriate device support module exists. If it doesn't, an error message is issued and processing is terminated with the PACT field still set to TRUE. This ensures that processes will no longer be called for this record. Thus error storms will not occur.
2. If PACT is FALSE and OMSL is CLOSED_LOOP, recGblGetLinkValue is called to read the current value of VAL. See *"Soft Output"*. If the return status of recGblGetLinkValue is zero then UDF is set to FALSE.
3. Check severity and write the new value. See *"Simulation Mode"* and *Invalid Output Action Fields* for details on how the simulation mode and the INVALID alarm conditions affect output.
4. If PACT has been changed to TRUE, the device support write output routine has started but has not completed writing the new value. In this case, the processing routine merely returns, leaving PACT TRUE.
5. Check to see if monitors should be invoked.
    - Alarm monitors are invoked if the alarm status or severity has changed.
    - Archive and value change monitors are invoked if OVAL is not equal to VAL.
    - NSEV and NSTA are reset to 0.
6. Scan forward link if necessary, set PACT FALSE, and return.

Device Support

Fields Of Interest To Device Support

Each stringout output record must have an associated set of device support routines. The primary responsibility of the device support routines is to write a new value whenever write_stringout is called. The device support routines are primarily interested in the following fields:

Field	Summary	Type	DCT	Default	Read	Write	CA PP
PACT	Record active	UCHAR	No		Yes	No	No
DPVT	Device Private	NOACCESS	No		No	No	No
NSEV	New Alarm Severity	MENU menuAlarmSevr	No		Yes	No	No
NSTA	New Alarm Status	MENU menuAlarmStat	No		Yes	No	No
VAL	Current Value	STRING [40]	Yes		Yes	Yes	Yes
OUT	Output Specification	OUTLINK	Yes		Yes	Yes	No

Device Support Routines

Device support consists of the following routines:

report

```
long report(int level)
```

This optional routine is called by the IOC command `dbior` and is passed the report level that was requested by the user. It should print a report on the state of the device support to stdout. The `level` parameter may be used to output increasingly more detailed information at higher levels, or to select different types of information with different levels. Level zero should print no more than a small summary.

init

```
long init(int after)
```

This optional routine is called twice at IOC initialization time. The first call happens before any of the `init_record()` calls are made, with the integer parameter `after` set to 0. The second call happens after all of the `init_record()` calls have been made, with `after` set to 1.

init_record

```
long init_record(dbCommon *prec)
```

This routine is optional. If provided, it is called by the record support `init_record()` routine.

get_ioint_info

```
long get_ioint_info(int cmd, dbCommon *precord, IOSCANPVT *ppvt)
```

This routine is called by the ioEventScan system each time the record is added or deleted from an I/O event scan list. `cmd` has the value (0,1) if the record is being (added to, deleted from) an I/O event list. It must be provided for any device type that can use the ioEvent scanner.

write_stringout

```
long write_stringout(stringoutRecord *prec)
```

This routine must output a new value. It returns the following values:

- 0: Success.
- Other: Error.

Device Support for Soft Records

The `Soft Channel` device support module writes the current value of VAL.

Device support for DTYP `stdio` is provided for writing values to the stdout, stderr, or errlog streams. `INST_IO` addressing `@stdout`, `@stderr` or `@errlog` is used on the OUT link field to select the desired stream.

1.5.32 Sub-Array Record (subArray)

The normal use for the subArray record type is to obtain sub-arrays from waveform records. Setting either the number of elements (NELM) or index (INDX) fields causes the record to be processed anew so that applications in which the length and position of a sub-array in a waveform record vary dynamically can be implemented using standard EPICS operator interface tools.

The first element of the sub-array, that at location INDX in the referenced waveform record, can be displayed as a scalar, or the entire subarray (of length NELM) can be displayed in the same way as a waveform record. If there are fewer than NELM elements in the referenced waveform after the INDX, only the number of elements actually available are returned, and the number of elements read field (NORD) is set to reflect this. This record type does not support writing new values into waveform records.

Parameter Fields

The record-specific fields are described below, grouped by functionality.

Scan Parameters

The subArray record has the standard fields for specifying under what circumstances the record will be processed. These fields are listed in *Scan Fields*.

Read Parameters

The subArray's input link (INP) should be configured to reference the Waveform record. It should specify the VAL field of a Waveform record. The INP field can be a channel access link, in addition to a database link.

In addition, the DTYP field must specify a device support module. Currently, the only device support module is `Soft Channel`.

Field	Summary	Type	DCT	Default	Read	Write	CA PP
INP	Input Specification	INLINK	Yes		Yes	Yes	No
DTYP	Device Type	DEVICE	Yes		Yes	Yes	No

Array Parameters

These parameters determine the number of array elements (the array length) and the data type of those elements. The Field Type of Value (FTVL) field determines the data type of the array.

The user specifies the maximum number of elements that can be read into the subarray in the MALM field. This number should normally be equal to the number of elements of the Waveform array (found in the Waveform's NELM field). The MALM field is used to allocate memory. The subArray's Number of Elements (NELM) field is where the user specifies the actual number of elements that the subArray will extract. It should of course be no greater than MALM; if it is, the record processing routine sets it equal to MALM.

The INDX field determines the offset of the subArray record's array in relation to the Waveform's. For instance, if INDX is 2, then the subArray will read NELM elements starting with the third element of the Waveform's array. Thus, it equals the index number of the Waveform's array.

The actual sub-array is referenced by the VAL field.

Field	Summary	Type	DCT	Default	Read	Write	CA PP
FTVL	Field Type of Value	MENU menuFtype	Yes		Yes	No	No
VAL	Value	Set by FTVL	No		Yes	Yes	Yes
MALM	Maximum Elements	ULONG	Yes	1	Yes	No	No
NELM	Number of Elements	ULONG	Yes	1	Yes	Yes	Yes
INDX	Substring Index	ULONG	Yes		Yes	Yes	Yes

Operator Display Parameters

These parameters are used to present meaningful data to the operator. They display the value and other parameters of the subarray record either textually or graphically.

EGU is a string of up to 16 characters describing the engineering units (if any) of the values which the subArray holds. It is retrieved by the `get_units()` record support routine.

The HOPR and LOPR fields set the upper and lower display limits for the sub-array elements. Both the `get_graphic_double()` and `get_control_double()` record support routines retrieve these fields.

The PREC field determines the floating point precision with which to display VAL. It is used whenever the `get_precision()` record support routine is called.

See *Fields Common to All Record Types* for more on the record name (NAME) and description (DESC) fields.

Field	Summary	Type	DCT	Default	Read	Write	CA PP
EGU	Engineering Units	STRING [16]	Yes		Yes	Yes	No
HOPR	High Operating Range	DOUBLE	Yes		Yes	Yes	No
LOPR	Low Operating Range	DOUBLE	Yes		Yes	Yes	No
PREC	Display Precision	SHORT	Yes		Yes	Yes	No
NAME	Record Name	STRING [61]	No		Yes	No	No
DESC	Descriptor	STRING [41]	Yes		Yes	Yes	No

Alarm Parameters

The subarray record has the alarm parameters common to all record types. *Alarm Fields* lists the fields related to alarms that are common to all record types.

Run-time Parameters

These fields are not configurable by the user. They are used for the record's internal processing or to represent the current state of the record.

The NORD field holds the number of elements that were actually read into the array. It will be less than NELM whenever the sum of the NELM and INDX fields exceeds the number of existing elements found in the source array.

BPTR contains a pointer to the record's array.

Field	Summary	Type	DCT	Default	Read	Write	CA PP
NORD	Number elements read	LONG	No		Yes	No	No
BPTR	Buffer Pointer	NOACCESS	No		No	No	No

Record Support

Record Support Routines

init_record

```
long (*init_record)(struct dbCommon *precord, int pass)
```

Using MALM and FTVL, space for the array is allocated. The array address is stored in BPTR. This routine checks to see that device support is available and a device support read routine is defined. If either does not exist, an error message is issued and processing is terminated. If device support includes `init_record()`, it is called.

process

```
long (*process)(struct dbCommon *precord)
```

See *"Record Processing"*.

cvt_dbaddr

```
long (*cvt_dbaddr)(struct dbAddr *paddr)
```

This is called by `dbNameToAddr()`. It makes the dbAddr structure refer to the actual buffer holding the result.

get_array_info

```
long (*get_array_info)(struct dbAddr *paddr, long *no_elements, long *offset)
```

Retrieves NORD.

put_array_info

```
long (*put_array_info)(struct dbAddr *paddr, long nNew)
```

Sets NORD.

get_graphic_double

```
long (*get_graphic_double)(struct dbAddr *paddr, struct dbr_grDouble *p)
```

For the elements in the array, this routine routines HOPR and LOPR. For the INDX field, this routine returns MALM - 1 and 0. For NELM, it returns MALM and 1. For other fields, it calls `recGblGetGraphicDouble()`.

get_control_double

```
long (*get_control_double)(struct dbAddr *paddr, struct dbr_ctrlDouble *p)
```

For array elements, this routine retrieves HOPR and LOPR. Otherwise, `recGblGetControlDouble()` is called.

get_units

```
long (*get_units)(struct dbAddr *paddr, char *units)
```

Retrieves EGU.

get_precision

```
long (*get_precision)(const struct dbAddr *paddr, long *precision)
```

Retrieves PREC.

Record Processing

Routine process implements the following algorithm:

1. Check to see that the appropriate device support module exists. If it doesn't, an error message is issued and processing is terminated with the PACT field still set to TRUE. This ensures that processes will no longer be called for this record. Thus error storms will not occur.
2. Sanity check NELM and INDX. If NELM is greater than MALM it is set to MALM. If INDX is greater than or equal to MALM it is set to MALM-1.
3. Call the device support's `read_sa()` routine. This routine is expected to place the desired sub-array at the beginning of the buffer and set NORD to the number of elements of the sub-array that were read.
4. If PACT has been changed to TRUE, the device support read operation has started but has not completed writing the new value. In this case, the processing routine merely returns, leaving PACT TRUE. Otherwise, process sets PACT TRUE at this time. This asynchronous processing logic is not currently used but has been left in place.
5. Check to see if monitors should be invoked.
   - Alarm monitors are invoked if the alarm status or severity has changed.
   - Archive and value change monitors are always invoked.
   - NSEV and NSTA are reset to 0.
6. Scan forward link if necessary, set PACT FALSE, and return.

Device Support

Fields Of Interest To Device Support

The device support routines are primarily interested in the following fields:

Field	Summary	Type	DCT	Default	Read	Write	CA PP
PACT	Record active	UCHAR	No		Yes	No	No
DPVT	Device Private	NOACCESS	No		No	No	No
UDF	Undefined	UCHAR	Yes	1	Yes	Yes	Yes
NSEV	New Alarm Severity	MENU menuAlarmSevr	No		Yes	No	No
NSTA	New Alarm Status	MENU menuAlarmStat	No		Yes	No	No
INP	Input Specification	INLINK	Yes		Yes	Yes	No
FTVL	Field Type of Value	MENU menuFtype	Yes		Yes	No	No
MALM	Maximum Elements	ULONG	Yes	1	Yes	No	No
NELM	Number of Elements	ULONG	Yes	1	Yes	Yes	Yes
INDX	Substring Index	ULONG	Yes		Yes	Yes	Yes
BPTR	Buffer Pointer	NOACCESS	No		No	No	No
NORD	Number elements read	LONG	No		Yes	No	No

Device Support Routines (devSASoft.c)

Device support consists of the following routines:

long report(int level)

This optional routine is called by the IOC command `dbior` and is passed the report level that was requested by the user. It should print a report on the state of the device support to stdout. The `level` parameter may be used to output increasingly more detailed information at higher levels, or to select different types of information with different levels. Level zero should print no more than a small summary.

long init(int after)

This optional routine is called twice at IOC initialization time. The first call happens before any of the `init_record()` calls are made, with the integer parameter `after` set to 0. The second call happens after all of the `init_record()` calls have been made, with `after` set to 1.

init_record

```
long init_record(subArrayRecord *prec)
```

This routine is called by the record support `init_record()` routine.

read_sa

```
long read_sa(subArrayRecord *prec)
```

Enough of the source waveform is read into BPTR, from the beginning of the source, to include the requested sub-array. The sub-array is then copied to the beginning of the buffer. NORD is set to indicate how many elements of the sub-array were acquired.

Device Support For Soft Records

Only the device support module `Soft Channel` is currently provided.

Soft Channel

INP is expected to point to an array field of a waveform record or similar.

1.5.33 Subroutine Record (sub)

The subroutine record is used to call a C initialization routine and a recurring scan routine. There is no device support for this record.

Parameter Fields

The record-specific fields are described below, grouped by functionality.

Scan Parameters

The subroutine record has the standard fields for specifying under what circumstances it will be processed. These fields are described in *Scan Fields*.

Read Parameters

The subroutine record has twelve input links (INPA-INPL), each of which has a corresponding value field (A-L). These fields are used to retrieve and store values that can be passed to the subroutine that the record calls.

The input links can be either channel access or database links, or constants. When constants, the corresponding value field for the link is initialized with the constant value and the field's value can be changed at run-time via dbPuts. Otherwise, the values for (A-F) are fetched from the input links when the record is processed.

Field	Summary	Type	DCT	Default	Read	Write	CA PP
INPA	Input A	INLINK	Yes		Yes	Yes	No
INPB	Input B	INLINK	Yes		Yes	Yes	No
INPC	Input C	INLINK	Yes		Yes	Yes	No
INPD	Input D	INLINK	Yes		Yes	Yes	No
INPE	Input E	INLINK	Yes		Yes	Yes	No
INPF	Input F	INLINK	Yes		Yes	Yes	No
INPG	Input G	INLINK	Yes		Yes	Yes	No
INPH	Input H	INLINK	Yes		Yes	Yes	No
INPI	Input I	INLINK	Yes		Yes	Yes	No
INPJ	Input J	INLINK	Yes		Yes	Yes	No
INPK	Input K	INLINK	Yes		Yes	Yes	No
INPL	Input L	INLINK	Yes		Yes	Yes	No
A	Value of Input A	DOUBLE	No		Yes	Yes	Yes
B	Value of Input B	DOUBLE	No		Yes	Yes	Yes
C	Value of Input C	DOUBLE	No		Yes	Yes	Yes
D	Value of Input D	DOUBLE	No		Yes	Yes	Yes
E	Value of Input E	DOUBLE	No		Yes	Yes	Yes
F	Value of Input F	DOUBLE	No		Yes	Yes	Yes
G	Value of Input G	DOUBLE	No		Yes	Yes	Yes
H	Value of Input H	DOUBLE	No		Yes	Yes	Yes
I	Value of Input I	DOUBLE	No		Yes	Yes	Yes
J	Value of Input J	DOUBLE	No		Yes	Yes	Yes
K	Value of Input K	DOUBLE	No		Yes	Yes	Yes
L	Value of Input L	DOUBLE	No		Yes	Yes	Yes

Subroutine Connection

These fields are used to connect to the C subroutine. The name of the subroutine should be entered in the SNAM field.

Field	Summary	Type	DCT	Default	Read	Write	CA PP
INAM	Init Routine Name	STRING [40]	Yes		Yes	No	No
SNAM	Subroutine Name	STRING [40]	Yes		Yes	Yes	No

Operator Display Parameters

These parameters are used to present meaningful data to the operator. They display the value and other parameters of the subroutine either textually or graphically.

EGU is a string of up to 16 characters that could describe any units used by the subroutine record. It is retrieved by the `get_units` record support routine.

The HOPR and LOPR fields set the upper and lower display limits for the VAL, A-L, LA-LL, HIHI, LOLO, LOW, and HIGH fields. Both the `get_graphic_double` and `get_control_double` record support routines retrieve these fields.

The PREC field determines the floating point precision with which to display VAL. It is used whenever the `get_precision` record support routine is called.

See *Fields Common to All Record Types* for more on the record name (NAME) and description (DESC) fields.

Field	Summary	Type	DCT	Default	Read	Write	CA PP
EGU	Engineering Units	STRING [16]	Yes		Yes	Yes	No
HOPR	High Operating Range	DOUBLE	Yes		Yes	Yes	No
LOPR	Low Operating Range	DOUBLE	Yes		Yes	Yes	No
PREC	Display Precision	SHORT	Yes		Yes	Yes	No
NAME	Record Name	STRING [61]	No		Yes	No	No
DESC	Descriptor	STRING [41]	Yes		Yes	Yes	No

Alarm Parameters

The possible alarm conditions for subroutine records are the SCAN, READ, limit alarms, and an alarm that can be triggered if the subroutine returns a negative value. The SCAN and READ alarms are called by the record or device support routines. The limit alarms are configured by the user in the HIHI, LOLO, HIGH, and LOW fields using numerical values. They apply to the VAL field. For each of these fields, there is a corresponding severity field which can be either NO_ALARM, MINOR, or MAJOR.

The BRSV field is where the user can set the alarm severity in case the subroutine returns a negative value.

Alarm Fields lists the fields related to alarms that are common to all record types.

Field	Summary	Type	DCT	Default	Read	Write	CA PP
HIHI	Hihi Alarm Limit	DOUBLE	Yes		Yes	Yes	Yes
HIGH	High Alarm Limit	DOUBLE	Yes		Yes	Yes	Yes
LOW	Low Alarm Limit	DOUBLE	Yes		Yes	Yes	Yes
LOLO	Lolo Alarm Limit	DOUBLE	Yes		Yes	Yes	Yes
HHSV	Hihi Severity	MENU menuAlarmSevr	Yes		Yes	Yes	Yes
HSV	High Severity	MENU menuAlarmSevr	Yes		Yes	Yes	Yes
LSV	Low Severity	MENU menuAlarmSevr	Yes		Yes	Yes	Yes
LLSV	Lolo Severity	MENU menuAlarmSevr	Yes		Yes	Yes	Yes
BRSV	Bad Return Severity	MENU menuAlarmSevr	Yes		Yes	Yes	Yes
HYST	Alarm Deadband	DOUBLE	Yes		Yes	Yes	No

Monitor Parameters

These parameters are used to determine when to send monitors placed on the VAL field. The appropriate monitors are invoked when VAL differs from the values in the ALST and MLST run-time fields, i.e., when the value of VAL changes by more than the deadband specified in these fields. The ADEL and MDEL fields specify a minimum delta which the change must surpass before the value-change monitors are invoked. If these fields have a value of zero, everytime the value changes, a monitor will be triggered; if they have a value of -1, everytime the record is processed, monitors are triggered. The ADEL field is used by archive monitors and the MDEL field for all other types of monitors.

Field	Summary	Type	DCT	Default	Read	Write	CA PP
ADEL	Archive Deadband	DOUBLE	Yes		Yes	Yes	No
MDEL	Monitor Deadband	DOUBLE	Yes		Yes	Yes	No

Run-time Parameters

These parameters are used by the run-time code for processing the subroutine record. They are not configured using a database configuration tool. They represent the current state of the record. Many of them are used by the record processing routines or the monitors.

VAL should be set by the subroutine. SADR holds the subroutine address and is set by the record processing routine.

The rest of these fields–LALM, ALST, MLST, and the LA-LL fields–are used to implement the monitors. For example, when LA is not equal to A, the value-change monitors are called for that field.

Field	Summary	Type	DCT	Default	Read	Write	CA PP
VAL	Result	DOUBLE	No		Yes	Yes	Yes
SADR	Subroutine Address	NOACCESS	No		No	No	No
LALM	Last Value Alarmed	DOUBLE	No		Yes	No	No
ALST	Last Value Archived	DOUBLE	No		Yes	No	No
MLST	Last Value Monitored	DOUBLE	No		Yes	No	No
LA	Prev Value of A	DOUBLE	No		Yes	No	No
LB	Prev Value of B	DOUBLE	No		Yes	No	No
LC	Prev Value of C	DOUBLE	No		Yes	No	No
LD	Prev Value of D	DOUBLE	No		Yes	No	No
LE	Prev Value of E	DOUBLE	No		Yes	No	No
LF	Prev Value of F	DOUBLE	No		Yes	No	No
LG	Prev Value of G	DOUBLE	No		Yes	No	No
LH	Prev Value of H	DOUBLE	No		Yes	No	No
LI	Prev Value of I	DOUBLE	No		Yes	No	No
LJ	Prev Value of J	DOUBLE	No		Yes	No	No
LK	Prev Value of K	DOUBLE	No		Yes	No	No
LL	Prev Value of L	DOUBLE	No		Yes	No	No

Record Support

Record Support Routines

init_record

```
long (*init_record)(struct dbCommon *precord, int pass)
```

For each constant input link, the corresponding value field is initialized with the constant value. For each input link that is of type PV_LINK, a channel access link is created.

If an initialization subroutine is defined, it is located and called.

The processing subroutine is located and its address stored in SADR.

process

```
long (*process)(struct dbCommon *precord)
```

See *"Record Processing"*.

get_units

```
long (*get_units)(struct dbAddr *paddr, char *units)
```

Retrieves EGU.

get_precision

```
long (*get_precision)(const struct dbAddr *paddr, long *precision)
```

Retrieves PREC when VAL is the field being referenced. Otherwise, calls `recGblGetPrec()`.

get_graphic_double

```
long (*get_graphic_double)(struct dbAddr *paddr, struct dbr_grDouble *p)
```

Sets the upper display and lower display limits for a field. If the field is VAL, A-L, LA-LL, HIHI, HIGH, LOW, or LOLO, the limits are set to HOPR and LOPR, else if the field has upper and lower limits defined they will be used, else the upper and lower maximum values for the field type will be used.

get_control_double

```
long (*get_control_double)(struct dbAddr *paddr, struct dbr_ctrlDouble *p)
```

Sets the upper control and the lower control limits for a field. If the field is VAL, A-L, LA-LL, HIHI, HIGH, LOW, or LOLO, the limits are set to HOPR and LOPR, else if the field has upper and lower limits defined they will be used, else the upper and lower maximum values for the field type will be used.

get_alarm_double

```
long (*get_alarm_double)(struct dbAddr *paddr, struct dbr_alDouble *p)
```

Sets the following values:

```
upper_alarm_limit = HIHI
upper_warning_limit = HIGH
lower_warning_limit = LOW
lower_alarm_limit = LOLO
```

Record Processing

Routine process implements the following algorithm:

1. If PACT is FALSE then fetch all arguments.
2. Call the subroutine and check return value.
   - Call subroutine
   - Set PACT TRUE
   - If return value is 1, return
3. Check alarms. This routine checks to see if the new VAL causes the alarm status and severity to change. If so, NSEV, NSTA and LALM are set. It also honors the alarm hysteresis factor (HYST). Thus the value must change by more than HYST before the alarm status and severity is lowered.
4. Check to see if monitors should be invoked.
   - Alarm monitors are invoked if the alarm status or severity has changed.
   - Archive and value change monitors are invoked if ADEL and MDEL conditions are met.
   - Monitors for A-L are invoked if value has changed.
   - NSEV and NSTA are reset to 0.
5. Scan forward link if necessary, set PACT FALSE, and return.

Example Synchronous Subroutine

This is an example subroutine that merely increments VAL each time process is called.

```
#include <stdio.h>
#include <dbDefs.h>
#include <subRecord.h>
#include <registryFunction.h>
#include <epicsExport.h>

static long subInit(struct subRecord *psub)
{
    printf("subInit was called\n");
    return 0;
}

static long subProcess(struct subRecord *psub)
{
    psub->val++;
    return 0;
}

epicsRegisterFunction(subInit);
epicsRegisterFunction(subProcess);
```

Example Asynchronous Subroutine

This example for a VxWorks IOC shows an asynchronous subroutine. It uses (actually misuses) fields A and B. Field A is taken as the number of seconds until asynchronous completion. Field B is a flag to decide if messages should be printed. Lets assume A > 0 and B = 1. The following sequence of actions will occcur:

1. subProcess is called with pact FALSE. It performs the following steps.
   - Computes, from A, the number of ticks until asynchronous completion should occur.
   - Prints a message stating that it is requesting an asynchronous callback.
   - Calls the vxWorks watchdog start routine.
   - Sets pact TRUE and returns a value of 0. This tells record support to complete without checking alarms, monitors, or the forward link.
2. When the time expires, the system wide callback task calls myCallback. myCallback locks the record, calls process, and unlocks the record.
3. Process again calls subProcess, but now pact is TRUE. Thus the following is done:
   - VAL is incremented.
   - A completion message is printed.
   - subProcess returns 0. The record processing routine will complete record processing.

   #include <types.h> #include <stdio.h> #include <wdLib.h> #include <callback.h> #include <dbDefs.h> #include <dbAccess.h> #include <subRecord.h>

   /* control block for callback*/ struct callback { epicsCallback callback; struct dbCommon *precord; WDOG_ID wd_id; };

   void myCallback(struct callback *pcallback) { struct dbCommon *precord=pcallback->precord; struct rset *prset=(struct rset *)(precord->rset); dbScanLock(precord); (*prset->process)(precord); dbScanUnlock(precord); }

   long subInit(struct subRecord *psub) { struct callback *pcallback; pcallback = (struct callback *)(calloc(1,sizeof(struct callback))); psub->dpvt = (void *)pcallback; callbackSetCallback(myCallback,pcallback); pcallback->precord = (struct dbCommon *)psub; pcallback->wd_id = wdCreate(); printf("subInit was called\n"); return 0; }

   long subProcess(struct subRecord *psub) { struct callback *pcallback=(struct callback*)(psub->dpvt); / *sub.inp must be a CONSTANT*/ if (psub->pact) { psub->val++; if (psub->b) printf("%s subProcess Completed\n", psub->name); return 0; } else { int wait_time = (long)(psub->a * vxTicksPerSecond); if (wait_time <= 0){ if (psub->b) printf("%s subProcess sync processing\n", psub->name); psub->pact = TRUE; return 0; } if (psub->b){ callbackSetPriority(psub->prio, pcallback); printf("%s Starting async processing\n", psub->name); wdStart(pcallback->wd_id, wait_time, callbackRequest, (int)pcallback); return 1; } } return 0; }

1.5.34 Waveform Record (waveform)

The waveform record type is used to interface waveform digitizers. The record stores its data in arrays. The array can contain any of the supported data types.

Parameter Fields

The record-specific fields are described below, grouped by functionality.

Scan Parameters

The waveform record has the standard fields for specifying under what circumstances the record will be processed. These fields are listed in *Scan Fields*.

Read Parameters

These fields are configurable by the user to specify how and from where the record reads its data. How the INP field is configured determines where the waveform gets its input. It can be a hardware address, a channel access or database link, or a constant. Only in records that use soft device support can the INP field be a channel access link, a database link, or a constant. Otherwise, the INP field must be a hardware address.

Fields related to waveform reading

Field	Summary	Type	DCT	Default	Read	Write	CA PP
DTYP	Device Type	DEVICE	Yes		Yes	Yes	No
INP	Input Specification	INLINK	Yes		Yes	Yes	No
NELM	Number of Elements	ULONG	Yes	1	Yes	No	No
FTVL	Field Type of Value	MENU menuFtype	Yes		Yes	No	No
RARM	Rearm the waveform	SHORT	Yes		Yes	Yes	Yes

The DTYP field must contain the name of the appropriate device support module. The values retrieved from the input link are placed in an array referenced by VAL. (If the INP link is a constant, elements can be placed in the array via dbPuts.) NELM specifies the number of elements that the array will hold, while FTVL specifies the data type of the elements (follow the link in the table above for a list of the available choices).

The RARM field used to cause some device types to re-arm when it was set to 1, but we don't know of any such devices any more.

Operator Display Parameters

These parameters are used to present meaningful data to the operator. They display the value and other parameters of the waveform either textually or graphically.

Fields related to *Operator Display*

Field	Summary	Type	DCT	Default	Read	Write	CA PP
EGU	Engineering Units	STRING [16]	Yes		Yes	Yes	No
HOPR	High Operating Range	DOUBLE	Yes		Yes	Yes	No
LOPR	Low Operating Range	DOUBLE	Yes		Yes	Yes	No
PREC	Display Precision	SHORT	Yes		Yes	Yes	No
NAME	Record Name	STRING [61]	No		Yes	No	No
DESC	Descriptor	STRING [41]	Yes		Yes	Yes	No

EGU is a string of up to 16 characters describing the units that the waveform measures. It is retrieved by the `get_units` record support routine.

The HOPR and LOPR fields set the upper and lower display limits for array elements referenced by the VAL field. Both the `get_graphic_double` and `get_control_double` record support routines retrieve these fields.

The PREC field determines the floating point precision with which to display the array values. It is used whenever the `get_precision` record support routine is called.

See *Fields Common to All Record Types* for more on the record name (NAME) and description (DESC) fields.

Alarm Parameters

The waveform record has the alarm parameters common to all record types. *Alarm Fields* lists the fields related to alarms that are common to all record types.

Monitor Parameters

These parameters are used to determine when to send monitors placed on the VAL field. The APST and MPST fields are a menu with choices `Always` and `On Change`. The default is `Always`, thus monitors will normally be sent every time the record processes. Selecting `On Change` causes a 32-bit hash of the VAL field buffer to be calculated and compared with the previous hash value every time the record processes; the monitor will only be sent if the hash is different, indicating that the buffer has changed. Note that there is a small chance that two different value buffers might result in the same hash value, so for critical systems `Always` may be a better choice, even though it re-sends duplicate data.

Field	Summary	Type	DCT	Default	Read	Write	CA PP
APST	Post Archive Monitors	MENU *waveformPOST*	Yes		Yes	Yes	No
MPST	Post Value Monitors	MENU *waveformPOST*	Yes		Yes	Yes	No
HASH	Hash of OnChange data.	ULONG	No		Yes	Yes	No

Menu waveformPOST

This menu defines the possible choices for `APST` and `MPST` fields:

Index	Identifier	Choice String
0	waveformPOST_Always	Always
1	waveformPOST_OnChange	On Change

Run-time Parameters

These parameters are used by the run-time code for processing the waveform. They are not configured using a configuration tool. Only the VAL field is modifiable at run-time.

VAL references the array where the waveform stores its data. The BPTR field holds the address of the array.

The NORD field indicates the number of elements that were read into the array.

The BUSY field permits asynchronous device support to collect array elements sequentially in multiple read cycles which may call the record's `process()` method many times before completing a read operation. Such a device would set BUSY to TRUE along with setting PACT at the start of acquisition (it could also set NORD to 0 and use it to keep track of how many elements have been received). After receiving the last element the `read_wf()` routine would clear BUSY which informs the record's `process()` method that the read has finished. Note that CA clients that perform gets of the VAL field can see partially filled arrays when this type of device support is used, so the BUSY field is almost never used today.

Field	Summary	Type	DCT	Default	Read	Write	CA PP
VAL	Value	Set by FTVL	No		Yes	Yes	Yes
BPTR	Buffer Pointer	NOACCESS	No		No	No	No
NORD	Number elements read	ULONG	No		Yes	No	No
BUSY	Busy Indicator	SHORT	No		Yes	No	No

Simulation Mode Parameters

The following fields are used to operate the record in simulation mode.

If SIMM (fetched through SIML) is YES, the record is put in SIMS severity and the value is fetched through SIOL. SSCN sets a different SCAN mechanism to use in simulation mode. SDLY sets a delay (in sec) that is used for asynchronous simulation processing.

See *Input Simulation Fields* for more information on simulation mode and its fields.

Field	Summary	Type	DCT	Default	Read	Write	CA PP
SIML	Simulation Mode Link	INLINK	Yes		Yes	Yes	No
SIMM	Simulation Mode	MENU menuYesNo	No		Yes	Yes	No
SIOL	Simulation Input Link	INLINK	Yes		Yes	Yes	No
SIMS	Simulation Mode Severity	MENU menuAlarmSevr	Yes		Yes	Yes	No
SDLY	Sim. Mode Async Delay	DOUBLE	Yes	-1.0	Yes	Yes	No
SSCN	Sim. Mode Scan	MENU menuScan	Yes	65535	Yes	Yes	No

Record Support

Record Support Routines

init_record

```
static long init_record(waveformRecord *prec, int pass)
```

Using NELM and FTVL space for the array is allocated. The array address is stored in the record.

This routine initializes SIMM with the value of SIML if SIML type is CONSTANT link or creates a channel access link if SIML type is PV_LINK. VAL is likewise initialized if SIOL is CONSTANT or PV_LINK.

This routine next checks to see that device support is available and a device support read routine is defined. If either does not exist, an error message is issued and processing is terminated

If device support includes `init_record()`, it is called.

process

```
static long process(waveformRecord *prec)
```

See *"Record Processing"* section below.

cvt_dbaddr

```
static long cvt_dbaddr(DBADDR *paddr)
```

This is called by dbNameToAddr. It makes the dbAddr structure refer to the actual buffer holding the result.

get_array_info

```
static long get_array_info(DBADDR *paddr, long *no_elements, long *offset)
```

Obtains values from the array referenced by VAL.

put_array_info

```
static long put_array_info(DBADDR *paddr, long nNew)
```

Writes values into the array referenced by VAL.

get_units

```
static long get_units(DBADDR *paddr, char *units)
```

Retrieves EGU.

get_prec

```
static long get_precision(DBADDR *paddr, long *precision)
```

Retrieves PREC if field is VAL field. Otherwise, calls `recGblGetPrec()`.

get_graphic_double

```
static long get_graphic_double(DBADDR *paddr, struct dbr_grDouble *pgd)
```

Sets the upper display and lower display limits for a field. If the field is VAL the limits are set to HOPR and LOPR, else if the field has upper and lower limits defined they will be used, else the upper and lower maximum values for the field type will be used.

Sets the following values:

```
upper_disp_limit = HOPR
lower_disp_limit = LOPR
```

get_control_double

```
static long get_control_double(DBADDR *paddr, struct dbr_ctrlDouble *pcd)
```

Sets the upper control and the lower control limits for a field. If the field is VAL the limits are set to HOPR and LOPR, else if the field has upper and lower limits defined they will be used, else the upper and lower maximum values for the field type will be used.

Sets the following values

```
upper_ctrl_limit = HOPR
lower_ctrl_limit = LOPR
```

Record Processing

Routine process implements the following algorithm:

1. Check to see that the appropriate device support module exists. If it doesn't, an error message is issued and processing is terminated with the PACT field still set to TRUE. This ensures that processes will no longer be called for this record. Thus error storms will not occur.
2. Call device support read routine.
3. If PACT has been changed to TRUE, the device support read routine has started but has not completed writing the new value. In this case, the processing routine merely returns, leaving PACT TRUE.
4. Check to see if monitors should be invoked.
    - Alarm monitors are invoked if the alarm status or severity has changed.
    - Archive and value change monitors are invoked if APST or MPST are Always or if the result of the hash calculation is different.
    - NSEV and NSTA are reset to 0.
5. Scan forward link if necessary, set PACT FALSE, and return.

Device Support

Fields Of Interest To Device Support

Each waveform record must have an associated set of device support routines. The primary responsibility of the device support routines is to obtain a new array value whenever read_wf is called. The device support routines are primarily interested in the following fields:

Field	Summary	Type	DCT	Default	Read	Write	CA PP
PACT	Record active	UCHAR	No		Yes	No	No
DPVT	Device Private	NOACCESS	No		No	No	No
NSEV	New Alarm Severity	MENU menuAlarmSevr	No		Yes	No	No
NSTA	New Alarm Status	MENU menuAlarmStat	No		Yes	No	No
INP	Input Specification	INLINK	Yes		Yes	Yes	No
NELM	Number of Elements	ULONG	Yes	1	Yes	No	No
FTVL	Field Type of Value	MENU menuFtype	Yes		Yes	No	No
RARM	Rearm the waveform	SHORT	Yes		Yes	Yes	Yes
BPTR	Buffer Pointer	NOACCESS	No		No	No	No
NORD	Number elements read	ULONG	No		Yes	No	No
BUSY	Busy Indicator	SHORT	No		Yes	No	No

Device Support Routines

Device support consists of the following routines:

report

```
long report(int level)
```

This optional routine is called by the IOC command `dbior` and is passed the report level that was requested by the user. It should print a report on the state of the device support to stdout. The `level` parameter may be used to output increasingly more detailed information at higher levels, or to select different types of information with different levels. Level zero should print no more than a small summary.

init

```
long init(int after)
```

This optional routine is called twice at IOC initialization time. The first call happens before any of the `init_record()` calls are made, with the integer parameter `after` set to 0. The second call happens after all of the `init_record()` calls have been made, with `after` set to 1.

init_record

```
long init_record(dbCommon *precord)
```

This routine is optional. If provided, it is called by the record support `init_record()` routine.

get_ioint_info

```
long get_ioint_info(int cmd, dbCommon *precord, IOSCANPVT *ppvt)
```

This routine is called by the ioEventScan system each time the record is added or deleted from an I/O event scan list. `cmd` has the value (0,1) if the record is being (added to, deleted from) an I/O event list. It must be provided for any device type that can use the ioEvent scanner.

read_wf

```
long read_wf(waveformRecord *prec)
```

This routine must provide a new input value. It returns the following values:

- 0: Success.
- Other: Error.

Device Support For Soft Records

The `Soft Channel` device support module is provided to read values from other records and store them in the VAL field. If INP is a constant link, then `read_wf()` does nothing. In this case, the record can be used to hold a fixed set of data or array values written from elsewhere. If INP is a valid link, the new array value is read from that link. NORD is set to the number of items received.

If the INP link type is constant, VAL is set from it in the `init_record()` routine and NORD is also set at that time.

CHAPTER

TWO

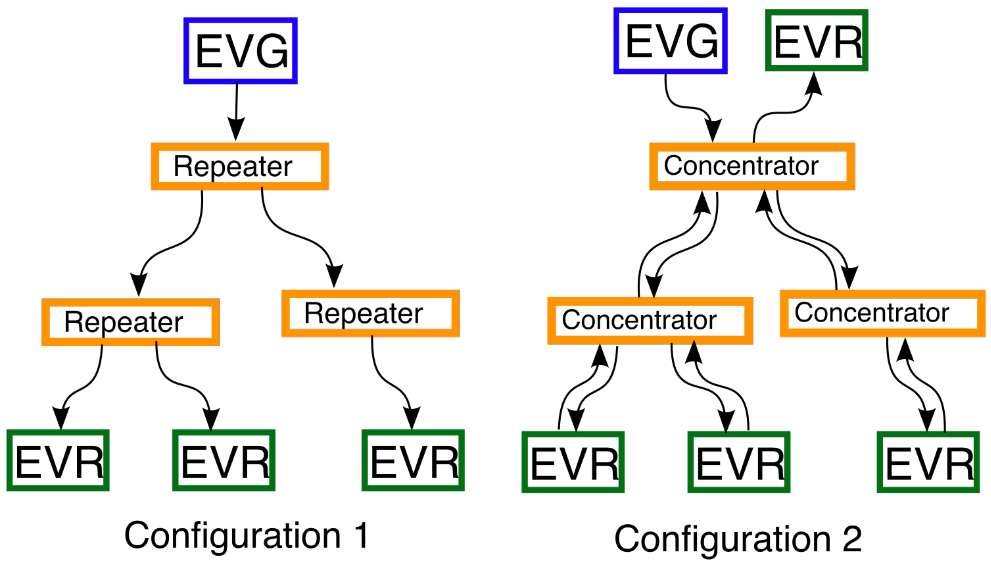
MRF TIMING SYSTEM REFERENCE

2.1 The MRF Timing System

The MRF Event System provides a complete timing distribution system including timing signal generation with only a few components. The system is capable of generating subharmonic frequency signals, triggers and sequences of events, etc. that are synchronous to an externally provided master clock reference and (optionally) another signal, for example mains voltage phase. Support for timestamps makes the system a global timebase and allows attaching timestamps to collected data and performed actions.

2.1.1 Timing System Principle of Operation

A basic setup of the timing system consists of an Event Generator (EVG), the distribution layer (Fan-Out, or Repeater/Concentrator) and Event Receivers (EVR). See the picture below.



In the basic use pattern (Configuration 1), the event stream is unidirectional, generated by the EVG and then multiplied using repeaters to a number of event receivers. Synchronicity is preserved in the distribution layer. Finally, the EVRs lock to the bitstream signal phase and thus are precisely synchronised to the bitstream, and consequently to each other with a high precision.

There is also the possibility for bi-directional signaling (Configuration 2), where EVRs can not only receive the event stream but also generate and send an event stream which will be forwarded "upstream" to the EVG via the distribution layer. In the upwards direction, the distribution layer nodes act as concentrators, multiplexing the streams from below into one stream going upwards. Obviously, as upstream events go through the concentrators, full determinism cannot be guaranteed; phase synchronicity with the master clock is preserved though.

The event system functions by transmitting a bit stream, called here event stream, between the system components. The event stream is a continuous flow of 16-bit data frames, generated and sent out by the event generator and sent out at the "event clock" rate. The event clock rate is derived from an externally generated RF signal or optionally an on-board clock generator. The event stream is phase locked to the clock reference.

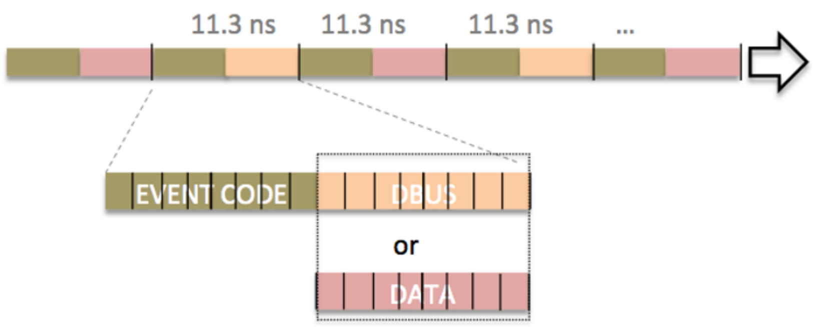
The physical media for transmission is optical fiber. Standard networking components (SFP modules, multi- or single mode fiber) are used to build the network. The network is fully dedicated to timing traffic, i.e., there is no need for bandwidth sharing. This makes it possible for the system to react to asynchronous events without jitter, or the having the need to schedule operations in advance.

Note: In earlier versions, to use the capabilities for bi-directional messaging, an EVR (hardware) was required to reside in the same system as the Event Generator. In the 300-series (with *delay compensation*), the Event Master (EVM) module contains two simplified EVRs, implemented in firmware, in addition to the EVG. In this case, a single EVM is sufficient for most purposes.

Event Stream

The event stream protocol is based on 8b10b encoded characters, which means that the actual bit rate is higher than the number of bits in the event frame. Ten bits are transmitted on the link for each 8-bit byte.

Each frame of the stream consists of two bytes. The first byte is dedicated for transmitting *timing events*, and always contains an *event code*. The second byte can be configured for use in two different ways, as *distributed bus* bits or *synchronous data transmission*. These will be explained in detail later.



(The above image shows a frame interval of 11.3 nanoseconds, corresponding to 88.0525 MHz event clock.)

Details about the event stream protocol can be found here.

2.1.2 Event Generator Overview

The Event Generator generates the event stream and sends it out to an array of Event Receivers.

The Event Generator has a number of functions:

- Generating and transmitting the *timing events*
- Transmitting the *Distributed Bus* bits
- Transmitting the *Synchronous Data* Buffer
- Acting as a *source for timestamps*.

Timing Events

Event codes can be understood as instructions to indicate that something has to happen and a corresponding action needs to be taken. The actions can be defined by the user.

For example, in the context of an accelerator, an event could be something like "send a beam pulse" and used to trigger a particle source to produce and feed a pulse to the accelerator. The 8-bit event codes can be configured by the user to have different meanings. The "send a beam pulse" event could be assigned the number 10, for example.

The event code will be inserted in the event stream and distributed via the distribution layer to several event receivers (EVR).

On the receiving side, the EVR can be configured to act in a number of ways when it receives the code. Possible actions can include:

- generating an hardware output (trigger) pulse, to trigger actions in some other components,
- generating a software interrupt, to trigger software actions
- actions related to managing timestamps
- a number of other possible actions, defined in the *Event Receiver documentation*.

Event Codes

A byte of 8 bits gives 256 different event codes. Actions for most of these codes can be freely configured by the user, however a few codes have special predefined functions. The special function event codes are listed in the table below.

Event Code	Code Name	EVG Function	EVR Function
0x00	Null Event Code	-	-
0x01–0x6F	-	User Defined	User Defined
0x70	Seconds '0'	-	Shift in '0' to LSB of Seconds Shift Register
0x71	Seconds '1'	-	Shift in '1' to LSB of Seconds Shift Register
0x72–0x78	-	User Defined	User Defined
0x79	Stop Event Log	-	-
0x7A	Heartbeat	-	-
0x7B	Synchronise Prescalers	-	Synchronise Prescaler Outputs
0x7C	Timestamp Counter Increment	-	Increment Timestamp Counter
0x7D	Timestamp Counter Reset	-	-
0x7E	Beacon event	-	-
0x7F	End of Sequence	-	-
0x80-FF	-	User Defined	User Defined

Table: Event Codes

Note: **Beacon events** are related to the active delay compensation and **shall not** be used in user applications.

The event codes are transmitted continuously. If there is no other event code to be transferred, the null event code (0x00) is transmitted. Every now and then a special 8B10B character K28.5 is transmitted instead of the null event code to allow the event receivers to synchronise on the correct word boundary on the serial bit stream.

Sources for timing events

Timing events can be generated from a number of different sources: physical input signals, from an internal event sequencer, software-generated events and events received from an upstream Event Generator.

Trigger Signal Inputs

There are eight trigger event inputs that can be configure to send out an event code on a stimulus. Each trigger event has its own programmable event code register and various enable bits.

Event Sequencer

Event sequencers provide a method of transmitting (or “playing back”) sequences of events stored in random access memory with defined timing. In the event generator there are two event sequencers. 8-bit event codes are stored in a RAM table together with a 32-bit time value (event address) relative to the start of sequence. Both sequencers can hold up to 2048 event code – time pairs.

The Sequencers may be triggered from several sources including software triggering, triggering on a multiplexed counter output or AC mains voltage synchronization logic output.

Uses for the sequencer

The event sequencers are typically used for sending out a precisely timed sequence of events, like an accelerator machine cycle. In this case, event codes that have been defined for different actions to be taken during the acceleration cycle are placed in the sequencer RAM together with the time interval between sending out the events. This is the most common use case for the sequencers. Typically the two sequencers are used in foreground/background combination, where the foreground sequencer is transmitting events, and the background sequencer can be prepared by software for the upcoming cycles. When the foreground sequencer finishes, the roles can be swapped and the backgound sequencer made active.

A more exotic use case for the sequencer could be sending out a complicated signal pattern, using the RAM contents in the recycle mode.

Software-generated Events

Events can be generated in software by writing into the Software Event register. This is useful for creating events that occur based on some higher-level conditions; for example an operator requesting a beam dump in a circular accelerator.

Upstream Events

Event Generators may be cascaded. The bitstream receiver in the event generator includes a first-in-first-out (FIFO) memory to synchronize incoming events which may be synchronized to a clock unrelated to the event clock.

Distributed Bus

The distributed bus allows transmission of eight simultaneous signals with half of the event clock rate time resolution (20 ns at 100 MHz event clock rate) from the event generator to the event receivers.

The distributed bus signals can be programmed to be available as hardware outputs on the event receiver.

Typical uses for the distributed bus are distributing hardware clock signals (that are slower that the event clock) or distributing status signals to a large number of receivers.

Timestamping support

The event system supports timestamping by providing:

- Facilities to distribute a seconds value to all receivers
- Facilities to support generation of sub-second timestamps, usually in the event clock resolution
- Precisely latching the timestamp in an Event Receiver, on request or when an event code has been received.

The timestamp support guarantees that all event receivers (and generators) in the same distribution setup will have precisely synchronized timestamps, up to the resolution of the event clock. For example, 100 MHz event clock results in highest timestamp resolution of 10 nanoseconds.

The seconds value to be distributed has to be provided to the Event Generator. This is typically sourced from an external GPS receiver.

Timestamp Generator

The model of time implemented by the MRF hardware is two 32-bit unsigned integers: counter, and “seconds”. The counter is maintained by each EVR and incremented quickly. The value of the “seconds” is sent periodically from the EVG at a lower rate.

Note that while it is referred to as “seconds” this value is an arbitrary integer that could have other meanings. Several methods of sending the seconds value to the EVG are possible:

External hardware

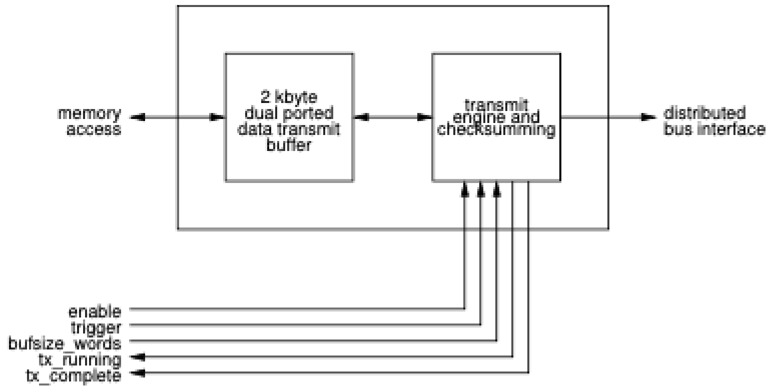
In this method, hardware is used to communicate with a GPS receiver over a serial (RS232) link to receive the timestamp and connect to two external inputs on the EVG. These inputs must be programmed to send the shift 0/1 codes.

Time from an NTP server

Time from a NTP server can be used without special hardware. This requires only a 1Hz (PPS) signal coming from the same source as the NTP time. Several commercial vendors supply dedicated NTP servers with builtin GPS receivers and 1Hz outputs. A software function is provided on the EVG which is triggered by the 1Hz signal. At the start of each second it sends the next second (current+1), which will be latched after the following 1Hz tick.

Synchronous Data buffer

A memory buffer of up to 2k bytes can be transmitted over the event link. This data buffer will be (synchronously) available to all EVRs that receive the event stream.

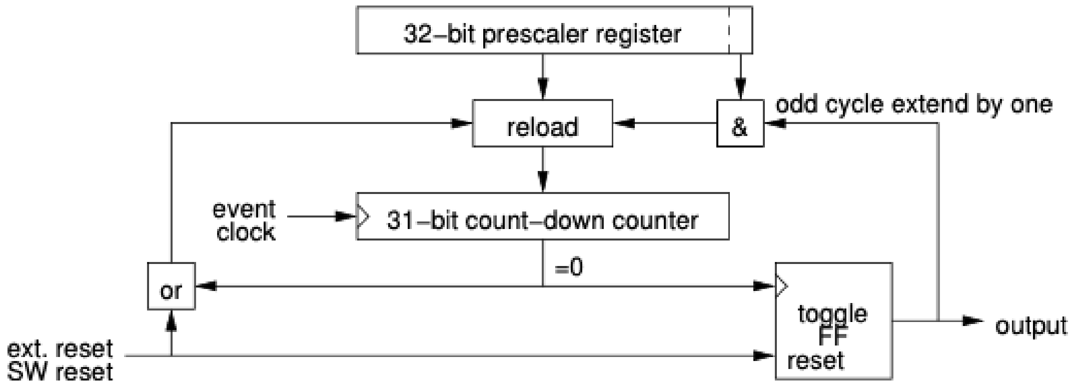

memory access
2 kbyte dual ported data transmit buffer
transmit engine and checksumming
distributed bus interface
enable
trigger
bufsize_words
tx_running
tx_complete

The data to be transmitted is stored in a 2 kbyte dual-ported memory starting from the lowest address 0. This memory is directly accessible via the memory interface in both generators and receivers.

Utility Functions in the Event Generator

Multiplexed Counters

Eight 32-bit multiplexed counters generate clock signals with programmable frequencies from event clock/2^{32}1 to event clock/2. Even divisors create 50% duty cycle signals. The counter outputs may be programmed to trigger events, drive distributed bus signals and trigger sequence RAMs. The output of multiplexed counter 7 is hard-wired to the mains voltage synchronization logic.



AC Line Synchronisation

The Event Generator provides synchronization to the mains voltage frequency or another external clock. The mains voltage frequency can be divided by an eight bit programmable divider. The output of the divider may be delayed by 0 to 25.5 ms by a phase shifter in 0.1 ms steps to be able to adjust the triggering position relative to mains voltage phase.

Event Clock RF Source

All operations on the event generator are synchronised to the event clock which is derived from an externally provided RF clock. For laboratory testing purposes an on-board fractional synthesiser may be used to deliver the event clock. The serial link bit rate is 20 times the event clock rate. The acceptable range for the event clock and bit rate is shown in the following table.

During operation the reference frequency should not be changed more than ±100 ppm.

	EventClock	BitRate
Minimum	50 MHz	1.0 Gb/s
Maximum	142.8 MHz	2.9 Gb/s

2.1.3 Event Receiver Overview

Event Receivers decode timing events and signals from an optical event stream transmitted by an Event Generator. Events and signals are received at the event clock rate. The event receivers lock to the phase of the upstream clock reference. Event Receivers convert event codes that are transmitted by an Event Generator to hardware outputs. They can also generate software interrupts and store the event codes with timestamps into FIFO memory to be read by a CPU.

Functional Description

After recovering the event clock the Event Receiver demultiplexes the event stream to 8-bit event codes and 8-bit distributed bus data. The distributed bus may be configured to share its bandwidth with time deterministic data transmission.

Event Decoding

Actions that the Event Receiver does when an event is received are configured by setting up *event mapping RAMs* in the EVR. The EVR provides two mapping RAMs of 256 × 128 bits each. The 128-bit data programmed into the corresponding memory location pointed to by the event code determines what actions will be taken.

Heartbeat Monitor

Heartbeat facility can be used to detect the loss of communication between the EVR and the EVG. A *heartbeat monitor* is provided to receive heartbeat events. Event code $7A is sent by the EVG periodically to reset the heartbeat counter. If no heartbeat event is received the counter times out (approx. 1.6 s) and a heartbeat flag is set. The Event Receiver may be programmed to generate a heartbeat interrupt at timeout.

Event FIFO and Timestamp Events

The Event System provides a global timebase to attach timestamps to collected data and performed actions. The time stamping system provides a 32-bit timestamp event counter and a 32-bit seconds counter. The timestamp event counter either counts received timestamp counter clock events or runs freely with a clock derived from the event clock. The event counter is also able to run on a clock provided on a distributed bus bit. When an event is received, the timestamp counters are latched and stored in an event FIFO. This way, a software driver can pick up the exact timestamp when the event was received and attach it to data, for example to an EPICS record timestamp.

Event Log

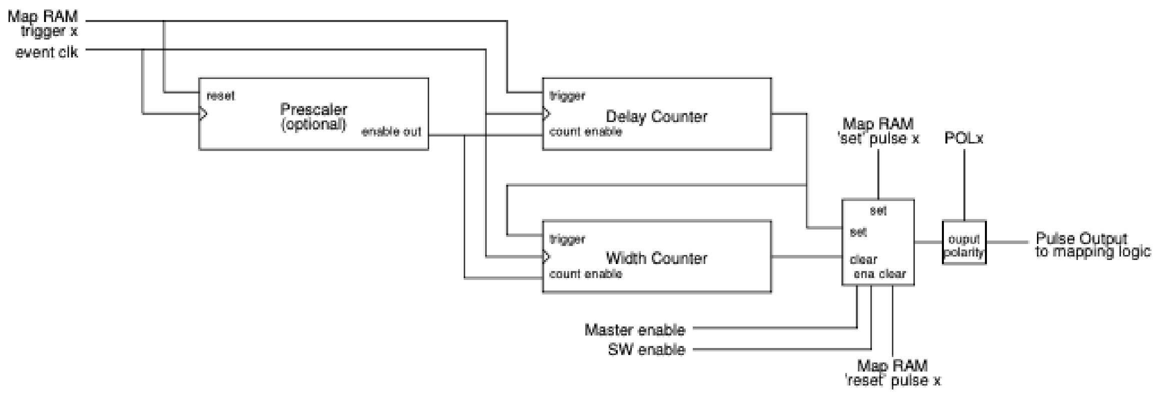
To debug or monitor the timing system performance, an event log facility is provided. Up to 512 events with timestamping information can be stored in the event log. The log is implemented as a ring buffer and is accessible as a memory region. Logging events can be stopped by an event or software.

Distributed Bus and Data Transmission

The distributed bus is able to carry eight simultaneous signals sampled with half the event clock rate over the fibre optic transmission media. The distributed bus signals may be output on programmable front panel outputs. The distributed bus bandwidth is shared by transmission of a configurable size data buffer to up to 2 kbytes.

Pulse Generators

Programmable pulse generators give a number of ways to configure how hardware (electrical/optical) outputs work. The structure of the pulse generation logic is shown in the figure below. Three signals from the mapping RAM control the output of the pulse: trigger, 'set' pulse and 'reset' pulse. A trigger causes the delay counter to start counting, when the end-of-count is reached the output pulse changes to the 'set' state and the width counter starts counting. At the end of the width count the output pulse is cleared. The mapping RAM signal 'set' and 'reset' cause the output to change state immediately without any delay.



Pulse Generator

Prescalers

The Event Receiver provides a number of programmable prescalers. The frequencies are programmable subharmonics of the event clock. A special event code *"reset prescalers"* ($7B) causes the prescalers to be synchronously reset in the whole system, so the frequency outputs will be in same phase across all event receivers.

Programmable Front Panel, Universal I/O and Backplane Connections

All outputs are programmable: each pulse generator output, prescaler and distributed bus bit can be mapped to any output. Starting with firmware version 0200 each output can have two sources which are logically OR'ed together.

Flip-flop Outputs (from FW version 0E0207)

There are 8 flip-flop outputs. Each of these is using two pulse generators, one for setting the output high and the other one for resetting the output low.

Front Panel Universal I/O Slots

Universal I/O slots provide different types of output with exchangeable Universal I/O modules. Each module provides two outputs e.g. two TTL output, two NIM output or two optical outputs. The source for these outputs is selected with mapping registers.

Synchronous Data Transmission

Pre-DC (Delay Compensation) event systems provided a way to to transmit configurable size data packets over the event system link. The buffer transmission size is configured in the Event Generator to up to 2 kbytes, and can be filled with arbitrary data by the host system. The Event Receiver is able to receive buffers of any size from 4 bytes to 2 kbytes in four byte (long word) increments.

Segmented Data Buffer

With the addition of delay compensation (300-series), a segmented data buffer has been introduced and it can coexist with the configurable size data buffer. The segmented data buffer is divided into 16 byte segments that allow updating only part of the buffer memory with the remaining segments left untouched.

External Event Input

An external hardware input is provided to be able to take an external pulse to generate an internal event. This event will be handled as any other received event.

2.1.4 Delay Compensation

In the 300-series event system, an active delay compensation feature was added. The delay compensation can be used to stabilize the system against e.g, thermal drifts of optical cables that affect the signal propagation time in the system. With different cable lengths, long distances and thermal gradients, the propagation delays could drift and disturb operation in cases where long-term timing stability is critical.

With the active delay compensation feature the Event Generator and distribution layer have been integrated into a single product, the Event Master (EVM).

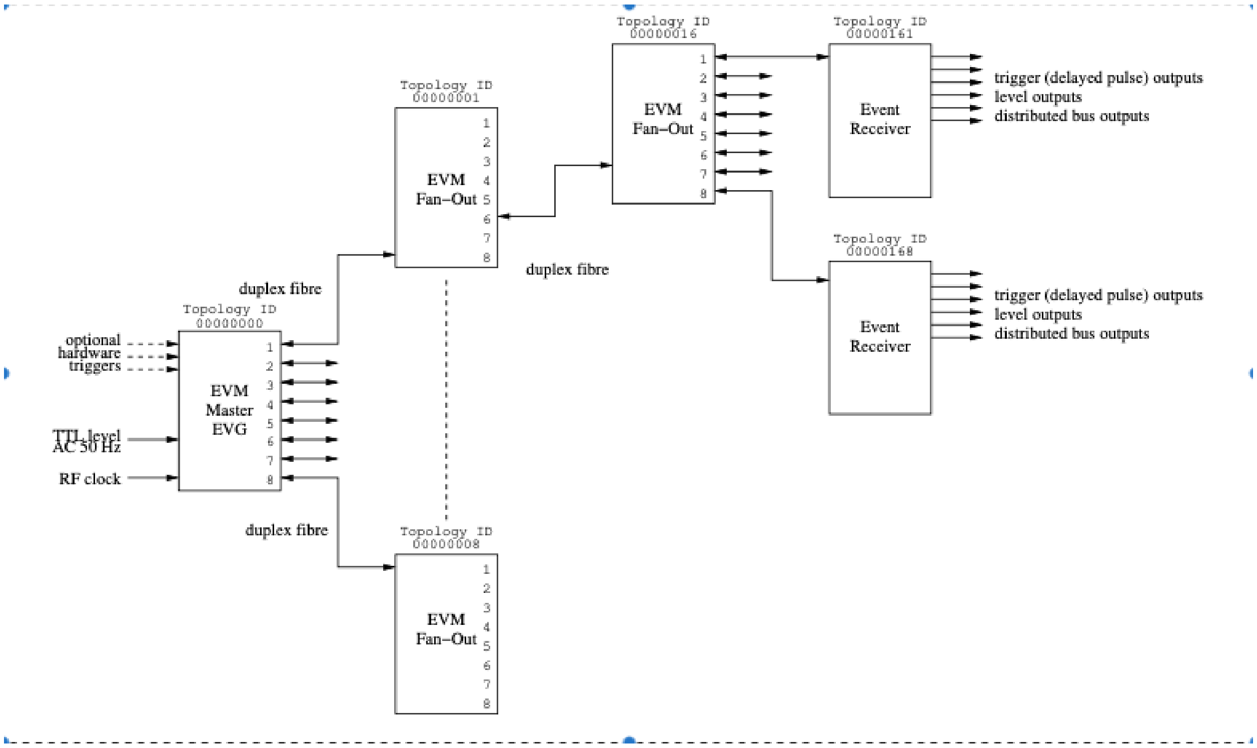


Figure: Timing System Topology (Active Delay Compensation, series 300)

Topology ID

Each device in the timing system is given an unique identifier, the Topology ID. The master EVM is given ID 0x00000000. The downstream devices are given IDs with the least significant four bits representing the port number the device is connected to. Each EVM left shifts its own ID by four bits and assigns the downstream port number to the lowest four bits to form the topology ID for the downstream devices in the next level. The topology IDs are represented above the devices in the example layout.

Active Delay Compensation

Delay compensation is achieved by measuring the propagation delay of events from the delay compensation master EVM through the distribution network up to the Event Receivers. At the last stage the EVR is aware of the delay through the network and adjusts an internal FIFO depth to match a programmed target delay value.

Timing System Master

The top node in the Timing System has the important task to generate periodic beacon events, to initialise and send out delay compensation data using the segmented data buffer. Only the top node can be the master and only one master is allowed in the system, all other EVMs have to be initialised in fan-out mode.

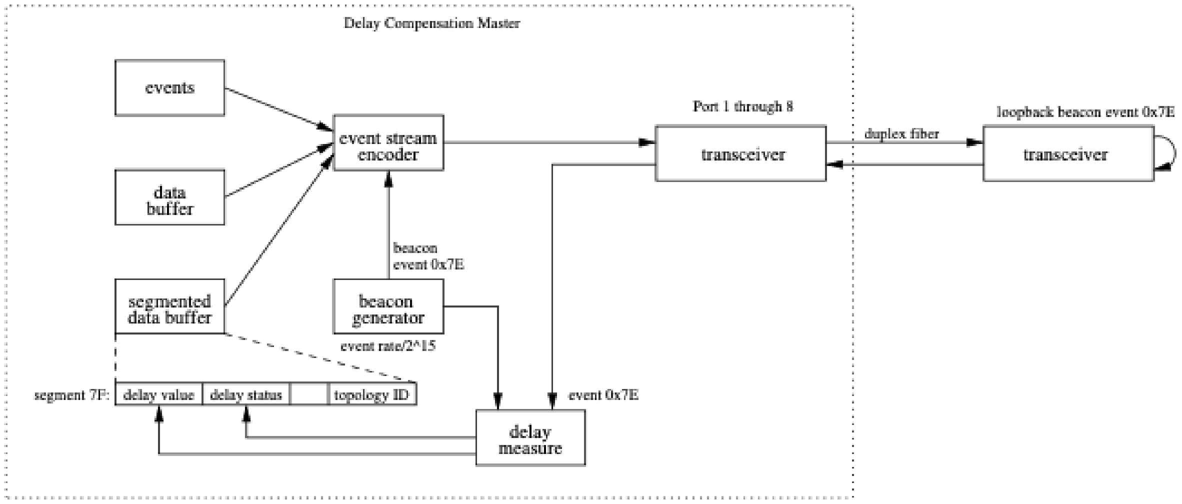


Figure: Timing System Master

The beacon generator sends out beacon events (code 0x7e) at a rate of event clock/2^{15}. When the next node receives the beacon it sends it immediately back to the master which measures the propagation delay of the beacon event.

Timing System Fan-Out

In EVMs configured as fan-outs, beacons from the Timing System Master are received by the port U transceiver. The recovered event clock from the transceiver is filtered by a clock cleaner external to the FPGA. Beacon events are propagated through the fan-out and the propagation delay is measured. Further on the beacon events get sent out on the fan-out ports and returned by the next level of fan-outs or event receivers. The loop delay for each port gets measured.

The fan-out receives the delay compensation segment on the segmented data buffer. This segment contains information about the delay value from the Timing System Master up to this fan-out and the delay value quality. The fan-out modifies the delay value and delay status fields in the DC segment for each port and sends out the new DC segments through ports 1 through 8.

Timing System Event Receiver

The Event Receiver receives the beacon event and returns it back immediately. Based on the recovered event clock from the gigabit transceiver the event receiver generates a local phase shifted and cleaned event clock. The majority of the event receiver logic is running in the cleaned event clock domain. A delay compensation FIFO separates the transceiver receive logic from the main event receiver logic.

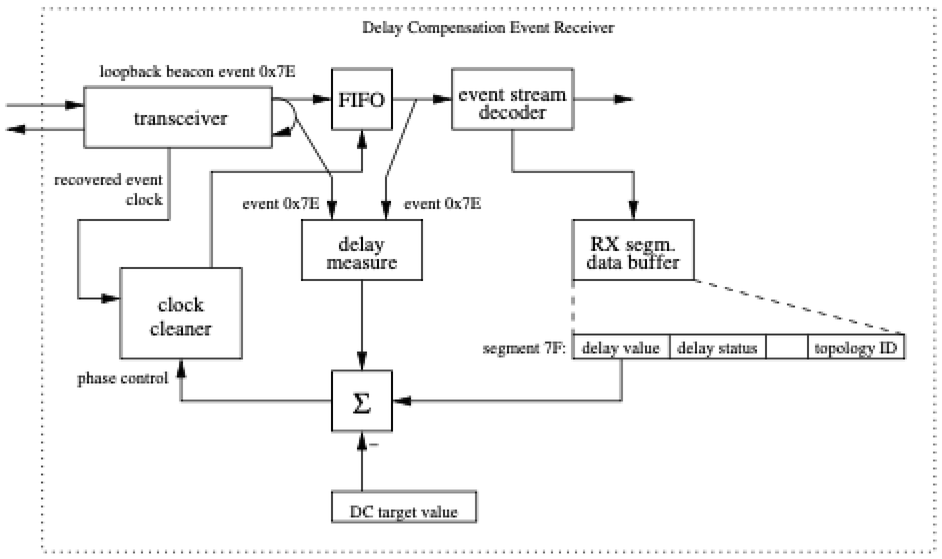


Figure 4: Timing System Event Receiver

The delay compensation segment of the segmented data buffer provides information of the fiber path delay from the timing master up to the event receiver. The event receiver adjusts the phase of the cleaned event clock to control the depth of the delay compensation FIFO.

2.2 Examples of usage scenarios

These examples use the MRF Linux API, to be found here. For register descriptions, see *EVG registermap* or *EVR registermap*.

2.2.1 Setting Up a Event System with Delay Compensation

In this example we are setting up a test system consisting of two VME-EVM-300 boards and to VME- EVR-300 boards. The first EVM (EVM1) is configured as the master and the seconds EVM (EVM2) as a fan-out. One EVR (EVR1) will be connected to the master (EVM1) and the other EVR (EVR2) to the fan-out (EVM2). The example setup is represented here:

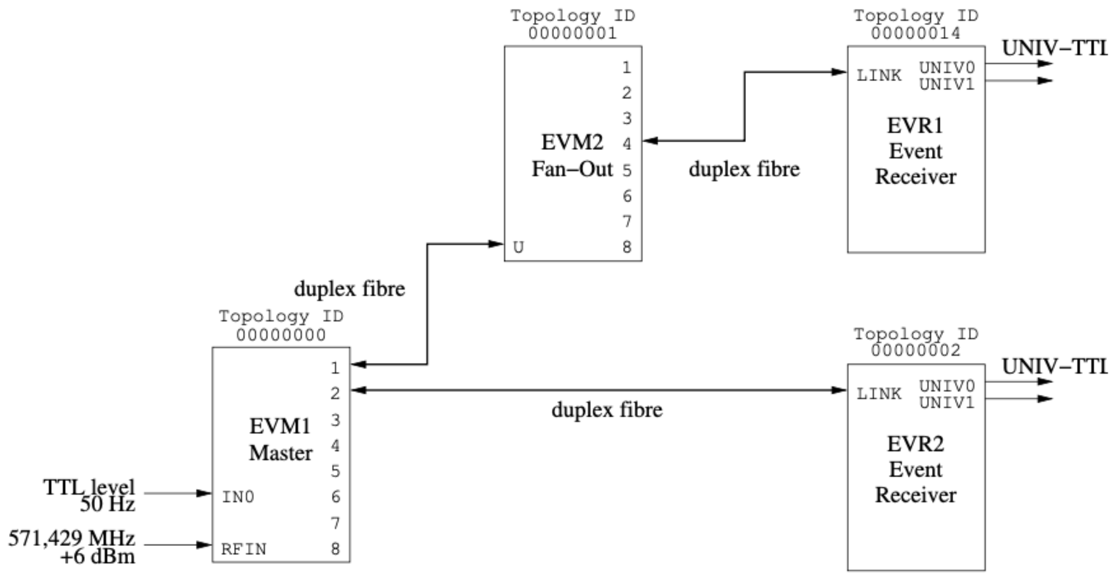


Figure: Example Setup

Initializing Master EVG

First we need to configure the master EVG to use the external RF input reference clock divided by four. After changing the clock source we need to reload the fractional synthesizer control word to force an internal reset.

The next step is to tell the top EVG that it is the master EVG and enable its beacon generator and delay compensation master responsibilities. Please note that the highest EVG in the system has to be the system master and if delay compensation is used it also has to have the beacon generator enabled. Only one EVG/EVM can be the system master and only one EVG is allowed to have the beacon generator enabled.

```
API calls                             Register access

EvgSetRFInput(evm1, 1, 3);        # *(evm1+0x50) = 0xc1030000;
EvgSetFracDiv(evm1, 0x0891c100); # *(evm1+0x80) = 0x0891c100;
EvgSystemMasterEnable(evm1, 1);  # *(evm1+0x04) = 0xe0c00000;
EvgBeaconEnable(evm1, 1);
EvgEnable(evm1, 1);
```

Initializing VME-EVM-300 as Fan-Out

The downstream EVM has to be configured to use the upstream EVG/EVM clock as its event clock and after this we need to (re)load the fractional synthesizer control word.

We enable the EVM and please note that both the system master bit and beacon generator bit are disabled.

```
API calls                         Register access

EvgSetRFInput(evm2, 4, 0x0c);     # *(evm2+0x50) = 0xc40c0000;
EvgSetFracDiv(evm2, 0x0891c100); # *(evm2+0x80) = 0x0891c100;
EvgEnable(evm2, 1);               # *(evm2+0x04) = 0xe0000000;
```

Initializing VME-EVR-300

We start with setting the fractional synthesizer operating frequency (reference for event clock) so that the EVR can lock to the received event stream.

The delay compensation logic measures/calculates a path delay from the master EVM/EVG to the EVR which consist of internal delays and fibre delays. The EVR has a receive FIFO and it adjusts the delay of this FIFO based on a target delay value and the actual path delay value. The delay value is a 32 bit value with a 16 bit integer part and a 16 bit fractional part. The integer part represents the delay in event clock cycles i.e. a value of 0x00010000 corresponds to an actual delay of one event clock cycle which at this examples rate is 7 ns.

In this example we set the target delay to 0x02100000 which is 3.696s.

```
 API calls                           Register access

 EvrSetFracDiv(evr1, 0x0891c100);    # *(evr1+0x80) = 0x0891c100;
 EvrSetTargetDelay(evr1, 0x02100000); # *(evr1+0xb0) = 0x02100000;
 EvrGetViolation(evr1, 1);           # *(evr1+0x08) = 0x00000001;
 EvrDCEnable(evr1, 1);               # *(evr1+0x04) = 0x80400000;
 EvrEnable(evr1, 1); |
```

The datapath delay value can be read from the EVR DCRxValue register at offset 0x0b8. For the example above with 2 m fiber patches the measured datapath delay value shows 0x0032cff0 (355.686 ns) for EVR1 and 0x00125eea (128.595 ns) for EVR2.

2.2.2 Generating an Event from AC input

A 50 Hz TTL level square wave signal is provided to the IN0 input on EVM1. We setup the input AC input divider to divide by 5, set the AC input logic to trigger event trigger 0 and we configure event trigger 0 to send out event code 0x01.

```
API calls                            Register access

EvgSetACInput(evm1, 0, 0, 5, 0);     # *(evm1+0x10) = 0x00000500;
EvgSetACMap(evm1, 0);                # *(evm1+0x14) = 0x00000001;
EvgSetTriggerEvent(evm1, 0, 0x01, 1); # *(evm1+0x100) = 0x00000101;
```

2.2.3 Receiving an Event and Generating an Output Pulse

To generate a pulse on a received event code in the EVR we need to setup the mapping RAM to trigger a pulse generator on an event and setup the pulse generator. We also need to map the pulse generator to the actual hardware output.

```
API calls                                     Register access

EvrSetPulseMap(evr1, 0, 0x01, 0, -1, -1);     # *(evr1+0x4014) = 0x00000001;
EvrSetPulseParams(evr1, 0, 0, 0, 1000);       # *(evr1+0x20C) = 0x000003e8;
EvrSetPulseProperties(evr1, 0, 0, 0, 0, 1, 1); # *(evr1+0x200) = 0x00000003;
EvrSetUnivOutMap(evr1, 0, 0x3f00);            # *(evr1+0x440) = 0x3f003f3f;
EvrMapRamEnable(evr1, 0, 1);                  # *(evr1+0x04) = 0x88400200;
EvrOutputEnable(evr1, 1);
```

Now we should see a 7s pulse on EVR1 UNIV0 output with a rate of 10 Hz. If we configure EVR2 the same way as the EVR1 in this example we should see a similar pulse on its UNIV0 output aligned with the output of EVR1.

2.2.4 Event Receiver Standalone Operation

Starting from firmware version 0207 capability to use the EVR as a stand-alone unit has been added. Functionality includes:

- Using the internal fractional synthesizer clock as a reference clock
- Generating internal events by software
- Generating internal event by one EVG type sequencer
- Generating internal event by external signals
- Internal events may be sent out on the TX link by setting the FWD bit for each event in the active mapping RAM

The example code below has been written for the mTCA-EVR-300, but with minor changes (remapping the outputs) it can be used for other form factors as well.

```
int evr_sa(volatile struct MrfErRegs *pEr)
{
  int i;
  EvrEnable(pEr, 1);
  if (!EvrGetEnable(pEr))
  {
    printf(ERROR_TEXT "Could not enable EVR!\n");
    return -1;
  }
  EvrSetIntClkMode(pEr, 1);
  /* Build configuration for EVR map RAMS */
  {
    int ram,code;
    /* Setup MAP ram: event code 0x01 to 0x04 trigger pulse generators 0 through 3*/
    ram = 0;
    for (i = 0; i < 4; i++)
  {
     code = 1+i;
     EvrSetLedEvent(pEr, ram, code, 1);
     /* Pulse Triggers start at code 1 */
     EvrSetPulseMap(pEr, ram, code, i, -1, -1);
  }
  /* Setup pulse generators and front panel TTL outputs*/
  for (i = 0; i < 4; i++)
  {
    EvrSetPulseParams(pEr, i, 1, 100, 100);
    EvrSetPulseProperties(pEr, i, 0, 0, 0, 1, 1);
    EvrSetFPOutMap(pEr, i, 0x3f00 | i);
  }

  /* Setup Prescaler 0 */
  EvrSetPrescaler(pEr, 0, 0x07ffff);
  /* Write some RAM events*/
  EvrSetSeqRamEvent(pEr, 0, 0, 0, 1);
  EvrSetSeqRamEvent(pEr, 0, 1, 0x001ff, 2);
  EvrSetSeqRamEvent(pEr, 0, 2, 0x002ff, 3);
  EvrSetSeqRamEvent(pEr, 0, 3, 0x003ff, 4);
  EvrSetSeqRamEvent(pEr, 0, 4, 0x04000, 0x7f);
```

(continues on next page)

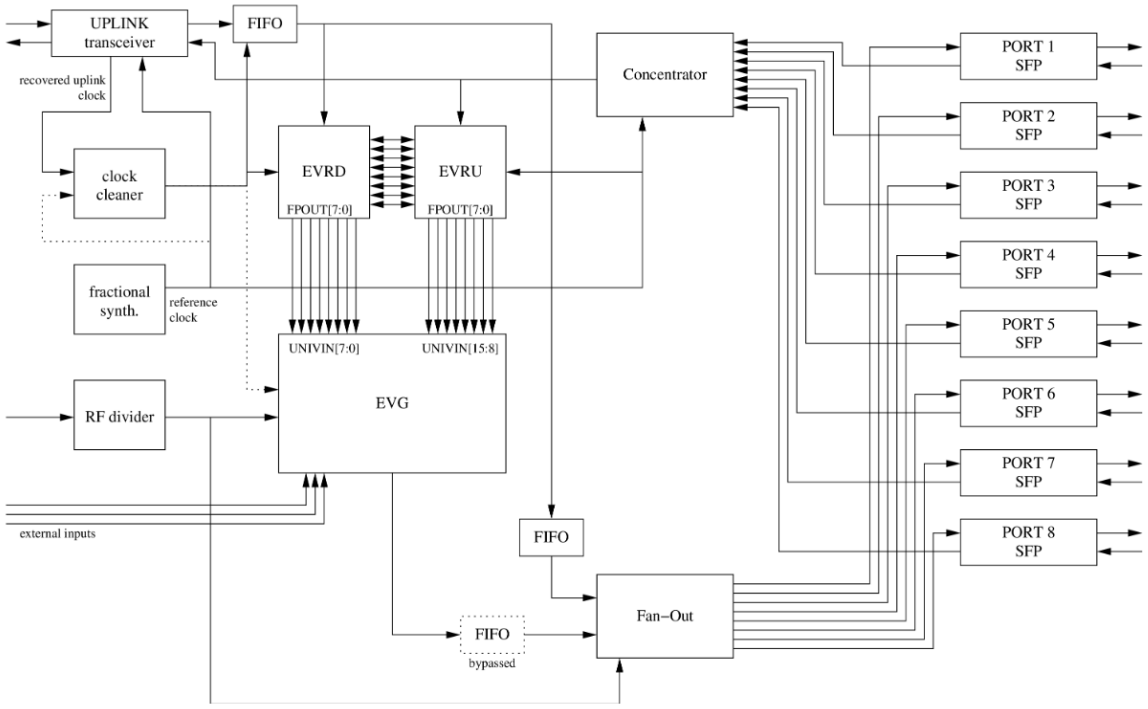
(continued from previous page)

```
  /* Setup sequence RAM to trigger from prescaler 0 */
  EvrSeqRamControl(pEr, 0, 1, 0, 0, 0, C_EVR_SIGNAL_MAP_PRESC+0);
  EvrMapRamEnable(pEr, 0, 1);
  EvrOutputEnable(pEr, 1);
  return 0;
}
```

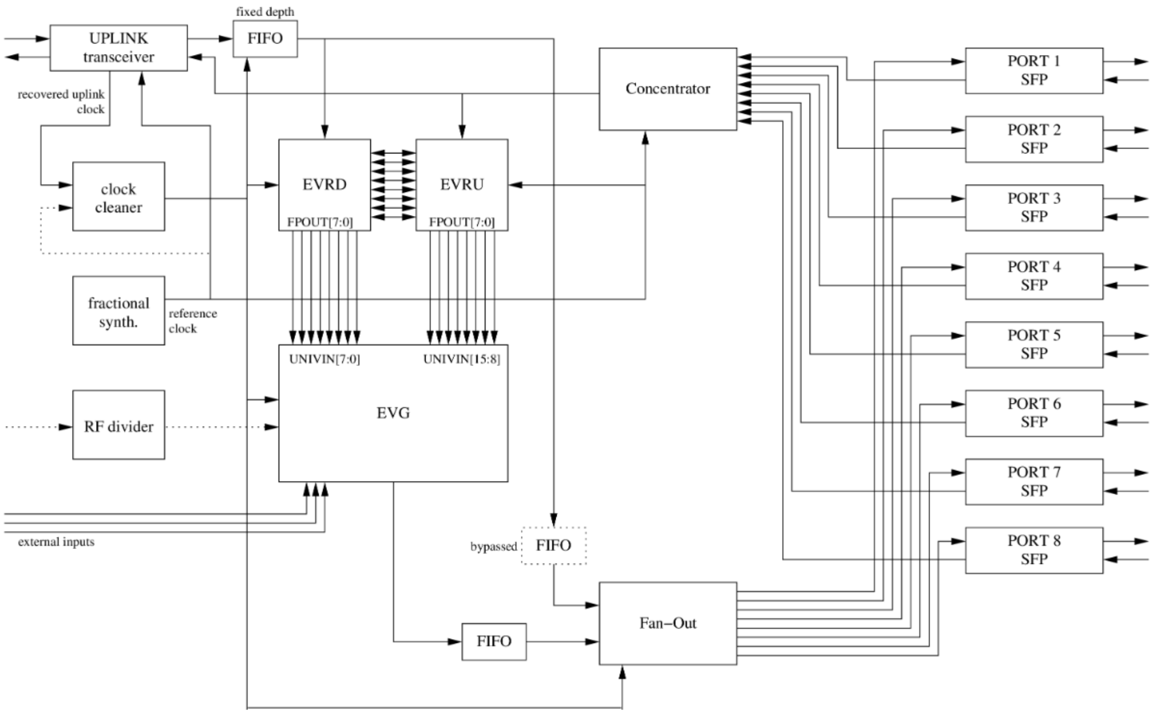
2.3 Event Master

Since the 300-series, Event Generator and Fanout/Concentrator are combined in the Event Master module that can be configured for either use; an Event Generator or Fanout-Concentrator.

Block diagram of EVM configured as an **Event Generator**:



Block diagram of EVM configured as a **Fanout/Concentrator**:



The essential differences are in clocking and FIFOs.

2.4 Fanout and Concentrator

When configured as a Fanout/Concentrator, the EVM has basically two tasks:

- Multiplying the event stream from one input link to multiple (up to 8) output links.
- Concentrating the event streams from multiple links to one upwards link.

Fanout/concentrators play also an important role in the *delay compensation*.

In EVMs configured as fan-outs *(DCMST = 0 and BCGEN = 0)* beacons from the Timing System Master are received by the port U transceiver. The recovered event clock from the transceiver is filtered by a clock cleaner external to the FPGA. A FIFO separates the cleaned event clock domain from the recovered clock domain. The depth of the FIFO is kept constant by adjusting the phase of the cleaned clock. Beacon events are propagated through the fan-out and the propagation delay from port U to the fan-out ports 1 through 8 is measured. This delay value can be read from the IntDCValue register.

Register map for this function can be found *here*.

2.5 Event Generator

The Event Generator generates the event stream and sends it out to an array of Event Receivers.

The Event Generator has a number of functions:

- Generating and transmitting the *timing events*
- Transmitting the *Distributed Bus* bits
- Transmitting the *Synchronous Data* Buffer
- Acting as a source for *timestamps*.

Events are sent out by the event generator as event frames (words) which consist of an eight bit event code and an eight bit distributed bus data byte. The event transfer rate is derived from an external RF clock or optionally an on-board clock generator. The optical event stream transmitted by the Event Generator is phase locked to the clock reference.

Register map for Event Generator function can be found *here*.

2.5.1 Event Generation

Timing events can be generated from a number of different sources: input signals, from an internal event sequencer, software-generated events and events received from an upstream Event Generator.

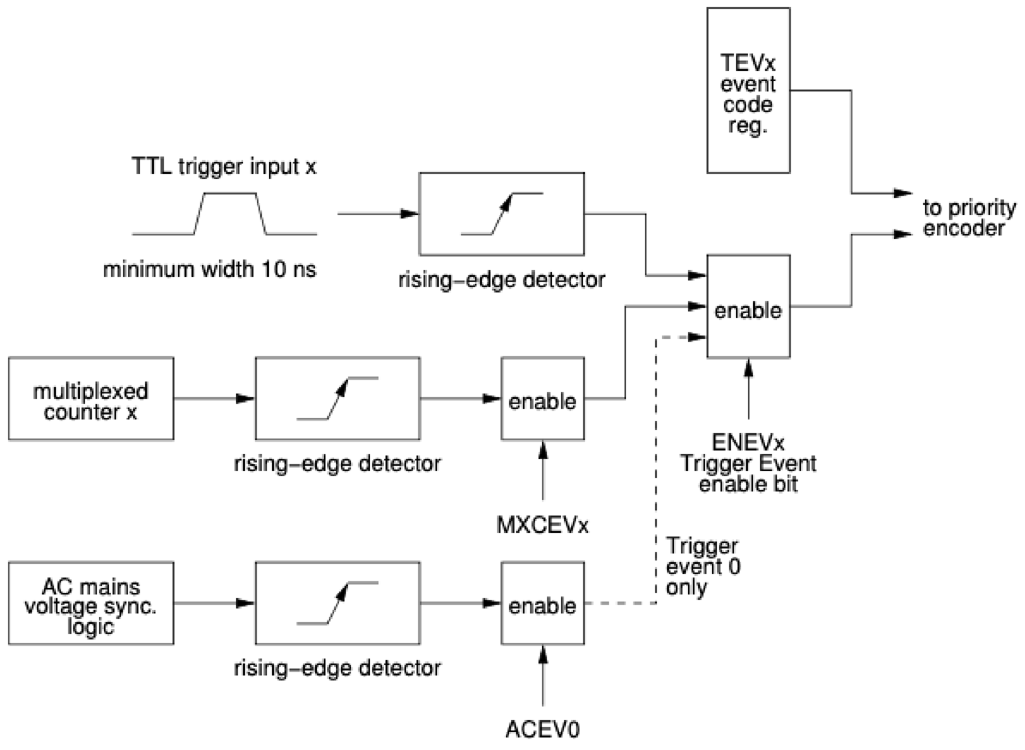
Only one event code may be transferred at a time. Should there be an event collision, i.e., two events should be sent at the same time, the one that has higher priority will be sent first. Event source priorities are resolved in a priority encoder.

Trigger Signal Inputs

There are eight trigger event inputs that can be configured to send out an event code on a stimulus. Each trigger event has its own programmable event code register and various enable bits. The event code transmitted is determined by contents of the corresponding event code register. The stimulus may be a detected rising edge on an external signal or a rising edge of a multiplexed counter output.

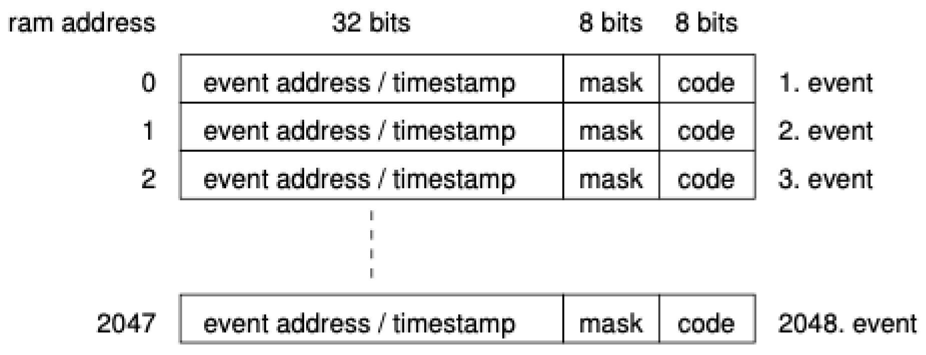
Trigger Event 0 has also the option of being triggered by a rising edge of the synchronization logic output signal (that is often used for synchronising with AC mains voltage).

The external inputs accept TTL level signals. The input logic is edge sensitive and the signals are subsequently synchronized internally to the event clock.



Event Sequencer

Event sequencers provide a method of transmitting or playing back sequences of events stored in random access memory with defined timing. In the event generator there are two event sequencers. 8-bit event codes are stored in a RAM table each attached with a 32-bit timestamp (event address) relative to the start of sequence. Both sequencers can hold up to 2048 event code – timestamp pairs.



ram address	32 bits	8 bits	8 bits	
0	event address / timestamp	mask	code	1. event
1	event address / timestamp	mask	code	2. event
2	event address / timestamp	mask	code	3. event
2047	event address / timestamp	mask	code	2048. event

Sequencer RAM Structure

The sequencer runs at the event clock rate. When the sequencer is triggered the internal event address counter starts counting. The counter value is compared to the event address of the next event in the RAM table. When the counter value matches or is greater than the timestamp in the RAM table, the attached event code is transmitted.

The time offset between two consecutive events in the RAM is allowed to be 1 to 2^{32} sequence clock cycles i.e. the internal event address counter rolls over to 0 when 0xffffffff is reached.

Starting with firmware version 0200 a mask field has been added. Bits in the mask field allow masking events from being send out based on external signal input states or software mask bits.

There are four **enable** signals:

– When mask enable bit is active ‘1’, enable event transmission only when HW signal is active high or software mask enable bit active ‘1’

And four **disable** signals:

– When mask disable bit is active ‘1’, disable event transmission when HW signal is active high or software mask disable bit is active ‘1’.

The Sequencers may be triggered from several sources including software triggering, triggering on a multiplexed counter output or AC mains voltage synchronization logic output.

The sequencer has three operating modes: single sequence, recycle and recycle with re-trigger (retrigger).

In the **single sequence** mode, the sequencer runs through the table until it reaches the end sequence code. After that, the sequencer is disabled, the timestamp (event address) counter and the RAM address are reset. The sequencer has to be re-enabled before it can run again.

In **recycle** mode, the sequence runs again immediately when it reaches the end sequence code. The sequencer then restarts from the beginning of the RAM.

In the **retrigger** mode, the sequencer stops when it reaches the end sequence code and waits for a trigger. RAM address and timestamp counter are both reset. When a new trigger arrives, the sequencer starts a new run. The difference to the single sequence mode is that the sequencer does not get disabled at end of sequence.

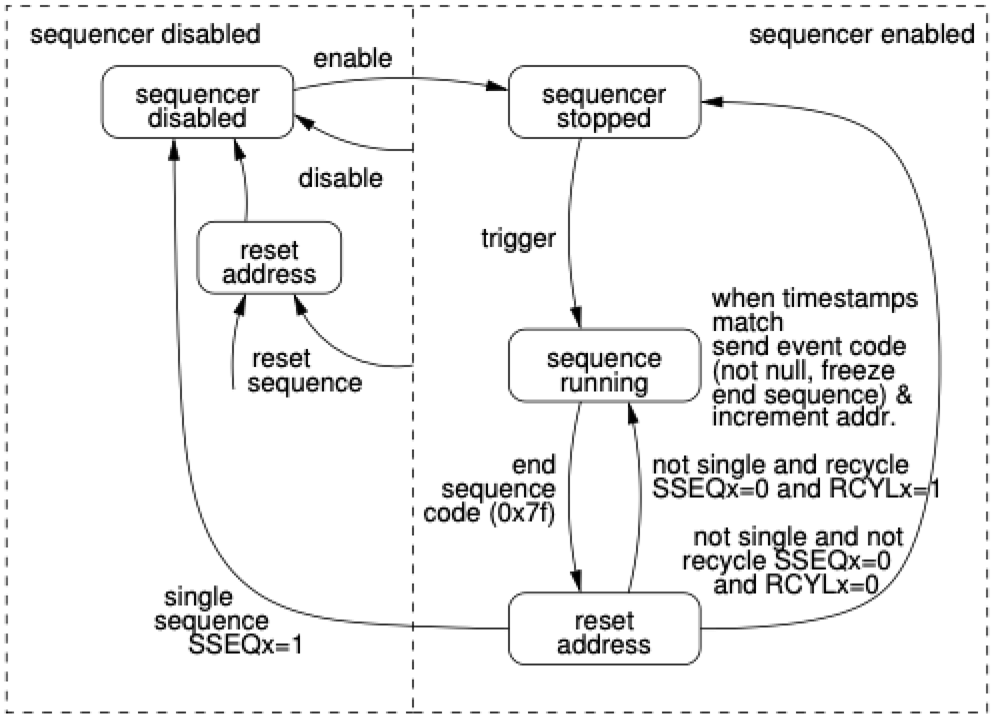
The sequencers are enabled by writing a ‘1’ bit to SQxEN in the *Sequence RAM control Register*. The RAMs may be disabled any time by writing a ‘1’ to SQxDIS bit. Disabling sequence RAMs does not reset the RAM address and timestamp registers. By writing a ‘1’ to the bit SQxRES in the Control Register the sequencer is both disabled and the RAM address and timestamp register is reset.

The contents of a sequencer RAM may be altered at any time, however, it is recommended only to modify RAM contents when the RAM is disabled.

There are two special event codes which are not transmitted, the null event code 0x00 and end sequence code 0x7f.

The null event code may be used if the time between two consecutive events should exceed 2^{32} event clock cycles by inserting a null event with a timestamp value of 0xffffffff. In this case the sequencer time will roll over from 0xffffffff to 0x00000000.

The end sequence code resets the sequencer RAM table address and timestamp register and depending on configuration bits, disables the sequencer (single sequence, SQxSNG=1) or restarts the sequence either immediately (recycle sequence, SQxREC=1) or waits for a new trigger (SQxREC=0).



Sequencer Interrupt Support

The sequencers provide several interrupts: a sequence start and sequence stop interrupt and two interrupts based on the position of the playback pointer in the sequencer RAM: a sequence halfway through interrupt and a sequence roll-over interrupt. The sequence start interrupt is issued when a sequencer is in enabled state, gets triggered and was not running before the trigger. A sequence stop interrupt is issued when the sequence is running and reaches the 'end of sequence' code.

Uses for the sequencer

The event sequencers are typically used for sending out a precisely timed sequence of events, like an accelerator machine cycle. In this case, event codes that have been defined for different actions to be taken during the acceleration cycle are placed in the sequencer RAM together with the time interval (event address) between sending out the events. This is the most common use case for the sequencers. Typically the two sequencers are used in foreground/background combination, where the foreground sequencer is transmitting events, and the background sequencer can be prepared by software for the upcoming cycles. When the foreground sequencer finishes, the roles can be swapped and the backgound sequencer made active.

A more exotic use case for the sequencer could be sending out a complicated signal pattern, using the RAM contents in the recycle mode.

Software-generated Events

Events can be generated in software by writing into the Software Event register. This is useful for creating events that occur based on some higher-level conditions; for example an operator requesting a beam dump in a circular accelerator.

Upstream Events

Event Generators may be cascaded. The bitstream receiver in the event generator includes a first-in-first-out (FIFO) memory to synchronize incoming events which may be synchronized to a clock unrelated to the event clock. Usually there are no events in the FIFO. An event code from an upstream EVG is transmitted as soon as there is no other event code to be transmitted.

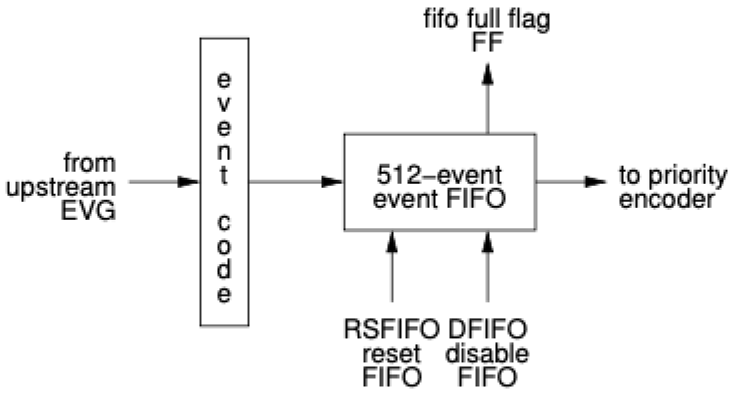


Figure: Upstream event processing.

2.5.2 Distributed Bus

The distributed bus allows transmission of eight simultaneous signals with half of the event clock rate time resolution (20 ns at 100 MHz event clock rate) from the event generator to the event receivers.

Note: Hard/firmware before Delay Compensation

Before introduction of delay compensation, if the data transfer feature was not in use, the highest time resolution for the distributed bus bits was equal to half of the event clock rate.

The source for distributed bus signals may be an external source or the signals may be generated with programmable *multiplexed counters* (MXC) inside the event generator. The bits of the distributed bus from external signals are sampled synchronously to the event clock.

The distributed bus signals can be programmed to be available as hardware outputs on the event receiver.

If there is an upstream EVG, the state of all distributed bus bits may be forwarded by the EVG.

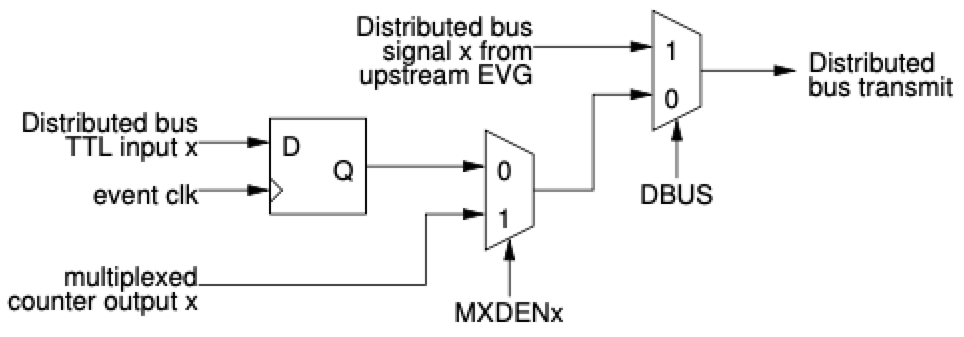


Figure: Distributed Bus signal source selection.

2.5.3 Timestamping support

The event system supports timestamping by providing:

- Facilities to distribute a seconds value to all receivers
- Facilities to support generation of sub-second timestamps, usually in the event clock resolution
- Precisely latching the timestamp in an Event Receiver, on request or when an event code has been received.

The timestamp support guarantees that all event receivers (and generators) in the same distribution will have precisely synchronized timestamp, up to the resolution of the event clock. For example, 100 MHz event clock results in highest timestamp resolution of 10 nanoseconds.

The seconds value to be distributed has to be provided to the Event Generator. This is typically sourced from an external GPS receiver.

Timestamp Generator

The model of time implemented by the MRF hardware is two 32-bit unsigned integers: counter, and "seconds". The counter is maintained by each EVR and incremented quickly. The value of the "seconds" is sent periodically from the EVG at a lower rate.

During each "second" 33 special codes (see sec. *Event Codes*) must be sent. The first 32 are the shift 0/1 codes which contain the value of the next "second". The last is the timestamp reset event. When received this code transfers the new "second" value out of the shift register, and resets the counter to zero. These actions start the next "second".

Note that while it is referred to as "seconds" this value is an arbitrary integer which can have other meanings. Currently only one time model is implemented, but implementing others is possible.

Standard (aka "Light Source") Time Model

In this model the "seconds" value is an actual 1Hz counter. The software default is the POSIX time of seconds since 1 Jan. 1970 UTC. Each new second is started with a trigger from an external 1Hz oscillator, usually a GPS receiver. Most GPS receivers have a one pulse per second (PPS) output. Time is converted to the EPICS epoch (1 Jan. 1990) for use in the IOC.

Several methods of sending the seconds value to the EVG are possible:

External hardware

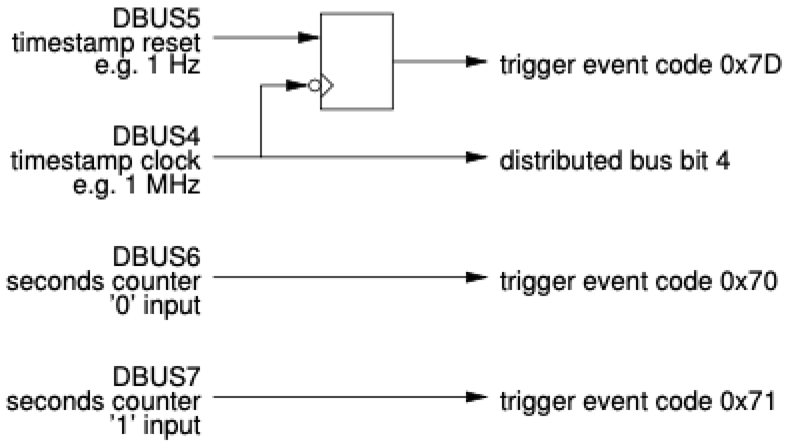
In this method, hardware is used to communicate with a GPS receiver over a serial (RS232) link to receive the timestamp and connect to two external inputs on the EVG. These inputs must be programmed to send the shift 0/1 codes.

Time from an NTP server

Time from a NTP server can be used without special hardware. This requires only a 1Hz (PPS) signal coming from the same source as the NTP time. Several commerial vendors supply dedicated NTP servers with builtin GPS receivers and 1Hz outputs. A software function is provided on the EVG which is triggered by the 1Hz signal. At the start of each second it sends the next second (current+1), which will be latched after the following 1Hz tick.

Timestamping Inputs

Starting from firmware version E306 a few distributed bus input signals have dual function: transition board input DBUS5-7 can be used to generate special event codes controlling the timestamping in Event Receivers.



The two clocks, timestamp clock and timestamp reset clock, are assumed to be rising edge aligned. In the EVG the timestamp reset clock is sampled with the falling edge of the timestamp clock. This is to prevent a race condition between the reset and clock signals. In the EVR the reset is synchronised with the timestamp clock.

The two seconds counter events are used to shift in a 32-bit seconds value between consecutive timestamp reset events. In the EVR the value of the seconds shift register is transferred to the seconds counter at the same time the higher running part of the timestamp counter is reset.

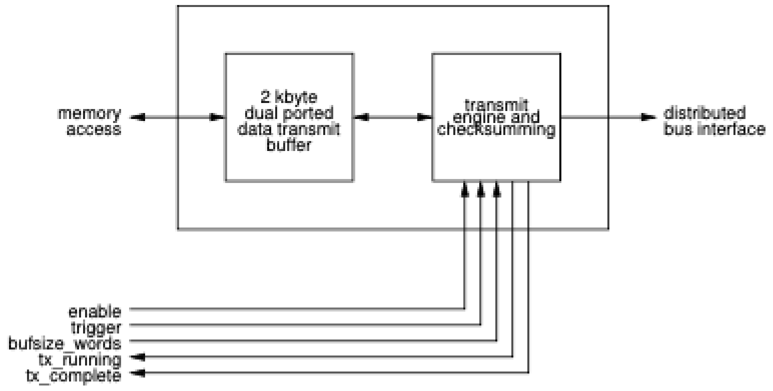
Logic has been added to automatically increment and send out the 32-bit seconds value. Using this feature requires the two externally supplied clocks as shown above, but the events 0x70 and 0x71 get generated automatically.

After the rising edge of the slower clock on DBUS4, the internal seconds counter is incremented and the 32 bit binary value is sent out LSB first as 32 events 0x70 and 0x71. The seconds counter can be updated by software by using the `TSValue` and `TSControl` registers.

The distributed bus event inputs can be enabled independently through the distributed bus event enable register. The events generated through these distributed bus input ports are given lowest priority.

2.5.4 Configurable Size Data Buffer

A buffer of up to 2k bytes can be transmitted over the event link. This data buffer will be (synchronously) available to all EVRs that receive the event stream.


memory access
2 kbyte dual ported data transmit buffer
transmit engine and checksumming
distributed bus interface
enable
trigger
bufsize_words
tx_running
tx_complete

The data to be transmitted is stored in a 2 kbyte dual-ported memory starting from the lowest address 0. This memory is directly accessible via the memory interface.

The transfer size is determined by bufsize register bits in four byte increments. The transmission is triggered by software. Two flags tx_running and tx_complete represent the status of transmission. Transmission utilises two K-characters to mark the start and end of the data transfer payload, the protocol looks following:

8B10B-character	Description
K28.0	Start of data transfer
Dxx.x	1st data byte (address 0)
Dxx.x	2nd data byte (address 1)
Dxx.x	3rd data byte (address 2)
Dxx.x	4th data byte (address 3)
…	…
Dxx.x	nth data byte (address n-1)
K28.1	End of data
Dxx.x	Checksum (MSB)
Dxx.x	Checksum(LSB)

Segmented Data Buffer Transmission

In addition to the configurable size data buffer a new way to transfer information is provided. The segmented data buffer memory is divided into 128 segments of 16 bytes each and it is possible to transmit the contents of a single segment or a block of consecutive segments without affecting contents of other segments.

With the introduction of active delay compensation in firmware version 0200, the use of the "data buffer mode" has become mandatory. The active delay compensation logic does use the last segment of the segmented data buffer memory for propagating delay compensation information and this segment is reserved for system use.

The data to be transmitted is stored in a 2 kbyte dual-ported memory starting from the lowest address 0. This memory is directly accessible from the memory interface (VME, PCI). The transfer size is determined by bufsize register bits in four byte increments. The transmission is triggered by software. Two flags, `tx_running` and `tx_complete` represent the status of transmission.

Transmission utilises two K-characters to mark the start and end of the data transfer payload, the protocol looks following:

8B10B-character	Description
K28.2	Start of data transfer
Dxx.x	Block address of 16 byte segment
Dxx.x	1st data byte (address 0)
Dxx.x	2nd data byte (address 1)
Dxx.x	3rd data byte (address 2)
Dxx.x	4th data byte (address 3)
…	…
Dxx.x	nth data byte (address n-1)
K28.1	End of data
Dxx.x	Checksum (MSB)
Dxx.x	Checksum(LSB)

Segmented Data Transfer Example

Delay Compensation and Topology ID data

The last segment is reserved for system management and is used to propagate delay compensation and topology data. The contents of the last segment are represented below. Please note that the word values are in little endian byte order.

segment	byte 0 - 3	byte 4 - 7	byte 8 - 11	byte 12 - 15
127	DCDelay	DCStatus	reserved	TopologyID

DCDelay represents the delay from DC master to receiving node. The value is a fixed point number with the point between the two 16 bit words. The delay is measured in event clock cycles.

DCStatus shows the quality of the delay value: 1 - initial lock, 3 - locked with precision < event clock cycle, 7 - fine precision.

TopologyID shows the geographical address of the node.

2.5.5 Programmable Outputs

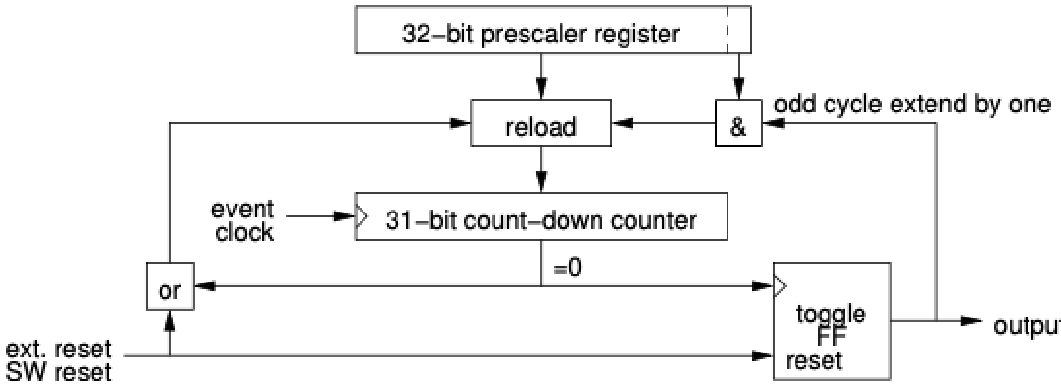
All the outputs are programmable: multiplexed counters and distributed bus bits can be mapped to any output. The mapping is shown in table below.

MappingID	Signal
0 to 31	(Reserved)
32	Distributed bus bit 0 (DBUS0)
…	…
39	Distributed bus bit 7 (DBUS7)
40	Multiplexed Counter 0
…	…
47	Multiplexed Counter 7
48	AC trigger logic output
49 to 61	(Reserved)
62	Force output high (logic 1)
63	Force output low (logic 0)

2.5.6 Utility Functions in the Event Generator

Multiplexed Counters

Eight 32-bit multiplexed counters generate clock signals with programmable frequencies from event clock/2^{32}1 to event clock/2. Even divisors create 50% duty cycle signals. The counter outputs may be programmed to trigger events, drive distributed bus signals and trigger sequence RAMs. The output of multiplexed counter 7 is hard-wired to the mains voltage synchronization logic.

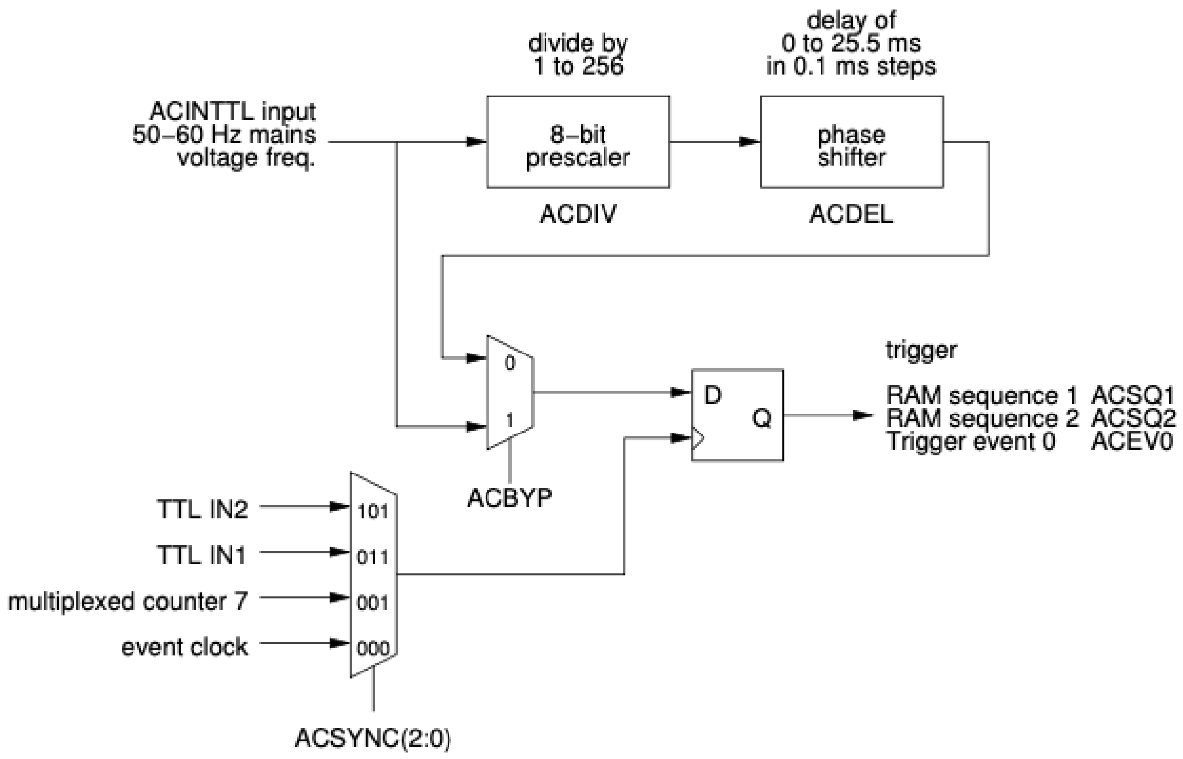


Each multiplexed counter consists of a 32-bit prescaler register and a 31-bit count-down counter which runs at the event clock rate. When count reaches zero, the output of a toggle flip-flop changes and the counter is reloaded from the prescaler register. If the least significant bit of the prescaler register is one, all odd cycles are extended by one clock cycle to support odd dividers. The multiplexed counters may be reset by software or hardware input. The reset state is defined by the multiplexed counter polarity register.

Prescaler value	DutyCycle	Frequency at 125MHz Event Clock
0,1 not allowed	undefined	undefined
2	50/50	62.5 MHz
3	33/66	41.7 MHz
4	50/50	31.25 MHz
5	40/60	25 MHz
…	…	…
2^{32}1	approx. 50/50	0.029 Hz

AC Line Synchronisation

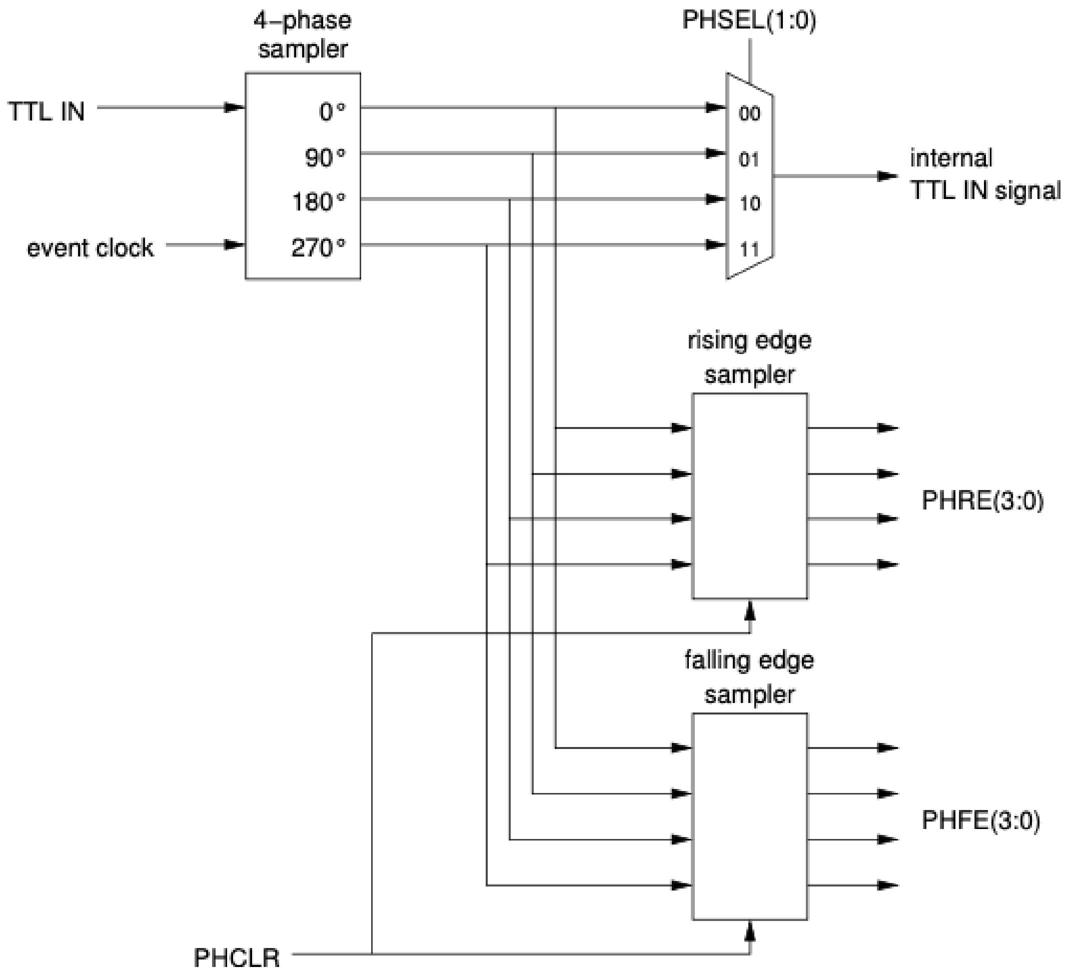
The Event Generator provides synchronization to the mains voltage frequency or another external clock. The mains voltage frequency can be divided by an eight bit programmable divider. The output of the divider may be delayed by 0 to 25.5 ms by a phase shifter in 0.1 ms steps to be able to adjust the triggering position relative to mains voltage phase. After this the signal synchronized to the event clock or the output of multiplexed counter 7. The option to synchronize to an external clock provided in front panel TTL input IN1 or IN2 has been added in firmware version 22000207.



The phase shifter operates with a clock of 1 MHz which introduces jitter. If the prescaler and phase shifter are not required this circuit may be bypassed. This also reduces jitter because the external trigger input is sampled directly with the event clock.

2.5.7 Front Panel TTL Input with Phase Monitoring

Starting from firmware 22000207 a new phase select and phase monitoring feature for the front panel TTL inputs has been added. This allows for monitoring the signal phase and selecting the sampling point of external signals that are phase locked to the event clock.



The external signal is sampled with four phases of the event clock, 0°, 90°, 180° and 270° and synchronized to the event clock. The signal being used for the internal FPGA logic is selected by the PHSEL bits in the phase monitoring register.

The phase monitoring logic detects rising and falling edges of the incoming signal and stores the phase offset in two registers **PHRE** for the rising edge and **PHFE** for the falling edge. The contents of the registers are updated on each edge detected and the values can be reset to 0000 for PHRE and 1111 for PHFE by writing a ‘1’ to the PHCLR bit.

PHRE value	Rising edge position
0000	Reset value, no edge detected
0001	Edge between 180° and 270°
0011	Edge between 90° and 180°
0111	Edge between 0° and 90°
1111	Edge between 270° and 0°

PHFE value	Falling edge position
1111	Reset value, no edge detected
1110	Edge between 180° and 270°
1100	Edge between 90° and 180°
1000	Edge between 0° and 90°
0000	Edge between 270° and 0°

If the input signal is phase locked to the event clock the phase monitoring values should be stable or toggling between two values if the signal is close to the clock sampling edge. A sampling point as far as possible from the transition point should be selected. Selecting the correct edge is not automated. The edge position of interest should be monitored by the user application and the correct phase should be selected by software.

PHRE value	Rising edge position	PHSEL
0000	Reset value, no edge detected	
0001	Edge between 180° and 270°	01, sample at 90°
0001/0011	Edge around 180°	00, sample at 0°
0011	Edge between 90° and 180°	00, sample at 0°
0011/0111	Edge around 90°	11, sample at 270°
0111	Edge between 0° and 90°	11, sample at 270°
0111/1111	Edge around 0°	10, sample at 180°
1111	Edge between 270° and 0°	10, sample at 180°
1111/0001	Edge around 270°	01, sample at 90°

Table: Phase Monitoring Rising Edge Select Values

PHFE value	Falling edge position	PHSEL
1111	Reset value, no edge detected	
1110	Edge between 180° and 270°	01, sample at 90°
1110/1100	Edge around 180°	00, sample at 0°
1100	Edge between 90° and 180°	00, sample at 0°
1100/1000	Edge around 90°	11, sample at 270°
1000	Edge between 0° and 90°	11, sample at 270°
1000/0000	Edge around 0°	10, sample at 180°
0000	Edge between 270° and 0°	10, sample at 180°
0000/1110	Edge around 270°	01, sample at 90°

Table: Phase Monitoring Falling Edge Select Values

In the DC firmware the distributed bus is operating at half rate of the event clock and when using an external clock with an even sub-harmonic the phase of the distributed bus transmission is arbitrary after restarting the system. To overcome this the phase monitoring inputs have a status bit that shows the phase of the distributed bus on the rising edge of the external input. The user can monitor this bit and verify that the phase is correct each time the system is restarted. If the phase is incorrect the phase may be toggled by writing a ‘1’ into the PHTOGG bit in the clock control register.

2.5.8 Event Clock RF Source

All operations on the event generator are synchronised to the event clock which is derived from an externally provided RF clock. For laboratory testing purposes an on-board fractional synthesiser may be used to deliver the event clock. The serial link bit rate is 20 times the event clock rate. The acceptable range for the event clock and bit rate is shown in the following table.

During operation the reference frequency should not be changed more than ±100 ppm.

	EventClock	BitRate
Minimum	50 MHz	1.0 Gb/s
Maximum	142.8 MHz	2.9 Gb/s

RF Clock and Event Clock

The event clock may be derived from an external RF clock signal. The front panel RF input is 50 ohm terminated and AC coupled to a LVPECL logic input, so either an ECL level clock signal or sine-wave signal with a level of maximum +10 dBm can be used.

Divider	RF Input Frequency	Event Clock	Bit Rate
÷1	50 MHz – 142.8 MHz	50 MHz–142.8 MHz	1.0 Gb/s – 2.9 Gb/s
÷2	100 MHz – 285.6 MHz	50 MHz–142.8 MHz	1.0 Gb/s – 2.9 Gb/s
÷3	150 MHz – 428.4 MHz	50 MHz–142.8 MHz	1.0 Gb/s – 2.9 Gb/s
÷4	200 MHz – 571.2 MHz	50 MHz–142.8 MHz	1.0 Gb/s – 2.9 Gb/s
÷5	250 MHz – 714 MHz	50 MHz–142.8 MHz	1.0 Gb/s – 2.9 Gb/s
÷6	300 MHz – 856.8 MHz	50 MHz–142.8 MHz	1.0 Gb/s – 2.9 Gb/s
÷7	350 MHz – 999.6 MHz	50 MHz–142.8 MHz	1.0 Gb/s – 2.9 Gb/s
÷8	400 MHz – 1.142 GHz	50 MHz–142.8 MHz	1.0 Gb/s – 2.9 Gb/s
÷9	450 MHz – 1.285 MHz	50 MHz–142.8 MHz	1.0 Gb/s – 2.9 Gb/s
÷10	500MHz–1.428GHz	50 MHz–142.8 MHz	1.0 Gb/s – 2.9 Gb/s
÷11	550MHz–1.571GHz	50 MHz–142.8 MHz	1.0 Gb/s – 2.9 Gb/s
÷12	600 MHz – 1.6 GHz	50 MHz–133 MHz	1.0 Gb/s – 2.667 Gb/s
÷14	700MHz–1.6GHz *)	50 MHz–114 MHz	1.0 Gb/s – 2.286 Gb/s
÷15	750MHz–1.6GHz *)	50 MHz–107 MHz	1.0 Gb/s – 2.133 Gb/s
÷16	800MHz–1.6GHz *)	50 MHz–100 MHz	1.0 Gb/s – 2.0 Gb/s
÷17	850MHz–1.6GHz *)	50 MHz–94 MHz	1.0 Gb/s – 1.882 Gb/s
÷18	900MHz–1.6GHz *)	50 MHz–88 MHz	1.0 Gb/s – 1.777 Gb/s
÷19	950MHz–1.6GHz *)	50 MHz–84 MHz	1.0 Gb/s – 1.684 Gb/s
÷20	1.0GHz–1.6GHz *)	50 MHz–80 MHz	1.0 Gb/s – 1.600 Gb/s
÷21	1.05GHz–1.6GHz *)	50 MHz–76 MHz	1.0 Gb/s – 1.523 Gb/s
÷22	1.1GHz–1.6GHz *)	50 MHz–72 MHz	1.0 Gb/s – 1.454 Gb/s
÷23	1.15GHz–1.6GHz *)	50 MHz–69 MHz	1.0 Gb/s – 1.391 Gb/s
÷24	1.2GHz–1.6GHz *)	50 MHz–66 MHz	1.0 Gb/s – 1.333 Gb/s
÷25	1.25GHz–1.6GHz *)	50 MHz–64 MHz	1.0 Gb/s – 1.280 Gb/s
÷26	1.3GHz–1.6GHz *)	50 MHz–61 MHz	1.0 Gb/s – 1.230 Gb/s
÷27	1.35GHz–1.6GHz *)	50 MHz–59 MHz	1.0 Gb/s – 1.185 Gb/s
÷28	1.4GHz–1.6GHz *)	50 MHz–57 MHz	1.0 Gb/s – 1.142 Gb/s
÷29	1.45GHz–1.6GHz *)	50 MHz–55 MHz	1.0 Gb/s – 1.103 Gb/s
÷30	1.5GHz–1.6GHz *)	50 MHz–53 MHz	1.0 Gb/s – 1.066 Gb/s
÷31	1.55GHz–1.6GHz *)	50 MHz–51 MHz	1.0 Gb/s – 1.032 Gb/s

continues on next page

Table 1 – continued from previous page

Divider	RF Input Frequency	Event Clock	Bit Rate
÷32	1.6 GHz *)	50 MHz	1.0 Gb/s

RF Input Requirements

*) Range limited by AD9515 maximum input frequency of 1.6 GHz

Fractional Synthesiser (EVM, distribution layer)

The event master requires a reference clock to be able to synchronise on the incoming event stream sent by the system master. A Microchip (formerly Micrel) SY87739L Protocol Transparent Fractional-N Synthesiser with a reference clock of 24 MHz is used.

The following table lists programming bit patterns for a few frequencies. Please note that before programming a new operating frequency in the fractional synthesizer the operating frequency (in MHz) has to be set in the UsecDivider register. This is essential as the board's PLL cannot lock if it does not know the frequency range to lock to.

Event Rate	Configuration Bit Pattern	Reference Output	Precision (theoretical)
142.8 MHz	0x0891C100	142.857 MHz	0
499.8 MHz/4 = 124.95 MHz	0x00FE816D	124.95 MHz	0
499.654 MHz/4 = 124.9135 MHz	0x0C928166	124.907 MHz	-52 ppm
476 MHz/4 = 119 MHz	0x018741AD	119 MHz	0
106.25 MHz (fibre channel)	0x049E81AD	106.25 MHz	0
499.8 MHz/5 = 99.96 MHz	0x025B41ED	99.956 MHz	-40 ppm
50 MHz	0x009743AD	50.0 MHz	0
499.8 MHz/10 = 49.98 MHz	0x025B43AD	49.978 MHz	-40 ppm
499.654MHz/4=124.9135MHz	0x0C928166	124.907MHz	-52 ppm
50 MHz	0x009743AD	50.0 MHz	0

2.6 Delay Compensation

With the active delay compensation feature the Event Generator and distribution layer have been integrated into a single product, the Event Master (EVM).

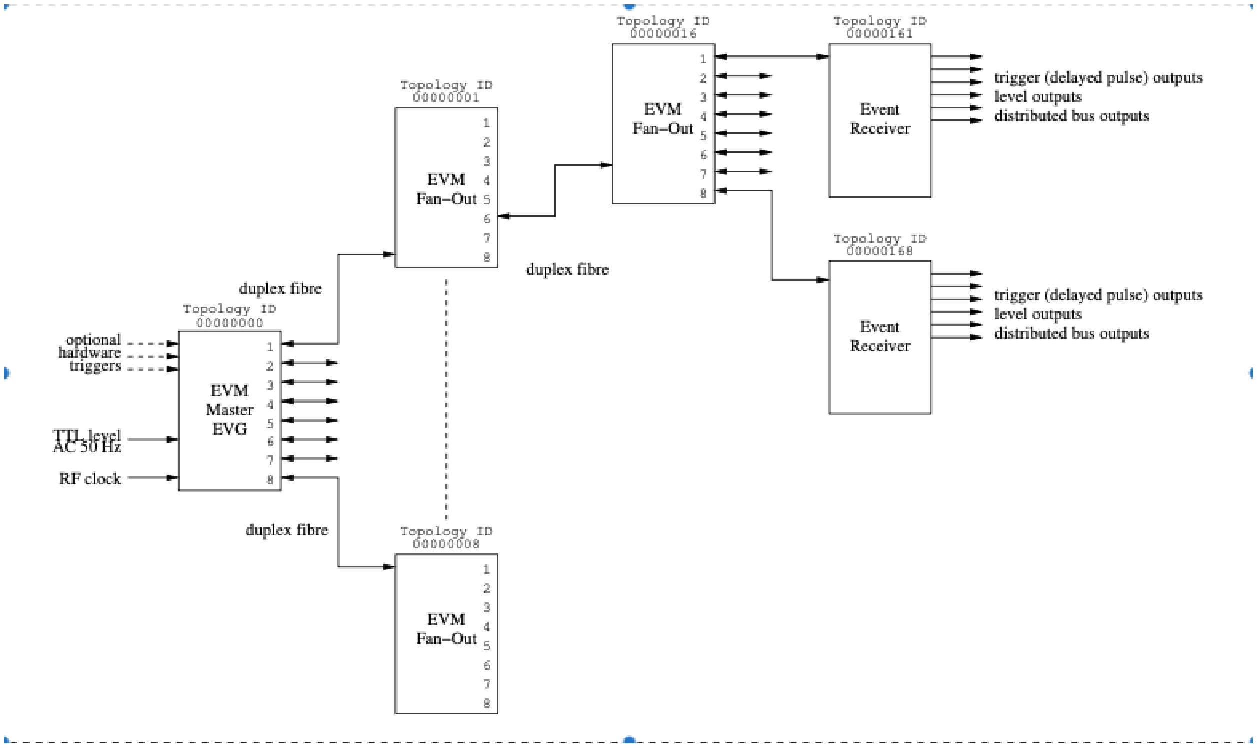


Figure 1: Timing System Topology (Active Delay Compensation, series 300)

2.6.1 Topology ID

Each device in the timing system is given an unique identifier, the Topology ID. The master EVM is given ID 0x00000000. The downstream devices are given IDs with the least significant four bits representing the port number the device is connected to. Each EVM left shifts its own ID by four bits and assigns the downstream port number to the lowest four bits to form the topology ID for the downstream devices in the next level. The topology IDs are represented above the devices in the example layout in figure 1.

2.6.2 Active Delay Compensation

Delay compensation is achieved in measuring the propagation delay of events from the delay compensation master EVM through the distribution network up to the Event Receivers. At the last stage the EVR is aware of the delay through the network and adjusts an internal FIFO depth to match a programmed target delay value.

Timing System Master

The top node in the Timing System has the important task to generate periodic beacon events, to initialise and send out delay compensation data using the segmented data buffer. Only the top node can be the master and only one master is allowed in the system, all other EVMs have to be initialised in fan-out mode.

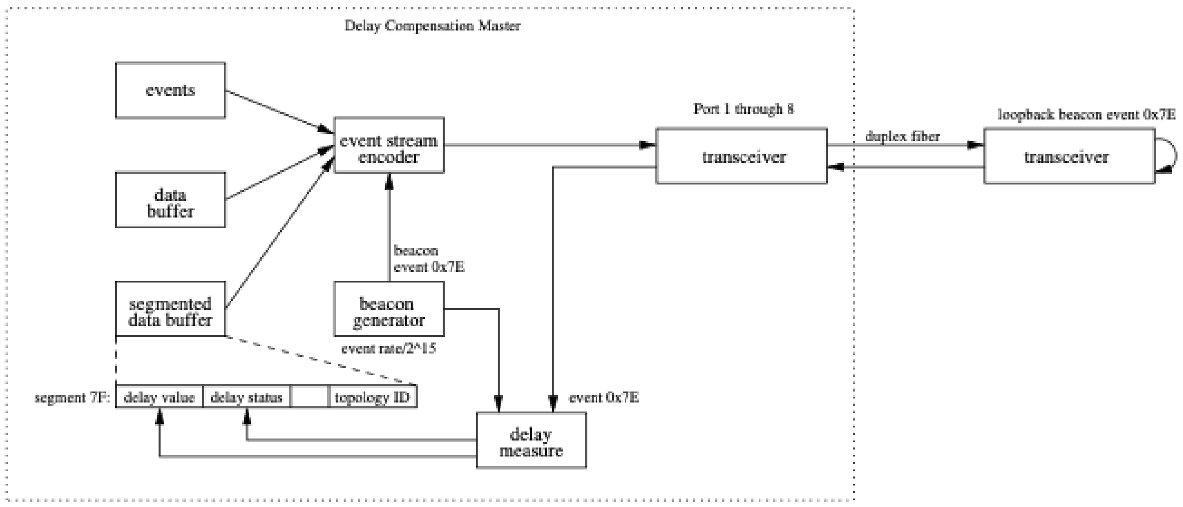


Figure 2: Timing System Master

The beacon generator sends out the beacon event (code 0x7e) at a rate of event clock/215. When the next node receives the beacon it sends it immediately back to the master which measures the propagation delay of the beacon event. The delay measurement precision improves with time and takes up to 15 minutes to stabilise. The delay value (half of the loop delay) and delay status for each SFP port is sent out using the segmented data buffer. In case the link returning the beacon (receiving side of port 1 through 8) is lost the measurement value is reset and the path delay value status is invalidated. Also if the delay value between consecutive measurements varies significantly (by more than +/- 4 event clock cycles) the delay measurement and delay value status is reset.

Timing System Fan-Out

In EVMs configured as fan-outs (DCMST = 0 and BCGEN = 0) beacons from the Timing System Master are received by the port U transceiver. The recovered event clock from the transceiver is filtered by a clock cleaner external to the FPGA. A FIFO separates the cleaned event clock domain from the recovered clock domain. The depth of the FIFO is kept constant by adjusting the phase of the cleaned clock. Beacon events are propagated through the fan-out and the propagation delay from port U to the fan-out ports 1 through 8 is measured. This delay value can be read from the IntDCValue register. Further on the beacon events get sent out on the fan-out ports and returned by the next level of fan-outs or event receivers. The loop delay for each port gets measured and the individual port delay values (half of the loop delay value) can be read from the registers Port1DCValue through Port8DCValue. Similar to the timing system master if the link returning the beacon (receiving side of port 1 through 8) is lost the measurement value is reset and the path delay value status is invalidated.

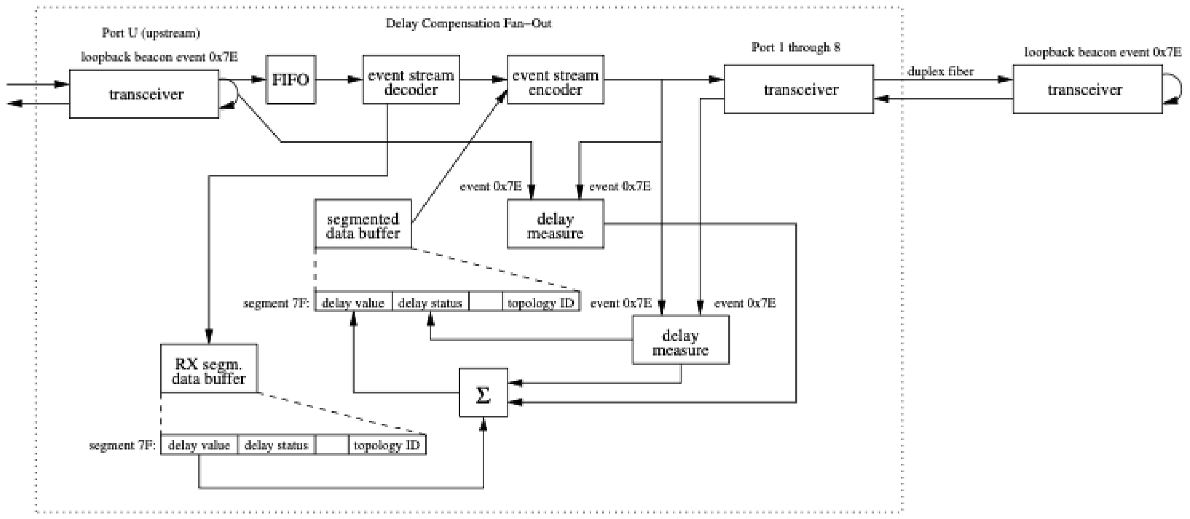


Figure: Timing System Fan-Out

The fan-out receives the delay compensation segment on the segmented data buffer. This segment contains information about the delay value from the Timing System Master up to this fan-out and the delay value quality. The intergrated delay value from the Timing System Master up to the fan-out can be retrieved from the UpDCValue register. The fan-out modifies the delay value and delay status fields in the DC segment for each port and sends out the new DC segments through ports 1 through 8.

2.7 EVG Function Register Map

The EVM module can be configured for use as an Event Generator (EVG) or a fanout/concentrator.

This is the register map when EVM is configured as an EVG.

Address	Register	Type	Description
0x000	Status	UINT32	*Status Register*
0x004	Control	UINT32	*Control Register*
0x008	IrqFlag	UINT32	*Interrupt Flag Register*
0x00C	IrqEnable	UINT32	*Interrupt Enable Register*
0x010	ACControl	UINT32	*AC divider control*
0x014	ACMap	UINT32	*AC trigger event mapping*
0x018	SWEvent	UINT32	*Software event Register*
0x01C	SegBufControl	UINT32	*Segmented Data Buffer Control Register*
0x020	DataBufControl	UINT32	*Data Buffer Control Register*
0x024	DBusMap	UINT32	*Distributed Bus Mapping Register*
0x028	DBusEvents	UINT32	*Distributed Bus Timestamping Events Register*
0x02C	FWVersion	UINT32	*Firmware Version Register*
0x034	TSControl	UINT32	*Timestamp event generator control Register*
0x038	TSValue	UINT32	Timestamp event generator value Register
0x040	FPInput	UINT32	Front Panel Input state Register
0x044	UnivInput	UINT32	Universal Input state Register
0x048	TBInput	UINT32	Transition Board Input state Register
0x04C	UsecDivider	UINT32	*Divider to get from Event Clock to 1 MHz*
0x050	ClockControl	UINT32	*Event Clock Control Register*

continues on next page

Table 2 – continued from previous page

Address	Register	Type	Description
0x060	EvanControl	UINT32	*Event Analyser Control Register*
0x064	EvanCode	UINT32	Event Analyser Distributed Bus and Event Code Register
0x068	EvanTimeHigh	UINT32	Event Analyser Time Counter (bits 63 – 32)
0x06C	EvanTimeLow	UINT32	Event Analyser Time Counter (bits 31 – 0)
0x070	SeqRamCtrl0	UINT32	*Sequence RAM 0 Control Register*
0x074	SeqRamCtrl1	UINT32	*Sequence RAM 1 Control Register*
0x080	FracDiv	UINT32	*Micrel SY87739L Fractional Divider Configuration Word*
0x0A0	SPIData	UINT32	*SPI Data Register*
0x0A4	SPIControl	UINT32	*SPI Control Register*
0x100	EvTrig0	UINT32	*Event Trigger 0 Register*
0x104	EvTrig1	UINT32	Event Trigger 1 Register
0x108	EvTrig2	UINT32	Event Trigger 2 Register
0x10C	EvTrig3	UINT32	Event Trigger 3 Register
0x110	EvTrig4	UINT32	Event Trigger 4 Register
0x114	EvTrig5	UINT32	Event Trigger 5 Register
0x118	EvTrig6	UINT32	Event Trigger 6 Register
0x11C	EvTrig7	UINT32	Event Trigger 7 Register
0x180	MXCCtrl0	UINT32	*Multiplexed Counter 0 Control Register*
0x184	MXCPresc0	UINT32	Multiplexed Counter 0 Prescaler Register
0x188	MXCCtrl1	UINT32	Multiplexed Counter 1 Control Register
0x18C	MXCPresc1	UINT32	Multiplexed Counter 1 Prescaler Register
0x190	MXCCtrl2	UINT32	Multiplexed Counter 2 Control Register
0x194	MXCPresc2	UINT32	Multiplexed Counter 2 Prescaler Register
0x198	MXCCtrl3	UINT32	Multiplexed Counter 3 Control Register
0x19C	MXCPresc3	UINT32	Multiplexed Counter 3 Prescaler Register
0x1A0	MXCCtrl4	UINT32	Multiplexed Counter 4 Control Register
0x1A4	MXCPresc4	UINT32	Multiplexed Counter 4 Prescaler Register
0x1A8	MXCCtrl5	UINT32	Multiplexed Counter 5 Control Register
0x1AC	MXCPresc5	UINT32	Multiplexed Counter 5 Prescaler Register
0x1B0	MXCCtrl6	UINT32	Multiplexed Counter 6 Control Register
0x1B4	MXCPresc6	UINT32	Multiplexed Counter 6 Prescaler Register
0x1B8	MXCCtrl7	UINT32	Multiplexed Counter 7 Control Register
0x1BC	MXCPresc7	UINT32	Multiplexed Counter 7 Prescaler Register
0x400	FPOutMap0	UINT16	Front Panel Output 0 Mapping Register
0x402	FPOutMap1	UINT16	Front Panel Output 1 Mapping Register
0x404	FPOutMap2	UINT16	Front Panel Output 2 Mapping Register
0x406	FPOutMap3	UINT16	Front Panel Output 3 Mapping Register
0x440	UnivOutMap0	UINT16	Universal Output 0 Mapping Register
0x442	UnivOutMap1	UINT16	Universal Output 1 Mapping Register
0x444	UnivOutMap2	UINT16	Universal Output 2 Mapping Register
0x446	UnivOutMap3	UINT16	Universal Output 3 Mapping Register
0x448	UnivOutMap4	UINT16	Universal Output 4 Mapping Register
0x44A	UnivOutMap5	UINT16	Universal Output 5 Mapping Register
0x44C	UnivOutMap6	UINT16	Universal Output 6 Mapping Register
0x44E	UnivOutMap7	UINT16	Universal Output 7 Mapping Register
0x450	UnivOutMap8	UINT16	Universal Output 8 Mapping Register
0x452	UnivOutMap9	UINT16	Universal Output 9 Mapping Register
0x480	TBOutMap0	UINT16	*Transition Board Output 0 Mapping Register*
0x482	TBOutMap1	UINT16	Transition Board Output 1 Mapping Register

continues on next page

Table 2 – continued from previous page

Address	Register	Type	Description
0x484	TBOutMap2	UINT16	Transition Board Output 2 Mapping Register
0x486	TBOutMap3	UINT16	Transition Board Output 3 Mapping Register
0x488	TBOutMap4	UINT16	Transition Board Output 4 Mapping Register
0x48A	TBOutMap5	UINT16	Transition Board Output 5 Mapping Register
0x48C	TBOutMap6	UINT16	Transition Board Output 6 Mapping Register
0x48E	TBOutMap7	UINT16	Transition Board Output 7 Mapping Register
0x490	TBOutMap8	UINT16	Transition Board Output 8 Mapping Register
0x492	TBOutMap9	UINT16	Transition Board Output 9 Mapping Register
0x494	TBOutMap10	UINT16	Transition Board Output 10 Mapping Register
0x496	TBOutMap11	UINT16	Transition Board Output 11 Mapping Register
0x498	TBOutMap12	UINT16	Transition Board Output 12 Mapping Register
0x49A	TBOutMap13	UINT16	Transition Board Output 13 Mapping Register
0x49C	TBOutMap14	UINT16	Transition Board Output 14 Mapping Register
0x49E	TBOutMap15	UINT16	Transition Board Output 15 Mapping Register
0x500	FPInMap0	UINT32	*Front Panel Input 0 Mapping Register*
0x504	FPInMap1	UINT32	Front Panel Input 1 Mapping Register
0x508	FPInMap2	UINT32	Front Panel Input 2 Mapping Register
0x520	FPPhMon0	UINT32	*Front Panel Input 0 Phase Monitoring Register*
0x524	FPPhMon1	UINT32	Front Panel Input 1 Phase Monitoring Register
0x528	FPPhMon2	UINT32	Front Panel Input 2 Phase Monitoring Register
0x540	UnivInMap0	UINT32	Front Panel Universal Input 0 Map Register
0x544	UnivInMap1	UINT32	Front Panel Universal Input 1 Map Register
0x548	UnivInMap2	UINT32	Front Panel Universal Input 2 Map Register
0x54C	UnivInMap3	UINT32	Front Panel Universal Input 3 Map Register
0x550	UnivInMap4	UINT32	Front Panel Universal Input 4 Map Register
0x554	UnivInMap5	UINT32	Front Panel Universal Input 5 Map Register
0x558	UnivInMap6	UINT32	Front Panel Universal Input 6 Map Register
0x55C	UnivInMap7	UINT32	Front Panel Universal Input 7 Map Register
0x560	UnivInMap8	UINT32	Front Panel Universal Input 8 Map Register
0x564	UnivInMap9	UINT32	Front Panel Universal Input 9 Map Register
0x600	TBInMap0	UINT32	*Transition Board Input 0 Mapping Register*
0x604	TBInMap1	UINT32	Transition Board Input 1 Mapping Register
0x608	TBInMap2	UINT32	Transition Board Input 2 Mapping Register
0x60C	TBInMap3	UINT32	Transition Board Input 3 Mapping Register
0x610	TBInMap4	UINT32	Transition Board Input 4 Mapping Register
0x614	TBInMap5	UINT32	Transition Board Input 5 Mapping Register
0x618	TBInMap6	UINT32	Transition Board Input 6 Mapping Register
0x61C	TBInMap7	UINT32	Transition Board Input 7 Mapping Register
0x620	TBInMap8	UINT32	Transition Board Input 8 Mapping Register
0x624	TBInMap9	UINT32	Transition Board Input 9 Mapping Register
0x628	TBInMap10	UINT32	Transition Board Input 10 Mapping Register
0x62C	TBInMap11	UINT32	Transition Board Input 11 Mapping Register
0x630	TBInMap12	UINT32	Transition Board Input 12 Mapping Register
0x634	TBInMap13	UINT32	Transition Board Input 13 Mapping Register
0x638	TBInMap14	UINT32	Transition Board Input 14 Mapping Register
0x63C	TBInMap15	UINT32	Transition Board Input 15 Mapping Register
0x800 – 0xFFF	DataBuf		Data Buffer Transmit Memory
0x1000 – 0x10FF	configROM		
0x1100 – 0x11FF	scratchRAM		

continues on next page

Table 2 – continued from previous page

Address	Register	Type	Description
0x1200 – 0x12FF	SFPEEPROM		Upstream SFP Transceiver EEPROM contents (SFP address 0xA0)
0x1300 – 0x13FF	SFPDIAG		Upstream SFP Transceiver diagnostics (SFP address 0xA2)
0x2000 – 0x27FF	SegBuf		Segmented Data Buffer Transmit Memory
0x8000 – 0xBFFF	SeqRam0		Sequence RAM 0
0xC000 – 0xFFFF	SeqRam1		Sequence RAM 1

2.7.1 Register descriptions

Status Register

address	bit 31	bit 30	bit 29	bit 28	bit 27	bit 26	bit 25	bit 24
0x000	RDB7	RDB6	RDB5	RDB4	RDB3	RDB2	RDB1	RDB0
	bit 23	**bit 22**	**bit 21**	**bit 20**	**bit 19**	**bit 18**	**bit 17**	**bit 16**
0x001	TDB7	TDB6	TDB5	TDB4	TDB3	TDB2	TDB1	TDB0

Bit	Function
RDB7	Status of received distributed bus bit 7 (from upstream EVG)
RDB6	Status of received distributed bus bit 6 (from upstream EVG)
RDB5	Status of received distributed bus bit 5 (from upstream EVG)
RDB4	Status of received distributed bus bit 4 (from upstream EVG)
RDB3	Status of received distributed bus bit 3 (from upstream EVG)
RDB2	Status of received distributed bus bit 2 (from upstream EVG)
RDB1	Status of received distributed bus bit 1 (from upstream EVG)
RDB0	Status of received distributed bus bit 0 (from upstream EVG)
TDB7	Status of transmitted distributed bus bit 7
TDB6	Status of transmitted distributed bus bit 6
TDB5	Status of transmitted distributed bus bit 5
TDB4	Status of transmitted distributed bus bit 4
TDB3	Status of transmitted distributed bus bit 3
TDB2	Status of transmitted distributed bus bit 2
TDB1	Status of transmitted distributed bus bit 1
TDB0	Status of transmitted distributed bus bit 0

Control Register

address	bit 31	bit 30	bit 29	bit 28	bit 27	bit 26	bit 25	bit 24
0x004	EVGEN	RXDIS	RXPWD	FIFORS		SRST	LEMDE	MXCRES
	bit 23	**bit 22**	**bit 21**	**bit 20**	**bit 19**	**bit 18**	**bit 17**	**bit 16**
0x005	BCGEN	DCMST						SRALT

Bit	Function
EVGEN	Event Generator Master enable
RXDIS	Disable event reception
RXPWD	Receiver Power down
FIFORS	Reset RX Event Fifo
SRST	Soft reset IP
LEMDE	Little endian mode (cPCI-EVG-300)
	0 – PCI core in big endian mode (power up default)
	1 – PCI core in little endian mode
MXCRES	Write 1 to reset multiplexed counters
BCGEN	Delay Compensation Beacon generator enable
	0 – Beacon generator disabled
	1 – Beacon generator enabled, sends out 0x7E events and delay compensation
	data. Set only for master EVG in system
DCMST	System Master enable
	0 – System Master disabled
	1 – System Master enabled – has to be set and only for master EVG in system
SRALT	(reserved)

Interrupt Flag Register

address	bit 31	bit 30	bit 29	bit 28	bit 27	bit 26	bit 25	bit 24
0x008								
	bit 23	**bit 22**	**bit 21**	**bit 20**	**bit 19**	**bit 18**	**bit 17**	**bit 16**
0x009			IFSOV1	IFSOV0			IFSHF1	IFSHF0
	bit 15	bit 14	bit 13	bit 12	bit 11	bit 10	bit 9	bit 8
0x00A			IFSSTO1	IFSSTO0			IFSSTA1	IFSSTA0
	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
0x00B		IFEXT	IFDBUF				IFFF	IFVIO

Bit	Function
IFSOV1	Sequence RAM 1 sequence roll over interrupt flag
IFSOV0	Sequence RAM 0 sequence roll over interrupt flag
IFSHF1	Sequence RAM 1 sequence halfway through interrupt flag
IFSHF0	Sequence RAM 0 sequence halfway through interrupt flag
IFSSTO1	Sequence RAM 1 sequence stop interrupt flag
IFSSTO0	Sequence RAM 0 sequence stop interrupt flag
IFSSTA1	Sequence RAM 1 sequence start interrupt flag
IFSSTA0	Sequence RAM 0 sequence start interrupt flag
IFEXT	External Interrupt flag
IFDBUF	Data buffer flag
IFFF	RX Event FIFO full flag
IFVIO	Port U Receiver violation flag

Interrupt Enable Register

address	bit 31	bit 30	bit 29	bit 28	bit 27	bit 26	bit 25	bit 24
0x00C	IRQEN	PCIIE						
	bit 23	bit 22	bit 21	bit 20	bit 19	bit 18	bit 17	bit 16
0x00D			IESOV1	IESOV0			IESHF1	IESHF0
	bit 15	bit 14	bit 13	bit 12	bit 11	bit 10	bit 9	bit 8
0x00E			IESSTO1	IESSTO0			IESSTA1	IESSTA0
	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
0x00F		IEEXT	IEDBUF				IEFF	IEVIO

Bit	Function
IRQEN	Master interrupt enable
PCIIE	PCI core interrupt enable (cPCI-EVG-300)
	This bit is used by the low level driver to disable further interrupts before the
	first interrupt has been handled in user space
IESOV1	Sequence RAM 1 sequence roll over interrupt enable
IESOV0	Sequence RAM 0 sequence roll over interrupt enable
IESHF1	Sequence RAM 1 sequence halfway through interrupt enable
IESHF0	Sequence RAM 0 sequence halfway through interrupt enable
IESSTO1	Sequence RAM 1 sequence stop interrupt enable
IESSTO0	Sequence RAM 0 sequence stop interrupt enable
IESSTA1	Sequence RAM 1 sequence start interrupt enable
IESSTA0	Sequence RAM 0 sequence start interrupt enable
IEEXT	External interrupt enable
IEDBUF	Data buffer interrupt enable
IEFF	Event FIFO full interrupt enable
IEVIO	Receiver violation interrupt enable

AC Trigger Control Register

address	bit 23	bit 22	bit 21	bit 20	bit 19	bit 18	bit 17	bit 16
0x011					ACSYN2	ACSYN1	ACBYP	ACSYN0
	bit 15	bit 14	bit 13	bit 12	bit 11	bit 10	bit 9	bit 8
0x012	ACDIV7	ACDIV6	ACDIV5	ACDIV4	ACDIV3	ACDIV2	ACDIV1	ACDIV0
	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
0x013	ACPS7	ACPS6	ACPS5	ACPS4	ACPS3	ACPS2	ACPS1	ACPS0

Bit	Function
ACBYP	AC divider and phase shifter bypass (0 = divider/phase shifter enabled, 1 =
	divider/phase shifter bypassed)
ACSYN(2:0)	Synchronization select
	000 = Event clock
	001 = Multiplexed counter 7 output
	011 = Front panel TTL input IN1
	101 = Front panel TTL input IN2
ACDIV(7:0)	AC Trigger divider (8-bit value)
ACPS(7:0)	AC Trigger Phase shift (8-bit value)

AC Trigger Mapping Register

address	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
0x017	ACM7	ACM6	ACM5	ACM4	ACM3	ACM2	ACM1	ACM0

Bit	Function
ACM7	If set AC circuit triggers Event Trigger 7
ACM6	If set AC circuit triggers Event Trigger 6
ACM5	If set AC circuit triggers Event Trigger 5
ACM4	If set AC circuit triggers Event Trigger 4
ACM3	If set AC circuit triggers Event Trigger 3
ACM2	If set AC circuit triggers Event Trigger 2
ACM1	If set AC circuit triggers Event Trigger 1
ACM0	If set AC circuit triggers Event Trigger 0

Software Event Register

address	bit 15	bit 14	bit 13	bit 12	bit 11	bit 10	bit 9	bit 8
0x01A							SWPEND	SWENA
	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
0x01B	CODE7	CODE6	CODE5	CODE4	CODE3	CODE2	CODE1	CODE0

Bit	Function
SWPEND	Event code waiting to be sent out (read-only). A new event code may be written
	to the event code register when this bit reads ‘0’.
SWENA	Enable software event
	When enabled ‘1’ a new event will be sent out when event code is written to
	the event code register.
CODE(7:0)	Event Code (8-bit value)

Segmented Data Buffer Control Register

address	bit 31	bit 30	bit 29	bit 28	bit 27	bit 26	bit 25	bit 24
0x01C	SADDR(7)	SADDR(6)	SADDR(5)	SADDR(4)	SADDR(3)	SADDR(2)	SADDR(1)	SADDR(0)
	bit 23	bit 22	bit 21	bit 20	bit 19	bit 18	bit 17	bit 16
0x01D				TXCPT	TXRUN	TRIG	ENA	1
	bit 15	bit 14	bit 13	bit 12	bit 11	bit 10	bit 9	bit 8
0x01E						DTSZ(10)	DTSZ(9)	DTSZ(8)
	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
0x01F	DTSZ(7)	DTSZ(6)	DTSZ(5)	DTSZ(4)	DTSZ(3)	DTSZ(2)	0	0

Bit	Function
SADDR	Transfer Start Segment Address (16 byte segments)
TXCPT	Data Buffer Transmission Complete
TXRUN	Data Buffer Transmission Running – set when data transmission has been
	triggered and has not been completed yet
TRIG	Data Buffer Trigger Transmission
	Write ‘1’ to start transmission of data in buffer
ENA	Data Buffer Transmission enable
	‘0’ – data transmission engine disabled
	‘1’ – data transmission engine enabled
DTSZ(10:8)	Data Transfer size 4 bytes to 2k in four byte increments

Data Buffer Control Register

address	bit 31	bit 30	bit 29	bit 28	bit 27	bit 26	bit 25	bit 24
0x020								
	bit 23	bit 22	bit 21	bit 20	bit 19	bit 18	bit 17	bit 16
0x021				TXCPT	TXRUN	TRIG	ENA	1
	bit 15	bit 14	bit 13	bit 12	bit 11	bit 10	bit 9	bit 8
0x022						DTSZ(10)	DTSZ(9)	DTSZ(8)
	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
0x023	DTSZ(7)	DTSZ(6)	DTSZ(5)	DTSZ(4)	DTSZ(3)	DTSZ(2)	0	0

Bit	Function
TXCPT	Data Buffer Transmission Complete
TXRUN	Data Buffer Transmission Running – set when data transmission has been
	triggered and has not been completed yet
TRIG	Data Buffer Trigger Transmission
	Write ‘1’ to start transmission of data in buffer
ENA	Data Buffer Transmission enable
	‘0’ – data transmission engine disabled
	‘1’ – data transmission engine enabled
DTSZ	Data Transfer size 4 bytes to 2k in four byte increments

Distributed Bus Mapping Register

address	bit 31	bit 30	bit 29	bit 28	bit 27	bit 26	bit 25	bit 24
0x024	DBMAP7(3)	DBMAP7(2)	DBMAP7(1)	DBMAP7(0)	DBMAP6(3)	DBMAP6(2)	DBMAP6(1)	DBMAP6(0)
	bit 23	bit 22	bit 21	bit 20	bit 19	bit 18	bit 17	bit 16
0x025	DBMAP5(3)	DBMAP5(2)	DBMAP5(1)	DBMAP5(0)	DBMAP4(3)	DBMAP4(2)	DBMAP4(1)	DBMAP4(0)
	bit 15	bit 14	bit 13	bit 12	bit 11	bit 10	bit 9	bit 8
0x026	DBMAP3(3)	DBMAP3(2)	DBMAP3(1)	DBMAP3(0)	DBMAP2(3)	DBMAP2(2)	DBMAP2(1)	DBMAP2(0)
	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
0x027	DBMAP1(3)	DBMAP1(2)	DBMAP1(1)	DBMAP1(0)	DBMAP0(3)	DBMAP0(2)	DBMAP0(1)	DBMAP0(0)

Bit	Function
DBMAP7(3:0)	Distributed Bus Bit 7 Mapping:
	0 – Off, output logic ‘0’
	1 – take bus bit from external input
	2 – Multiplexed counter output mapped to distributed bus bit
	3 – Distributed bus bit forwarded from upstream EVG
DBMAP6(3:0)	Distributed Bus Bit 7 Mapping (see above for mappings)
DBMAP5(3:0)	Distributed Bus Bit 7 Mapping (see above for mappings)
DBMAP4(3:0)	Distributed Bus Bit 7 Mapping (see above for mappings)
DBMAP3(3:0)	Distributed Bus Bit 7 Mapping (see above for mappings)
DBMAP2(3:0)	Distributed Bus Bit 7 Mapping (see above for mappings)
DBMAP1(3:0)	Distributed Bus Bit 7 Mapping (see above for mappings)
DBMAP0(3:0)	Distributed Bus Bit 7 Mapping (see above for mappings)

Distributed Bus Event Enable Register

address	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
0x02B	DBEV7	DBEV6	DBEV5					

Bit	Function
DBEV5	Distributed bus input 5 “Timestamp reset” 0x7D event enable
DBEV6	Distributed bus input 6 “Seconds ‘0’” 0x70 event enable
DBEV7	Distributed bus input 7 “Seconds ‘1’” 0x71 event enable

FPGA Firmware Version Register

address	bit 31	bit 30	bit 29	bit 28	bit 27	bit 26	bit 25	bit 24
0x02C	0	0	1	0	FF3	FF2	FF1	FF0
	bit 23	bit 22	bit 21	bit 20	bit 19	bit 18	bit 17	bit 16
0x02D	Subrelease ID							
	bit 15	bit 14	bit 13	bit 12	bit 11	bit 10	bit 9	bit 8
0x02E	Firmware ID							
	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
0x02F	Revision ID							

Bit	Function
Form Factor FF(0:3)	
	0 – CompactPCI 3U
	1 – PMC
	2 – VME64x
	3 – CompactRIO
	4 – CompactPCI 6U
	6 – PXIe
	7 – PCIe
	8 – mTCA.4

Bit	Function
Subrelease ID	For production releases the subrelease ID counts up from 00.
	For pre-releases this ID is used “backwards” counting down from ff i.e. when
	approacing release 22000207, we have prereleases 22FF0206, 22FE0206,
	22FD0206 etc. in this order.
Firmware ID	00 – Modular Register Map firmware (no delay compensation)
	01 – Reserved
	02 – Delay Compensation firmware
Revision ID	See end of manual

Timestamp Generator Control Register

address	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
0x037							TSGENA	TSGLOAD

Bit	Function
TSGENA	Timestamp Generator Enable (‘0’ = disable, ‘1’ = enable)
TSGLOAD	Timestamp Generator Load new value into Timestamp Counter
	Write ‘1’ to load new value

Microsecond Divider Register

address	bit 15 .. bit 0
0x04E	Rounded integer value of 1s * event clock

This register shall be written with an integer value of the event clock rate in MHz. For 100 MHz event clock this register should read 100, for 50 MHz event clock this register should read 50. This value is used to set the parameters for the clock cleaner PLL and e.g. for the phase shifter in the AC input logic.

Clock Control Register

address	bit 31	bit 30	bit 29	bit 28	bit 27	bit 26	bit 25	bit 24
0x050	PLLLOCK	BWSEL2	BWSEL1	BWSEL0		RFSEL2	RFSEL1	RFSEL0
	bit 23	bit 22	bit 21	bit 20	bit 19	bit 18	bit 17	bit 16
0x051	PHTOGG		RFDIV5	RFDIV4	RFDIV3	RFDIV2	RFDIV1	RFDIV0
	bit 15	bit 14	bit 13	bit 12	bit 11	bit 10	bit 9	bit 8
0x052							CGLOCK	

Bit	Function
PLLLOCK	Clock cleaner locked
BWSEL2:0	PLL Bandwidth Select (see Silicon Labs Si5317 datasheet)
	000 – Si5317, BW setting HM (lowest loop bandwidth)
	001 – Si5317, BW setting HL
	010 – Si5317, BW setting MH
	011 – Si5317, BW setting MM
	100 – Si5317, BW setting ML (highest loop bandwidth)
PHTOGG	Distributed bus phase toggle
RFDIV5-0	External RF divider select:
	000000 – RF/1
	000001 – RF/2
	000010 – RF/3
	000011 – RF/4
	000100 – RF/5
	000101 – RF/6
	000110 – RF/7
	000111 – RF/8
	001000 – RF/9
	001001 – RF/10
	001010 – RF/11
	001011 – RF/12
	001100 – OFF
	001101 – RF/14
	001110 – RF/15
	001111 – RF/16
	010000 – RF/17
	010001 – RF/18
	010010 – RF/19
	010011 – RF/20
	010100 – RF/21
	010101 – RF/22
	010110 – RF/23
	010111 – RF/24
	011000 – RF/25
	011001 – RF/26

continues on next page

Table 3 – continued from previous page

Bit	Function
	011010 – RF/27
	011011 – RF/28
	011100 – RF/29
	011101 – RF/30
	011110 – RF/31
	011111 – RF/32
RFSEL2-0	RF reference select:
	000 – Use internal reference (fractional synthesizer)
	001 – Use external RF reference (front panel input through divider)
	010 – PXIe 100 MHz clock
	100 – Use recovered RX clock, Fan-Out mode
	101 – Use external RF reference for downstream ports, internal reference for
	upstream port, Fan-Out mode, event rate down conversion
	110 – PXIe 10 MHz clock through clock multiplier
	111 – Recovered clock /2 decimate mode, event rate is halved
CGLOCK	Micrel SY87739L reference clock locked (read-only)

Note: Please note that after changing the Event clock source the fractional synthesizer control word must be reloaded to initialize an internal reset.

Event Analyser Control Register

address	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
0x063				EVANE	EVARS	EVAOF	EVAEN	EVACR

Bit	Function
EVANE	Event Analyser FIFO not empty flag:
	0 – FIFO empty
	1 – FIFO not empty, events in FIFO
EVARS	Event Analyser Reset
	0 – not in reset
	1 – reset
EVAOF	Event Analyser FIFO overflow flag:
	0 – no overflow
	1 – FIFO overflow
EVAEN	Event Analyser enable
	0 – Event Analyser disabled
	1 – Event Analyser enabled
EVACR	Event Analyser 64 bit counter reset
	0 – Counter running
	1 – Counter reset to zero.

Sequence RAM Control Registers

address	bit 31	bit 30	bit 29	bit 28	bit 27	bit 26	bit 25	bit 24
0x070							SQ0RUN	SQ0ENA
	bit 23	bit 22	bit 21	bit 20	bit 19	bit 18	bit 17	bit 16
0x071	SQ0XTR	SQ0XEN	SQ0SWT	SQ0SNG	SQ0REC	SQ0RES	SQ0DIS	SQ0EN
	bit 15	bit 14	bit 13	bit 12	bit 11	bit 10	bit 9	bit 8
0x072	SQSWMASK3	SQSWMASK2	SQSWMASK1	SQSWMASK0	SQSWENA3	SQSWENA2	SQSWENA1	SQSWENA0
	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
0x073	SQ0TSEL7	SQ0TSEL6	SQ0TSEL5	SQ0TSEL4	SQ0TSEL3	SQ0TSEL2	SQ0TSEL1	SQ0TSEL0

address	bit 31	bit 30	bit 29	bit 28	bit 27	bit 26	bit 25	bit 24
0x074							SQ1RUN	SQ1ENA
	bit 23	bit 22	bit 21	bit 20	bit 19	bit 18	bit 17	bit 16
0x075	SQ1XTR	SQ1XEN	SQ1SWT	SQ1SNG	SQ1REC	SQ1RES	SQ1DIS	SQ1EN
	bit 15	bit 14	bit 13	bit 12	bit 11	bit 10	bit 9	bit 8
0x076	SQSWMASK3	SQSWMASK2	SQSWMASK1	SQSWMASK0	SQSWENA3	SQSWENA2	SQSWENA1	SQSWENA0
	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
0x077	SQ1TSEL7	SQ1TSEL6	SQ1TSEL5	SQ1TSEL4	SQ1TSEL3	SQ1TSEL2	SQ1TSEL1	SQ1TSEL0

Bit	Function
SQxRUN	Sequence RAM running flag (read-only)
SQxENA	Sequence RAM enabled flag (read_only)
SQxSWT	Sequence RAM software trigger, write ‘1’ to trigger
SQxSNG	Sequence RAM single mode
SQxREC	Sequence RAM recycle mode
SQxRES	Sequence RAM reset, write ‘1’ to reset
SQxDIS	Sequence RAM disable, write ‘1’ to disable
SQxEN	Sequence RAM enable, write ‘1’ to enable/arm
SQxXEN	Sequence RAM allow external enable, ‘1’ - allow
SQxXTR	Sequence RAM allow external trigger enable, ‘1’ - allow
SQSWMASK	Sequence RAM SW mask register, the mask bits are common for all RAMS
SQSWENA	Sequence RAM SW enable register, the mask bits are common for all RAMS
SQxTSEL	Sequence RAM trigger select:
	0 – trigger from MXC0
	1 – trigger from MXC1
	2 – trigger from MXC2
	3 – trigger from MXC3
	4 – trigger from MXC4
	5 – trigger from MXC5
	6 – trigger from MXC6
	7 – trigger from MXC7
	16 – trigger from AC synchronization logic
	17 – trigger from sequence RAM 0 software trigger
	18 – trigger from sequence RAM 1 software trigger
	19 – trigger always immediately when enabled
	24 – trigger from sequence RAM 0 external trigger
	25 – trigger from sequence RAM 1 external trigger
	31 – trigger disabled (default after power up)

SY87739L Fractional Divider Configuration Word

Configuration Word	Frequency with 24 MHz reference oscillator
0x0891C100	142.857 MHz
0x00DE816D	125 MHz
0x00FE816D	124.95 MHz
0x0C928166	124.9087 MHz
0x018741AD	119 MHz
0x072F01AD	114.24 MHz
0x049E81AD	106.25 MHz
0x008201AD	100 MHz
0x025B41ED	99.956 MHz
0x0187422D	89.25 MHz
0x0082822D	81 MHz
0x0106822D	80 MHz
0x019E822D	78.900 MHz
0x018742AD	71.4 MHz
0x0C9282A6	62.454 MHz
0x009743AD	50 MHz
0x0C25B43AD	49.978 MHz
0x0176C36D	49.965 MHz

SPI Configuration Flash Registers

address	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
0x0A3	SPI-DATA7	SPI-DATA6	SPI-DATA5	SPI-DATA4	SPI-DATA3	SPI-DATA2	SPI-DATA1	SPI-DATA0
	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
0x0A7	E	RRDY	TRDY	TMT	TOE	ROE	OE	SSO

Bit	Function
SPIDATA(7:0)	Read SPI data byte / Write SPI data byte
E	Overrun Error flag
RRDY	Receiver ready, if ‘1’ data byte waiting in SPI_DATA
TRDY	Transmitter ready, if ‘1’ SPI_DATA is ready to accept new transmit data byte
TMT	Transmitter empty, if ‘1’ data byte has been transmitted
TOE	Transmitter overrun error
ROE	Receiver overrun error
OE	Output enable for SPI pins, ‘1’ enable SPI pins
SSO	Slave select output enable for SPI slave device, ‘1’ device selected

Event Trigger Registers

address	bit 15	bit 14	bit 13	bit 12	bit 11	bit 10	bit 9	bit 8
0x102								EVEN0
	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
0x103	EVCD0(7)	EVCD0(6)	EVCD0(5)	EVCD0(4)	EVCD0(3)	EVCD0(2)	EVCD0(1)	EVCD0(0)
	bit 15	bit 14	bit 13	bit 12	bit 11	bit 10	bit 9	bit 8
0x106								EVEN1
	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
0x107	EVCD1(7)	EVCD1(6)	EVCD1(5)	EVCD1(4)	EVCD1(3)	EVCD1(2)	EVCD1(1)	EVCD1(0)
	bit 15	bit 14	bit 13	bit 12	bit 11	bit 10	bit 9	bit 8
0x10A								EVEN2
	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
0x10B	EVCD2(7)	EVCD2(6)	EVCD2(5)	EVCD2(4)	EVCD2(3)	EVCD2(2)	EVCD2(1)	EVCD2(0)
	bit 15	bit 14	bit 13	bit 12	bit 11	bit 10	bit 9	bit 8
0x10E								EVEN3
	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
0x10F	EVCD3(7)	EVCD3(6)	EVCD3(5)	EVCD3(4)	EVCD3(3)	EVCD3(2)	EVCD3(1)	EVCD3(0)
	bit 15	bit 14	bit 13	bit 12	bit 11	bit 10	bit 9	bit 8
0x112								EVEN4
	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
0x113	EVCD4(7)	EVCD4(6)	EVCD4(5)	EVCD4(4)	EVCD4(3)	EVCD4(2)	EVCD4(1)	EVCD4(0)
	bit 15	bit 14	bit 13	bit 12	bit 11	bit 10	bit 9	bit 8
0x116								EVEN5
	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
0x117	EVCD5(7)	EVCD5(6)	EVCD5(5)	EVCD5(4)	EVCD5(3)	EVCD5(2)	EVCD5(1)	EVCD5(0)
	bit 15	bit 14	bit 13	bit 12	bit 11	bit 10	bit 9	bit 8
0x11A								EVEN6
	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
0x11B	EVCD6(7)	EVCD6(6)	EVCD6(5)	EVCD6(4)	EVCD6(3)	EVCD6(2)	EVCD6(1)	EVCD6(0)
	bit 15	bit 14	bit 13	bit 12	bit 11	bit 10	bit 9	bit 8
0x11E								EVEN7
	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
0x11F	EVCD7(7)	EVCD7(6)	EVCD7(5)	EVCD7(4)	EVCD7(3)	EVCD7(2)	EVCD7(1)	EVCD7(0)

Bit	Function
EVENx	Enable Event Trigger x
EVCDx	Event Trigger Code for Event trigger x

Multiplexed Counter Registers

address	bit 31	bit 30	bit 29	bit 28	bit 27	bit 26	bit 25	bit 24
0x180	MXC0	MXCP0						
	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
0x183	MX0EV7	MX0EV6	MX0EV5	MX0EV4	MX0EV3	MX0EV2	MX0EV1	MX0EV
	bit 31	bit 0						
0x184	Multiplexed Counter 0 Prescaler							
	bit 31	bit 30	bit 29	bit 28	bit 27	bit 26	bit 25	bit 24

continues on next pag

Table 5 – continued from previous page

address	bit 31	bit 30	bit 29	bit 28	bit 27	bit 26	bit 25	bit 24
0x188	MXC1	MXCP1						
	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
0x18B	MX1EV7	MX1EV6	MX1EV5	MX1EV4	MX1EV3	MX1EV2	MX1EV1	MX1EV
	bit 31	bit 0						
0x18C	Multiplexed Counter 1 Prescaler							
	bit 31	bit 30	bit 29	bit 28	bit 27	bit 26	bit 25	bit 24
0x190	MXC2	MXCP2						
	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
0x193	MX2EV7	MX2EV6	MX2EV5	MX2EV4	MX2EV3	MX2EV2	MX2EV1	MX2EV
	bit 31	bit 0						
0x194	Multiplexed Counter 2 Prescaler							
	bit 31	bit 30	bit 29	bit 28	bit 27	bit 26	bit 25	bit 24
0x198	MXC3	MXCP3						
	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
0x19B	MX3EV7	MX3EV6	MX3EV5	MX3EV4	MX3EV3	MX3EV2	MX3EV1	MX3EV
	bit 31	bit 0						
0x19C	Multiplexed Counter 3 Prescaler							
	bit 31	bit 30	bit 29	bit 28	bit 27	bit 26	bit 25	bit 24
0x1A0	MXC4	MXCP4						
	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
0x1A3	MX4EV7	MX4EV6	MX4EV5	MX4EV4	MX4EV3	MX4EV2	MX4EV1	MX4EV
	bit 31	bit 0						
0x1A4	Multiplexed Counter 4 Prescaler							
	bit 31	bit 30	bit 29	bit 28	bit 27	bit 26	bit 25	bit 24
0x1A8	MXC5	MXCP5						
	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
0x1AB	MX5EV7	MX5EV6	MX5EV5	MX5EV4	MX5EV3	MX5EV2	MX5EV1	MX5EV
	bit 31	bit 0						
0x1AC	Multiplexed Counter 5 Prescaler							
	bit 31	bit 30	bit 29	bit 28	bit 27	bit 26	bit 25	bit 24
0x1B0	MXC6	MXCP6						
	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
0x1B3	MX6EV7	MX6EV6	MX6EV5	MX6EV4	MX6EV3	MX6EV2	MX6EV1	MX6EV
	bit 31	bit 0						
0x1B4	Multiplexed Counter 6 Prescaler							
	bit 31	bit 30	bit 29	bit 28	bit 27	bit 26	bit 25	bit 24
0x1B8	MXC7	MXCP7						
	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
0x1BB	MX7EV7	MX7EV6	MX7EV5	MX7EV4	MX7EV3	MX7EV2	MX7EV1	MX7EV
	bit 31	bit 0						
0x1BC	Multiplexed Counter 7 Prescaler							

Bit	Function
MXCx	Multiplexed counter output status (read-only)
MXPx	Multiplexed counter output polarity
MXxEV7	Map rising edge of multiplexed counter x to send out event trigger 7
MXxEV6	Map rising edge of multiplexed counter x to send out event trigger 6
MXxEV5	Map rising edge of multiplexed counter x to send out event trigger 5
MXxEV4	Map rising edge of multiplexed counter x to send out event trigger 4
MXxEV3	Map rising edge of multiplexed counter x to send out event trigger 3
MXxEV2	Map rising edge of multiplexed counter x to send out event trigger 2
MXxEV1	Map rising edge of multiplexed counter x to send out event trigger 1
MXxEV0	Map rising edge of multiplexed counter x to send out event trigger 0

Transition Board Output Mapping Registers

address	bit 15 to bit 0
0x480	Transition Board Output 0 Mapping ID (see *the table for mapping IDs*)
0x482	Transition Board Output 1 Mapping ID
…	
0x41E	Transition Board Output 15 Mapping ID

Front Panel Input Mapping Registers

address	bit 31	bit 30	bit 29	bit 28	bit 27	bit 26	bit 25	bit 24
0x500	FP0SQMK3	FP0SQMK2	FP0SQMK1	FP0SQMK0	FP0SQEEN3	FP0SQEEN2	FP0SQEEN1	FP0SQEEN0/FP0IRQ
	bit 23	bit 22	bit 21	bit 20	bit 19	bit 18	bit 17	bit 16
0x501	FP0DB7	FP0DB6	FP0DB5	FP0DB4	FP0DB3	FP0DB2	FP0DB1	FP0DB0
	bit 15	bit 14	bit 13	bit 12	bit 11	bit 10	bit 9	bit 8
0x502			FP0SEN1	FP0SEN0			FP0SEQ1	FP0SEQ0
	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
0x503	FP0EV7	FP0EV6	FP0EV5	FP0EV4	FP0EV3	FP0EV2	FP0EV1	FP0EV0
	bit 31	bit 30	bit 29	bit 28	bit 27	bit 26	bit 25	bit 24
0x504	FP1SQMK3	FP1SQMK2	FP1SQMK1	FP1SQMK0	FP1SQEEN3	FP1SQEEN2	FP1SQEEN1	FP1SQEEN0/FP1IRQ
	bit 23	bit 22	bit 21	bit 20	bit 19	bit 18	bit 17	bit 16
0x505	FP1DB7	FP1DB6	FP1DB5	FP1DB4	FP1DB3	FP1DB2	FP1DB1	FP1DB0
	bit 15	bit 14	bit 13	bit 12	bit 11	bit 10	bit 9	bit 8
0x506			FP1SEN1	FP1SEN0			FP1SEQ1	FP1SEQ0
	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
0x507	FP1EV7	FP1EV6	FP1EV5	FP1EV4	FP1EV3	FP1EV2	FP1EV1	FP1EV0
	bit 31	bit 30	bit 29	bit 28	bit 27	bit 26	bit 25	bit 24
0x508	FP2SQMK3	FP2SQMK2	FP2SQMK1	FP2SQMK0	FP2SQEEN3	FP2SQEEN2	FP2SQEEN1	FP2SQEEN0/FP2IRQ
	bit 23	bit 22	bit 21	bit 20	bit 19	bit 18	bit 17	bit 16
0x509	FP2DB7	FP2DB6	FP2DB5	FP2DB4	FP2DB3	FP2DB2	FP2DB1	FP2DB0
	bit 15	bit 14	bit 13	bit 12	bit 11	bit 10	bit 9	bit 8
0x50A			FP2SEN1	FP2SEN0			FP2SEQ1	FP2SEQ0
	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
0x50B	FP2EV7	FP2EV6	FP2EV5	FP2EV4	FP2EV3	FP2EV2	FP2EV1	FP2EV0

Bit	Function
FPxSQMKy	Map Front panel Input x to Sequence Event Mask bit y
FPxSQEENy	Map Front panel Input x to Sequence Event Enable bit y
FPxIRQ	Map Front panel Input x to External Interrupt
FPxDB7	Map Front panel Input x to Distributed Bus bit 7
FPxDB6	Map Front panel Input x to Distributed Bus bit 6
FPxDB5	Map Front panel Input x to Distributed Bus bit 5
FPxDB4	Map Front panel Input x to Distributed Bus bit 4
FPxDB3	Map Front panel Input x to Distributed Bus bit 3
FPxDB2	Map Front panel Input x to Distributed Bus bit 2
FPxDB1	Map Front panel Input x to Distributed Bus bit 1
FPxDB0	Map Front panel Input x to Distributed Bus bit 0
FPxSEN1	Map Front panel Input x to Sequence External Enable 1
FPxSEN0	Map Front panel Input x to Sequence External Enable 0
FPxSEQ1	Map Front panel Input x to Sequence Trigger 1
FPxSEQ0	Map Front panel Input x to Sequence Trigger 0
FPxEV7	Map Front panel Input x to Event Trigger 7
FPxEV6	Map Front panel Input x to Event Trigger 6
FPxEV5	Map Front panel Input x to Event Trigger 5
FPxEV4	Map Front panel Input x to Event Trigger 4
FPxEV3	Map Front panel Input x to Event Trigger 3
FPxEV2	Map Front panel Input x to Event Trigger 2
FPxEV1	Map Front panel Input x to Event Trigger 1
FPxEV0	Map Front panel Input x to Event Trigger 0

2.7.2 Front Panel Input Phase Monitoring Registers

address	bit 31	bit 30	bit 29	bit 28	bit 27	bit 26	bit 25	bit 24
0x520	PHCLR0	DBPH0					PHSEL0(1)	PHSEL0(0)
	bit 15	bit 14	bit 13	bit 12	bit 11	bit 10	bit 9	bit 8
0x522					PHFE0(3)	PHFE0(2)	PHFE0(1)	PHFE0(0)
	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
0x523					PHRE0(3)	PHRE0(2)	PHRE0(1)	PHRE0(0)

address	bit 31	bit 30	bit 29	bit 28	bit 27	bit 26	bit 25	bit 24
0x524	PHCLR1	DBPH1					PHSEL1(1)	PHSEL1(0)
	bit 15	bit 14	bit 13	bit 12	bit 11	bit 10	bit 9	bit 8
0x526					PHFE1(3)	PHFE1(2)	PHFE1(1)	PHFE1(0)
	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
0x527					PHRE1(3)	PHRE1(2)	PHRE1(1)	PHRE1(0)

address	bit 31	bit 30	bit 29	bit 28	bit 27	bit 26	bit 25	bit 24
0x528	PHCLR2	DBPH2					PHSEL2(1)	PHSEL2(0)
	bit 15	bit 14	bit 13	bit 12	bit 11	bit 10	bit 9	bit 8
0x52A					PHFE2(3)	PHFE2(2)	PHFE2(1)	PHFE2(0)
	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
0x52B					PHRE2(3)	PHRE2(2)	PHRE2(1)	PHRE2(0)

Bit	Function
PHCLRx	Reset phase monitoring registers by writing ‘1’
DBPHx	Distributed bus phase sampled on rising edge of input signal
PHSELx(1:0)	Input phase select
	00 - Sample input with 0° event clock
	01 - Sample input with 90° event clock
	10 - Sample input with 180° event clock
	11 - Sample input with 270° event clock
PHFEx(3:0)	Falling edge phase monitoring register
PHREx(3:0)	Rising edge phase monitoring register

Transition Board Input Mapping Registers

address	bit 31	bit 30	bit 29	bit 28	bit 27	bit 26	bit 25	bit 24
0x540	TI0SQMK(3)	TI0SQMK(2)	TI0SQMK(1)	TI0SQMK(0)	TI0SQEEN(3)	TI0SQEEN(2)	TI0SQEEN(1)	TI0SQEEN(0) / TI0IRQ
	bit 23	bit 22	bit 21	bit 20	bit 19	bit 18	bit 17	bit 16
0x541	TI0DB7	TI0DB6	TI0DB5	TI0DB4	TI0DB3	TI0DB2	TI0DB1	TI0DB0
	bit 15	bit 14	bit 13	bit 12	bit 11	bit 10	bit 9	bit 8
0x542			TI0SEN1	TI0SEN0			TI0SEQ1	TI0SEQ0
	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
0x543	TI0EV7	TI0EV6	TI0EV5	TI0EV4	TI0EV3	TI0EV2	TI0EV1	TI0EV0
	bit 31	bit 30	bit 29	bit 28	bit 27	bit 26	bit 25	bit 24
0x544	TI1SQMK(3)	TI1SQMK(2)	TI1SQMK(1)	TI1SQMK(0)	TI1SQEEN(3)	TI1SQEEN(2)	TI1SQEEN(1)	TI1SQEEN(0) / TI1IRQ
	bit 23	bit 22	bit 21	bit 20	bit 19	bit 18	bit 17	bit 16
0x545	TI1DB7	TI1DB6	TI1DB5	TI1DB4	TI1DB3	TI1DB2	TI1DB1	TI1DB0
	bit 15	bit 14	bit 13	bit 12	bit 11	bit 10	bit 9	bit 8
0x546			TI1SEN1	TI1SEN0			TI1SEQ1	TI1SEQ0
	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
0x547	TI1EV7	TI1EV6	TI1EV5	TI1EV4	TI1EV3	TI1EV2	TI1EV1	TI1EV0
…								
	bit 31	bit 30	bit 29	bit 28	bit 27	bit 26	bit 25	bit 24
0x55C	TI5SQMK(3)	TI5SQMK(2)	TI5SQMK(1)	TI5SQMK(0)	TI5SQEEN(3)	TI5SQEEN(2)	TI5SQEEN(1)	TI5SQEEN(0) / TI5IRQ
	bit 23	bit 22	bit 21	bit 20	bit 19	bit 18	bit 17	bit 16
0x55D	TI15DB7	TI15DB6	TI15DB5	TI15DB4	TI15DB3	TI15DB2	TI15DB1	TI15DB0
	bit 15	bit 14	bit 13	bit 12	bit 11	bit 10	bit 9	bit 8
0x55E			TI15SEN1	TI15SEN0			TI15SEQ1	TI15SEQ0
	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit1	bit 0
0x55F	TI15EV7	TI15EV6	TI15EV5	TI15EV4	TI15EV3	TI15EV2	TI15EV1	TI15EV0

Bit	Function
TIxSQMKy	Map Transition Board Input x to Sequence Event Mask bit y
TIxSQEENy	Map Transition Board Input x to Sequence Event Enable bit y
TIxIRQ	Map Transition Board Input x to External Interrupt
TIxDB7	Map Transition Board Input x to Distributed Bus bit 7
TIxDB6	Map Transition Board Input x to Distributed Bus bit 6
TIxDB5	Map Transition Board Input x to Distributed Bus bit 5
TIxDB4	Map Transition Board Input x to Distributed Bus bit 4
TIxDB3	Map Transition Board Input x to Distributed Bus bit 3
TIxDB2	Map Transition Board Input x to Distributed Bus bit 2
TIxDB1	Map Transition Board Input x to Distributed Bus bit 1
TIxDB0	Map Transition Board Input x to Distributed Bus bit 0
TIxSEN1	Map Transition Board Input x to Sequence External Enable 1
TIxSEN0	Map Transition Board Input x to Sequence External Enable 0
TIxSEQ1	Map Transition Board Input x to Sequence Trigger 1
TIxSEQ0	Map Transition Board Input x to Sequence Trigger 0
TIxEV7	Map Transition Board Input x to Event Trigger 7
TIxEV6	Map Transition Board Input x to Event Trigger 6
TIxEV5	Map Transition Board Input x to Event Trigger 5
TIxEV4	Map Transition Board Input x to Event Trigger 4
TIxEV3	Map Transition Board Input x to Event Trigger 3
TIxEV2	Map Transition Board Input x to Event Trigger 2
TIxEV1	Map Transition Board Input x to Event Trigger 1
TIxEV0	Map Transition Board Input x to Event Trigger 0

Note: All enabled input signals are OR'ed together. So if e.g. distributed bus bit 0 has two sources from universal input 0 and 1, if either of the inputs is active high also the distributed bus is active high.

2.7.3 Embedded Event Receivers

The EVM-300 firmware includes two embedded event receivers. The downstream event receiver (EVRD) receives the event stream from port U whereas the upstream event receiver (EVRU) receives the concentrated event stream from ports 1 through 8.

The downstream event receiver (EVRD) is located in the EVG register map at offset 0x20000 through 0x2ffff and the upstream event receiver (EVRU) is located in the EVG register map at offset 0x30000 through 0x3ffff. The event receiver register map follows the description further in this document.

The event master has a number of internal signals which are connected following:

Signal destination	Signal source
EVG UNIVIN(0)	EVRD FPOUT(0)
EVG UNIVIN(1)	EVRD FPOUT(1)
EVG UNIVIN(2)	EVRD FPOUT(2)
EVG UNIVIN(3)	EVRD FPOUT(3)
EVG UNIVIN(4)	EVRD FPOUT(4)
EVG UNIVIN(5)	EVRD FPOUT(5)
EVG UNIVIN(6)	EVRD FPOUT(6)
EVG UNIVIN(7)	EVRD FPOUT(7)

continues on next page

Table 6 – continued from previous page

Signal destination	Signal source
EVG UNIVIN(8)	EVRU FPOUT(0)
EVG UNIVIN(9)	EVRU FPOUT(1)
EVG UNIVIN(10)	EVRU FPOUT(2)
EVG UNIVIN(11)	EVRU FPOUT(3)
EVG UNIVIN(12)	EVRU FPOUT(4)
EVG UNIVIN(13)	EVRU FPOUT(5)
EVG UNIVIN(14)	EVRU FPOUT(6)
EVG UNIVIN(15)	EVRU FPOUT(7)
EVRD FPIN(0)	EVRU FPOUT(0)
EVRD FPIN(1)	EVRU FPOUT(1)
EVRD FPIN(2)	EVRU FPOUT(2)
EVRD FPIN(3)	EVRU FPOUT(3)
EVRD FPIN(4)	EVRU FPOUT(4)
EVRD FPIN(5)	EVRU FPOUT(5)
EVRD FPIN(6)	EVRU FPOUT(6)
EVRD FPIN(7)	EVRU FPOUT(7)
EVRU FPIN(0)	EVRD FPOUT(0)
EVRU FPIN(1)	EVRD FPOUT(1)
EVRU FPIN(2)	EVRD FPOUT(2)
EVRU FPIN(3)	EVRD FPOUT(3)
EVRU FPIN(4)	EVRD FPOUT(4)
EVRU FPIN(5)	EVRD FPOUT(5)
EVRU FPIN(6)	EVRD FPOUT(6)
EVRU FPIN(7)	EVRD FPOUT(7)

Address	Register	Type	Description
0x000	Status	UINT32	Status Register
0x004	Control	UINT32	Control Register
0x010	UpDCValue	UINT32	Upstream Data Compensation Delay Value
0x014	FIFODCValue	UINT32	Receive FIFO Data Compensation Delay Value
0x018	IntDCValue	UINT32	FCT Internal Datapath Data Compensation Delay Value
0x02C	TopologyID	UINT32	Timing Node Topology ID
0x040	Port1DCValue	UINT32	Port 1 loop delay value
0x044	Port2DCValue	UINT32	Port 2 loop delay value
0x048	Port3DCValue	UINT32	Port 3 loop delay value
0x04C	Port4DCValue	UINT32	Port 4 loop delay value
0x050	Port5DCValue	UINT32	Port 5 loop delay value
0x054	Port6DCValue	UINT32	Port 6 loop delay value
0x058	Port7DCValue	UINT32	Port 7 loop delay value
0x05C	Port8DCValue	UINT32	Port 8 loop delay value
0x1000 – 0x10FF	SFP1EEPROM		Port 1 SFP Transceiver EEPROM contents (SFP address 0xA0)
0x1100 – 0x11FF	SFP1DIAG		Port 1 SFP Transceiver diagnostics (SFP address 0xA2)
0x1200 – 0x12FF	SFP2EEPROM		Port 2 SFP Transceiver EEPROM contents (SFP address 0xA0)
0x1300 – 0x13FF	SFP2DIAG		Port 2 SFP Transceiver diagnostics (SFP address 0xA2)
0x1400 – 0x14FF	SFP3EEPROM		Port 3 SFP Transceiver EEPROM contents (SFP address 0xA0)
0x1500 – 0x15FF	SFP3DIAG		Port 3 SFP Transceiver diagnostics (SFP address 0xA2)
0x1600 – 0x16FF	SFP4EEPROM		Port 4 SFP Transceiver EEPROM contents (SFP address 0xA0)
0x1700 – 0x17FF	SFP4DIAG		Port 4 SFP Transceiver diagnostics (SFP address 0xA2)

continues on next page

Table 7 – continued from previous page

Address	Register	Type	Description
0x1800 – 0x18FF	SFP5EEPROM		Port 5 SFP Transceiver EEPROM contents (SFP address 0xA0)
0x1900 – 0x19FF	SFP5DIAG		Port 5 SFP Transceiver diagnostics (SFP address 0xA2)
0x1A00 – 0x1AFF	SFP6EEPROM		Port 6 SFP Transceiver EEPROM contents (SFP address 0xA0)
0x1B00 – 0x1BFF	SFP6DIAG		Port 6 SFP Transceiver diagnostics (SFP address 0xA2)
0x1C00 – 0x1CFF	SFP7EEPROM		Port 7 SFP Transceiver EEPROM contents (SFP address 0xA0)
0x1D00 – 0x1DFF	SFP7DIAG		Port 7 SFP Transceiver diagnostics (SFP address 0xA2)
0x1E00 – 0x1EFF	SFP8EEPROM		Port 8 SFP Transceiver EEPROM contents (SFP address 0xA0)
0x1F00 – 0x1FFF	SFP8DIAG		Port 8 SFP Transceiver diagnostics (SFP address 0xA2)

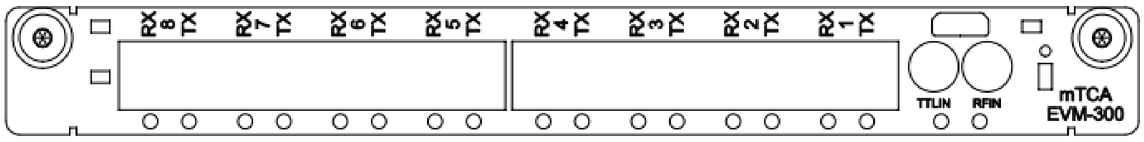
address	bit 23	bit 22	bit 21	bit 20	bit 19	bit 18	bit 17	bit 16
0x001	LINK8	LINK7	LINK6	LINK5	LINK4	LINK3	LINK2	LINK1
	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
0x003	VIO8	VIO7	VIO6	VIO5	VIO4	VIO3	VIO2	VIO1

Bit	Function
LINK8	Port 8 RX Status, 1 – link up, 0 – link down
LINK7	Port 7 RX Status, 1 – link up, 0 – link down
LINK6	Port 6 RX Status, 1 – link up, 0 – link down
LINK5	Port 5 RX Status, 1 – link up, 0 – link down
LINK4	Port 4 RX Status, 1 – link up, 0 – link down
LINK3	Port 3 RX Status, 1 – link up, 0 – link down
LINK2	Port 2 RX Status, 1 – link up, 0 – link down
LINK1	Port 1 RX Status, 1 – link up, 0 – link down
VIO8	Port 8 RX Violation
VIO7	Port 7 RX Violation
VIO6	Port 6 RX Violation
VIO5	Port 5 RX Violation
VIO4	Port 4 RX Violation
VIO3	Port 3 RX Violation
VIO2	Port 2 RX Violation
VIO1	Port 1 RX Violation

address	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
0x007	CLRV8	CLRV7	CLRV6	CLRV5	CLRV4	CLRV3	CLRV2	CLRV1

Bit	Function
CLRV8	Clear RX Violation Port 8
CLRV7	Clear RX Violation Port 7
CLRV6	Clear RX Violation Port 6
CLRV5	Clear RX Violation Port 5
CLRV4	Clear RX Violation Port 4
CLRV3	Clear RX Violation Port 3
CLRV2	Clear RX Violation Port 2
CLRV1	Clear RX Violation Port 1

2.8 MTCA-EVM-300



Connector / Led	Style	Level	Description
RFIN	LEMO	RF+10dBm	RF Reference Input
TTLIN	LEMO	TTL	ACIN / TTL0 Trigger input
TX1	LC	optical	Fan-Out Port 1 Transmit (TX 1)
RX1	LC	optical	Concentrator Port 1 Receiver (RX 1)
TX2	LC	optical	Fan-Out Port 2 Transmit (TX 2)
RX2	LC	optical	Concentrator Port 2 Receiver (RX 2)
TX3	LC	optical	Fan-Out Port 3 Transmit (TX 3)
RX3	LC	optical	Concentrator Port 3 Receiver (RX 3)
TX4	LC	optical	Fan-Out Port 4 Transmit (TX 4)
RX4	LC	optical	Concentrator Port 4 Receiver (RX 4)
TX5	LC	optical	Fan-Out Port 5 Transmit (TX 5)
RX5	LC	optical	Concentrator Port 5 Receiver (RX 5)
TX6	LC	optical	Fan-Out Port 6 Transmit (TX 6)
RX6	LC	optical	Concentrator Port 6 Receiver (RX 6)
TX7	LC	optical	Fan-Out Port 7 Transmit (TX 7)
RX7	LC	optical	Concentrator Port 7 Receiver (RX 7)
TX8 (UP)	LC	optical	Upstream Transmit Optical Output (TX)
RX8 (UP)	LC	optical	Upstream Receiver Optical Input (RX)

2.8.1 TTL Input Levels

The mTCA-EVM-300 has one front panel TTL input. The input is terminated with 50 ohm to ground and is 5V tolerant even when powered down.

Input specifications are following:

parameter	value
connector type	LEMO EPK.00.250.NTN
input impedance	50 ohm
V_{IH}	> 2.3 V
V_{IL}	< 1.0 V

2.8.2 Register Mapping

The mTCA-EVM-300 uses the following PCI IDs.

ID name	value	description
PCI Vendor ID	0x10ee	Xilinx
PCI Device ID	0x7011	Kintex 7
PCI Subdevice VID	0x1a3e	Micro-Research Finland Oy
PCI Subdevice DID	0x232c	mTCA-EVM-300

All EVM functions are memory mapped through PCI BAR0.

Address	function	description
0x00000-0x0FFFF	EVG	Event generator
0x10000-0x1FFFF	FCT	Fan-out registers
0x20000-0x2FFFF	EVRD	Downstream Event Receiver
0x30000-0x3FFFF	EVRU	Upstream Event Receiver

2.9 EVM Firmware Version Change Log

FW Version	Date	Changes	Af
0200	11.06.2015	- Prototype release	V
0201	24.09.2015	- Added segmented data buffer	V
		Fixed Port 1 TX polarity	
010202	01.10.2015	- Changed receive FIFO delay target to 00060000	V
		Added LED test mode (production testing)	
		Removed test signals from TBOUT	
010203	23.11.2015	- Added changes for running with a slower clock on fan-out.	V
020203	18.12.2015	- Changes to data buffer forwarding	V
		Changes for rate conversion forwarding, using internal div/2.	
0204	12.01.2016	- /2 rate conversion working on events, dbuf and dbits.	V
		Improvements to delay measurement system.	
0205	13.04.2016	- Moved delay compensation segment from segment 0 to	V
		last segment in memory.	
		Fixed front panel TTL input order.	
		Fixed race condition in segmented memory buffer trans-	
		mission that caused dropped software buffers.	
FB0206	23.12.2016	- Added upstream and downstream event receivers.	V
		Changed beacon event from 0x7A to 0x7E.	
		Added topology ID	
		Added delay measurement validity information to delay VME-EVM-300	
		compensation data	
000207	19.01.2017	- Added front panel input phase monitoring and phase	V
		select features.	
		Added external AC input synhronisation features.	
010207	09.02.2017	- Fixed occasional dropped out downstream and upstream VME-EVM-300	
data buffers/segmented data buffers.			
030207	03.05.2017	- Added RF input monitoring logic to automatically recover	V

continues

Table 8 – continued from previous page

FW Version	Date	Changes	Af
		from lost RF signal.	
		Added a way to toggle distributed bus transmission	
		phase when an external AC synchronisation clock is used.	
040207	23.05.2017	- Fixed readout of diagnostics information on single	VM
		mode transceivers.	
050207	26.06.2017	- Fixed transceiver_channel to turn off receiver on first	VM
		error to prevent propagation of errors up stream.	

2.10 FCT Function Register Map

The EVM module can be configured for use as an Event Generator (EVG) or a fanout/concentrator.

This is the register map when EVM is configured as a fanout/concentrator.

Address	Register	Type	Description
0x000	Status	UINT32	*Status Register*
0x004	Control	UINT32	*Control Register*
0x010	UpDCValue	UINT32	Upstream Data Compensation Delay Value
0x014	FIFODCValue	UINT32	Receive FIFO Data Compensation Delay Value
0x018	IntDCValue	UINT32	FCT Internal Datapath Data Compensation Delay Value
0x02C	TopologyID	UINT32	Timing Node Topology ID
0x040	Port1DCValue	UINT32	Port 1 loop delay value
0x044	Port2DCValue	UINT32	Port 2 loop delay value
0x048	Port3DCValue	UINT32	Port 3 loop delay value
0x04C	Port4DCValue	UINT32	Port 4 loop delay value
0x050	Port5DCValue	UINT32	Port 5 loop delay value
0x054	Port6DCValue	UINT32	Port 6 loop delay value
0x058	Port7DCValue	UINT32	Port 6 loop delay value
0x05C	Port8DCValue	UINT32	Port 8 loop delay value
0x1000 – 0x10FF	SFP1EEPROM		Port 1 SFP Transceiver EEPROM contents (SFP address 0xA0)
0x1100 – 0x11FF	SFP1DIAG		Port 1 SFP Transceiver diagnostics (SFP address 0xA2)
0x1200 – 0x12FF	SFP2EEPROM		Port 2 SFP Transceiver EEPROM contents (SFP address 0xA0)
0x1300 – 0x13FF	SFP2DIAG		Port 2 SFP Transceiver diagnostics (SFP address 0xA2)
0x1400 – 0x14FF	SFP3EEPROM		Port 3 SFP Transceiver EEPROM contents (SFP address 0xA0)
0x1500 – 0x15FF	SFP3DIAG		Port 3 SFP Transceiver diagnostics (SFP address 0xA2)
0x1600 – 0x16FF	SFP4EEPROM		Port 4 SFP Transceiver EEPROM contents (SFP address 0xA0)
0x1700 – 0x17FF	SFP4DIAG		Port 4 SFP Transceiver diagnostics (SFP address 0xA2)
0x1800 – 0x18FF	SFP5EEPROM		Port 5 SFP Transceiver EEPROM contents (SFP address 0xA0)
0x1900 – 0x19FF	SFP5DIAG		Port 5 SFP Transceiver diagnostics (SFP address 0xA2)
0x1A00 – 0x1AFF	SFP6EEPROM		Port 6 SFP Transceiver EEPROM contents (SFP address 0xA0)
0x1B00 – 0x1BFF	SFP6DIAG		Port 6 SFP Transceiver diagnostics (SFP address 0xA2)
0x1C00 – 0x1CFF	SFP7EEPROM		Port 7 SFP Transceiver EEPROM contents (SFP address 0xA0)
0x1D00 – 0x1DFF	SFP7DIAG		Port 7 SFP Transceiver diagnostics (SFP address 0xA2)
0x1E00 – 0x1EFF	SFP8EEPROM		Port 8 SFP Transceiver EEPROM contents (SFP address 0xA0)
0x1F00 – 0x1FFF	SFP8DIAG		Port 8 SFP Transceiver diagnostics (SFP address 0xA2)

Some information about the transceiver EEPROM contents can be found in this document.

2.10.1 Status Register

address	bit 23	bit 22	bit 21	bit 20	bit 19	bit 18	bit 17	bit 16
0x001	LINK8	LINK7	LINK6	LINK5	LINK4	LINK3	LINK2	LINK1
	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
0x003	VIO8	VIO7	VIO6	VIO5	VIO4	VIO3	VIO2	VIO1

Bit	Function
LINK8	Port8 RXStatus, 1–link up, 0–link down
LINK7	Port7 RXStatus, 1–link up, 0–link down
LINK6	Port6 RXStatus, 1–link up, 0–link down
LINK5	Port5 RXStatus, 1–link up, 0–link down
LINK4	Port4 RXStatus, 1–link up, 0–link down
LINK3	Port3 RXStatus, 1–link up, 0–link down
LINK2	Port2 RXStatus, 1–link up, 0–link down
LINK1	Port1 RXStatus, 1–link up, 0–link down
VIO8	Port 8 RX Violation
VIO7	Port 7 RX Violation
VIO6	Port 6 RX Violation
VIO5	Port 5 RX Violation
VIO4	Port 4 RX Violation
VIO3	Port 3 RX Violation
VIO2	Port 2 RX Violation
VIO1	Port 1 RX Violation

2.10.2 Control Register

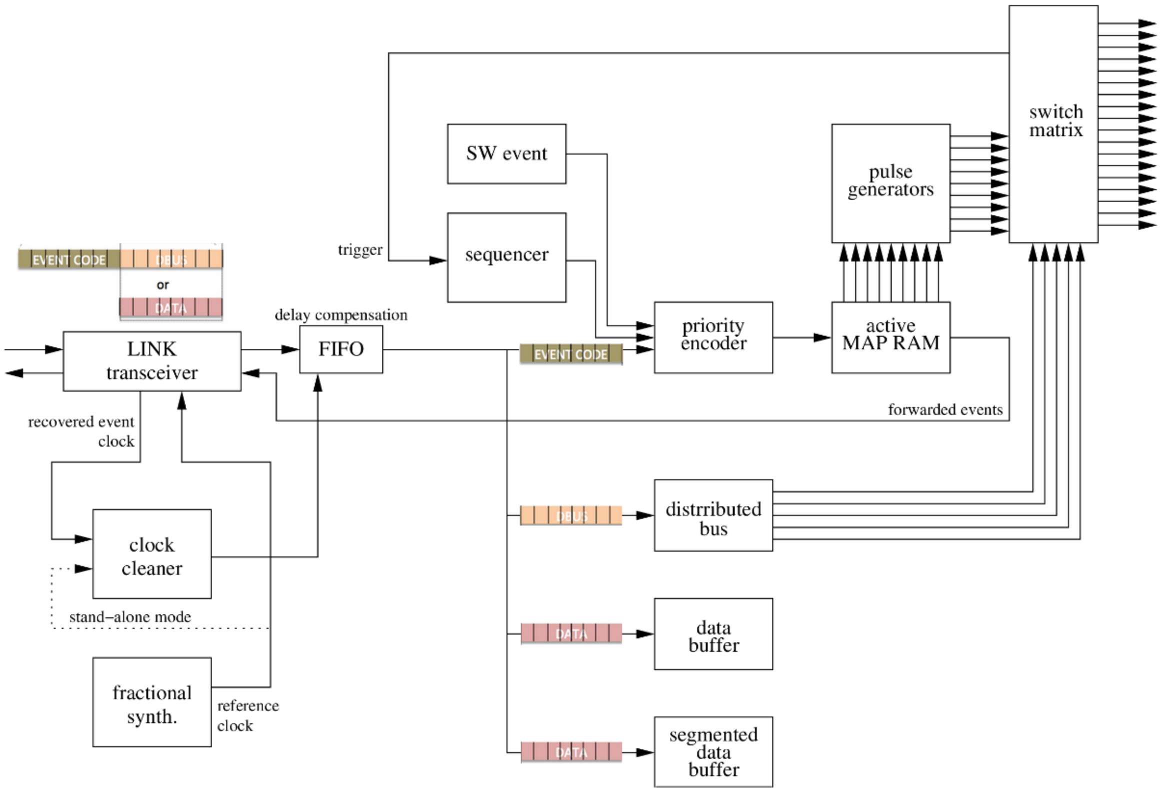
address	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
0x007	CLRV8	CLRV7	CLRV6	CLRV5	CLRV4	CLRV3	CLRV2	CLRV1

Bit	Function
CLRV8	Clear RX Violation Port 8
CLRV7	Clear RX Violation Port 7
CLRV6	Clear RX Violation Port 6
CLRV5	Clear RX Violation Port 5
CLRV4	Clear RX Violation Port 4
CLRV3	Clear RX Violation Port 3
CLRV2	Clear RX Violation Port 2
CLRV1	Clear RX Violation Port 1

2.11 Event Receiver

Event Receivers decode timing events and signals from an optical event stream transmitted by an Event Generator. Events and signals are received at predefined rate the event clock that is usually divided down from an accelerators main RF reference. The event receivers lock to the phase event clock of the Event Generator and are thus phase locked to the RF reference. Event Receivers convert event codes transmitted by an Event Generator to hardware outputs. They can also generate software interrupts and store the event codes with globally distributed timestamps into FIFO memory to be read by a CPU.

Block diagram of the Event Receiver (simplified).



2.11.1 Functional Description

After recovering the event clock the Event Receiver demultiplexes the event stream to 8-bit event codes and 8-bit distributed bus data. The distributed bus may be configured to share its bandwidth with time deterministic data transmission.

2.11.2 Event Decoding

The actions that the Event Receiver takes when it receives an event code are configured by means of the Mapping RAMs. Each EVR provides two mapping RAMs of 256 × 128 bits. Only one of the RAMs can be active at a time, however both RAMs may be modified at any time. The event code is applied to the address lines of the active mapping RAM. The 128-bit data programmed into a specific memory location pointed to by the event code determines what actions will be taken. The rough classification of actions is as described in the table below.

Event code	Offset	Internal functions	Pulse Triggers	'Set' Pulse	'Reset' Pulse
0x00	0x0000	4 bytes/32 bits	4 bytes/32 bits	4 bytes/32 bits	4 bytes/32 bits
0x01	0x0010	4 bytes/32 bits	4 bytes/32 bits	4 bytes/32 bits	4 bytes/32 bits
0x02	0x0020	4 bytes/32 bits	4 bytes/32 bits	4 bytes/32 bits	4 bytes/32 bits
…	…	…	…	…	…
0xFF	0x0FF0	4 bytes/32 bits	4 bytes/32 bits	4 bytes/32 bits	4 bytes/32 bits

Function mapping

There are 32 bits (96 to 127) that are reserved for internal functions, some of which are by default mapped to event codes as shown in the table below. The remaining 96 bits control internal pulse generators. For each pulse generator there is one bit to trigger the pulse generator, one bit to set the pulse generator output and one bit to clear the pulse generator output.

Map bit	Default event code	Function
127	n/a	Save event in FIFO
126	n/a	Latch timestamp
125	n/a	Led event
124	n/a	Forward event from RX to TX
123	0x79	Stop event log
122	n/a	Log event
102 to 121	n/a	(Reserved)
101	0x7a	Heartbeat event
100	0x7b	Reset Prescalers
99	0x7d	Timestamp reset event (TS counter reset)
98	0x7c	Timestamp clock event (TS counter increment)
97	0x71	Seconds shift register '1'
96	0x70	Seconds shift register '0'
80 to 95	n/a	(Reserved)
79	n/a	Trigger pulse generator 15
…	n/a	
64	n/a	Trigger pulse generator 0
48 to 63	n/a	(Reserved)
47	n/a	Set pulse generator 15 output high
…	n/a	
32	n/a	Set pulse generator 0 output high
16 to 31	n/a	(Reserved)
15	n/a	Reset pulse generator 15 output low
…	n/a	
0	n/a	Reset pulse generator 0 output low

2.11.3 Heartbeat Monitor

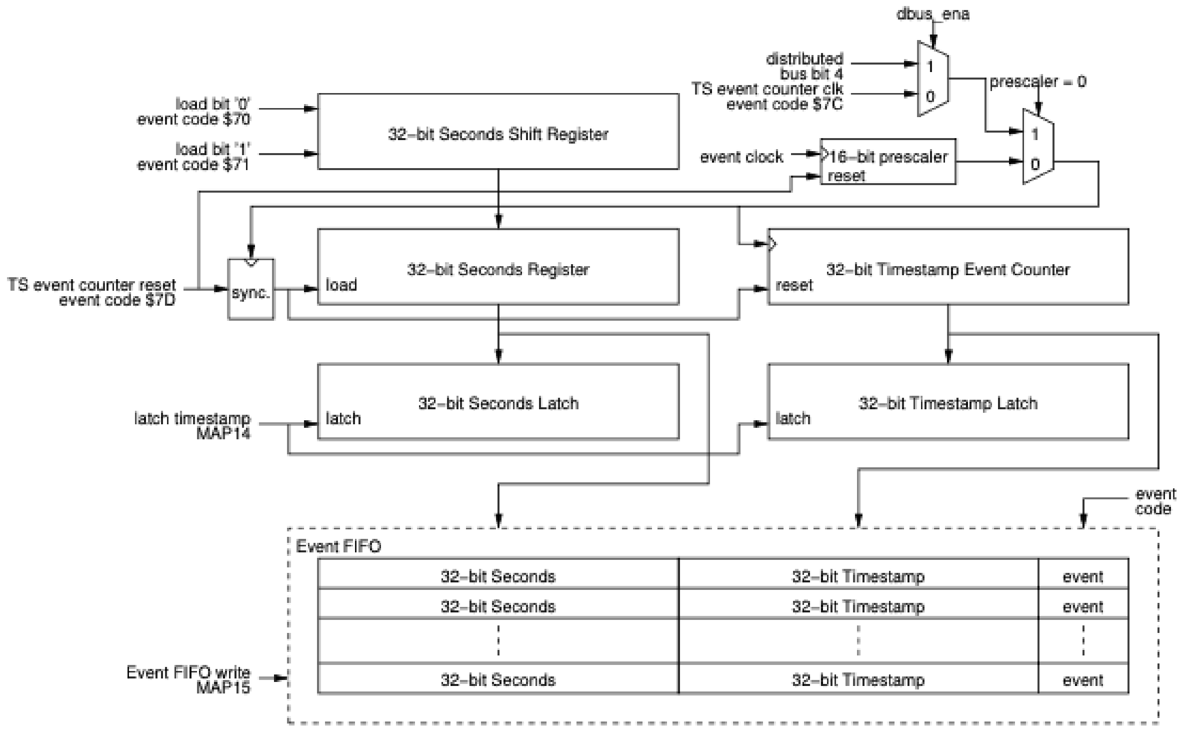
A heartbeat monitor is provided to receive heartbeat events. Event code $7A is by default set up to reset the heartbeat counter. If no heartbeat event is received the counter times out (approx. 1.6 s) and a heartbeat flag is set. The Event Receiver may be programmed to generate a heartbeat interrupt at timeout.

2.11.4 Event FIFO and Timestamp Events

The Event System provides a global timebase to attach timestamps to collected data and performed actions. The time stamping system consists of a 32-bit timestamp event counter and a 32-bit seconds counter. The timestamp event counter either counts received timestamp counter clock events or runs freely with a clock derived from the event clock. The event counter is also able to run on a clock provided on a distributed bus bit.

The event counter clock source is determined by the prescaler control register. The timestamp event counter is cleared at the next event counter rising clock edge after receiving a timestamp event counter reset event. The seconds counter is updated serially by loading zeros and ones (see mapping register bits) into a shift register MSB first. The seconds register is updated from the shift register at the same time the timestamp event counter is reset.

The timestamp event counter and seconds counter contents may be latched into a timestamp latch. Latching is determined by the active event map RAM and may be enabled for any event code. An event FIFO memory is implemented to store selected event codes with attached timing information. The 80-bit wide FIFO can hold up to 511 events. The recorded event is stored along with 32-bit seconds counter contents and 32-bit timestamp event counter contents at the time of reception. The event FIFO as well as the timestamp counter and latch are accessible by software.



2.11.5 Event Log

Up to 512 events with timestamping information can be stored in the event log. The log is implemented as a ring buffer and is accessible as a memory region. Logging events can be stopped by an event or software.

2.11.6 Distributed Bus and Data Transmission

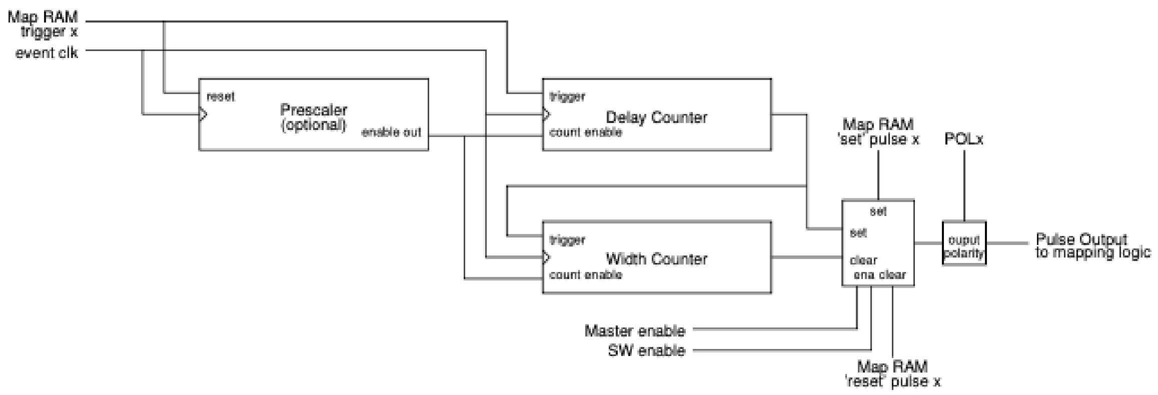
The distributed bus is able to carry eight simultaneous signals sampled with half the event clock rate over the fibre optic transmission media. The distributed bus signals may be output on programmable front panel outputs. The distributed bus bandwidth is shared by transmission of a configurable size data buffer to up to 2 kbytes.

2.11.7 Pulse Generators

The structure of the pulse generation logic is shown in the figure below. Three signals from the mapping RAM control the output of the pulse: trigger, 'set' pulse and 'reset' pulse. A trigger causes the delay counter to start counting, when the end-of-count is reached the output pulse changes to the 'set' state and the width counter starts counting. At the end of the width count the output pulse is cleared. The mapping RAM signal 'set' and 'reset' cause the output to change state immediately without any delay.

Starting from firmware version 0200 pulse generators can also be triggered from rising edges of distributed bus signals or EVR internal prescalers.

32 bit registers are reserved for both counters and the prescaler, however, the prescaler is not necessarily implemented for all channels and may be hard coded to 1 in case the prescaler is omitted. Software may write 0xFFFFFFFF to these registers and read out the actual width or hard-coded value of the register. For example, if the width counter is limited to 16 bits a read will return 0x0000FFFF after a write of 0xFFFFFFFF.



Pulse Generator

Pulse Generator Gates

Depending on firmware revision/form factor a number of pulse generators are configured as event triggered gates only and can be used to mask or enable pulse generator triggers.

The VME-EVR-300, PCIe-EVR- 300DC and mTCA-EVR-300 have four pulse generators configured as gates, pulse generators 28 to 31 which correspond gates 0 to 3.

2.11.8 Prescalers

The Event Receiver provides a number of programmable prescalers. The frequencies are programmable and are derived from the event clock. A special event code reset prescalers $7B causes the prescalers to be synchronously reset, so the frequency outputs will be in same phase across all event receivers.

2.11.9 Programmable Front Panel, Universal I/O and Backplane Connections

All outputs are programmable: each pulse generator output, prescaler and distributed bus bit can be mapped to any output. Starting with firmware version 0200 each output can have two sources which are logically OR'ed together. The mapping for a single source is shown in table below.

Each output has a two byte mapping register and each byte corresponds a single source. An unused mapping source should be set to 63 (0x3f). In case of a bidirectional signal to tri-state set both bytes to 61 (0x3d).

Table 18: Output mapping values

Mapping ID	Signal
0 to n-1	Pulse generator output (number n of pulse generators depends on HW and firmware version)
n to 31	(Reserved)
32	Distributed bus bit 0 (DBUS0)
…	…
39	Distributed bus bit 7 (DBUS7)
40	Prescaler 0
41	Prescaler 1
42	Prescaler 2
43 to 47	(Reserved)
48	Flip-flop output 0
…	…
55	Flip-flop output 7
56 to 58	(Reserved)
59	Event clock output (only on PXIe-EVR-300)
60	Event clock output with 180° phase shift (only on PXIe-EVR-300)
61	Tri-state output (for PCIe-EVR-300DC with input module populated in IFB-300's Universal I/O slot)
62	Force output high (logic 1)
63	Force output low (logic 0)

2.11.10 Flip-flop Outputs (from FW version 0E0207)

There are 8 flip-flop outputs. Each of these is using two pulse generators, one for setting the output high and the other one for resetting the output low. In the table below you can see the relationship between flip-flops and pulse generators and the output mapping IDs.

flip-flop	mappingID	Set	Reset
0	48	Pulse gen. 0	Pulse gen. 1
1	49	Pulse gen. 2	Pulse gen. 3
2	50	Pulse gen. 4	Pulse gen. 5
3	51	Pulse gen. 6	Pulse gen. 7
4	52	Pulse gen. 8	Pulse gen. 9
5	53	Pulse gen. 10	Pulse gen. 11
6	54	Pulse gen. 12	Pulse gen. 13
7	55	Pulse gen. 14	Pulse gen. 15

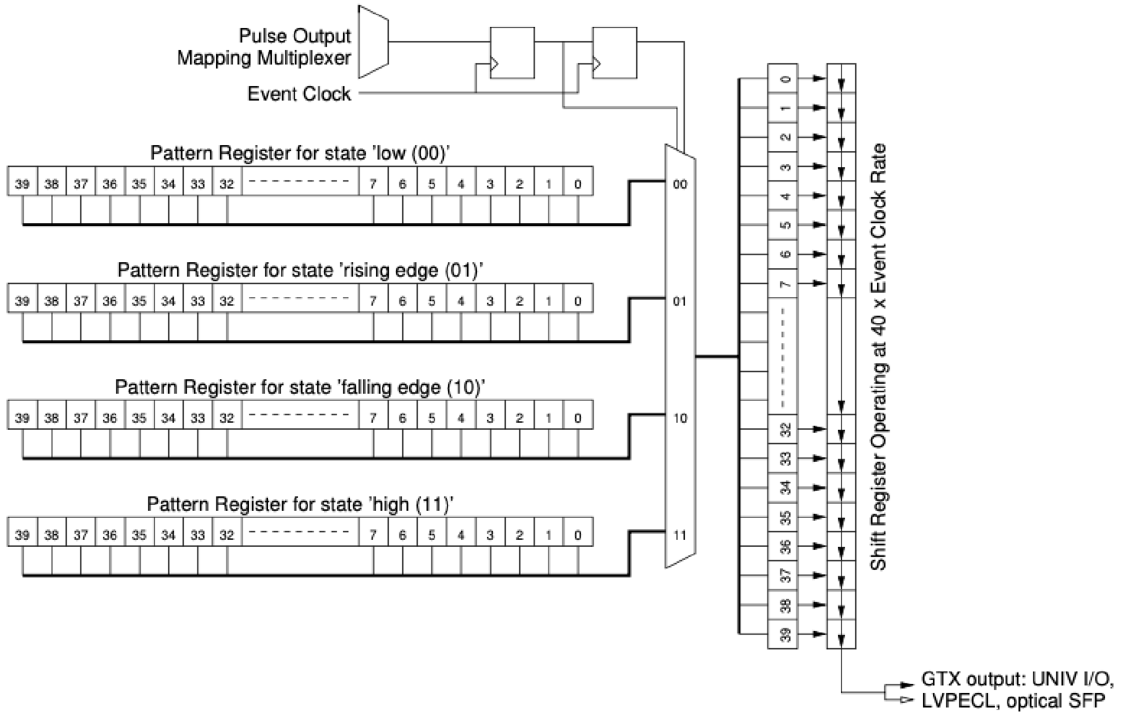
2.11.11 Front Panel Universal I/O Slots

Universal I/O slots provide different types of output with exchangeable Universal I/O modules. Each module provides two outputs e.g. two TTL output, two NIM output or two optical outputs. The source for these outputs is selected with mapping registers.

VME-EVR-300 GTX Front Panel Outputs and mTCA-EVR TCLKA/TCLKB Clocks The VME-EVR-300 has four GTX front panel outputs, two in Universal I/O slot UNIV6/UNIV7 and CML outputs CML0 and CML1. The GTX Outputs provide low jitter signals with special outputs. The outputs can work in different configurations: pulse mode, pattern mode and frequency mode. The difference com- pared to the CML output of the VME-EVR-230RF is that instead of 20 bits per event clock cycle the GTX outputs have 40 bits per event clock cycle doubling the resolution to 200 ps/bit at an event clock of 125 MHz. The mTCA-EVR-300 TCLKA and TCLKB backplane clock operate the same way as VME-EVR-300 GTX front panel outputs. The pulse mapping is controlled through UNIV16 (TCLKA) and UNIV17 (TCLKB) mapping registers.

2.11.12 GTX Pulse Mode

The source for these outputs is selected in a similar way than the standard outputs using mapping registers, however, the output logic monitors the state of this signal and distinguishes between state low (00), rising edge (01), high state (11) and falling edge (10). Based on the state a 40 bit pattern is sent out with a bit rate of 40 times the event clock rate.


Pulse Output
Mapping Multiplexer
Event Clock
Pattern Register for state 'low (00)'
Pattern Register for state 'rising edge (01)'
Pattern Register for state 'falling edge (10)'
Pattern Register for state 'high (11)'
00
01
10
11
Shift Register Operating at 40 x Event Clock Rate
GTX output: UNIV I/O,
LVPECL, optical SFP

- When the source for a GTX output is low and was low one event clock cycle earlier (state low), the GTX output repeats the 40 bit pattern stored in pattern_00 register.
- When the source for a GTX output is high and was low one event clock cycle earlier (state rising), the GTX output sends out the 40 bit pattern stored in pattern_01 register.
- When the source for a GTX output is high and was high one event clock cycle earlier (state high), the GTX output repeats the 40 bit pattern stored in pattern_11 register.
- When the source for a GTX output is low and was high one event clock cycle earlier (state falling), the GTX output sends out the 40 bit pattern stored in pattern_10 register.

For an event clock of 125 MHz the duration of one single GTX output bit is 200 ps. These outputs allow for producing fine grained adjustable output pulses and clock frequencies.

2.11.13 GTX Frequency Mode

In frequency mode one can generate clocks where the clock period can be defined in steps of 1/40th part of the event clock cycle i.e. 200 ps step with an event clock of 125 MHz. There are some limitations, however:

- Clock high time and clock low time must be 40/40th event clock period steps
- Clock high time and clock low time must be < 65536/40th event clock period steps

The clock output can be synchronized by one of the pulse generators, distributed bus signal etc. When a rising edge of the mapped output signal is detected the frequency generator takes its output value from the trigger level bit and the counter value from the trigger position register. Thus one can adjust the phase of the synchronized clock in 1/40th steps of the event clock period. To change the generated clock phase in respect to the trigger we can select the trigger polarity by bit CMLTL in the CML Control register and the trigger position also in the CML Control register.

2.11.14 GTX Pattern Mode

In pattern mode one can generate arbitrary bit patterns taking into account following:

```
-   The pattern length is a multiple of 40 bits, where each bit is
    1/40th of the event clock period
-   Maximum length of the arbitrary pattern is 40 × 2048 bits
-   A pattern can be triggered from any pulse generator, distributed
    bus bit etc. When triggered the pattern generator starts sending
    40 bit words from the pattern memory sequentially starting from
    position 0. This goes on until the pattern length set by the
    samples register has been reached.
-   If the pattern generator is in recycle mode the pattern
    continues immediately from position 0 of the pattern memory.
-   If the pattern generator is in single pattern mode, the pattern
    stops and the 40 bit word from the last position of the pattern
    memory (2047) is sent out until the pattern generator is
    triggered again.
```

2.11.15 Configurable Size Data Buffer (EVR)

Pre-DC (Delay Compensation) event systems provided a way to to transmit configurable size data packets that may be transmitted over the event system link. The buffer transmission size is configured in the Event Generator to up to 2 kbytes. The Event Receiver is able to receive buffers of any size from 4 bytes to 2 kbytes in four byte (long word) increments.

2.11.16 Segmented Data Buffer

With the addition of delay compensation a segmented data buffer has been introduced and it can coexist with the configurable size data buffer. The segmented data buffer is divided into 16 byte segments that allow updating only part of the buffer memory with the remaining segments left untouched.

When starting a data transmission the Event Generator first sends the starting segment number that defines the starting address in the buffer. The data buffer address offset is the segment number * 16 bytes. The Event Receiver writes the received bytes into the data buffer and when transmission is complete a receive complete flag is raised for the starting segment of the packet transmission. The transmission can overlap several segments, however, the flag is raised only for the starting segment. If there is a checksum mismatch the checksum error flag for the starting segment is set. In case the receive complete flag already was set before the new data was received an segment overflow flag is set. Flags are cleared by writing a '1' to the receive flag. Each segment has a receive data counter and after completion of the transfer the receive data counter of the starting segment is updated with the actual number of bytes received in the transmission.

The procedure to receive a segmented data buffer is following:

- check that receive complete flag for received segment is set
- check that starting segment overflow flag is cleared
- read transmission size from segment receive data counter
- copy segment data from segmented data buffer memory into system RAM
- verify that starting segment overflow flag is still cleared
- clear segment receive complete flag

Starting with firmware 0205 the delay compensation logic uses the last 16 byte segment of the segmented data buffer for delay compensation data.

2.11.17 Interrupt Generation

The Event Receiver has multiple interrupt sources which all have their own enable and flag bits. The following events may be programmed to generate an interrupt:

- Receiver link state change
- Receiver violation: bit error or the loss of signal.
- Lost heartbeat: heartbeat monitor timeout.
- Write operation of an event to the event FIFO.
- Event FIFO is full.
- Data Buffer reception complete.

In addition to the events listed above an interrupt can be generated from one of the pulse generator outputs, distributed bus bits or prescalers. The pulse interrupt can be mapped in a similar way as the front panel outputs.

2.11.18 External Event Input

An external hardware input is provided to be able to take an external pulse to generate an internal event. This event will be handled as any other received event.

2.11.19 Programmable Reference Clock

The event receiver requires a reference clock to be able to synchronise on the incoming event stream sent by the event generator. For flexibility a programmable reference clock is provided to allow the use of the equipment in various applications with varying frequency requirements. Please note before programming a new operating frequency with the fractional synthesizer the operating frequency (in MHz) has to be set in the UsecDivider register. This is essential as the event receiver's PLL cannot lock if it does not know the frequency range to lock to.

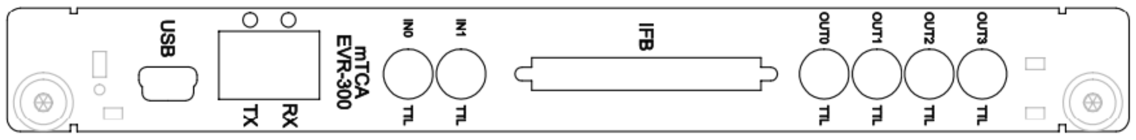
2.11.20 Fractional Synthesiser

The clock reference for the event receiver is generated on-board the event receiver using a fractional synthesiser. A Microchip (formerly Micrel) SY87739L Protocol Transparent Fractional-N Synthesiser with a reference clock of 24 MHz is used. The following table lists programming bit patterns for a few frequencies.

Event Rate	Configuration Bit Pattern	Reference Output	Precision (theoretical)
142.8 MHz	0x0891C100	142.857 MHz	0
499.8 MHz/4 = 124.95 MHz	0x00FE816D	124.95 MHz	0
499.654 MHz/4 = 124.9135 MHz	0x0C928166	124.907 MHz	-52 ppm
476 MHz/4 = 119 MHz	0x018741AD	119 MHz	0
106.25 MHz (fibre channel)	0x049E81AD	106.25 MHz	0
499.8 MHz/5 = 99.96 MHz	0x025B41ED	99.956 MHz	-40 ppm
50 MHz	0x009743AD	50.0 MHz	0
499.8 MHz/10 = 49.98 MHz	0x025B43AD	49.978 MHz	-40 ppm
499.654MHz/4=124.9135MHz	0x0C928166	124.907MHz	-52 ppm
50 MHz	0x009743AD	50.0 MHz	0

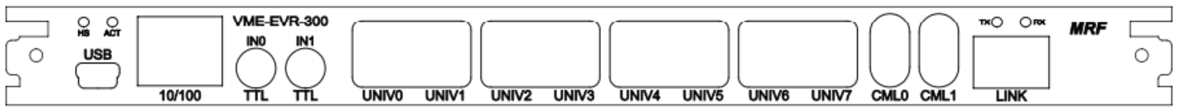
The event receiver reference clock is required to be in ±100 ppm range of the event generator event clock.

2.12 MTCA-EVR-300



Connector / Led	Style	Level	Description
USB	Micro-USB		MMC diagnostics serial port / JTAG interface
Link TX (SFP)	LC	Optical 850 nm	Event link Transmit
			Green: TX enable
			Red: Fract.syn. not locked
			Blue: Event out
Link RX (SFP)	LC	Optical 850 nm	Event link Receiver
			Green: link up
			Red: link violation detected
			Blue: event led
IFB	VHDCI	LVDS	IFB-300 Interface Box connection
IN0	LEMO	TTL	FPTTL0 Trigger input
IN1	LEMO	TTL (3.3V / 5V)	FPTTL1 Trigger input
OUT0	LEMO	3.3V LVTTL	TTL Front panel output 0
OUT1	LEMO	3.3V LVTTL	TTL Front panel output 1
OUT2	LEMO	3.3V LVTTL	TTL Front panel output 2
OUT3	LEMO	3.3V LVTTL	TTL Front panel output 3
TCLKA	mTCA.4	LVDS	TCLKA clock on backplane
			This signal is driven by CML/GTX logic block 0
			Mapped as Universal Output 16
TCLKB	mTCA.4	LVDS	TCLKB clock on backplane
			This signal is driven by CML/GTX logic block 1
			Mapped as Universal Output 17
RX17	mTCA.4	MLVDS	Backplane output 0
TX17	mTCA.4	MLVDS	Backplane output 1
RX18	mTCA.4	MLVDS	Backplane output 2
TX18	mTCA.4	MLVDS	Backplane output 3
RX19	mTCA.4	MLVDS	Backplane output 4
TX19	mTCA.4	MLVDS	Backplane output 5
RX20	mTCA.4	MLVDS	Backplane output 6
TX20	mTCA.4	MLVDS	Backplane output 7

2.13 VME-EVR-300



2.13.1 VME-EVR-300 Front Panel Connections

The front panel of the Event Receiver includes the following connections and status leds:

Connector / Led	Style	Level	Description
HS	Red Led		Module Failure
HS	Blue Led		Module Powered Down
ACT	3-color Led		SAM3X Activity Led
USB	Micro-USB		SAM3X Serial port / JTAG interface
10/100	RJ45		SAM3X Ethernet Interface
IN0	LEMO	TTL (3.3V / 5V)	FPTTL0 Trigger input
IN1	LEMO	TTL (3.3V / 5V)	FPTTL1 Trigger input
UNIV0/1	Universal slot		Universal Output 0/1
UNIV2/3	Universal slot		Universal Output 2/3
UNIV4/5	Universal slot		Universal Output 4/5
UNIV6/7	Universal slot		Universal Output 6/7
			The output signals come through CML/GTX logic block 0/1
CML0	LEMO EPY	CML	Mapped as Universal Output 8
			The output signals come through CML/GTX logic block 2
CML1	LEMO EPY	CML	Mapped as Universal Output 9
			The output signals come through CML/GTX logic block 3
Link TX (SFP)	LC	Optical 850 nm	Event link Transmit
Link RX (SFP)	LC	Optical 850 nm	Event link Receiver

VME TTL Input Levels

The VME-EVR-300 has two front panel TTL inputs. The inputs have a configurable input termination than can be set by a jumper. The input can be terminated with 50 ohm to ground or 220 ohm to +3.3V. The front panel inputs are 5V tolerant even when powered down.

Input specifications are following:

parameter	value
connector type	LEMO EPK.00.250.NTN
input impedance	50ohm
V_{IH}	> 2.3 V
V_{IL}	< 1.0 V

2.14 Event Receiver Registermap

Event Receiver register/memory map.

2.14.1 Register Map

Address	Register	Type	Description
0x000	Status	UINT32	*Status Register*
0x004	Control	UINT32	*Control Register*
0x008	IrqFlag	UINT32	*Interrupt Flag Register*
0x00C	IrqEnable	UINT32	*Interrupt Enable Register*
0x010	PulseIrqMap	UINT32	*Mapping register for pulse interrupt*
0x018	SWEvent	UINT32	*Software event register*
0x01C	PCIIrqEnable	UINT32	*PCI Interrupt Enable Register*
0x020	DataBufCtrl	UINT32	*Data Buffer Control and Status Register*
0x024	TxDataBufCtrl	UINT32	*TX Data Buffer Control and Status Register*
0x028	TxSegBufCtrl	UINT32	*TX Segmented Data Buffer Control and Status Register*
0x02C	FWVersion	UINT32	*Firmware Version Register*
0x040	EvCntPresc	UINT32	Event Counter Prescaler
0x04C	UsecDivider	UINT32	Divider to get from Event Clock to 1 MHz
0x050	ClockControl	UINT32	*Event Clock Control Register*
0x05C	SecSR	UINT32	Seconds Shift Register
0x060	SecCounter	UINT32	Timestamp Seconds Counter
0x064	EventCounter	UINT32	Timestamp Event Counter
0x068	SecLatch	UINT32	Timestamp Seconds Counter Latch
0x06C	EvCntLatch	UINT32	Timestamp Event Counter Latch
0x070	EvFIFOSec	UINT32	Event FIFO Seconds Register
0x074	EvFIFOEvCnt	UINT32	Event FIFO Event Counter Register
0x078	EvFIFOCode	UINT16	Event FIFO Event Code Register
0x07C	LogStatus	UINT32	Event Log Status Register
0x080	FracDiv	UINT32	*Micrel SY87739L Fractional Divider Configuration Word*
0x090	GPIODir	UINT32	Front Panel UnivIO GPIO signal direction
0x094	GPIOIn	UINT32	Front Panel UnivIO GPIO input register
0x098	GPIOOut	UINT32	Front Panel UnivIO GPIO output register

continues on next page

Table 11 – continued from previous page

Address	Register	Type	Description
0x0A0	SPIData	UINT32	*SPI Data Register*
0x0A4	SPIControl	UINT32	*SPI Control Register*
0x0B0	DCTarget	UINT32	Delay Compensation Target Value
0x0B4	DCRxValue	UINT32	Delay Compensation Transmission Path Delay Value
0x0B8	DCIntValue	UINT32	Delay Compensation Internal Delay Value
0x0BC	DCStatus	UINT32	*Delay Compensation Status Register*
0x0C0	TopologyID	UINT32	Timing Node Topology ID
0x0E0	SeqRamCtrl	UINT32	*Sequence RAM Control Register*
0x100	Prescaler0	UINT32	Prescaler 0 Divider
0x104	Prescaler1	UINT32	Prescaler 1 Divider
0x108	Prescaler2	UINT32	Prescaler 2 Divider
0x10C	Prescaler3	UINT32	Prescaler 3 Divider
0x110	Prescaler4	UINT32	Prescaler 4 Divider
0x114	Prescaler5	UINT32	Prescaler 5 Divider
0x118	Prescaler6	UINT32	Prescaler 6 Divider
0x11C	Prescaler7	UINT32	Prescaler 7 Divider
0x120	PrescPhase0	UINT32	Prescaler 0 Phase Offset Register
0x124	PrescPhase1	UINT32	Prescaler 1 Phase Offset Register
0x128	PrescPhase2	UINT32	Prescaler 2 Phase Offset Register
0x12C	PrescPhase3	UINT32	Prescaler 3 Phase Offset Register
0x130	PrescPhase4	UINT32	Prescaler 4 Phase Offset Register
0x134	PrescPhase5	UINT32	Prescaler 5 Phase Offset Register
0x138	PrescPhase6	UINT32	Prescaler 6 Phase Offset Register
0x13C	PrescPhase7	UINT32	Prescaler 7 Phase Offset Register
0x140	PrescTrig0	UINT32	*Prescaler 0 Pulse Generator Trigger Register*
0x144	PrescTrig1	UINT32	Prescaler 1 Pulse Generator Trigger Register
0x148	PrescTrig2	UINT32	Prescaler 2 Pulse Generator Trigger Register
0x14C	PrescTrig3	UINT32	Prescaler 3 Pulse Generator Trigger Register
0x150	PrescTrig4	UINT32	Prescaler 4 Pulse Generator Trigger Register
0x154	PrescTrig5	UINT32	Prescaler 5 Pulse Generator Trigger Register
0x158	PrescTrig6	UINT32	Prescaler 6 Pulse Generator Trigger Register
0x15C	PrescTrig7	UINT32	Prescaler 7 Pulse Generator Trigger Register
0x180	DBusTrig0	UINT32	*DBus Bit 0 Pulse Generator Trigger Register*
0x184	DBusTrig1	UINT32	DBus Bit 1 Pulse Generator Trigger Register
0x188	DBusTrig2	UINT32	DBus Bit 2 Pulse Generator Trigger Register
0x18C	DBusTrig3	UINT32	DBus Bit 3 Pulse Generator Trigger Register
0x190	DBusTrig4	UINT32	DBus Bit 4 Pulse Generator Trigger Register
0x194	DBusTrig5	UINT32	DBus Bit 5 Pulse Generator Trigger Register
0x198	DBusTrig6	UINT32	DBus Bit 6 Pulse Generator Trigger Register
0x19C	DBusTrig7	UINT32	DBus Bit 7 Pulse Generator Trigger Register
0x200	Pulse0Ctrl	UINT32	*Pulse 0 Control Register*
0x204	Pulse0Presc	UINT32	Pulse 0 Prescaler Register
0x208	Pulse0Delay	UINT32	Pulse 0 Delay Register
0x20C	Pulse0Width	UINT32	Pulse 0 Width Register
0x210	Pulse 1 Registers		
0x220	Pulse 2 Registers		
…			
0x3F0	Pulse 31 Registers		
0x400	FPOutMap0	UINT16	Front Panel Output 0 Map Register

continues on next page

Table 11 – continued from previous page

Address	Register	Type	Description
0x402	FPOutMap1	UINT16	Front Panel Output 1 Map Register
0x404	FPOutMap2	UINT16	Front Panel Output 2 Map Register
0x406	FPOutMap3	UINT16	Front Panel Output 3 Map Register
0x408	FPOutMap4	UINT16	Front Panel Output 4 Map Register
0x40A	FPOutMap5	UINT16	Front Panel Output 5 Map Register
0x40C	FPOutMap6	UINT16	Front Panel Output 6 Map Register
0x40E	FPOutMap7	UINT16	Front Panel Output 7 Map Register
0x440	UnivOutMap0	UINT16	Front Panel Universal Output 0 Map Register
0x442	UnivOutMap1	UINT16	Front Panel Universal Output 1 Map Register
0x444	UnivOutMap2	UINT16	Front Panel Universal Output 2 Map Register
0x446	UnivOutMap3	UINT16	Front Panel Universal Output 3 Map Register
0x448	UnivOutMap4	UINT16	Front Panel Universal Output 4 Map Register
0x44A	UnivOutMap5	UINT16	Front Panel Universal Output 5 Map Register
0x44C	UnivOutMap6	UINT16	Front Panel Universal Output 6 Map Register
0x44E	UnivOutMap7	UINT16	Front Panel Universal Output 7 Map Register
0x450	UnivOutMap8	UINT16	Front Panel Universal Output 8 Map Register
0x452	UnivOutMap9	UINT16	Front Panel Universal Output 9 Map Register
0x454	UnivOutMap10	UINT16	Front Panel Universal Output 10 Map Register
0x456	UnivOutMap11	UINT16	Front Panel Universal Output 11 Map Register
0x458	UnivOutMap12	UINT16	Front Panel Universal Output 12 Map Register
0x45A	UnivOutMap13	UINT16	Front Panel Universal Output 13 Map Register
0x45C	UnivOutMap14	UINT16	Front Panel Universal Output 14 Map Register
0x45E	UnivOutMap15	UINT16	Front Panel Universal Output 15 Map Register
0x460	UnivOutMap16	UINT16	Front Panel Universal Output 16 Map Register
0x462	UnivOutMap17	UINT16	Front Panel Universal Output 17 Map Register
0x480	TBOutMap0	UINT16	Transition Board Output 0 Map Register
0x482	TBOutMap1	UINT16	Transition Board Output 1 Map Register
0x484	TBOutMap2	UINT16	Transition Board Output 2 Map Register
0x486	TBOutMap3	UINT16	Transition Board Output 3 Map Register
0x488	TBOutMap4	UINT16	Transition Board Output 4 Map Register
0x48A	TBOutMap5	UINT16	Transition Board Output 5 Map Register
0x48C	TBOutMap6	UINT16	Transition Board Output 6 Map Register
0x48E	TBOutMap7	UINT16	Transition Board Output 7 Map Register
0x490	TBOutMap8	UINT16	Transition Board Output 8 Map Register
0x492	TBOutMap9	UINT16	Transition Board Output 9 Map Register
0x494	TBOutMap10	UINT16	Transition Board Output 10 Map Register
0x496	TBOutMap11	UINT16	Transition Board Output 11 Map Register
0x498	TBOutMap12	UINT16	Transition Board Output 12 Map Register
0x49A	TBOutMap13	UINT16	Transition Board Output 13 Map Register
0x49C	TBOutMap14	UINT16	Transition Board Output 14 Map Register
0x49E	TBOutMap15	UINT16	Transition Board Output 15 Map Register
0x4A0	TBOutMap16	UINT16	Transition Board Output 16 Map Register
0x4A2	TBOutMap17	UINT16	Transition Board Output 17 Map Register
0x4A4	TBOutMap18	UINT16	Transition Board Output 18 Map Register
0x4A6	TBOutMap19	UINT16	Transition Board Output 19 Map Register
0x4A8	TBOutMap20	UINT16	Transition Board Output 20 Map Register
0x4AA	TBOutMap21	UINT16	Transition Board Output 21 Map Register
0x4AC	TBOutMap22	UINT16	Transition Board Output 22 Map Register
0x4AE	TBOutMap23	UINT16	Transition Board Output 23 Map Register

continues on next page

Table 11 – continued from previous page

Address	Register	Type	Description
0x4B0	TBOutMap24	UINT16	Transition Board Output 24 Map Register
0x4B2	TBOutMap25	UINT16	Transition Board Output 25 Map Register
0x4B4	TBOutMap26	UINT16	Transition Board Output 26 Map Register
0x4B6	TBOutMap27	UINT16	Transition Board Output 27 Map Register
0x4B8	TBOutMap28	UINT16	Transition Board Output 28 Map Register
0x4BA	TBOutMap29	UINT16	Transition Board Output 29 Map Register
0x4BC	TBOutMap30	UINT16	Transition Board Output 30 Map Register
0x4BE	TBOutMap31	UINT16	Transition Board Output 31 Map Register
0x4C0	BPOutMap0	UINT16	Backplane Output 0 Map Register
0x4C2	BPOutMap1	UINT16	Backplane Output 1 Map Register
0x4C4	BPOutMap2	UINT16	Backplane Output 2 Map Register
0x4C6	BPOutMap3	UINT16	Backplane Output 3 Map Register
0x4C8	BPOutMap4	UINT16	Backplane Output 4 Map Register
0x4CA	BPOutMap5	UINT16	Backplane Output 5 Map Register
0x4CC	BPOutMap6	UINT16	Backplane Output 6 Map Register
0x4CE	BPOutMap7	UINT16	Backplane Output 7 Map Register
0x500	FPInMap0	UINT32	*Front Panel Input 0 Mapping Register*
0x504	FPInMap1	UINT32	Front Panel Input 1 Mapping Register
0x510	UnivInMap0	UINT32	Universal Input 0 Mapping Register
0x514	UnivInMap1	UINT32	Universal Input 1 Mapping Register
…			
0x560	BPInMap0	UINT32	Backplane Input 0 Mapping Register
0x564	BPInMap1	UINT32	Backplane Input 1 Mapping Register
…			
0x610	GTX0Ctrl	UINT32	UNIV6 / GTX0CML Output Control Register
0x614	GTX0HP	UINT16	UNIV6 / GTX0 Output High Period Count
0x616	GTX0LP	UINT16	UNIV6 / GTX0 Output Low Period Count
0x618	GTX0Samp	UINT32	UNIV6 / GTX0 Output Number of 40 bit word patterns
0x630	GTX1Ctrl	UINT32	UNIV7 / GTX1CML 5 Output Control Register
0x634	GTX1HP	UINT16	UNIV7 / GTX1 Output High Period Count
0x636	GTX1LP	UINT16	UNIV7 / GTX1 Output Low Period Count
0x638	GTX1Samp	UINT32	UNIV7 / GTX1 Output Number of 40 bit word patterns
0x650	GTX2Ctrl	UINT32	CML 0 / GTX2 Output Control Register
0x654	GTX2HP	UINT16	CML 0 / GTX2 Output High Period Count
0x656	GTX2LP	UINT16	CML 0 / GTX2 Output Low Period Count
0x658	GTX2Samp	UINT32	CML 0 / GTX2 Output Number of 40 bit word patterns
0x670	GTX3Ctrl	UINT32	CML1 / GTX3 Output Control Register
0x674	GTX3HP	UINT16	CML1 / GTX3 Output High Period Count
0x676	GTX3LP	UINT16	CML1 / GTX3 Output Low Period Count
0x678	GTX3Samp	UINT32	CML1 / GTX3 Output Number of 40 bit word patterns
0x800 – 0xFFF	DataBuf		Data Buffer Receive Memory
0x1000 – 0x17FF	Diagnostics counters		
0x1800 – 0x1FFF	TxDataBuf		Data Buffer Transmit Memory
0x2000 – 0x3FFF	EventLog		512 x 16 byte position Event Log
0x4000 – 0x4FFF	MapRam1		Event Mapping RAM 1
0x5000 – 0x5FFF	MapRam2		Event Mapping RAM 2
0x8000 – 0x80FF	configROM		
0x8100 – 0x81FF	scratchRAM		
0x8200 – 0x82FF	SFPEEPROM		SFP Transceiver EEPROM contents (SFP address 0xA0)

continues on next page

Table 11 – continued from previous page

Address	Register	Type	Description
0x8300 – 0x83FF	SFPDIAG		SFP Transceiver diagnostics (SFP address 0xA2)
0x8800	DataBufRXSize0	UINT32	Segmented Data Buffer Segment 0 Receive Size Register
0x8804	SDataBufRXSize0	UINT32	Segmented Data Buffer Segment 1 Receive Size Register
0x89FC	SDataBufRXSize127	UINT32	Segmented Data Buffer Segment 127 Receive Size Register
0x8F80 – 0x8F8F	SDataBufSIrqEna		*Segmented Data Buffer Segment Interrupt Enable Register*
0x8FA0 – 0x8FAF	SDataBufCSFlag		*Segmented Data Buffer Segment Checksum Flags*
0x8FC0 – 0x8FCF	SDataBufOVFlag		*Segmented Data Buffer Segment Overflow Flags*
0x8FE0 – 0x8FEF	SDataBufRxFlag		*Segmented Data Buffer Segment Receive Flags*
0x9000 – 0x97FF	SDataBufData		Segmented Data Buffer Segment Data Memory
0xA000 – 0xA7FF	SDataBufData		Segmented Data Buffer Transmit Memory
0xC000 – 0xFFFF	SeqRam		Sequence RAM
0x20000 – 0x23FFF	GTX0MEM	Pattern memory:	
			16k bytes GTX output 0 (VME-EVR-300)
0x24000 – 0x27FFF	GTX1MEM	Pattern memory:	
			16k bytes GTX output 1 (VME-EVR-300)
0x28000 – 0x2BFFF	GTX2MEM	Pattern memory:	
			16k bytes GTX output 2 (VME-EVR-300)
0x2C000 – 0x2FFFF	GTX3MEM	Pattern memory:	
			16k bytes GTX output 3 (VME-EVR-300)

2.14.2 Status Register

address	bit 31	bit 30	bit 29	bit 28	bit 27	bit 26	bit 25	bit 24
0x000	DBUS7	DBUS6	DBUS5	DBUS4	DBUS3	DBUS2	DBUS1	DBUS0
address	bit 23	bit 22	bit 21	bit 20	bit 19	bit 18	bit 17	bit 16
0x001								LEGVIO
address	bit 15	bit 14	bit 13	bit 12	bit 11	bit 10	bit 9	bit 8
0x002								
address	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
0x003	SFPMOD	LINK	LOGSTP					

Bit	Function
DBUS7	Read status of DBUS bit 7
DBUS6	Read status of DBUS bit 6
DBUS5	Read status of DBUS bit 5
DBUS4	Read status of DBUS bit 4
DBUS3	Read status of DBUS bit 3
DBUS2	Read status of DBUS bit 2
DBUS1	Read status of DBUS bit 1
DBUS0	Read status of DBUS bit 0
LEGVIO	Legacy VIO (series 100, 200 and 230)
SFPMOD	SFP module status:
	‘0’ – plugged in
	‘1’ – no module installed
LINK	Link status:
	‘0’ – link down
	‘1’ – link up
LOGSTP	Event Log stopped flag

2.14.3 Control Register

address	bit 31	bit 30	bit 29	bit 28	bit 27	bit 26	bit 25	bit 24
0x004	EVREN	EVFWD	TXLP	RXLP	OUTEN	SRST	LEMDE	GTXIO
address	bit 23	bit 22	bit 21	bit 20	bit 19	bit 18	bit 17	bit 16
0x005		DCENA						
address	bit 15	bit 14	bit 13	bit 12	bit 11	bit 10	bit 9	bit 8
0x006		TSDBUS	RSTS			LTS	MAPEN	MAPRS
address	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
0x007	LOGRS	LOGEN	LOGDIS	LOGSE	RSFIFO			

Bit	Function
EVREN	Event Receiver Master enable
EVFWD	Event forwarding enable:
	0 – Events not forwarded
	1 – Events received with forward bit in mapping RAM set are sent back on TX
TXLP	Transmitter loopback:
	0 – Receive signal from SFP transceiver (normal operation)
	1 – Loopback EVR TX into EVR RX
RXLP	Receiver loopback:
	0 – Transmit signal from EVR on SFP transceiver TX
	1 – Loopback SFP RX on SFP TX
OUTEN	Output enable for FPGA external components / IFB-300 (cPCI-EVRTG-300, PCIe-EVR-300, PXIe-EVR-300I)

continues on next page

Table 12 – continued from previous page

Bit	Function
	0 – disable outputs
	1 – enable outputs
SRST	Soft reset IP
LEMDE	Little endian mode (cPCI-EVR-300, PCIe-EVR-300)
	0 – PCI core in big endian mode (power up default)
	1 – PCI core in little endian mode
GTXIO	GUN-TX output hardware inhibit override
	0 – honor hardware inhibit signal (default)
	1 – inhibit override, don't care about hardware inhibit input state
	DCENA Delay compensation mode enable
	0 – Delay compensation mode disable (receive FIFO depth controlled by DC Target).
	1 – Delay compensation mode enable (receive FIFO depth controlled by DC Target - Datapath Delay).
TSDBUS	Use timestamp counter clock on DBUS4
RSTS	Reset Timestamp. Write 1 to reset timestamp event counter and timestamp latch.
LTS	Latch Timestamp: Write 1 to latch timestamp from timestamp event counter to timestamp latch.
MAPEN	Event mapping RAM enable.
MAPRS	Mapping RAM select bit for event decoding:
	0 – select mapping RAM 1
	1 – select mapping RAM 2.
LOGRS	Reset Event Log. Write 1 to reset log.
LOGEN	Enable Event Log. Write 1 to (re)enable event log.
LOGDIS	Disable Event Log. Write 1 to disable event log.
LOGSE	Log Stop Event Enable.
RSFIFO	Reset Event FIFO. Write 1 to clear event FIFO.

2.14.4 Interrupt Flag Register

address	bit 31	bit 30	bit 29	bit 28	bit 27	bit 26	bit 25	bit 24
0x008								
address	bit 23	bit 22	bit 21	bit 20	bit 19	bit 18	bit 17	bit 16
0x009				IFSOV				IFSHF
	bit 15	bit 14	bit 13	bit 12	bit 11	bit 10	bit 9	bit 8
0x00A				IFSSTO				IFSSTA
address	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
0x00B	IFSEGD	IFLINK	IFDBUF	IFHW	IFEV	IFHB	IFFF	IFVIO

Bit	Function
IFSOV	Sequence RAM sequence roll over interrupt flag
IFSHF	Sequence RAM sequence halfway through interrupt flag
IFSSTO	Sequence RAM sequence stop interrupt flag
IFSSTA	Sequence RAM sequence start interrupt flag
IFSEGD	Segmented data buffer interrupt flag
IFLINK	Link state change interrupt flag
IFDBUF	Data buffer interrupt flag
IFHW	Hardware interrupt flag (mapped signal)
IFEV	Event interrupt flag
IFHB	Heartbeat interrupt flag
IFFF	Event FIFO full flag
IFVIO	Receiver violation flag

2.14.5 Interrupt Enable Register

address	bit 31	bit 30	bit 29	bit 28	bit 27	bit 26	bit 25	bit 24
0x00C	IRQEN							
address	bit 23	bit 22	bit 21	bit 20	bit 19	bit 18	bit 17	bit 16
0x00D				IESOV				IESHF
	bit 15	bit 14	bit 13	bit 12	bit 11	bit 10	bit 9	bit 8
0x00E				IESSTO				IESSTA
address	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
0x00F	IESEGD	IELINK	IEDBUF	IEHW	IEEV	IEHB	IEFF	IEVIO

Bit	Function
IRQEN	Master interrupt enable:
	0 – disable all interrupts
	1 – allow interrupts
IESOV	Sequence RAM sequence roll over interrupt enable
IESHF	Sequence RAM sequence halfway through interrupt enable
IESSTO	Sequence RAM sequence stop interrupt enable
IESSTA	Sequence RAM sequence start interrupt enable
IESEGD	Segmented data buffer interrupt enable
IELINK	Link state change interrupt enable
IEDBUF	Data buffer interrupt enable
IEHW	Hardware interrupt enable (mapped signal)
IEEV	Event interrupt enable
IEHB	Heartbeat interrupt enable
IEFF	Event FIFO full interrupt enable
IEVIO	Receiver violation interrupt enable

2.14.6 Hardware Interrupt Mapping Register

address	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
0x013	MapID [7]	MapID [6]	MapID [5]	MapID [4]	MapID [3]	MapID [2]	MapID [1]	MapID [0]

Table: mapping IDs

Mapping ID	Signal
0 to n-1	Pulse generator output (number n of pulse generators depends on HW and firmware version)
n to 31	(Reserved)
32	Distributed bus bit 0 (DBUS0)
…	…
39	Distributed bus bit 7 (DBUS7)
40	Prescaler 0
41	Prescaler 1
42	Prescaler 2
43 to 47	(Reserved)
48	Flip-flop output 0
…	…
55	Flip-flop output 7
56	to 58 (Reserved)
59	Event clock output (only on PXIe-EVR-300)
60	Event clock output with 180° phase shift (only on PXIe-EVR-300)
61	Tri-state output (for PCIe-EVR-300DC with input module populated in IFB-300's Universal I/O slot)
62	Force output high (logic 1)
63	Force output low (logic 0)

Flip-flop Outputs (from FW version 0E0207)

There are 8 flip-flop outputs. Each of these is using two pulse generators, one for setting the output high and the other one for resetting the output low. In the table below you can see the relationship between flip-flops and pulse generators and the output mapping IDs.

flip-flop	Mapping ID	Set	Reset
0	48	Pulse gen. 0	Pulse gen. 1
1	49	Pulse gen. 2	Pulse gen. 3
2	50	Pulse gen. 4	Pulse gen. 5
3	51	Pulse gen. 6	Pulse gen. 7
4	52	Pulse gen. 8	Pulse gen. 9
5	53	Pulse gen. 10	Pulse gen. 11
6	54	Pulse gen. 12	Pulse gen. 13
7	55	Pulse gen. 14	Pulse gen. 15

2.14.7 Software Event Register

address	bit 15	bit 14	bit 13	bit 12	bit 11	bit 10	bit 9	bit 8
0x01A							SWPEND	SWENA
address	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
0x01B	EVC[7]	EVC[6]	EVC[5]	EVC[4]	EVC[3]	EVC[2]	EVC[1]	EVC[0]

Bit	Function
SWPEND	Event code waiting to be inserted (read-only). A new event code may be written
	to the event code register when this bit reads ‘0’.
SWENA	Enable software event
	When enabled ‘1’ a new event will be inserted into the receive event stream
	when event code is written to the event code register.
EVC[7:0]	Event Code to be inserted into receive event stream

2.14.8 PCI Interrupt Enable Register

address	bit 31	bit 30	bit 29	bit 28	bit 27	bit 26	bit 25	bit 24
0x01c		PCIIE						

Bit	Function
PCIIE	PCI core interrupt enable (PCIe-EVR-300DC, mTCA-EVR-300)

This bit is used by the low level driver to disable further interrupts before the first interrupt has been handled in user space

2.14.9 Receive Data Buffer Control and Status Register

address	bit 15	bit 14	bit 13	bit 12	bit 11	bit 10	bit 9	bit 8
0x022	DBRX/DBENA	DBRDY/DBDIS	DBCS	DBEN	RXSIZE[11]	RXSIZE[10]	RXSIZE[9]	RXSIZE[8]
address	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
0x023	RXSIZE[7]	RXSIZE[6]	RXSIZE[5]	RXSIZE[4]	RXSIZE[3]	RXSIZE[2]	RXSIZE[1]	RXSIZE[0]

Bit	Function
DBRX	Data Buffer Receiving (read-only)
DBENA	Set-up for Single Reception (write '1' to set-up)
DBRDY	Data Buffer Transmit Complete / Interrupt Flag
DBDIS	Stop Reception (write '1' to stop/disable)
DBCS	Data Buffer Checksum Error (read-only)
	Flag is cleared by writing '1' to DBRX or DBRDY or disabling data buffer
DBEN	Data Buffer Enable Data Buffer Mode
	'0' – Distributed bus not shared with data transmission, full speed distributed bus
	'1' – Distributed bus shared with data transmission, half speed distributed bus
RXSIZE	Data Buffer Received Buffer Size (read-only).

Transmit Data Buffer Control Register

address	bit 31	bit 30	bit 29	bit 28	bit 27	bit 26	bit 25	bit 24
0x024								
address	bit 23	bit 22	bit 21	bit 20	bit 19	bit 18	bit 17	bit 16
0x025				TXCPT	TXRUN	TRIG	ENA	1
address	bit 15	bit 14	bit 13	bit 12	bit 11	bit 10	bit 9	bit 8
0x026						DTSZ[10]	DTSZ[9]	DTSZ[8]
address	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
0x027	DTSZ[7]	DTSZ[6]	DTSZ[5]	DTSZ[4]	DTSZ[3]	DTSZ[2]	0	0

Bit	Function
TXCPT	Data Buffer Transmission Complete
TXRUN	Data Buffer Transmission Running – set when data transmission has been triggered and has not been completed yet
TRIG	Data Buffer Trigger Transmission
	Write '1' to start transmission of data in buffer
ENA	Data Buffer Transmission enable
	'0' – data transmission engine disabled
	'1' – data transmission engine enabled
DTSZ(10:2)	Data Transfer size 4 bytes to 2k in four byte increments

Transmit Segemented Data Buffer Control Register

ad-dress	bit 31	bit 30	bit 29	bit 28	bit 27	bit 26	bit 25	bit 24
0x028	SADDR[7]	SADDR[6]	SADDR[5]	SADDR[4]	SADDR[3]	SADDR[2]	SADDR[1]	SADDR[0]
address	bit 23	bit 22	bit 21	bit 20	bit 19	bit 18	bit 17	bit 16
0x029				TXCPT	TXRUN	TRIG	ENA	MODE
address	bit 15	bit 14	bit 13	bit 12	bit 11	bit 10	bit 9	bit 8
0x02A						DTSZ[10]	DTSZ[9]	DTSZ[8]
address	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
0x02B	DTSZ[7]	DTSZ[6]	DTSZ[5]	DTSZ[4]	DTSZ[3]	DTSZ[2]	0	0

Bit	Function
SADDR	Transfer Start Segment Address (16 byte segments)
TXCPT	Data Buffer Transmission Complete
TXRUN	Data Buffer Transmission Running – set when data transmission has been triggered and has not been completed yet
TRIG	ata Buffer Trigger Transmission
	Write ‘1’ to start transmission of data in buffer
ENA	Data Buffer Transmission enable
	‘0’ – data transmission engine disabled
	‘1’ – data transmission engine enabled
MODE	Distributed bus sharing mode
	‘0’ – distributed bus not shared with data transmission
	‘1’ – distributed bus shared with data transmission
DTSZ(10:2)	Data Transfer size 4 bytes to 2k in four byte increments

2.14.10 FPGA Firmware Version Register

address	bit 31	bit 30	bit 29	bit 28	bit 27	bit 26	bit 25	bit 24
0x02C				EVR = 0x1	FF[3]	FF[2]	FF[1]	FF[0]
address	bit 23	bit 22	bit 21	bit 20	bit 19	bit 18	bit 17	bit 16
0x02D	SID[7]	SID[6]	SID[5]	SID[4]	SID[3]	SID[2]	SID[1]	SID[0]
address	bit 15	bit 14	bit 13	bit 12	bit 11	bit 10	bit 9	bit 8
0x02E	FID[7]	FID[6]	FID[5]	FID[4]	FID[3]	FID[2]	FID[1]	FID[0]
address	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
0x02F	RID[7]	RID[6]	RID[5]	RID[4]	RID[3]	RID[2]	RID[1]	RID[0]

Bit	Function
Form Factor FF(3:0)	0 – CompactPCI 3U
	1 – PMC
	2 – VME64x
	3 – CompactRIO
	4 – CompactPCI 6U
	6 – PXIe
	7 – PCIe
	8 – mTCA.4
Subrelease ID SID(7:0)	For production releases the subrelease ID counts up from 00.
	For pre-releases this ID is used “backwards” counting down from ff i.e. when
	approaching release 12000207, we have prereleases 12FF0206, 12FE0206,
	12FD0206 etc. in this order.
Firmware ID FID(7:0)	
	00 – Modular Register Map firmware (no delay compensation)
	01 – Reserved
	02 – Delay Compensation firmware
Revision ID RID(7:0)	See end of manual

2.14.11 Clock Control Register

address	bit 31	bit 30	bit 29	bit 28	bit 27	bit 26	bit 25	bit 24
0x050	PLL-LOCK	BWSEL[2]	BWSEL[1]	BWSEL[0]		CLKMD[1]	CLKMD[0]	
address	bit 15	bit 14	bit 13	bit 12	bit 11	bit 10	bit 9	bit 8
0x052							CGLOCK	

Bit	Function
PLL-LOCK	Clock cleaner PLL Locked (read-only) to receiver recovered clock
BWSEL2:0	PLL Bandwidth Select (see Silicon Labs Si5317 datasheet)
	000 – Si5317, BW setting HM (lowest loop bandwidth)
	001 – Si5317, BW setting HL
	010 – Si5317, BW setting MH
	011 – Si5317, BW setting MM
	100 – Si5317, BW setting ML (highest loop bandwidth)
CLKMD1:0	Event clock mode
	00 – Event clock synchronized to upstream EVG. Event clock continues to run with same frequency if link is lost.
	01 – Event clock synchronized to local fractional synthesizer reference.
	10 – Event clock synchronized to upstream EVG. Fall back to local reference if upstream link is lost.
	11 – Event clock synchronized to upstream EVG. Event clock is stopped if link is lost.
CGLOCK	Micrel fractional synthesizer SY87739L locked (read-only). This serves as the
	reference clock for the FPGA internal transceiver and indicates that a valid
	configuration word has been set in the FracDiv control register.

Event FIFO

Note that reading the FIFO event code registers pulls the event code and timestamp/seconds value from the FIFO for access. The correct order to read an event from FIFO is to first read the event code register and after this the timestamp/seconds registers in any order. Every read access to the FIFO event register pulls a new event from the FIFO if it is not empty.

SY87739L Fractional Divider Configuration Word

The fractional synthesizer serves as the reference clock for the FPGA internal transceiver.

Configuration Word	Frequency with 24 MHz reference oscillator
0x0891C100	142.857 MHz
0x00DE816D	125 MHz
0x00FE816D	124.95 MHz
0x0C928166	124.9087 MHz
0x018741AD	119 MHz
0x072F01AD	114.24 MHz
0x049E81AD	106.25 MHz
0x008201AD	100 MHz
0x025B41ED	99.956 MHz
0x0187422D	89.25 MHz
0x0082822D	81 MHz
0x0106822D	80 MHz
0x019E822D	78.900 MHz
0x018742AD	71.4 MHz
0x0C9282A6	62.454 MHz
0x009743AD	50 MHz
0x0C25B43AD	49.978 MHz
0x0176C36D	49.965 MHz

2.14.12 SPI Configuration Flash Registers

address	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
0x0A3	SPI-DATA[7]	SPI-DATA[6]	SPI-DATA[5]	SPI-DATA[4]	SPI-DATA[3]	SPI-DATA[2]	SPI-DATA[1]	SPI-DATA[0]
address	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
0x0A7	E	RRDY	TRDY	TMT	TOE	ROE	OE	SSO

Bit	Function
SPIDATA(7:0)	Read SPI data byte / Write SPI data byte
E	Overrun Error flag
RRDY	Receiver ready, if ‘1’ data byte waiting in SPI_DATA
TRDY	Transmitter ready, if ‘1’ SPI_DATA is ready to accept new transmit data byte
TMT	Transmitter empty, if ‘1’ data byte has been transmitted
TOE	Transmitter overrun error
ROE	Receiver overrun error
OE	Output enable for SPI pins, ‘1’ enable SPI pins
SSO	Slave select output enable for SPI slave device, ‘1’ device selected

2.14.13 Delay Compensation Status Register

address	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
0x0BE						PDVLD[2]	PDVLD[1]	PDVLD[0]
address	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
0x0BF					DLYL	DLYS		DCLOCK

Bit	Function
PDVLD(2:0)	Path delay valid
	000 – Path delay value not valid from master EVM to EVR
	001 – Path delay value valid (coarse/quick acquisition)
	011 – Path delay value valid (medium precision/slow acquisition)
	111 – Path delay value valid (fine precision/slow acquisition)
DLYL	Delay setting too long (delay shorter than target)
DLYS	Delay setting too short (delay longer than target)
DCLOCK	Delay fifo locked to setting/delay value

2.14.14 Sequence RAM Control Register

address	bit 31	bit 30	bit 29	bit 28	bit 27	bit 26	bit 25	bit 24
0x0E0							SQRUN	SQENA
address	bit 23	bit 22	bit 21	bit 20	bit 19	bit 18	bit 17	bit 16
0x0E1			SQSWT	SQSNG	SQREC	SQRES	SQDIS	SQEN
address	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
0x0E3	SQTSEL[7]	SQTSEL[6]	SQTSEL[5]	SQTSEL[4]	SQTSEL[3]	SQTSEL[2]	SQTSEL[1]	SQTSEL[0]

Bit	Function
SQRUN	Sequence RAM running flag (read-only)
SQENA	Sequence RAM enabled flag (read_only)
SQSWT	Sequence RAM software trigger, write '1' to trigger
SQSNG	Sequence RAM single mode
SQREC	Sequence RAM recycle mode
SQRES	Sequence RAM reset, write '1' to reset
SQDIS	Sequence RAM disable, write '1' to disable
SQEN	Sequence RAM enable, write '1' to enable/arm
SQTSEL	0 to n-1 – Pulse generator output
	n to 31 – (Reserved)
	32 to 39 – Distributed bus bit 0 (DBUS0) to bit 7 (DBUS7)
	40 to 47 – Prescaler 0 to Prescaler 7
	48 to 58 – (Reserved)
	61 – Software trigger
	62 – Continuous trigger
	63 – Trigger disabled

2.14.15 Prescaler Pulse Trigger Registers

Each bit in the Prescaler Pulse Trigger Register corresponds to one pulse generator trigger. If for instance bit 0 is set, pulse generator 0 gets trigger on each 0 to 1 transition of the corresponding prescaler.

2.14.16 Distributed Bus Pulse Trigger Registers

Each bit in the Distributed Bus Pulse Trigger Register corresponds to one pulse generator trigger. If for instance bit 0 is set, pulse generator 0 gets trigger on each 0 to 1 transition of the corresponding distributed bus bit.

2.14.17 Pulse Generator Registers

address	bit 31	bit 30	bit 29	bit 28	bit 27	bit 26	bit 25	bit 24
0x200	PxMASK[7]	PxMASK[6]	PxMASK[5]	PxMASK[4]	PxMASK[3]	PxMASK[2]	PxMASK[1]	PxMASK[0]
address	bit 23	bit 22	bit 21	bit 20	bit 19	bit 18	bit 17	bit 16
0x201	PxENA[7]	PxENA[6]	PxENA[5]	PxENA[4]	PxENA[3]	PxENA[2]	PxENA[1]	PxENA[0]
address	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
0x203	PxOUT	PxSWS	PxSWR	PxPOL	PxMRE	PxMSE	PxMTE	PxENA

address	bit 31…bit 0
0x204	Pulse Generator 0 Prescaler Register
address	bit 31…bit 0
0x208	Pulse Generator 0 Delay Register
address	bit 31…bit 0
0x20C	Pulse Generator 0 Width Register

Bit	Function
PxMASK(7:0)	Pulse HW Mask Register
	0 – HW masking disabled
	1 – HW masking enabled. When corresponding gate bit is active ‘1’ pulse triggers are blocked
PxENA(7:0)	Pulse HW Enable Register
	0 – HW enabling inactive
	1 – HW enabling active. When corresponding gate bit is inactive ‘0’ pulse triggers are blocked
PxOUT	Pulse Generator Output (read-only)
PxSWS	Pulse Generator Software Set
PxSWC	Pulse Generator Software Reset
PxPOL	Pulse Generator Output Polarity
	0 – normal polarity
	1 – inverted polarity
PxMRE	Pulse Generator Event Mapping RAM Reset Event Enable
	0 – Reset events disabled
	1 – Mapped Reset Events reset pulse generator output
PxMSE	Pulse Generator Event Mapping RAM Set Event Enable
	0 – Set events disabled
	1 – Mapped Set Events set pulse generator output
PxMTE	Pulse Generator Event Mapping RAM Trigger Event Enable
	0 – Event Triggers disabled
	1 – Mapped Trigger Events trigger pulse generator
PxENA	Pulse Generator Enable
	0 – generator disabled
	1 – generator enabled

2.14.18 Input Mapping Registers

The same bit mapping applies to Front Panel Inputs, Universal Inputs and Backplane Inputs.

address	bit 31	bit 30	bit 29	bit 28	bit 27	bit 26	bit 25	bit 24
0x500	FPIN0		EXTLV0	BCKLE0	EXTLE0	EXTED0	BCKEV0	EXTEV0
address	bit 23	bit 22	bit 21	bit 20	bit 19	bit 18	bit 17	bit 16
0x501	T0DB7	T0DB6	T0DB5	T0DB4	T0DB3	T0DB2	T0DB1	T0DB0

address	bit 15 … bit 8
0x502	Backward Event Code Register for front panel input 0
address	bit 7 … bit 0
0x503	External Event Code Register for front panel input 0

address	bit 31	bit 30	bit 29	bit 28	bit 27	bit 26	bit 25	bit 24
0x504	FPIN1		EXTLV1	BCKLE1	EXTLE1	EXTED1	BCKEV1	EXTEV1
address	bit 23	bit 22	bit 21	bit 20	bit 19	bit 18	bit 17	bit 16
0x505	T1DB7	T1DB6	T1DB5	T1DB4	T1DB3	T1DB2	T1DB1	T1DB0

address	bit 15 … bit 8
0x506	Backward Event Code Register for front panel input 1
address	bit 7 … bit 0
0x507	External Event Code Register for front panel input 1

Bit	Function
FPINx	Front panel Input x state.
	0 – low
	1 – high
EXTLVx	Backward HW Event Level Sensitivity for input x
	0 – active high
	1 – active low
BCKLEx	Backward HW Event Level Trigger enable for input x
	0 – disable level events
	1 – enable level events, send out backward event code every 1 us when input is
	active (see EXTLVx for level sensitivity)
EXTLEx	External HW Event Level Trigger enable for input x
	0 – disable level events
	1 – enable level events, apply external event code to active mapping RAM
	every 1 us when input is active (see EXTLVx for level sensitivity)
EXTEDx	Backward HW Event Edge Sensitivity for input x
	0 – trigger on rising edge
	1 – trigger on falling edge
BCKEVx	Backward HW Event Edge Trigger Enable for input x
	0 – disable backward HW event
	1 – enable backward HW event, send out backward event code on detected
	edge of hardware input (see EXTEDx bit for edge)
EXTEVx	External HW Event Enable for input x
	0 – disable external HW event
	1 – enable external HW event, apply external event code to active mapping
	RAM on edge of hardware input
TxDB7-TxDB0	Backward distributed bus bit enable:
	0 – disable distributed bus bit
	1 – enable distributed bus bit control from hardware input: e.g. when TxDB7
	is ‘1’ the hardware input x state is sent out on distributed bus bit 7.

address	bit 31…bit 16
0x610	Frequency mode trigger position

address	bit 15	bit 14	bit 13	bit 12	bit 11	bit 10	bit 9	bit 8
0x612					GTX3MD	GTX2MD	GTXPH1	GTXPH0
address	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
0x613	CMLRC	CMLTL	CMLMD(1:0)			CMLRES	CMLPWD	CMLENA

Bit	Function
GTX3MD	GUN-TX-300 Mode (cPCI-EVRTG-300 only)
	0 – CML/GTX Mode
	1 – SFP output in GUN-TX-300 Mode
GTX2MD	GUN-TX-203 Mode (cPCI-EVRTG-300 only)
	0 – CML/GTX Mode
	1 – SFP output in GUN-TX-203 Mode
GTXPH1:0	GUN-TX-203 Trigger output phase shift (cPCI-EVRTG-300 only)
	00 – no delay
	01 – output pulse delayed by ¼ event clock period (2 ns)
	10 – output pulse delayed by ½ event clock period (4 ns)
	11 – output pulse delayed by ¾ event clock period (6 ns)
CMLRC	CML Pattern recycle
CMLTL	CML Frequency mode trigger level
CMLMD	CML Mode Select:
	00 = classic mode
	01 = frequency mode
	10 = pattern mode
	11 = undefined
CMLRES	CML Reset
	1 = reset CML output (default on EVR power up)
	0 = normal operation
CMLPWD	CML Power Down
	1 = CML outputs powered down (default on EVR power up)
	0 = normal operation
CMLENA	CML Enable
	0 = CML output disabled (default on EVR power up)
	1 = CML output enabled

2.14.19 Data Buffer Segment Interrupt Enable Register

address	bit 31	bit 30	bit 29	bit 28	bit 27	bit 26	bit 25	bit 24
0x8F80	DBIE00	DBIE01	DBIE02	DBIE03	DBIE04	DBIE05	DBIE06	DBIE07
address	bit 23	bit 22	bit 21	bit 20	bit 19	bit 18	bit 17	bit 16
0x8F81	DBIE08	DBIE09	DBIE0A	DBIE0B	DBIE0C	DBIE0D	DBIE0E	DBIE0F
…								
address	bit 15	bit 14	bit 13	bit 12	bit 11	bit 10	bit 9	bit 8
0x8F8E	DBIE70	DBIE71	DBIE72	DBIE73	DBIE74	DBIE75	DBIE76	DBIE77
address	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
0x8F8F	DBIE78	DBIE79	DBIE7A	DBIE7B	DBIE7C	DBIE7D	DBIE7E	DBIE7F

Bit	Function
DBIExx	Data Buffer Segment (16-byte segments) Interrupt Enable:
	0 – Interrupt for segment disabled
	1 – Interrupt for segment enabled
	An interrupt will occur when the segment's receive flag is active. To enable
	Data Buffer interrupts the IEDBUF bit in the Interrupt Enable Register has to be set.

2.14.20 Data Buffer Checksum Flag Register

address	bit 31	bit 30	bit 29	bit 28	bit 27	bit 26	bit 25	bit 24
0x8FA0	DBCS00	DBCS01	DBCS02	DBCS03	DBCS04	DBCS05	DBCS06	DBCS07
address	bit 23	bit 22	bit 21	bit 20	bit 19	bit 18	bit 17	bit 16
0x8FA1	DBCS08	DBCS09	DBCS0A	DBCS0B	DBCS0C	DBCS0D	DBCS0E	DBCS0F
address	bit 15	bit 14	bit 13	bit 12	bit 11	bit 10	bit 9	bit 8
0x8FAE	DBCS70	DBCS71	DBCS72	DBCS73	DBCS74	DBCS75	DBCS76	DBCS77
address	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
0x8FAF	DBCS78	DBCS79	DBCS7A	DBCS7B	DBCS7C	DBCS7D	DBCS7E	DBCS7F

Bit	Function
DBCSxx	Data Buffer Segment (16-byte segments) Checksum Flag:
	0 – Checksum OK
	1 – Checksum error
	This flag is cleared by writing a '1' into the segment's DBRXxx bit in the DataBufRxFlag register.

Data Buffer Overflow Flag Register

address	bit 31	bit 30	bit 29	bit 28	bit 27	bit 26	bit 25	bit 24
0x8FC0	DBOV00	DBOV01	DBOV02	DBOV03	DBOV04	DBOV05	DBOV06	DBOV07
address	bit 23	bit 22	bit 21	bit 20	bit 19	bit 18	bit 17	bit 16
0x8FC1	DBOV08	DBOV09	DBOV0A	DBOV0B	DBOV0C	DBOV0D	DBOV0E	DBOV0F
…								
address	bit 15	bit 14	bit 13	bit 12	bit 11	bit 10	bit 9	bit 8
0x8FCE	DBOV70	DBOV71	DBOV72	DBOV73	DBOV74	DBOV75	DBOV76	DBOV77
address	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
0x8FCF	DBOV78	DBOV79	DBOV7A	DBOV7B	DBOV7C	DBOV7D	DBOV7E	DBOV7F

Bit	Function
DBOVxx	Data Buffer Segment (16-byte segments) Overflow Flag:
	0 – No overflow condition
	1 – Overflow: a new packet has been received before the DBRX flag for this segment was cleared
	This flag is cleared by writing a ‘1’ into the segment’s DBRXxx bit in the DataBufRxFlag register.

2.14.21 Data Buffer Receive Flag Register

address	bit 31	bit 30	bit 29	bit 28	bit 27	bit 26	bit 25	bit 24
0x8FE0	DBRX00	DBRX01	DBRX02	DBRX03	DBRX04	DBRX05	DBRX06	DBRX07
address	bit 23	bit 22	bit 21	bit 20	bit 19	bit 18	bit 17	bit 16
0x8FE1	DBRX08	DBRX09	DBRX0A	DBRX0B	DBRX0C	DBRX0D	DBRX0E	DBRX0F
address	bit 15	bit 14	bit 13	bit 12	bit 11	bit 10	bit 9	bit 8
0x8FEE	DBRX70	DBRX71	DBRX72	DBRX73	DBRX74	DBRX75	DBRX76	DBRX77
address	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
0x8FEF	DBRX78	DBRX79	DBRX7A	DBRX7B	DBRX7C	DBRX7D	DBRX7E	DBRX7F

Bit	Function
DBRXxx	Data Buffer Segment (16-byte segments) Receive Flag:
	0 – No packet received
	1 – Data packet received in this segment
	This flag is cleared by writing a ‘1’ into the segment’s DBRXxx bit.

2.14.22 SFP Module EEPROM and Diagnostics

Small Form Factor Pluggable (SFP) transceiver modules provide a means to identify the module by accessing an EEPROM. As an advanced feature some modules also support reading dynamic information including module temperature, receive and transmit power levels etc. from the module. The EVR gives access to all of this information through a memory window of 2 × 256 bytes. The first 256 bytes consist of the EEPROM values and the rest of the advanced values.

BASE ID FIELDS

Byte #	Field size	Notes	Value
0	1	Type of serial transceiver	0x03 = SFP transceiver
1	1	Extended identifier of type serial transceiver	0x04 = serial ID module definition
2	1	Code for connector type 0x07 = LC	
3 – 10	8	Code for electronic compatibility or optical compatibility	
11	1	Code for serial encoding algorithm	
12	1	Nominal bit rate, units of 100 MBits/sec	
13	1	Reserved	
14	1	Link length supported for 9/125 m fiber, units of km	
15	1	Link length supported for 9/125 m fiber, units of 100 m	
16	1	Link length supported for 50/125 m fiber, units of 10 m	
17	1	Link length supported for 62.5/125 m fiber, units of 10 m	
18	1	Link length supported for copper, units of meters	
19	1	Reserved	
20 – 35	16	SFP transceiver vendor name (ASCII)	
36	1	Reserved	
37 – 39	3	SFP transceiver vendor IEEE company ID	
40 – 55	16	Part number provided by SFP transceiver vendor (ASCII)	
56 – 59	4	Revision level for part number provided by vendor (ASCII)	
60 – 62	3	Reserved	
63	1	Check code for Base ID Fields	

EXTENDED ID FIELDS

Byte #	Field size	Notes	Value
64 – 65	2	Indicated which optional SFP signals are implemented	
66	1	Upper bit rate margin, units of %	
67	1	Lower bit rate margin, units of %	
68 – 83	16	Serial number provided by vendor (ASCII)	
84 – 91	8	Vendor's manufacturing date code	
92 – 94	3	Reserved	
95	1	Check code for the Extended ID Fields	

VENDOR SPECIFIC ID FIELDS

Byte #	Field size	Notes	Value
96 – 127	32	Vendor specific data	
128 – 255		Reserved	

ENHANCED FEATURE SET MEMORY

Byte #	Field size	Notes
256 – 257	2	Temp H Alarm Signed twos complement integer in increments of 1/256 °C
258 – 259	2	Temp L Alarm Signed twos complement integer in increments of 1/256 °C
260 – 261	2	Temp H Warning Signed twos complement integer in increments of 1/256 °C
262 – 263	2	Temp L Warning Signed twos complement integer in increments of 1/256 °C
264 – 265	2	VCC H Alarm Supply voltage decoded as unsigned integer in increments of 100 V
266 – 267	2	VCC L Alarm Supply voltage decoded as unsigned integer in increments of 100 V
268 – 269	2	VCC H Warning Supply voltage decoded as unsigned integer in increments of 100 V
270 – 271	2	VCC L Warning Supply voltage decoded as unsigned integer in increments of 100 V
272 – 273	2	Tx Bias H Alarm Laser bias current decoded as unsigned integer in increment of 2 A
274 – 275	2	Tx Bias L Alarm Laser bias current decoded as unsigned integer in increment of 2 A
276 – 277	2	Tx Bias H Warning Laser bias current decoded as unsigned integer in increment of 2 A
278 – 279	2	Tx Bias L Warning Laser bias current decoded as unsigned integer in increment of 2 A
280 – 281	2	Tx Power H Alarm Transmitter average optical power decoded as unsigned integer in increments of 0.1 W
282 – 283	2	Tx Power L Alarm Transmitter average optical power decoded as unsigned integer in increments of 0.1 W
284 – 285	2	Tx Power H Warning Transmitter average optical power decoded as unsigned integer in increments of 0.1
286 – 287	2	Tx Power L Warning Transmitter average optical power decoded as unsigned integer in increments of 0.1
288 – 289	2	Rx Power H Alarm Receiver average optical power decoded as unsigned integer in increments of 0.1 W
290 – 291	2	Rx Power L Alarm Receiver average optical power decoded as unsigned integer in increments of 0.1 W
292 – 293	2	Rx Power H Warning Receiver average optical power decoded as unsigned integer in increments of 0.1 W
294 – 295	2	Rx Power L Warning Receiver average optical power decoded as unsigned integer in increments of 0.1 W
296 – 311	16	Reserved
312 – 350		External Calibration Constants
351	1	Checksum for Bytes 256 – 350

Table 13 – continued from previous page

Byte #	Field size	Notes
352 – 353	2	Real Time Temperature Signed twos complement integer in increments of 1/256 °C
354 – 355	2	Real Time VCC Power SupplyVoltage Supply voltage decoded as unsigned integer in increments of 100 V
356 – 357	2	Real Time Tx Bias Current Laser bias current decoded as unsigned integer in increment of 2 A
358 – 359	2	Real Time Tx Power Transmitter average optical power decoded as unsigned integer in in- crements of 0.1
360 – 361	2	Real Time Rx Power Receiver average optical power decoded as unsigned integer in increments of 0.1 W
362 – 365	4	Reserved
366	1	Status/Control
367	1	Reserved
368	1	Alarm Flags
369	1	Alarm Flags cont.
370 – 371	2	Reserved
372	1	Warning Flags
373	1	Warning Flags cont.
374 – 511		Reserved/Vendor Specific

2.14.23 Hardware Configuration Summary

	VME-EVR-300	mTCA-EVR-300	PCIe-EVR-300DC
Pulse Generators	24	16	16
FP TTL inputs	2	2	0
FP TTL outputs	0	4	0
FP GTX outputs	4[1]	0	0
FP UNIV I/O / slots	4	0	16 / 8[2]
FP UNIV GPIO pins / slots 16/4 0 / 0 0 / 0			
TB Outputs	16	32	0
TB Inputs	16	32	0
Prescalers	8 x 32 bit	8 x 32 bit	8 x 32 bit

Pulse Generator: Prescaler, Delay, Pulse Width Range (bits)

	VME-EVR-300	mTCA-EVR-300	PCIe-EVR-300DC
0	16, 32, 32	16, 32, 32	16, 32, 32
1	16, 32, 32	16, 32, 32	16, 32, 32
2	16, 32, 32	16, 32, 32	16, 32, 32
3	16, 32, 32	16, 32, 32	16, 32, 32
4	0, 32, 16	0, 32, 16	0, 32, 16
5	0, 32, 16	0, 32, 16	0, 32, 16
6	0, 32, 16	0, 32, 16	0, 32, 16
7	0, 32, 16	0, 32, 16	0, 32, 16
8	0, 32, 16	0, 32, 16	0, 32, 16
9	0, 32, 16	0, 32, 16	0, 32, 16
10	0, 32, 16	0, 32, 16	0, 32, 16
11	0, 32, 16	0, 32, 16	0, 32, 16
12	0, 32, 16	0, 32, 16	0, 32, 16
13	0, 32, 16	0, 32, 16	0, 32, 16
14	0, 32, 16	0, 32, 16	0, 32, 16
15	0, 32, 16	0, 32, 16	0, 32, 16

(^1) One Universal I/O slot (2 outputs), 2 x CML output (^2) Universal I/O is available on the external I/O box

2.14.24 PCIe-EVR-300DC and IFB-300 Connections

Due to its small bracket the PCIe-EVR-300DC has only a SFP transceiver and a micro-SCSI type con- nector to interface to the IFB-300. The cable between the PCIe-EVR-300DC and IFB-300 should be connected/disconnected only when powered down.

Connector / Led Style Level Description Link TX (SFP) LC Optical 850 nm Event link Transmit Green: TX enable Red: Fract.syn. not locked Blue: Event out Link RX (SFP) Next to micro-SCSI

LC Optical 850 nm Event link Receiver Green: link up Red: link violation detected Blue: event led

The interface board IFB-300 has eight Universal I/O slots which can be populated with various types of Universal I/O modules. If an input module is populated in any slot a jumper has to be mounted in that slot's two pin header with marking "Insert jumper for input module". Please note that if an input module is mounted the corresponding Universal Output Mapping has to be tri-stated. Refer to Table 1: Signal mapping IDs for details.

Universal Slot 0/1 signals are hard-wired to the TTLIN 0/1 signals.

Figure 22: IFB-300 Front Panel

Connector / Led Style Level Description UNIV0/1 Universal slot TTL Input / Universal I/O 0/1 UNIV2/3 Universal slot Universal I/O 2/3 UNIV4/5 Universal slot Universal I/O 4/5 UNIV6/7 Universal slot Universal I/O 6/7 UNIV8/9 Universal slot Universal I/O 8/9 UNIV10/11 Universal slot Universal I/O 10/11 UNIV12/13 Universal slot Universal I/O 12/13 UNIV14/15 Universal slot Universal I/O 14/15 LINK Green led RX link up EVIN Yellow led RX event in EVOUT Yellow led RX event led (mapped) RXFAIL Red led RX violation detected

2.14.25 mTCA-EVR-300 Connections

Figure 23: mTCA-EVR-300 Front Panel

Connector / Led Style Level Description USB Micro-USB MMC diagnostics serial port / JTAG interface Link TX (SFP) LC Optical 850 nm Event link Transmit Green: TX enable Red: Fract.syn. not locked Blue: Event out Link RX (SFP) LC Optical 850 nm Event link Receiver Green: link up Red: link violation detected Blue: event led IFB VHDCI LVDS IFB-300 Interface Box connection IN0 LEMO TTL FPTTL0 Trigger input IN1 LEMO TTL (3.3V / 5V) FPTTL1 Trigger input OUT0 LEMO 3.3V LVTTL TTL Front panel output 0 OUT1 LEMO 3.3V LVTTL TTL Front panel output 1 OUT2 LEMO 3.3V LVTTL TTL Front panel output 2 OUT3 LEMO 3.3V LVTTL TTL Front panel output 3 TCLKA mTCA.4 LVDS TCLKA clock on backplane This signal is driven by CML/GTX logic block 0 Mapped as Universal Output 16 TCLKB mTCA.4 LVDS TCLKB clock on backplane This signal is driven by CML/GTX logic block 1 Mapped as Universal Output 17 RX17 mTCA.4 MLVDS Backplane output 0 TX17 mTCA.4 MLVDS Backplane output 1 RX18 mTCA.4 MLVDS Backplane output 2 TX18 mTCA.4 MLVDS Backplane output 3 RX19 mTCA.4 MLVDS Backplane output 4 TX19 mTCA.4 MLVDS Backplane output 5 RX20 mTCA.4 MLVDS Backplane output 6 TX20 mTCA.4 MLVDS Backplane output 7

2.14.26 PCIe-EVR-300DC Firmware Upgrade

The PCIe-EVR-300DC firmware image can be upgraded with the following command after loading the driver in Linux:

dd if=new_image.bit of=/dev/era1

A power cycle is required to load the new configuration image on the PCIe-EVR-300DC.

2.15 EVR Firmware Version Change Log

FW Version	Date	Changes
0200	11.06.2015	- Prototype release
0201	24.09.2015	- Added segmented data buffer block status flags
		- Changed delay compensation FIFO depth from 2k to 4k event cycles
		- Added DCM modulation to improve jitter performance
0203	12.01.2016	- Delay compensation amendments, non-GTX outputs are compensated properly
0204	25.01.2016	- First release for PCIe-EVR-300DC
		- Fixed segmented data buffer flag writes
0204	03.02.2016	- Fixed initial values of GTX outputs
		- GTX output aligment
0205	07.04.2016	- Changed PCIe-EVR-300DC class code to 0x118000.

Table 14 – continued from previous page

FW Version	Date	Changes
		- Moved delay compensation data from first segment to last segment.
		- Fixed dual output mapping for transition board outputs.
		- Added backplane signals to mTCA-EVR.
		- Added delay compensation disabled mode to be able to use DC capable EVRs with pre-DC EVG and f
0206	12.08.2016	- Relocated segmented data buffer to new address location.
		- Replaced earlier data buffer in its original position (maintaining compatibility with 230 series protocol
		- Changed segmented data buffer protocol to use K28.2 as a start symbol
0207	30.08.2016	- Added stand-alone capability: using its internal reference
		the EVR can now operate as a stand-alone pulse generator without event link.
		- EVR can operate as a simple EVG by forwarding internal events
		- Added software event capability
		- Added one EVG type sequencer
030207	23.12.2016	- Changed beacon event code from 0x7a to 0x7e.
		- Added status bits for delay compensation path delay value validity.
		- Added register for topology ID.
040207	09.1.2017	- Repaired “trigger allways” problem with triggering sequencer with pulse generator 19.
		- Added mapping 61 for sequencer software triggering.
050207	19.1.2017	- Fixed running on internal reference for VME-EVR-300.VME-EVR-300
060207	9.2.2017	- Added configurability to handling a lost event clock:
		continue, stop, fallback to reference clock.
		- Further fix to running on internal reference for VME-EVR-300.
070207	6.4.2017	- Fixed CML/GTX operation in stand-alone mode without receiver event stream.
		- Fixed mapping of TCLKA/TCLKB backplane clocks on mTCA-EVR-300.
080207	7.8.2017	- PCIe AXI to OPB bridge fix for overloapping read/write
		operation during block transfers.
		- Added pullup to MODU_SDA and MODU_DEF0.
090207	27.2.2018	- Changes to get design built on Vivado 2017.4
0A0207	18.9.2018	- Changed number of external inputs to 16.
0D0207	20.5.2019	- Added programmable phase shift to prescalers.
0E0207	2.7.2019	- Fix to event FIFO.
		- Added flip-flop outputs.

2.16 Epics device driver for MRF Event Generator (EVG)

::: author [Jayesh Shah, NSLS2, BNL jshah@bnl.gov] :::

\

::: date [Web version/RTD: January 21, 2024] [Last Updated: September 28, 2011] [Last Updated: Ju 28, 2011]

:::

2.16.1 The Source

Source code for the mrfioc2 module, including the EVG support is available in the EPICS modules repository in Github.

VCS Checkout

```
$ git clone https://github.com/epics-modules/mrfioc2.git
```

Currently the driver supports the following models: (To be completed)

- VME-EVG-220
- VME-EVG-230
- cPCI-EVG-220
- cPCI-EVG-230
- PXI-EVG-220
- VME-EVM-300
- MTCA-EVM-300

The required software that this driver depends on:

- EPICS Base >= 3.14.10,
- devLib2 >= 2.8, and
- the MSI tool (included in Base >= 3.15.1).

2.16.2 IOC Deployment

This section outlines a general strategy for adding an EVG to an IOC.

VMEbus based hardware

The VME bus based EVGs are configured using the mrmEvgSetupVME() IOC shell function.

```
mrmEvgSetupVME (
 const char\* id, // EVG card ID
 epicsInt32 slot, // VME slot
 epicsUInt32 vmeAddress, // Desired VME address in A24 space
 epicsInt32 irqLevel // IRQ Level
 epicsInt32 irqVector, // Desired interrupt vector number
)
```

Example call:

```
mrmEvgSetupVME(EVG1,5,0x20000000,3,0x26)
```

In this example EVG1 is defined to be the VME card in slot 5 on VME crate. It is given the A32 base address of 0x20000000 and configured to interrupt on level 3 with vector 0x26.

You can look at example startup script(st.cmd file) for EVG in ./mrfioc2/iocBoot/iocevgmrm directory.

VME64x card configuration

VME64x allows for jumpless configuration of the card, but does not support automatic assignment of resources. Selection of an unused address range and IRQ level/vector is left to the user.

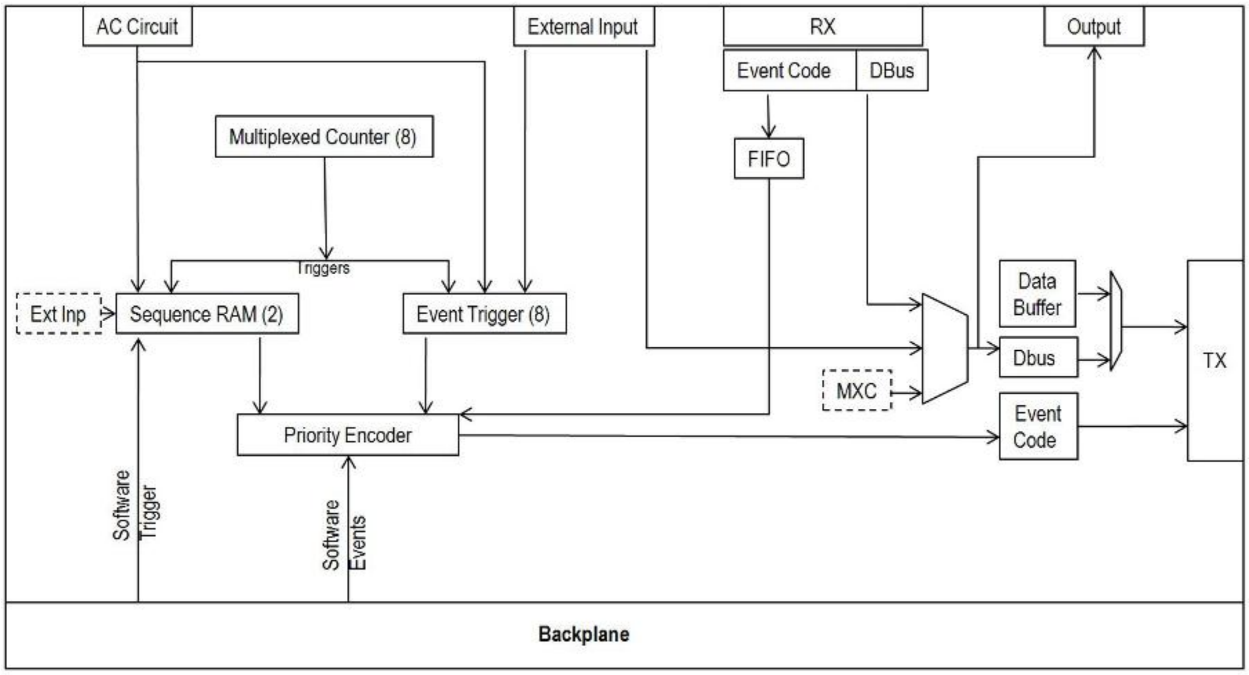
Before setup is done the VME64 identifer fields are verified so that specifying an incorrect slot number is detected and setup will safely abort.

PCI or PCIe based hardware

For PCI (or PCIe) - based EVG or EVM, use

```
mrmEvgSetupPCI (
        const char* id,         // Card Identifier
        const char *spec,       // ID spec. or Bus number
        int d,                              // Device number
        int f)                              // Function number
```

2.16.3 Classes/Sub-Components



EVG

Global EVG Options:

- **Enable** (bo/bi): EVG enable and disable.

Event Clock

All the operations on EVG are synchronized to the event clock, which is derived from either externally provided RF clock or from an on-board fractional synthesizer. Use of the on-board fractional synthesiser is mainly intended for laboratory testing purposes.

The serial link bit rate is 20 times the event clock rate. The acceptable range for the event clock and bit rate is shown in the following table.

	Event Clock	**Bit Rate**
Minimum	50 MHz	1.0 Gb/s
Maximum	142.8 MHz	2.9 Gb/s

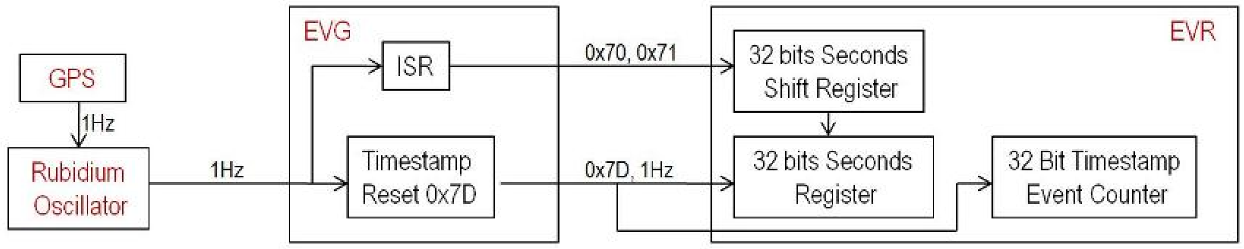
see: evgMrmApp/Db/evgEvtClk.db

- **Source** (bo/bi): The event clock may be derived from external RF clock signal or from an on-board fractional synthesizer.
- **RF reference frequency** (ao/ai) : Set the RF Input frequency in MHz. Frequency can range from 50 to 1600.
- **RF Divider** (longout/longin): Divider to derive desired event clock from RF reference frequency.
- **Fractional Synthesizer frequency** (ao/ai): This frequency could be used to derive event clock.
- **Event Clock Frequency Readback** (ai): Gets the current event clock frequency in MHz.

Timestamping

The Event System provides a global timebase to attach timestamps to all collected data and performed actions at EVR. The time stamping system consists of 32-bit timestamp event counter and a 32-bit seconds counter.

This driver provides you an option of doing timestamping calculations in software as compared to the dedicated hardware as used at few places.



Following are the EVR requirements for accurate timestamping:

- At the start of every second, receive the event code 0x7D which would load the 32-bit seconds count from shift register into the seconds register of EVR and reset the timestamp event counter.
- Have the next 32-bit seconds count shifted in the shift register of EVR before the end of the current second. The shift register is updated serially by loading zeros and ones on receipt of event code 0x70 and 0x71 respectively.

Timestamping at EVG:

For timestamp EVG needs a pulse from the time source at the start of every second. EVG used this 1 pulse per second input to address both requirements of EVR timestamping.

- The first requirement is addressed by using Trigger Events of EVG. We can configure one of the trigger events to send out event code 0x7D when it receives a pulse from the 1PPS source.
- For addressing second requirement EVG uses software events. When timestamping starts the EVG driver obtains the current time from epicsGeneralTime interface(which inturn is synced to a accurate time source) and stores it locally. Now the driver uses the 1 pulse per second output from the time source to update the seconds count of the locally stored time and then sends out next second using event codes 0x70 and 0x71 via software events.

Driver handles different error scenarios:

- EVG uses timer with 1PPS input signal. If it does not detect the signal in some '1 + delta' second the timer goes off and it raises an major alarm and timestamping stops. Once EVG receives the pulse from the 1PPS source it starts the timer again and if the timer does not go off for 5 consecutive pulses then the EVG starts sending timestamps again.
- Before sending out the timestamps to EVR (i.e. the 32-bit seconds count), EVG compares the [stored time](updated by 1 PPS) with the [current time] (obtained from epicsGeneralTime). If they do not match an minor alarm is raised but the stored time is sent as the current time to EVR.

Advantages:

- Using minimum number of EVG inputs for the timestamping purpose.

see: evgMrmApp/Db/evgMrm.db

Records associated with EVG time stamping:

- **Synchronize Timestamp** (bo): Sync the current time with the NTP server.
- **1PPS source** for Timestamping:
  - **Timestamp Input** (mbbo/mbbi):
    - None : Stop timestamping
    - Front : Front Panel Input
    - Univ : Universal Input
    - Rear : Rear Transitional Input

Software Events

Software event is used to send out an event by writing that event code to a particular register in EVG.

- **Enable** (bo/bi): Enable/Disable the transmission of Software Events.
- **Event Code** (longout/longin): Sends out the event code onto the event stream. Event code can range from 0 to 255.

Trigger Events

see: evgMrmApp/Db/evgTrigEvt.db

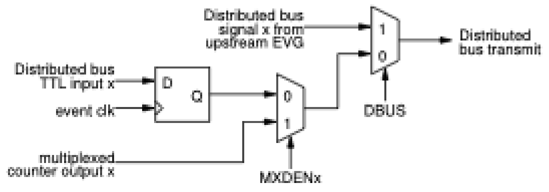
Trigger events are used to send out event codes into the event stream every time an input trigger is received. The stimulus can be a rising edge on an external input signal, a multiplexed counter output or the ac signal.

- **Enable** (bo/bi): Enable/Disable the transmission of Trigger Events.
- **Event Code** (longout/longin): Sets the event code to be sent out, whenever a trigger is received. Event Code can range from 0 to 255.
- **Trigger Source** (mbbo): The trigger could come from one or multiple sources. It could come from any of the external inputs and/or any multiplexed counter output and/or from ac signal. If multiple trigger sources are selected then those signal are OR'ed together and the resulting signal works as the trigger.

Distributed bus

The distributed bus allows transmission of eight simultaneous signals with half of the event clock rate time resolution (for example, 20 ns at 100 MHz event clock rate). The source for distributed bus signals may come from an external source or the signals may be generated with programmable multiplexed counters (MXC) inside the event generator.

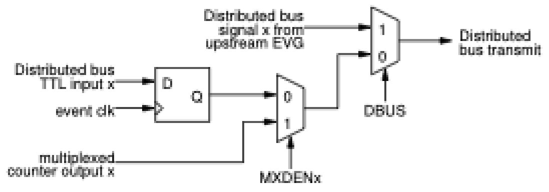
The distributed bus signals may be programmed to be available as hardware outputs on the event receiver.


Distributed bus signal x from upstream EVG
Distributed bus transmit
Distributed bus TTL input x
event clk
D
Q
0
1
DBUS
multiplexed counter output x
MXDENx

- **Signal Source/Map** (mbbo): The bits of the distributed bus can be driven by selecting one of the following sources.
  - Ext Inp : Sampling of the external input signals at event rate.
  - MXC : Sampling of the corresponding multiplexed counter output at event rate.
  - Upstream EVG : Forwarding the state of distributed bus bit of upstream EVG.
- **Selecting the input** (bo): When the source for the distributed bus signals is external input signal, we need to specify which input signal needs to be mapped onto the distributed bus. If multiple inputs are mapped onto a single distributed bus bit then those signals are logically OR'ed together and the resulting signal is used to drive the distributed bus bit.

Multiplexed Counter

There are 8 32-bit multiplexed counters that generate clock signals with programmable frequencies from event clock/2^{32}-1 to event clock/2. The counter outputs may be programmed to trigger events, drive distributed bus signals and trigger sequence RAMs.


Distributed bus signal x from upstream EVG
Distributed bus transmit
Distributed bus TTL input x
event clk
D
Q
0
1
0
1
DBUS
multiplexed counter output x
MXDENx

see evgMrmApp/Db/evgMxc.db

- **Polarity** (bo/bi): Set the Multiplex Counter(Mxc) output polarity.
- **Frequency** (ao/ai): Request a signal with a particular frequency.
- **Prescaler** (longout/longin): Used as counter to produce a signal with a particular frequency.
- **Reset** Reset all the multiplexed counters. After reset all the counters are in phase/sync with each other.

Prescaler value	DutyCycle	Frequency at 125MHz Event Clock
0,1 not allowed	undefined	undefined
2	50/50	62.5 MHz
3	33/66	41.7 MHz
4	50/50	31.25 MHz
5	40/60	25 MHz
…	…	…
2^{32}1	approx. 50/50	0.029 Hz

Input

VME-EVG-230 has 2 Front panel, 4 Universal and 16 Transitional Inputs.

- **External Input Interrupt** (bo): Enable or Disable the External Interrupt. When enabled, an interrupt is received on every rising edge the input signal.

Output

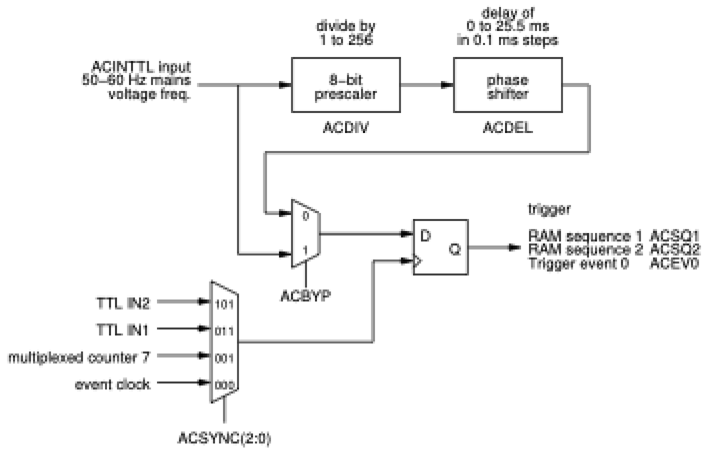
It is used to configure the 4 front panel outputs and 4 four front panel universal outputs.

- **Source**(mbbo/mbbi): The output could be mapped to
  - Any of the eight distributed bus bits
  - Forced logic 1
  - Forced logic 0.

AC Trigger

EVG provides synchronization to the mains voltage frequency or another external clock.

see: evgMrmApp/Db/evgAcTrig.db



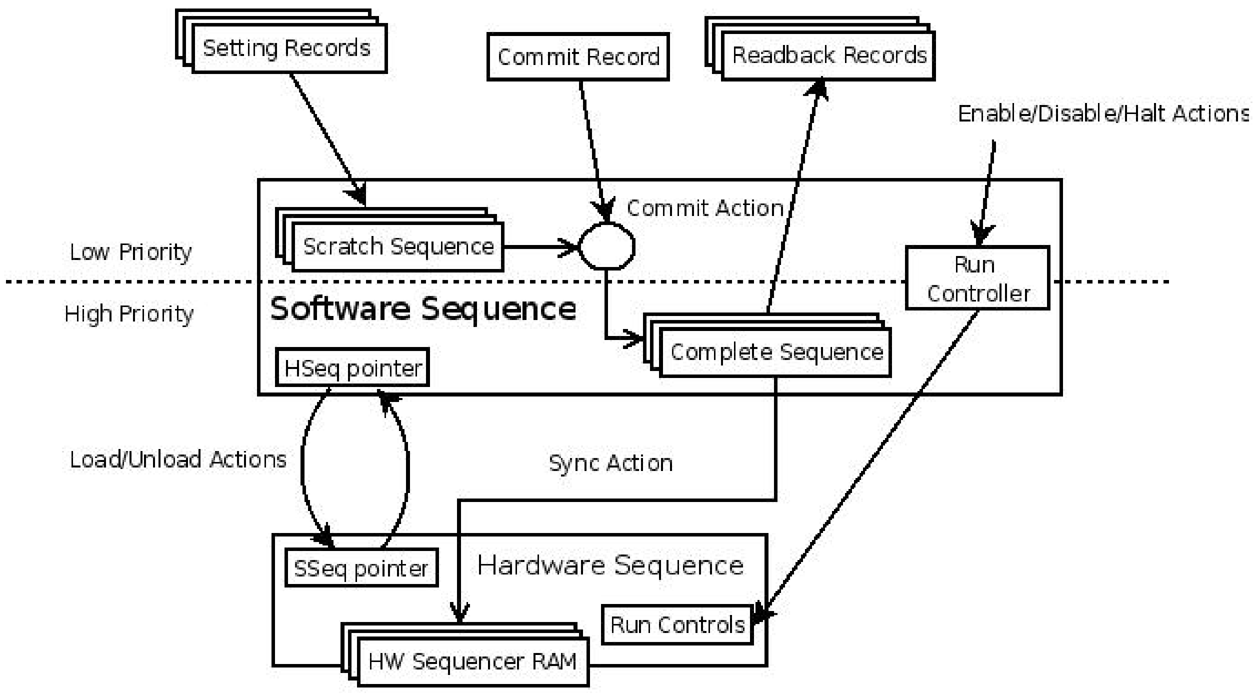
- **Divider**(longout/longin): The mains voltage frequency can be divided by an eight bit programmable divider.
- **Phase**(ao/ai): The output of the divider may be delayed by 0 to 25.5 ms by a phase shifter in 0.1ms steps to adjust the triggering position relative to mains voltage phase.
- **AC Bypass**(bo/bi): It is set to bypass the AC divider and phase shifter circuitry.
- **Sync** (bo/bi): The AC Trigger could be synchronized either with event clock or the output of multiplexed counter 7.

Event Sequencer

Event Sequencer provides a method of transmitting or playing back sequences of events stored in random access memory with defined timing. The EVG has 2 sequence RAMs (sequencers or hard sequence). The sequencer can hold up to 2048 <event code, mask, timeStamp> 3-item tuples. When the sequencer is triggered, an internal counter starts counting. When the counter value matches the timeStamp of the next event, the attached event code is transmitted. Starting with firmware version 0200 a mask field has been added. Bits in the mask field allow masking events from being send out based on external signal input states or software mask bits.

see mrmShared/Db/mrmSoftSeq.template and evgMrmApp/Db/evgSoftSeq.template

Functional block diagram of device support for event sequencer



Device support for sequencer introduces a concept of software sequence(a.k.a. soft sequence). The existence of the software and hardware sequences is an abstraction made to separate the process of assembling a sequence from the process of placing it into hardware. Software sequence maintains a complete ready to run copy of all sequences in the IOC at all times. The IOC is then free to choose which sequence to place into hardware. Since this is a local operation it can be done quickly and efficiently. The IOC can have any number of these soft sequences but at a time the number of these soft sequences that can be loaded into the EVG hardware is restricted by the number of hardware sequences.

As shown in the picture above IOC maintains 2 copies of sequencer data (i.e. Event Code's, Timestamps, Trigger Source and Run Mode). Scratch sequence and complete sequence. Users are allowed to make changes to the scratch sequence directly. Scratch sequence is like the working copy. When user are satisfied with the changes made to the working copy then they can [‘commit’]the soft sequence which will update the complete sequence with the scratch sequence. If the software sequence has an associated hardware sequence with it then the complete sequence is copied to the hardware on **commit**. This is the Sync operation of sequencer.

Parts of the sequence:

- **Event Code List**(waveform): It is used to set the list of the eventCodes of the soft sequence. These eventCodes are transmitted whenever the timeStamp associated with eventCode matches the counter value of sequencer.
- **Timestamp List**(waveform): It is used to set the timeStamps for the events in the soft sequence.
- **Timestamp Input Mode**(bo): There are two mode to enter the timestamping data in the sequencer i.e. EGU and TICKS.
  - EGU: In EGU mode user can enter the timestamps in units of seconds, milli-seconds, micro-seconds or nano-seconds.
  - TICKS: Here user can provide timestamps in terms of Event Clock ticks.
  - All the timestamp values are offset from the time the sequencer receives the trigger.
- **Timestamp Resolution**(mbbo) : If the timestamp input mode is EGU user can use this record to give the units to time.

  - Sec - Input/Output sequencer timestamps in seconds
  - mSec - Input/Output sequencer timestamps in micro-seconds
  - uSec - Input/Output sequencer timestamps in milli-seconds
  - nSec - Input/Output sequencer timestamps in nano-seconds

- **Run Mode**(mbbo/mbbi): Run mode is used determine what will the sequencer do at the end of the sequence. where mode could be any of the following:
  - Single : Disarms the sequencer at the end of the sequence.
  - Automatic : Restarts the sequence immediately after the end of the sequence.
  - Normal : At the end of the sequence, the sequencer rearms and waits for the trigger to restart the sequence.
- **Trigger Source**(mbbo/mbbi): Trigger Src is used to select the source of the trigger, which starts the sequencer.
  - Mxc : Trigger from MXC0 - MXC7
  - AC : Trigger from AC sync logic
  - Software : Trigger from RAM0/RAM1 software trigger.
  - External : Trigger is received from any external input.

Above records only deal with the scratch copy of the soft sequence. They do not directly interact with the hardware sequence.

A soft sequence could be in different states like LOADED or UNLOADED, COMMITTED or DIRTY, ENABLED or DISABLED.

- **Load**(bo): If successful, load causes a soft sequence to move from UNLOADED state to LOADED state. In the LOADED state, an hard sequence is assigned to a soft sequence. If the soft sequence is already in LOADED state then load will return with an error message. The operation will fail if all the hard sequences are already assigned. An allocation scheme ensures that at any given time, each hard sequence is connected to only one soft sequence. Load also copies the last committed data to the hardware.
- **Unload**(bo): The unload causes the soft sequence to enter into UNLOADED state. This operation cannot fail. In unloaded state the assignment of a hard sequence to a soft sequence is released.
- **Commit**(bo): Whenever you modify a soft sequence, the scratch copy in the soft sequence is modified (Refer to evg-seq diagram). Commit causes the changes from the 'scratch sequence' to be copied to the 'complete sequence'. If the soft sequence is loaded, commit also initiates sync operation and copies the changes from complete sequence to the hardware. Modifying the sequenceRam while it is running gives undefined behavior hence 'commit' makes sure that the changes are not written to the hardware while it is running. Hence it waits for the current sequence to finish before writing to the hardware sequence.
- **Enable**(bo): It puts the soft sequence in the ENABLED state. In enabled state, a loaded sequence is armed and waits for the trigger. If is already in ENABLED state the record does nothing.
- **Disable**(bo): In DISABLED state the armed sequence is disarmed, so even if the sequencer receives the trigger the sequence is does not run again.
- **Pause**(bo): This stops the currently running sequence(if any) and then disarms it. Pause leaves the sequence in DISABLED state. When the sequence starts running again(Arm + Trigger), it continues the from where it was stopped.
- **Abort**(bo): This causes the currently running sequence(if any) to stop and then disarmed. Abort leaves the sequence in DISABLED state. After disarming it also resets the timestamp and eventCode registers. So when the sequence starts running again(Arm + Trigger), it continues the from the start.

Caveats for sequencer

- In the Event Code and Timestamp arrays provided by user are of different lengths then the length of the sequence would be the length of the smaller of the two arrays. The remaining extra elements of the longer array would be ignored.
- Driver by defaults puts the 'End of Sequence (0x7f)' event code at the end of the sequence and it will be sent 'evgEndOfSeqBuf' event clock tick after the last event in the sequence has been sent out. Which currently defaults to five event clock ticks. If user provides 0x7f with a timestamp then that would be used instead of the default one.
- If a soft sequence is uncommitted and running then when the IOC restarts the sequence would be in uncommitted state but wont be running i.e. last committed sequence is lost.

PyQt script. (Front end for Event Code and Timestamp arrays)

- You need to install PyQt4 to run this python script. Debian package is pyqt4-dev-tools.
- You can have timestamp as 'zero' for the first event code in the sequencer. So this will allow the first event code in the sequencer to be sent out immediately after sequencer receives the trigger. But adding 'zero' as timestamp anywhere else(other than for first event code) is an error and the sequence would be truncated as soon as a zero is encountered. e.g. timestamp array: 0x20, 0x30, 0, 0x40 would be truncated to 0x20, 0x30. (Just first two elements before zero.)

Acknowledgment

Thanks for all the help and support

- Michael Davidsaver, NSLS2, BNL.
- Eric Bjorklund, LANSCE, LANL.

2.16.4 EVG Device Support Reference

The following sections list the properties for all sub-units with functional descriptions.

Global Properties in this section apply to the EVG as a whole. See: evgMrmApp/Db/evgMrm.db

Enable Implemented for: bo DTYP: EVG Master enable for the EVG. If not set then very little will happen.

Event Clock See: evgMrmApp/Db/evgEvtClk.db

Clock Source Implemented for: bo DTYP: EVG Evt Clk Source Selects either the internal fractional synthesizer, or the external clock input (RFIN).

RF Divider Implemented for: longout DTYP: EVG RF Divider Sets the programmable divider which converts the external clock input (RFIN) to the Event clock frequency. Valid only for the external clock source.

RF Ref. Frequency Implemented for: ao DTYP: EVG Clk RFref When using the external clock input (RFIN). This property must be set to the frequency being given to RFIN. The EVG is not able to measure this, so it must be provided by the user. This number is used to convert Event clock ticks into real time (nanoseconds). An incorrect setting will result in incorrect delays and periods being calculated.

Synth. Frequency Implemented for: ao DTYP: EVG Clk When using the internal fractional synthesizer this property sets the Event clock frequency used. If the fractional synthesizer is not able to produce the requested frequency then it will attempt to find a frequency as close as possible.

Event Clock Frequency Implemented for: ai DTYP: EVG Clk Current Event clock frequency. When using the external clock this is a readback of the value of the RF Ref.

Frequency property.

When using the internal fractional synthesizer this is a readback of the actual output frequency, which may be different then what was requested with the Synth. Frequency.

9.3 AC Line Sync. The AC Line Sync unit is a trigger source which relates to the phase of its input. The trigger is given on the closest tick of the syncroniszation source. Typically this is used to provide a trigger from the facility power mains. See: evgMrmApp/Db/evgAcTrig.db

9.3.1 Divider Implemented for: longout DTYP: EVG AC Divider The mains voltage frequency can be divided by an eight bit programmable divider.

9.3.2 Phase Implemented for: ao DTYP: EVG AC Phase The output of the divider may be delayed by 0 to 25.5 ms by a phase shifter in 0.1ms steps to adjust the triggering position relative to mains voltage phase.

9.3.3 Bypass Implemented for: bo DTYP: EVG AC Bypass Bypass the AC divider and phase shifter circuitry. Equivalent to setting divide by 1 and phase 0.

9.3.4 Synchronization Source Implemented for: bo DTYP: EVG AC Sync The AC Trigger could be synchronized either with event clock or the output of multiplexed counter 7.

9.4 Software Event See: evgMrmApp/Db/evgSoftEvt.db Implemented for: longout DTYP: EVG Soft Evt When processed immediately queues the code stored in the value field to be sent over the event link. Software events have the lowest priority in the queue and will be sent in the next otherwise empty frame. Only one software event can be queued. If there are more then one records then each will wait until it can queue its code, and will continue to wait until the code is sent.

32

9.5 Event Triggers See: evgMrmApp/Db/evgSoftEvt.db

9.6 Inputs 9.7 Outputs 9.8 DBus Bits 9.9 Multiplexed Counters 9.10 Software Sequences 9.11 Data Buffer Tx See section 8.10.2 on page 29.

2.17 EPICS device driver for MRF Event Receiver (EVR)

Event Receiver is a component of the MRF Timing system. Introduction to the MRF timing system can be found *here*.

Further details on the hardware implementation can be found on the same site, in particular about the *Event Receiver*.

More information on the Micro Research hardware can be found on their website http://www.mrf.fi/.

2.17.1 What is Available?

The software discussed below can be found on the 'EPICS modules' site on Github, in the 'mrfioc2' repository.

https://github.com/epics-modules/mrfioc2

2.17.2 Prerequisites

Build system required modules

EPICS Base (>= 3.14.10) : EPICS Core
https://epics-controls.org/resources-and-support/base/

MSI Macro expansion tool (required only for EPICS Base <3.15.0)
http://www.aps.anl.gov/epics/extensions/msi/index.php

devLib2 (>= 2.9): PCI/VME64x Hardware access library
https://github.com/epics-modules/devlib2/

Build system optional modules. Not required, but highly recommended.

autosave: Automatic save and restore on boot
https://github.com/epics-modules/autosave

iocstats : Runtime IOC statistics (CPU load, . . .)
https://github.com/epics-modules/iocStats
https://www.slac.stanford.edu/grp/ssrl/spear/epics/site/devIocStats/

Target operating system requirements

RTEMS: >= 4.9.x

vxWorks: >=6.7

Linux kernel: >= 3.2.1 (earlier versions may work)

Source

VCS Checkout

```
$ git clone https://github.com/epics-modules/mrfioc2.git
```

Edit 'configure/CONFIG_SITE' and 'configure/RELEASE' then run make.

Supported Hardware

The following devices are supported.

Name	#FP[1]	# FP UNIV[2]	#FP Inputs[3]	RTM[4]
VME-EVR-230[5]	4	4	2	Yes
VME-EVR-230RF	7[6]	2	2	Yes
PMC-EVR-230	3	0	1	No
CPCI-EVR-230	0	4	2	Yes[7]
cPCI-EVRTG-300	2[8]	2	1[9]	No
cPCI-EVR-300	0	12	2	No
PCIe-EVR-300DC	0	0	0	Yes[13]
mTCA-EVR-300[10]	4	4/0	2	Yes
VME-EVR-300	4[12]	4	2	Yes

[1] Front panel outputs (TTL)
[2] Front panel universal output sockets
[3] Front panel inputs
[4] Supports Rear Transition Module
[5] This device has not been tested
[6] Outputs 4,5,6 are CML
[7] Supports PCI side-by-side module
[8] GTX outputs
[9] Special GTX interlock
[13] Extension "box", connected with a cable.
[10] Two hardware flavors exist, one with 2x UNIV I/O sockets, the other with an IFB-300 connector,
[12] One Universal I/O slot, 2 x CML (GTX) outputs.

2.17.3 Overview of the Driver

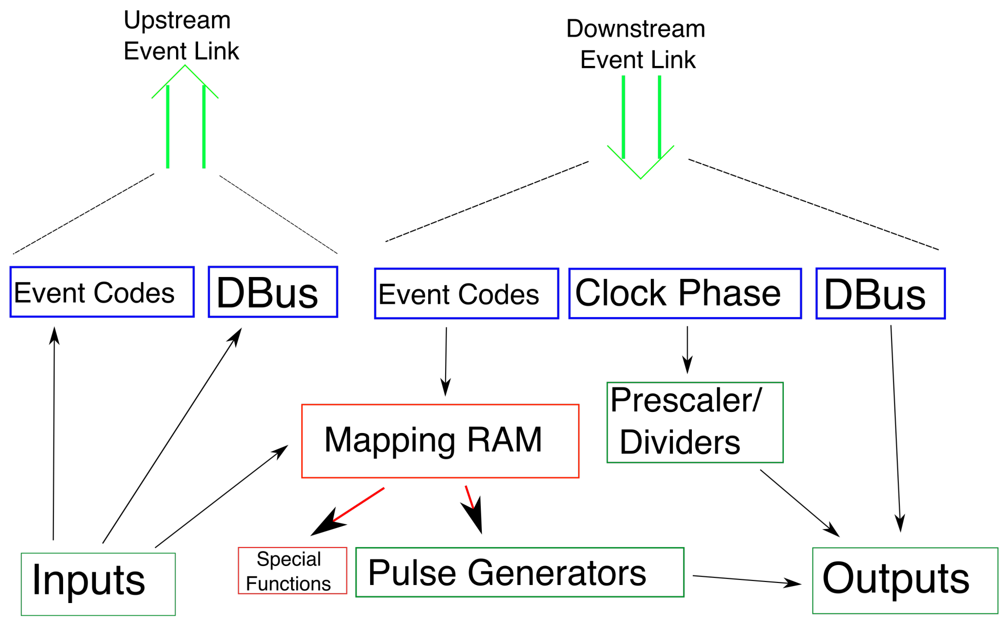
The purpose of this document is to act as a guide for using the 'mrfioc2' EPICS support module for the Micro Research Finland (MRF) timing system[11]. It describes software for using the Event Receiver (EVR).

This document is an overview of the driver and its capabilities. It is not a fully up-to-date document with all details, even if the contents are thought to be accurate. The intent of this document is to introduce the concepts and help understanding how to use the driver. For studying the implementation, the important locations in the source tree relating to the EVR are included.

The source repository contains a full API documentation, produced with Doxygen.

2.17.4 Receiver Functions

Internally an EVR can be thought of as a number of logical sub-units that connect the upstream and downstream event links to the local inputs and outputs. These sub-units include: the Event Mapping Ram, Pulse Generators, Prescalers (clock dividers), and the logical controls for the physical inputs and outputs.



Logical connections inside an EVR

Pulse Generators

Each *pulse generator* has a an associated Delay, Width, Polarity (active low/high), and (sometimes) a Prescaler (clock divider). When triggered by the Mapping Ram it will wait for the Delay time in its inactive state. Then it will transition to its active state, wait for the Width time before transitioning back to its inactive state.

Resolution of the delay and width is determined by the prescaler. A setting of 1 gives the best resolution.

In addition, the Mapping Ram can force a Pulse Generator into either state (Active/Inactive).

[11] List of supported hardware given in section [Supported Hardware]](#supported-hardware).

Event Mapping Ram

The *Event Mapping Ram* is a table used to define the actions to be taken by an EVR when it receives a particular event code number. The mapping it defines is a many-to-many relation; one event can cause several actions, and one action can be caused by several events.

The actions which can be taken can be grouped into two categories: Special actions, and Pulse Generator actions. Special actions include those related to timestamp distribution, and the system heartbeat tick (see *Special Function Mappings* for a complete list). Each Pulse Generator has three mappable actions: Set (force active), Reset (force inactive), and Trigger (start delay program). Most applications will use Trigger mappings.

Prescalers (Clock Divider)

Prescaler sub-units take the EVR's local oscillator and output a lower frequency clock which is phased locked to the local clock, which is in sync with the global master clock. The lower frequency must be an integer divisor of the Event clock.

To provide known phase relationships, all dividers can be synchronously reset when a mapped event code is received. This is the Reset PS action. See *Special Function Mappings*.

Outputs (TTL)

This sub-unit represents a physical output on the EVR. Each output may be connected to one source: a Distributed Bus bit, a Prescaler, or a Pulse Generator (see *Function Mapping* for a complete list).

Outputs (CML and GTX)

Current Mode Logic outputs can send a bit pattern at the bit rate of the event link bit clock (20x the Event Clock). This pattern may be specified in one of three possible ways:

- As four 20 bit sub-patterns (rising, high, falling, and low).
- As two periods (high and low). These specify a square wave with variable frequency and duty factor.
- As an arbitrary bit pattern (<= 40940 bits) which begins when the output goes [TODO: high or low?].

In the sub-pattern mode, the rising and falling edge patterns are transmitted when the output level changes, while the high and low patterns are repeated in between level changes.

The GTX outputs that are found only on selected models like VME-EVR-300, function similarly to the CML outputs, however at twice the frequency. Thus for these devices patterns are 40 bits.

Inputs

An EVR's local TTL input can cause several actions when triggered. It may be directly connected to one of the upstream *Distributed Bus* bits, it may cause an event to be sent on the upstream links, or applied to the local Mapping Ram.

The rising edge of a local input can be timestamped.

Global Timestamp Reception

Each EVR receives synchronous time broadcasts from an EVG. Software may query the current time at any point. The arrival time of event codes can be saved as well. This can be accomplished with the 'event' record device support.

Each EVR may be configured with a different method of incrementing the timestamp counter. See section *Timestamp Sources*.

In addition to being slaved to an EVG, those EVR models/firmware which provide a Software Event transmission function can send timestamps as well. This can be used to simulate timestamps in a standalone environment such as a test lab. see the TimeSrc property in *EVG Functions (DC firmware)*.

TimeSrc=0

: The default, which disables EVR timestamp generation.

TimeSrc=1

: In External mode the EVR will send a timestamp when event 125 is received. Reception of 125 can be either from an input, or for DC EVRs the sequencer.

TimeSrc=2

: In Sys Clock mode, the EVR will generate a software 125 event based on the system clock. This is the simplest standalone mode.

Data Buffer Tx/Rx

A recipient can register callback functions for each Protocol ID. It will then be shown the body of every buffer arriving with this ID.

A default recipient is provided which stores data in a waveform record.

2.17.5 IOC Deployment

This section outlines a general strategy for adding an EVR to an IOC. First general information is presented, followed by a section describing the extra steps needed to use mrfioc2 under Linux.

An example IOC shell script is included as "iocBoot/iocevrmrm/st.cmd".

Device names

All EVGs and EVRs in an IOC are identified by an unique name. This is first given in the IOC shell functions described below, and repeated in the INP or OUT field of all database records which reference it. Both EVGs, and EVRs share the same namespace. This restriction is needed since some code is shared between these two devices.

2.17.6 VME64x Device Configuration

The VME bus based EVRs and EVGs are configured using one of the following IOC shell functions.

```
# Receiver
mrmEvrSetupVME("anEVR", 3, 0x30000000, 4, 0x28)
```

In this example EVR "anEVR" is defined to be the VME card in slot 3. It is given the A32 base address of 0x30000000 and configured to interrupt on level 4 with vector 0x28.

2.17.7 PCI Device Configuration

PCI bus cards are identified with the mrmEvrSetupPCI() IOC shell function.

Since PCI devices are automatically configured only the geographic address (bus:device.function) needs to be provided. This information can usually be found at boot time (RTEMS) or in /proc/bus/pci/devices (Linux).

The IOC shell function devPCIShow() is also provided to list PCI devices in the system.

```
# Receiver
mrmEvrSetupPCI("PMC", "1:2.0")
```

This example defines EVR "PMC" to be bus 1 device 2 function 0.

Support for using mTCA slot number is available on some targets (Linux only as of devlib2 2.9). This does any automatic lookup of PCI address from slot number. Be aware that PCIe "slot" numbers, while stable across reboots, may change with hardware configuration, firmware, or OS upgrades.

```
mrmEvrSetupPCI("PMC", "slot=5")
```

PCI Setup in Linux

In order to use PCI EVRs in the Linux operating system a small kernel driver must be built and loaded. The source for this driver is found in 'mrmShared/linux/'. This directory contains a Makefile for use by the Linux kernel build system (not EPICS).

To build the driver you must have access to a configured copy of the kernel source used to build the target system's kernel. If the build and target systems use the same kernel, then the location will likely be '/lib/modules/'uname -r'/build'. In case of a cross-built kernel the location will be elsewhere.

To build the module for use on the host system:

```
$ make -C /location/of/mrmShared/linux \
KERNELDIR=/lib/modules/`uname -r`/build modules_install
$ sudo depmod -a
$ sudo modprobe mrf
```

Building for a cross-target might look like:

```
$ make -C /location/of/mrmShared/linux \
KERNELDIR=/location/of/kernel/src \
ARCH=arm CROSS_COMPILE=/usr/local/bin/arm- \
INSTALL_MOD_PATH=/location/of/target/root \
modules_install
```

Once the module is installed on the running target the special device file associated with each EVR must be created. If your target system is running UDEV this will happen automatically. See mrmShared/linux/README for example UDEV config. If UDEV is not present, then you must do the following.

```
# grep mrf /proc/devices
254 mrf
# mknod -m 666 /dev/uio0 c 254 0
```

If may be necessary to change the file permission to allow the IOC process to open it. UDEV users may find one of the following commands useful for constructing a rules file.

```
# udevinfo -a -p $(udevinfo -q path -n /dev/uio0 )

# udevadm info -a -p $(udevadm info -q path -n /dev/uio0 )
```

Each additional device adds one to the number (uio1, uio2, ...).

Once the device file exists with the correct permissions the IOC will be able to location it based on the bus:device.function given an to mrmEvrSetupPCI().

2.17.8 Example Databases

The mrfioc2 module includes example database templates for all supported devices (see *Supported Hardware*). While each is fully functional, it is expected that most sites will make modifications. It is suggested that the original be left unchanged and a copy be made with the institute name and other information as a suffix. (evr-pmc-230.substitutions becomes evr-pmc-230-nsls2.substitutions).

The authors would like to encourage users to send their customized databases back so that they may be included as examples in future releases of mrfioc2.

The templates consist of a substitutions file for each model (MTCA, PMC, cPCI, VME-RF). These templates instanciate the correct number of records for the inputs/outputs found on each device. It also includes entries for event mappings and database events which will be frequent targets for customization.

Each substitutions file will be expanded during the build process with the MSI utility to create a database file with two undefined macros (P and C). 'SYS' and 'D' define a common prefix shared by all PVs and must be unique in the system. 'EVR' is a card name also given as the first argument of one of the mrmEvrSetup*() IOC shell functions (unique in each IOC).

Thus an IOC with two identical VME cards could use a configuration like:

```
mrmEvrSetupVME("evr1",5,0x20000000,3,0x26)
mrmEvrSetupVME("evr2",6,0x21000000,3,0x28)
dbLoadRecords("evr-vmerf-230.db", "SYS=test, D=evr:a, EVR=evr1")
dbLoadRecords("evr-vmerf-230.db", "SYS=test, D=evr:b, EVR=evr2")
```

autosave

All example database files include "info()" entries to generate autosave request files. The example IOC shell script "iocBoot/iocevrmrm/st.cmd" includes the following to configure autosave.

```
save_restoreDebug(2)
dbLoadRecords("db/save_restoreStatus.db", "P=mrftest:")
save_restoreSet_status_prefix("mrftest:")

set_savefile_path("${mnt}/as","/save")
set_requestfile_path("${mnt}/as","/req")
```

This enables some extra debug information which is useful for testing, and loads the autosave on-line status database. It also sets the locations where .sav and .req files will be searched for.

```
set_pass0_restoreFile("mrf_settings.sav")
set_pass0_restoreFile("mrf_values.sav")
set_pass1_restoreFile("mrf_values.sav")
set_pass1_restoreFile("mrf_waveforms.sav")
```

Sets three files which will be loaded. The "values" are loaded twices as is the autosave convention.

```
iocInit()

makeAutosaveFileFromDbInfo("as/req/mrf_settings.req", "autosaveFields_pass0")
makeAutosaveFileFromDbInfo("as/req/mrf_values.req", "autosaveFields")
makeAutosaveFileFromDbInfo("as/req/mrf_waveforms.req", "autosaveFields_pass1")
```

After the IOC has started the request files are generated. This is where the "info()" entries in the database files are used.

```
create_monitor_set("mrf_settings.req", 5 , "")
create_monitor_set("mrf_values.req", 5 , "")
create_monitor_set("mrf_waveforms.req", 30 , "")
```

Finally the request files are re-read and monitor sets are created.

2.17.9 Testing Procedures

This section presents several step by step procedures which may be useful when testing the function of hardware and software.

In the "documentation/demo/" directory several IOC shell script files with the commands given in this section as well as other examples.

EVG and EVR Checkout

This procedure requires both a generator, receiver, and a fiber jumper cable to connect them.

It is assumed that no cables are connected to the front panel of either EVG or EVR. The example "iocBoot/iocevrmrm/st.cmd" script is used with `SYS=TST` and `D=evr` for the receiver and `D=evg` for the generator. Verify this with the following commands at the IOC shell.

```
>dbgrep("*Link:Clk-SP")
TST{evr}Link:Clk-SP
>dbgrep("*FracSynFreq-SP")
TST{evg-EvtClk}FracSynFreq-SP
```

The following examples use the IOC shell commands `dbpr()` and `dbpf()`. Remote use of `caput` and `caget` is also possible.

```
>dbpf("TST{evg-EvtClk}Source-Sel","FracSyn")
>dbpf("TST{evg-EvtClk}FracSynFreq-SP","125.0")
>dbpf("TST{evr}Link:Clk-SP","125.0")
>dbpf("TST{evr}Ena-Sel","Enabled")
>dbpr("TST{evr}Link-Sts")
...
... VAL: 0
```

This sets the event link speed on both the EVR and EVG. The EVG is commanded to use its internal synthesizer instead of an external clock.

Now use the fiber jumper cable to connect the TX port of the generator to the RX port of the receiver. (The Tx port will have a faint red light coming from it).

Once connected the red link fail LED should go off and the link status PV should read OK (1).

```
>dbpr("TST{evr}Link-Sts")
...
... VAL: 1
```

At this point the receivier has locked to the generator signal, but no data is being sent. This includes the heartbeat event. Thus the heartbeat timeout counter should be increasing.

```
>dbpr("TST{evr}Cnt:LinkTimo-I")
...
... VAL: 45
>dbpr("TST{evr}Cnt:LinkTimo-I")
...
... VAL: 47
```

Now we will set up the generator to send a periodic event code.

```
>dbpf("TST{evg-Mxc:0}Prescaler-SP", "125000000")
>dbpr("TST{evg-Mxc:0}Frequency-RB",1)
...
EGU: Hz ...
... VAL: 1
>dbpf("TST{evg-TrigEvt:0}EvtCode-SP", "122")
>dbpf("TST{evg-TrigEvt:0}TrigSrc-Sel", "Mxc0")
>dbpf("TST{evg-TrigEvt:1}EvtCode-SP", "125")
>dbpf("TST{evg-TrigEvt:1}TrigSrc-Sel", "Mxc0")
>dbpf("TST{evr}Evt:Blink0-SP", "125")
```

This configures multiplexed counter 0 (Mxc #0) to trigger on the event clock frequency divided by 125000000. In this case this gives 1Hz. Trigger event #0 is then configured to send event code 122, and trigger event #1 to send code 125, when Mxc #0 triggers.

At this point both the EVG's amber EVENT OUT led and the EVR's EVENT IN led should flash at 1Hz.

For diagnostics the EVR's Blink0 mapping is configured to blink the EVR's EVENT OUT led when event code 125 is received. Setting to 0 will cause it to stop blinking.

Event code 122 is the heartbeat reset event. Since it is being sent the link timeout counter should no longer be increasing.

```
>dbpr("TST{evr}Cnt:LinkTimo-I")
...
... VAL: 120
>dbpr("TST{evr}Cnt:LinkTimo-I")
...
... VAL: 120
```

At this point, if the system is given an NTP server the EVG will get a correct (but unsynchronized) time and messages similar to the following will be printed.

```
Starting timestamping
epicsTime: Wed Jun 01 2011 17:54:53.000000000
TS becomes valid after fault 4de6b533
```

The first two lines come from the EVG and indicate that it is sending a timestamp. The third line comes from the EVR and indicates that it is receiving a correct timestamp.

The counter for the 1Hz event should now be increasing.

```
>dbpr("TST{evr}1hzCnt-I")
... VAL: 5
>dbpr("TST{evr}1hzCnt-I")
... VAL: 6
```

Timestamp Test

An external 1Hz pulse generator is required for this test. It should be connected to front panel input 0 on the EVG. This is LEMO connector expecting a TTL signal.

```
>dbpr("TST{evr}Link-Sts")
...
... VAL: 1
```

If the event link status is not OK then perform setup as described in the previous test.

Check the current time source status

```
>generalTimeReport(2)
Backwards time errors prevented 0 times.

Current Time Providers:     "EVR", priority = 50
        Current Time not available
    "NTP", priority = 100
        Current Time is 2011-06-02 10:23:26.058125.
    "OS Clock", priority = 999
        Current Time is 2011-06-02 10:23:26.057101.

Event Time Providers:
    "EVR", priority = 50
```

This shows that the NTP time source is functioning. This is required for this test.

```
>dbpf("TST{evg-TrigEvt:1}EvtCode-SP", "125")
>dbpf("TST{evg-TrigEvt:1}TrigSrc-Sel", "Front0")
>dbpf("TST{evr}Evt:Blink0-SP", "125")
```

Sends event code 125 on the rising edge for front panel input 0. For diagnostics sets the blink mapping. If the led is not blinking then check the 1Hz pulse generator.

```
dbpr("TST{evr}Time:Valid-Sts")
...
... VAL: 1
```

Indicates that the EVR has received a valid time

```
>generalTimeReport(2)
Backwards time errors prevented 0 times.

Current Time Providers:     "EVR", priority = 50
        Current Time is 2011-06-02 10:26:50.683808.
    "NTP", priority = 100
        Current Time is 2011-06-02 10:26:50.681220.
```

(continues on next page)

(continued from previous page)

```
    "OS Clock", priority = 999
        Current Time is 2011-06-02 10:26:50.683854.

Event Time Providers:
    "EVR", priority = 50
```

Shows that a valid time is now being reported.

```
$ camonitor TST{evr:3}Time-I
TST{evr:3}Time-I               2011-06-02 10:32:11.999993 Thu, 02 Jun 2011 10:32:12 -0400
TST{evr:3}Time-I               2011-06-02 10:32:12.999993 Thu, 02 Jun 2011 10:32:13 -0400
TST{evr:3}Time-I               2011-06-02 10:32:13.999993 Thu, 02 Jun 2011 10:32:14 -0400
TST{evr:3}Time-I               2011-06-02 10:32:14.999993 Thu, 02 Jun 2011 10:32:15 -0400
```

The timestamp indicator record takes its record timestamp from the arrival of the 125 event code. As can be seen, this time is stored immediately before the sub-seconds is zeroed. This can be verified by switching this.

```
$ caget TST{evr:3}Time-I.TSE
TST{evr:3}Time-I.TSE           125
$ caput TST{evr:3}Time-I.TSE 0
Old : TST{evr:3}Time-I.TSE           125
New : TST{evr:3}Time-I.TSE           0
$ camonitor TST{evr:3}Time-I
TST{evr:3}Time-I               2011-06-02 10:35:31.005655 Thu, 02 Jun 2011 10:35:31 -0400
TST{evr:3}Time-I               2011-06-02 10:35:32.005655 Thu, 02 Jun 2011 10:35:32 -0400
TST{evr:3}Time-I               2011-06-02 10:35:33.005655 Thu, 02 Jun 2011 10:35:33 -0400
TST{evr:3}Time-I               2011-06-02 10:35:34.005655 Thu, 02 Jun 2011 10:35:34 -0400
```

Now a time latched by software when this record is processed. For real-time system this time should be stable.

2.17.10 Firmware Update

300-series Devices

- PCIe-EVR-300DC
- mTCA-EVR-300
- mTCA-EVM-300

These devices support upgrade of firmware through PCIe register access. As such, a failed upgrade will result in an unusable device.

To test if a card may be upgrade with this mechanism, run *flashinfo* and *flashread* command. The following shows a device which can be upgraded.

```
epics> mrmEvrSetupPCI("EVR1", "03:00.0")
  ...
epics> flashinfo("EVR1:FLASH")
Vendor: 20 (Micron)
Device: ba
ID: 18
Capacity: 0x1000000
```

(continues on next page)

(continued from previous page)

```
Sector: 0x10000
Page: 0x100
S/N:  23 51 61 31 16 00 14 00 31 26 05 15 ee 45
epics> flashread("EVR1:FLASH", 0, 64)
00090ff0 0ff00ff0 0ff00000 0161001f
70636965 65767233 30306463 3b557365
7249443d 30584646 46464646 46460062
000c376b 37307466 62673637 36006300
epics>
```

Before upgrading, it is suggested to backup the existing firmware. If the size of the existing firmware is known, then this size can be used. Otherwise, use the capacity reported by *flashinfo*. All Xilinx bit files for a particular device typically have the same size.

In this example of a PCIe-EVR-300DC with the 207.0 firmware, the exact size is 3011417 bytes, which we arbitrarily round up to 3MB.

```
epics> flashread("EVR1:FLASH", 0, 0x300000, "PCIe-EVR-300DC.207.0.backup.bit")
| 3080192
  ...
```

Now write the new firmware file.

```
epics> flashwrite("EVR1:FLASH", 0, "PCIe-EVR-300DC.207.6.bit")
```

If the update process is interrupted, **do not power cycle**! Re-run the update process to completion.

After the write completes successfully, power cycle the card to load the new bit file.

VME EVRs and EVGs

Update for VME cards is accomplished through the ethernet jack label “10 BaseT”. The procedure covered in the MRF manual.

cPCI-EVRTG-300

Undocumented.

PMC-EVR-230

Firmware update for the PMC module EVR is accomplished through a JTAG interface as with the cPCI-EVRTG-300. For reasons of physical space the JTAG wires are not brought to a connector, but connected to 4 I/O pins of the PLX 9030 PCI bridge chip. In order to control these pins and update the firmware some additional software is needed. Software update may be performed by using either the parallel port support or through JTAG pins. The running Kernel must be built with the CONFIG_GENERIC_GPIO and CONFIG_GPIO_SYSFS options if the latter approach is to be used.

If the parallel port support is available, a message is printed to the kernel log when the Linux kernel module provided with mrfioc2 (mrmShared/linux) is loaded.

```
Emulating cable: Minimal
```

The kernel module also exposes the 4 I/O pins via the Linux GPIO API. The 4 pins are numbered in the order: TCK, TMS, TDO, and TDI. The number of the first pin is printed to the kernel log when the MRF kernel module is loaded.

```
GPIO setup ok, JTAG available at bit 252
```

In this example the 4 pins would be TCK=252, TMS=253, TDO=254, and TDI=255.

Creating an SVF file from a BIT file

The firmware file will likely be supplied in one of two formats having the extensions .bit or .svf. If the provided file has the extension .svf then proceed to section Programming with UrJTAG.

To convert a .bit file to a .svf file it is necessary to get the iMPACT programming tool from Xilinx. The easiest way to do this is with the "Lab Tools" bundle.

http://www.xilinx.com/support/download/index.htm

The following instructions are for iMPACT version 14.2.

1. Install and run the iMPACT program.
2. When prompted to create a project click cancel
3. On the left side of the main window is a pane titled "iMPACT FLows". Double click on "Create PROM File"
4. Select "Xilinx Flash/PROM" and click the first green arrow.
5. Select "Platform Flash" and "xcf08p" and click "Add Storage Device" then the second green arrow.
6. Select an output file name and path. Ensure that the file format is MCS. Click OK
7. Several small dialogs will appear. When prompted to "Add device" select the .bit file provided by MRF.
8. When prompted to add another device click No.
9. On the left side of the main window is a pane titled "iMPACT Processes". Double click on "Generate File".
10. The .mcs file should now be written.
11. Exit and restart iMPACT.

See http://www.xilinx.com/support/documentation/user_guides/ug161.pdf starting on page 67 for more detailed instructions.

1. Create a new iMPACT project. Select "Prepare a Boundary-Scan File" and the SVF format.
2. When prompted, select a name for the resulting .svf file
3. When prompted to "Assign New Configuration File" select the .mcs file just created.
4. When prompted to select a PROM type choose "xcf08p"
5. An icon representing the PROM should now appear as the only entry in the JTAG chain.
6. Right click on this icon and select Program.
7. In the dialog which appears check Verify and click OK.
8. The .svf file should now be written.
9. Exit iMPACT

Programming with UrJTAG

http://urjtag.org/

As of August 2012 support to the Linux GPIO "cable" was not included in any UrJTAG release. It is necessary to checkout and build the development version (commit id b6945fc65 from 9 Aug. 2012 works). This requires the Git version control tool. To build and use UrJTAG on target system, there may be a need to install certain packages in the system.

```
$ sudo apt-get install pciutils make autoconf autopoint libtool
pkg-config bison libusb-1.0-0-dev libusb-dev flex python-dev
```

With all necessary tools available, configure and build UrJTAG.

```
$ git clone git://urjtag.git.sourceforge.net/gitroot/urjtag/urjtag
$ cd urjtag/urjtags
$ ./autogen.sh --disable-nls --disable-python --prefix=$PWD/usr
$ make && make install
```

Firmware update may be performed using the parallel port support if available, e.g. when loading the kernel driver:

```
$ sudo modprobe uio
$ sudo modprobe parport
$ sudo insmod mrf.ko
$ dmesg
...
[   69.046938] mrf-pci 0000:08:0d.0: MRF Setup complete
[   69.047007] mrf-pci 0000:09:0e.0: PCI IRQ 72 -> rerouted to legacy IRQ 16
[   69.047589] mrf-pci 0000:09:0e.0: GPIOC 00249412
[   69.047626] mrf-pci 0000:09:0e.0: GPIO setup ok, JTAG available at bit 252
[   69.144196] mrf-pci 0000:09:0e.0: Emulating cable: Minimal
[   69.144239] mrf-pci 0000:09:0e.0: MRF Setup complete
...
```

The "Emulating cable: Minimal" message indicates that Minimal JTAG cable type can be used to communicate with a device. A ppdev device should be available for usage with UrJTAG:

```
$ sudo modprobe ppdev
$ dmesg
...
[   69.028268] ppdev: user-space parallel port driver
...
$ ls /dev | grep parport
parport0
```

On the target system run UrJTAG as root:

```
# ./usr/bin/jtag
jtag> cable Minimal ppdev /dev/parport0
Initializing ppdev port /dev/parport0
jtag> detect
IR length: 26
Chain length: 2
Device Id: 00100001001000111110000010010011 (0x2123E093)
  Manufacturer: Xilinx (0x093)
```

(continues on next page)

(continued from previous page)

```
  Part(0):      xc2vp4 (0x123E)
  Stepping:     2
  Filename:     /epics/urjtag/share/urjtag/xilinx/xc2vp4/xc2vp4
Device Id: 11100101000001010111000010010011 (0xE5057093)
  Manufacturer: Xilinx (0x093)
  Part(1):      xcf08p (0x5057)
  Stepping:     14
  Filename:     /epics/urjtag/share/urjtag/xilinx/xcf08p/xcf08p
jtag> part 1
jtag> svf /location/of/pmc-prom.svf stop progress
```

Alternatively, a GPIO cable may be utilized if the kernel was built with options required (CONFIG_GENERIC_GPIO and CONFIG_GPIO_SYSFS), on the target system run UrJTAG as root (or a user which can export and use GPIO pins).

```
# ./usr/bin/jtag
jtag> cable gpio tck=252 tms=253 tdo=254 tdi=255
jtag> detect
IR length: 26
Chain length: 2
Device Id: 00100001001000111110000010010011 (0x2123E093)
  Manufacturer: Xilinx (0x093)
  Part(0):      xc2vp4 (0x123E)
  Stepping:     2
  Filename:     /epics/urjtag/share/urjtag/xilinx/xc2vp4/xc2vp4
Device Id: 11100101000001010111000010010011 (0xE5057093)
  Manufacturer: Xilinx (0x093)
  Part(1):      xcf08p (0x5057)
  Stepping:     14
  Filename:     /epics/urjtag/share/urjtag/xilinx/xcf08p/xcf08p
jtag> part 1
jtag> svf /location/of/pmc-prom.svf stop progress
```

Note that the device IDs may not be correctly recognized. This will not effect the programming process.

If no errors are printed then the update process was successful. The new firmware will not be loaded until the PMC module is reset (power cycle system).

2.17.11 NTPD Time Source

It is possible to use an EVR as a time source for the system NTP daemon on Linux. This is implemented using the shared memory clock driver (#28).

http://www.eecis.udel.edu/~mills/ntp/html/drivers/driver28.html

An IOC is configured to write data to a shared memory segment by adding a line to its start script.

```
time2ntp("evrname", N)
```

Here "evrname" is the same name given when configuring the EVR (see [Device Names](#Device names). The memory segment ID number N must be between 0 and 4 inclusive. The NTP daemon enforces that segments 0 and 1 require root permissions to use. Segments 2, 3, and 4 can be accessed by an unprivileged user.

It is suggested to use an unprivileged segment to avoid running the IOC as root. However, this would allow any user on the system to effectively control NTPD. So it is not recommended for systems with untrusted users.

The NTP daemon is configured from the file */etc/ntp.conf*. On Debian Linux systems using DHCP it will be necessary to modify */etc/dhcp/dhclient-exit-hooks.d/ntp* instead.

```
server 127.127.28.N minpoll 1 maxpoll 2 prefer
fudge 127.127.28.N refid EVR
```

This will configure NTPD to read time from segment N. Here N must match what was specified for *time2ntp()*.

When functioning correctly NTPD status should look like:

```
$ ntpq -p
remote           refid      st t when poll reach   delay   offset  jitter
==========================================================================
+time.cs.nsls2.l .GPS.       1 u   29   64  377    2.684   -0.001   0.089
*SHM(3)          .EVR.       0 l    7    8  377    0.000    0.000   0.001
```

The shared memory interface can only be used to provide time with microsecond precision. So this measurement, taken from a production NSLS2 server, showing a jitter of $\pm1$ microsecond is the best which can be obtained.

If the propagation time from the time source to the EVR is known, then the offset can be given by adding "time1 0.XXX" to the 'fudge' line in *ntp.conf*.

2.17.12 Buffered Timestamp Capture

Some applications are interested in the precise reception timestamp of an asynchronous event code. For example, an External event code from an EVR Input. Further, if this Input/event code occurs at a high rate, it is preferable for software to process reception times in batches.

The motivating use case for this feature was monitoring of a rotational encoder which produces a pulse on crossing a particular angle. The times of this crossing are needed to calculate frequency and phase. Further, crossing occur at ~1KHz.

Buffers are setup by loading instances of the `db/mrmevrtsbuf.db` database. Many buffers may be loaded. While un-useful it is possible to associate multiple buffers with the same event code.

```
dbLoadRecords("db/evr-pcie-300dc.db","SYS=TST, D=evr:1, EVR=EVR,\
               FEVT=125")
dbLoadRecords("db/mrmevrtsbuf.db", "SYS=TST, D=evr:1-ts:1, EVR=EVR,\
               CODE=20, TRIG=10, FLUSH=TimesRelFlush")
```

In this example, the (optional) CODE and TRIG macros name two event codes. CODE=20 is the event for which the reception time will be captured. The (also optional) TRIG=10 is an event for which reception will cause the internal buffer of timestamps to be flushed to a waveform record. Alternately, flushing can be triggered by another record.

The CODE and TRIG macros are setting the default values of fields which may be changed at runtime.

Each waveform record which present timestamps does so in a format determined by the FLUSH macro.

TimesRelFlush

: Elements are times in nanoseconds relative to the flushing action (either flush event code, or manual flush). The time of the flushing action is stored as the record timestamp. Element values are always negative. This is the default if FLUSH is not set.

TimesRelFirst

: Elements are times in nanoseconds relative to the time of the first event received after a previous flush. The time of the first event is stored in the record timestamp. Element values are always positive, and the first element value is always zero.

2.17.13 Implementation Details

Details of some parts of the driver which may be useful in understanding (and trouble shooting) the behavior of the driver.

Event code FIFO Buffer

Each EVR implements a hardware First In First Out buffer for event codes. When certain "interesting" event code numbers are received the code and arrival time are placed in this buffer. Two interrupt condition are generated by the FIFO: not empty, and full. The first is asserted when the first event added, and cleared when the last event is removed. The second occurs when last free entry in the buffer is consumed. Further event occurrences are lost.

When the not empty interrupt occurs the fifo drain task (named EVRFIFO in epicsThreadShowAll()) is woken up by a message queue. This task runs at scan high priority (90). Once awakened it will remove at most 512 event codes from the buffer before sleeping again. The number 512 is an arbitrary number chosen to prevent the starvation of lower priority tasks if a high frequency event code is accidentally mapped into the FIFO. A minimum sleep time is enforced by the **mrmEvrFIFOPeriod** variable. This governs the maximum rate that events can be reported through the FIFO. Setting to 0 will disable it.

Each of the event codes 1-255 has an IOSCANPVT and a list of callback functions (type EVR::eventCallback) which will be invoked when the event occurs.

An invocation of an IOSCANPVT list may place an arbitrary number of CALLBACKs into the message queue of the three EPICS callback scan tasks (High, Medium, and Low). If these message queues are overflowed then CALLBACK in other drivers my be lost. The scanIoRequest() function does not report this error prior to Base 3.15.0.2.

To avoid this disastrous occurrence the EVR driver will not re-run the scan list for an event, until all actions **at all priorities** from the previous run have finished. This is implemented by placing a special sentinel CALLBACK in all three queues. An event will not be re-run until all three of the CALLBACK have run.

The FIFO servicing code can indicate two error conditions. Occurrences of these errors are recorded in the `FIFO Overflow Count` and `FIFO Over rate` counters.

The `FIFO Overflow Count` gives the number of times the hardware FIFO buffer has overflowed. This is a serious error since arbitrary event code (including the timestamping codes) will be lost.

The `FIFO Over rate` counter counts the number of times any event reoccurred before the actions of the last occurrence were finished processing. This is less serious since other event codes are not effected.

Data Buffer reception

Each EVR can receive a single data buffer. Once a data message has been received, the reception engine is disabled to allow time to download the buffer. Then the engine can be re-enabled in preparation for the next message. An interrupt is generated when the message has been fully received, and the engine disabled.

Instead of a separate thread, buffer reception is implemented as a two stage callback run by the High (first) and Medium (second) priority scan tasks. The first callback copies the buffer into memory and immediately re-enables buffer reception, it then passes the data to the second callback. This callback passes the buffer to a list of user callback functions which have registered interest in the Protocol ID found in the message header.

Timestamp validation

It is impossible to verify a time without a second trusted reference. Since such a reference is not generally available, the driver can only make some checks against corruption.

The seconds part of the timestamp should only change when the 1Hz reset event (125) is received from the EVG. Therefore a callback is attached to that event code. When a new seconds value arrives it is compared to the previous stored value. If it is exactly 1 greater then it is taken to be the new seconds value. If it is not then the EVR time is declared invalid.

When the time is invalid, it can only become valid after five sequential seconds values are received. Any out of sequence value resets the count.

2.17.14 EVR Device Support Reference

The EPICS support module for MRF devices consists of a number of supports which are generally tied to a specific logical sub-unit. Each sub-unit may be thought of as an object having a number of properties. For example, each Delay Generator has properties 'Delay' and 'Width'. These properties can be read or modified in several ways. A delay can specified as an integer number of ticks of its reference clock (hardware view), or in seconds as a floating point number (user view).

In this example the properties 'Delay' and 'Width' should be settable in exact integer as well as the more useful, but imprecise, floating point units (eg. seconds). This needs to be accomplished by two different device supports (longout, and ao). Of course it is also useful to have some confirmation that settings have been applied so read-backs are desireable (longin, ai).

Some of the device supports defined are as follows. The full list is given in **`mrfCommon/src/mrfCommon.dbd`**.

```
device(longin , INST_IO, devLIFromUINT32, "Obj Prop uint32")
device(longin , INST_IO, devLIFromUINT16, "Obj Prop uint16")
device(longin , INST_IO, devLIFromBool,   "Obj Prop bool")

device(ai , INST_IO, devAOFromDouble, "Obj Prop double")
device(ai , INST_IO, devAOFromUINT32, "Obj Prop uint32")
device(ai , INST_IO, devAOFromUINT16, "Obj Prop uint16")
```

Unless otherwise noted, all device support use **INST_IO** input/output links with the format:

```
@OBJ=$(OBJECTNAME), PROP=Property Name
```

Since the Pulser sub-unit has the property 'Delay' which supports both integer and float settings, the following database can be constructed.

```
record(ao, "$(PN)Delay-SP")
{
  field(DTYP, "Obj Prop double")
  field(OUT , "@OBJ=$(OBJ), PROP=Delay")
  field(PINI, "YES")
  field(DESC, "Pulse Generator $(PID)")
  field(FLNK, "$(PN)Delay-RB")
}
record(ai, "$(PN)Delay-RB")
{
  field(DTYP, "Obj Prop double")
  field(INP , "@OBJ=$(OBJ), PROP=Delay")
```

(continues on next page)

(continued from previous page)

```
  field(FLNK, "$(PN)Delay:Raw-RB")
}
record(longin, "$(PN)Delay:Raw-RB")
{
  field(DTYP, "Obj Prop uint32")
  field(INP , "@OBJ=$(OBJ), PROP=Delay")
}
```

This provides setting in engineering units and readbacks in both EGU and raw for the delay property.

Note: It is inadvisable to have to more then one output record pointing to the same property of the same device. However, it is allowed since there are cases where this is desireable.

Note: Documentation of individual device support may be found in the example database files.

2.17.15 Global Properties

Properties in this section apply to the EVR as a whole. Records accessing properties in this section will have DTYP set to “EVR”.

See: *evrApp/Db/evrbase.db*

Name	Record type(s)	Description
Enable	bo, bi	Master enable for the EVR.
PLL Lock Status	bi	
Link Status	bi	*Event link status*
Timestamp Valid	bi	*Validity of the timestamp*
Model	longin	*Hardware model*
Version	longin	
Sw Version		
FIFO Overflow Count	longin	
Fifo Over Rate	longin	
HB Timeout Count		
Clock	ao, ai	
Timestamp Source	longout, longin	*Selects timestamp source*
Timestamp Clock	ao, ai	
Timestamp Prescaler	longin	
Timestamp	stringin	
Event Clock TD Div	longin	
Receive Error Count	longin	

For example, the boolean property **Enable** could be written by the following record:

```
record(bo, "$(P)ena " ) {
  field ( DTYP, " Obj Prop bool " )
  field (OUT , "@OBJ=$(OBJ), PROP=Enable")
}
```

PLL Lock Status

Implemented for: bi

This indicates whether the phase locked loop which synchronizes an EVR's local oscillator with the phase of the EVG's oscillator. Outputs will not be stable unless the PLL is locked.

Except for immediately (1sec) after a change to the fractional synthesizer this property should always read as true (locked). Reading false for longer than one second is likely an indication that the fractional synthesizer is misconfigured, or that a hardware fault has occurred.

Link Status

Indicates when the event link is active. This means that the receiver sees light, and that valid data is being decoded.

A reading of false may be caused by a number of conditions including: EVG down, fiber unplugged or broken, and/or incorrect fractional synthesizer frequency.

Timestamp Valid

Indicates if the EVR has a current, valid timestamp. Conditions under which the timestamp is declared invalid include:

- TS counter reset event received, but “seconds” value not updated.
- Found timestamp with previous invalid value. Catches old timestamp in buffers.
- TS counter exceeded limit (eg. missed reset event)
- New seconds value is less then the last valid values, or more then two greater then the last valid value. (Light Soure time model only). This will reject single bad values sent by the EVG.
- Event Link error (Status is error)

The timestamp will become valid when a new seconds value is received from the EVG.

Model

The hardware model number.

Version

The firmware version number.

Sw Version

A string describing the version of the driver software. This is captured when the driver is compiled

FIFO Overflow Count

Counts the number of hardware event buffer overflows. There is a single hard- ware buffer for all event codes. When it overflows arbitrary events will fail to be delivered to software. This can cause the timestamp to falsely be invalidated, and can disrupt database processing which depends on event reception. This is a serious error which should be corrected.

Note: An overflow does not effect physical outputs.

FIFO Over rate

Counts overflows in all of the per event software buffers. This indicates that the period between successive events is shorter then the runtime of the code (callbacks, and database processing) that is causes. Extra events are being dropped and cause no action. Actions of other event codes are not effected.

Clock

Frequency of an EVR's local oscillator. This must be close enough to the EVG master oscilator to allow the phase locked loop in the EVR to lock. The native analog units are Hertz (Hz). This can be changed with the LINR and ESLO fields. Use ESLO of 1e-6 to allow user setting/reading in MHz.

Timestamp Sources

Determines what causes the timestamp event counter to tick. There are three possible values:

- **Event clock**: Use an integer divisor of the EVR's local oscillator.
- **Mapped code(s)**: Increments the counter whenever certain events arrive. These codes can be defined using special mapping records.
- **DBus 4**: Increments on the 0->1 transition of DBus bit #4.

Timestamp Clock

Specifies the rate at which the timestamp event counter will be incremented. This determines the resolution of all timestamps. This setting is used in conjunction with the 'Timestamp Source'. When the timestamp source is set to "Event clock" this property is used to compute an integer divider from the EVR's local oscilator frequency to the given frequency. Since this may not be exact it is recommended to read back the actual divider setting via the "Timestamp Prescaler" property. In all modes this value is stored in memory and used to convert the timestamp event counter values from ticks to seconds. By default the analog units are Hertz (Hz). This can be changed with the LINR and ESLO fields. Use ESLO of 1e-6 to allow user setting/reading in MHz.

Timestamp Prescaler

When using the "Event clock" timestamp source this will return the actual divisor used. In other modes it reads 0.

Timestamp

When processed creates a human readable string with either the current event link time, or the event link time when code # was last received. If code is omitted or 0, the the current wall clock time is used. Code may also have any valid event number 1-255. Then it will print the time of the last received event.

Event Clock TS Div

This is an approximate divider from the event link frequency down to 1MHz. It is used to determine the heartbeat timeout.

Receive Error Count

The number of event link errors which have occurred.

2.17.16 Pulse Generator

Properties in this section apply to the Pulse Generator (Pulser) sub-unit named $(OBJ):Pul# where # is a number between 0 and 15. Records accessing properties in this section will have DTYP set to “EVR Pulser”.

See: evrApp/Db/evrpulser.db

Enable

Implemented for: bo, bi When not set, the output of the Pulse Generator will remain in its inactive state (normally low).

The generator must be enabled before mapped actions will have any effect.

Polarity

Implemented for: bo, bi Reverses the output polarity. When set, changes the Pulse Generator’s output from normally low to normally high.

Prescaler

Implemented for: longout, longin Decreases the resolution of both delay and width by an integer multiple. Determines the tick rate of the internal counters used for delay and width with respect to the EVR’s local oscillator.

Delay

Implemented for: ao, longout, ai, longin Determines the time between when the Pulse Generator is triggered and when it changes state from inactive to active (normally low to high). This can be given in integer ticks, or floating point seconds. This can be changed with the LINR and ESLO fields. Use ESLO of 1e6 to allow user setting/reading in microseconds.

Width

Implemented for: ao, longout, ai, longin Determines the time between when the Pulse Generator changes state from inactive to active (normally low to high), and when it changes back to inactive. This can be given in integer ticks, or floating point seconds. This can be changed with the LINR and ESLO fields. Use ESLO of 1e6 to allow user setting/reading in microseconds.

Prescaler (Clock Divider)

Properties in this section apply to the Prescaler sub-unit. Records accessing properties in this section will have DTYP set to “EVR Prescaler”. See: evrApp/Db/evrscale.db

Divide

Implemented for: longout, ao, longin Sets the integer divisor between the Event Clock and the sub-unit output. By default the analog units are Hertz (Hz). This can be changed with the LINR and ESLO fields. Use ESLO of 1e-6 to allow user setting/reading in MHz.

2.17.17 Output (TTL and CML)

Properties in this section apply to the Output sub-unit. Records accessing properties in this section will have DTYP set to “EVR Output”. See: evrMrmApp/Db/mrmevrout.db

Map

Implemented for: longout, longin The meaning of this value is determined by the specific implementation used. For the MRM implementation the following codes are valid.

#	Output Source
63	Force High
62	Force Low
42	Prescaler (Divider) 2
41	Prescaler (Divider) 1
40	Prescaler (Divider) 0
39	Distributed Bus Bit 7
38	Distributed Bus Bit 6
37	Distributed Bus Bit 5
36	Distributed Bus Bit 4
35	Distributed Bus Bit 3
34	Distributed Bus Bit 2
33	Distributed Bus Bit 1
32	Distributed Bus Bit 0
9	Pulse generator 9
8	Pulse generator 8
7	Pulse generator 7
6	Pulse generator 6
5	Pulse generator 5
4	Pulse generator 4
3	Pulse generator 3
2	Pulse generator 2
1	Pulse generator 1
0	Pulse generator 0

2.17.18 Output (CML only)

Additional properties for Current Mode Logic (CML) outputs. Records accessing properties in this section will have DTYP set to "EVR CML" with the exception of waveform records which have either "EVR CML Pattern Set" or "EVR CML Pattern Get".

See: evrApp/Db/evrcml.db

Enable

Implemented for: bo, bi Trigger permit.

Power

Implemented for: bo, bi Current driver on.

Reset

Implemented for: bo, bi Pattern reset.

Mode

Implemented for: mbbo Selects CML pattern mode. Possible values are: 4x Pattern (0), Frequency (1), Waveform (2).

4x Pattern Uses the Pat Rise, Pat High, Pat Fall, and Pat Low properties to store four 20 bit (0 -> 0xfff) sub-patterns.

Frequency Uses the Freq Trig Lvl, Counts High, and Counts Low properties Waveform Uses the bit pattern stored by the Pattern Set property.

Pat Rise/High/Fall/Low

Implemented for: longout, longin Each property stores a seperate 20-bit pattern (0 -> 0xfff). These patterns are sent during the four parts of a square wave. Rising and Falling patterns start as soon as the edge is detected and will interrupt the pattern currently being sent. The High and Low patterns are sent after an edge pattern is sent and will repeat until the next edge.

Freq Trig Lvl

Implemented for: bo, bi Synchronize forces to this level when in frequency mode.

Counts High/Low

Implemented for: longout, longin, ao, ai Stores a value which is the number of counts (long) or time (analog) of the high or low part of a square wave. The number of ticks must be >20 and the time must be greater then one period of the event clock.

2.17.19 Input

Properties in this section apply to the Input sub-unit. Records accessing properties in this section will have DTYP set to "EVR Input". See: evrApp/Db/evrin.db

Active Level

Implemented for: bo, bi When operating in level triggered mode, determines if codes are sent when the input level is low, or high.

Active Edge

Implemented for: bo, bi When operating in edge triggered mode, Determines if codes are sent on the falling or rising edge in the input signal.

External Mode

Implemented for: mbbo, mbbi Selects the condition (Level, Edge, None) in which to inject event codes into the local mapping ram.

These codes are treated as codes coming from the downstream event link.

External Code

Implemented for: longout, longin Sets the code which will be applied to the local mapping ram whenever the 'External Mode' condition is met.

Backwards Mode

Implemented for: mbbo, mbbi Selects the condition (Level, Edge, None) in which to send on the upstream event link.

Backwards Code

Implemented for: longout, longin Sets the code which will be sent on the upstream event link whenever the 'Backwards Mode' condition is met.

DBus Mask

Implemented for: mbbo, mbbi Sets the upstream Distributed Bus bit mask which is driven by this input.

2.17.20 Event Mapping

Properties in this section describe actions which should be taken when an event code is received.

Pulse Generator Mapping

Implemented for: longout

See: evrApp/Db/evrpulsermap.db

Causes a received event to trigger a Pulse Generator (Pulser) sub-unit, or force it into an active (set) or inactive (reset) state.

These records will have DTYP set to "EVR Pulser Mapping". Each record will cause one event to trigger, set, or reset one Pulse Generator. It is possible (and likely) that more then one record will interact with each event code or Pulse Generator. However, each pairing must be unique.

```
record ( longout, "$(P)$(N)$(M)" ) {
   field(DTYP, "EVR Pulser Mapping" )
   field(OUT , "@C=$(C) , I=$(PID) , Func=$(F)" )
   field(PINI , "YES" )
   field(DESC, "Mapping for Pulser $(PID)" )
   field(VAL , "$(EVT)" )
   field(LOPR, "0" )
   field(HOPR, "255" )
   field(DRVL, "0" )
   field(DRVH, "255" )
```

In this example the event $(EVT) specified in the VAL field will cause function $(F) on Pulse Generator # $(PID). Current functions are 'Trig', 'Reset', and 'Set'.

Special Function Mapping

Implemented for: longout

See: evrApp/Db/evrmap.db and compare with *register definition.*

Allows a number of special actions to be mapped to certains events. These actions include:

Blink An LED on the EVRs front panel will blink when the code is received.

Forward The received code will be immediately retransmits on the upstream event link.

Stop Log Freeze the circular event log buffer. An CPU interrupt will be raised which will cause the buffer to be downloaded. This might be a useful action to map to a fault event.

Log Include this event code in the circular event log.

Heartbeat This event resets the heartbeat timeout timer.

Reset PS Resets the phase of all prescalers.

TS reset Transfers the seconds timestamp from the shift register and zeros the sub-seconds part.

TS tick When the timestamp source is 'Mapped code' then any event with this mapping will cause the sub-seconds part of the timestamp to increment.

Shift 1 Shifts the current value of the seconds timestamp shift register up by one bit and sets the low bit to 1.

Shift 0 Shifts the current value of the seconds timestamp shift register up by one bit and sets the low bit to 0.

FIFO Bypass the automatic allocation mechanism and always include this code in the event FIFO.

In the following example the front panel LED on the EVR will blink whenever event 14 is received.

```
record (longout , "$(P)map: blink" ) {
   field(DTYP, "EVR Mapping" )
   field(OUT , "@C=$(C) , Func=Blink" )
   field(PINI , "YES" )
   field(VAL , "14" )
   field(LOPR, "0" )
   field(HOPR, "255" )
}
```

2.17.21 Database Events

Implemented for: longout See: evrApp/Db/evrevent.db

A device support for the 'event' recordtype is provided which uses the Event FIFO to record the arrival of certain interesting events.

When set to SCAN 'I/O Intr' the event record device support will process the record causing the requested DB event.

```
record (longout ,"$(P)$(N)" ) {
   field (DTYP,"EVR" )
   field (SCAN,"I/O Intr" )
   field ( INP ,"@Card=$(C) ,Code=$(CODE)" )
   field (VAL ,"$(EVNT)" )
   field (TSE ,"2" ) # from device support
   field (FLNK,"$(P)$(N) : count" )

}

   record (calc, "$(P)$(N):count" ) {
   field (SCAN, "Event" )
   field (EVNT, "$(EVNT)" )
   field (CALC, "A+1" )
   field (INPA , "$(P)$(N) : count NPP" )
   field (TSEL, "$(P)$(N) .TIME" )
}
```

In this example the hardware event code '$(CODE)' will cause the database event '$(EVNT)'.

Note:

that while both '$(CODE)' and '$(EVNT)' are numbers, they need not be the same. Hardware code 21 can cause DB event 40.

2.17.22 Data Buffer Rx

Records accessing properties in this section will have DTYP set to "MRM EVR Buf Rx". See: evrApp/Db/mrmevrbufrx.db

Enable Data

Implemented for: bo Selects Event link data mode. This chooses between DBus only, and DBus+Buffer modes. In DBus only mode Data Buffer reception is not possible.

Data Rx

Implemented for: waveform When a buffer with the given Protocol ID is received a copy is placed in this record. It is possible to have many records receiving the same Protocol ID.

Note:

In order to avoid extra copy overhead this record bypasses the normal scanning process. It function like "I/O Intr", however the SCAN field should be left as "Passive".

```
record ( waveform , "$(P)dbus:recv:u32" )

{
   field (DESC,"Recv Buffer" )
   field (DTYP,"MRM EVR Buf Rx" )
   field ( INP ,"@C=$(C) , Proto=$(PROTO) , P=Data Rx" )
   field (FTVL,"ULONG" )
   field (NELM,"2046" )
   info(autosaveFields_pass0 , "INP" )
}
```

2.17.23 Data Buffer Tx

Records accessing properties in this section will have DTYP set to "MRF Data Buf Tx".

This section is shared between the EVR and EVG.

Outgoing Event Data Mode

See: mrmShared/Db/databuftxCtrl.db Implemented for: bo Selects Event link data mode. This chooses between DBus only, and DBus+Buffer modes. In DBus only mode Data Buffer transmission is not possible.

Data Tx

See: mrmShared/Db/databuftx.db This records sends a block of data with the given Protocol ID.

```
record ( waveform , "$(P)dbus:send:u32" )
{
   field (DESC, " Send Buffer")
   field (DTYP, "MRF Data Buf Tx" )
   field ( INP , "@C=$ (C) , Proto=$ (PROTO) , P=Data Tx" )
   field (FTVL, "ULONG" )
   field (NELM, " 2046 " )
   info( autosaveFields_pass0 , "INP " )
   info( autosaveFields_pass1 , "VAL" )
}
```

Per-device Database Files

Several database are installed by default for use with certain devices. Use with different devices is not an error, but will result in warnings being printed for sub-units included in the database file, but not physically present.

- db/evr-cpci-230.db
- db/evr-cpci-300.db
- db/evr-mtca-300.db
- db/evr-pcie-300dc.db
- db/evr-pmc-230.db
- db/evr-tg-300.db
- db/evr-vmerf-230.db

Special Database Files

Several database files are provided to augment the per-device files. These optional files are not tied to a specific hardware sub-unit.

- db/evrevent.db

Adds a reception counter for a specific event code.

- db/mrmevrtsbuf.db

Adds a capture buffer for reception times of a certain, fast, event code.

- db/evralias.db

A set of alias() entries to give an alternative (application specific) name prefix(s) for anEVR pulser.

- db/databuftx.db
- db/mrmevrbufrx.db

Examples of sending and receiving a data buffer.

- db/evrNtp.db

Status for the builtin NTP clock driver.

EnablePLL

This indicates whether the phase locked loop which synchronizes an EVR's local oscillator with the phase of the EVG's oscillator. Outputs will not be stable unless the PLL is locked. Except for immediately (1sec) after a change to the fractional synthesizer this property should always read as true (locked). Reading false for longer then one second is likely an indication that the fractional synthesize is misconfigured, or that a hardware fault has occured.

INDEX