ANL/TD/CP-94927

LEU Conversion Feasibility Studies for the BMRR and HFBR*

R.B. Pond, N.A. Hanan, and J.E. Matos
Technology Development Division
Argonne National Laboratory
9700 South Cass Avenue
Argonne, Illinois 60439

To be presented at:
1997 ANS Winter Meeting and Embedded Topical Meetings
Albuquerque, New Mexico
November 16-20, 1997

* Work supported by the U.S. Department of Energy, Office of Nonproliferation and National Security, under Contract No. W-31-109-ENG-38.

DISCLAIMER

DISCLAIMER

Portions of this document may be illegible in electronic image products. Images are produced from the best available original document.

are: Soft Links, VME/VXI, CAMAC, Allen Bradley, GPIB, Bitbus, and a couple of special bus types.

2.2 C Structure Definition

The program dbToRecordtypeH creates a C structure definition file from the record description file. This C file is included by the record support module and also by any associated device support modules.

2.3 Record Support Module

A record support module contains a set of methods that are accessed by iocCore via a record support entry table having the following structure:

```
struct rset {  /* record support entry table */
   long            number;
   RECSUPFUN       init;
   RECSUPFUN       init_record;
   RECSUPFUN       process;
   ...
}
```

Each record support module must define a global variable for its record support entry table. This definition has the form:

```
struct rset aiRSET={
   RSETNUMBER,
   report,
   ...
}
```

The global variable is aiRSET. It is permissible for any support method to be null. During IOC initialization, EPICS locates the address of the record support entry table for each record type.

3 Extensible Device Support

A device support module contains a set of methods that are called by record support. These methods implement device-specific code. For example, there is one analog input record support module but a large collection of analog device support modules, one for each supported analog device. In order to create support for a new device, a new device definition and a device support module must be created.

3.1 Create a Device Definition

A device definition has the form:

```
device(ai,VME_IO,devAiXX,"XX device")
```

The first parameter is the record type to which this device support is related, and the second parameter is the bus type. The third argument is the name of the global variable that will be discovered at IOC initialization, and the last parameter is a DTYP menu choice string.

3.2 Create the Device Support Module

The methods for a device support module are described in a device support entry table which has the following structure:

```
struct dset {
   long            number;
   DEVSUPFUN       report;
   DEVSUPFUN       init;
   DEVSUPFUN       init_record;
   DEVSUPFUN       get_ioint_info;
   /* other methods are record type dependent*/
}
```

Each device support module must define a global variable for its device support entry table. It has the form:

```
struct dset devAiXX={
   6,
   report,
   ...
}
```

4 Implementation Summary

The preceding two sections described the syntax for defining a record type and a device. In addition, a syntax is defined for defining menus and record instances. The complete syntax is called the EPICS Database Definition Language.

EPICS databases can be accessed via two libraries: static database access and run-time database access. The static library never calls any record or device support method while the run-time library does. The static library is supported on the cross development system as well as on IOCs. The run-time library is supported only on IOCs.

In general there are four types of software that need access to EPICS records: Database Configuration Tools, Channel Access Clients, IOC software accessing records in the same IOC, and Record/Device Support accessing its own record instances.

Database Configuration Tools can only use static database access. Channel Access Clients access records via a channel access (CA) server that resides in each IOC. The CA server in turn uses a combination of static and run-time database access. Any IOC software can use a combination

of static and run-time database accesses to access records in the same IOC.

Record/Device Support uses the C structure definition to directly access record instances of their record type. This provides very fast access. A C structure is also generated for dbCommon. This structure is included by many components of the core IOC software. Thus the core software can access these fields quickly.

The early versions of EPICS did not have both static and run-time access libraries. Originally there was only one Database Configuration Tool that directly accessed the internal structures used to implement the database. Thus, whenever any change was made to the internal structures, the Database Configuration Tool also had to be changed. The static access library was created to fix this limitation as well as to allow additional Database Configuration Tools.

5 Discussion

Although the design of extensible support was not intentionally object oriented, the design does have an object-oriented flavor. Here we discuss the good and bad aspects of using object-oriented ideas.

Since the structures generated from a Record Description File contain only data and not methods, they are NOT similar to Java or C++ classes. Because methods are not present, a clear separation between static and run-time database access is possible. This is a good feature and should be kept. It also allows generation of C structures that can be used by either C or C++ code.

The record and device support entry tables are almost like Java interfaces or pure abstract C++ classes. We can state that EPICS databases are defined via a Database Definition Language and an Abstract Interface Definition.

The two main shortcomings of the existing implementation are: 1) only two interfaces are defined: RSETs and DSETs (actually a third called a driver entry table is also defined), and 2) the way hardware links are implemented makes it extremely difficult to support arbitrary bus types and additional hardware configuration information.

6 Possible Future improvements

The first change would be to support an arbitrary number of interfaces. An Interface Definition syntax can be developed that would allow automatic generation of C and C++ module templates. A few interface definitions will be used by core IOC software but additional interfaces can be defined. A library will be provided to register and locate interface implementations.

The second change would be to redo the existing implementation of links. This involves defining a Structure Definition Format, eliminating DBF_DEVICE, adding two new link types, DBF_IN_STRUCTURE and DBF_OUT_STRUCTURE, and replacing the existing device definition with a link definition.

6.1 Structure Definition

A structure definition is just like a record type definition except that it begins:

```
structure(<name>) {
  field(. . .
}
```

and does not include dbCommon.

6.2 New Link Types

Instead of DBF_INLINK and DBF_OUTLINK define the following:

DBF_INLINK, DBF_OUTLINK
: These provide the same functionality as existing soft DBF_INLINK, DBF_OUTLINK, i.e., they are NOT used for hardware links.

DBF_IN_STRUCTURE, DBF_OUT_STRUCTURE
: These are the replacements for the existing DBF_DEVICE and INP, OUT.

6.3 Link Replacement for Existing Device Definition

The existing device definition will be replaced by:

link(<record_type>.<field>,<structure>,<interface>,
"<choice>")

where:

<record_type>.<field>
: is the record type and the field within the record to which this definition applies.

<structure>
: is the name of a structure that contains additional information for this link.

<interface>
: is the name of an interface that will handle processing of this link. The methods supplied by an interface will be record-type specific. Methods, however, will be defined that implement existing functionality.

<choice>
: is a choice string for Database Configuration Tools and for choosing an interface for run-time processing.

For example, the existing definition

```
device(ai,VME_IO,devAiDvx2502,"DVX-2502")
```

could be replaced by

```
link(ai.INP,VME_IO,devAiDvx2502,"DVX-2502")
```

where VME_IO is now the name of a structure rather than a hard-coded definition in link.h.

A record instance file currently has a definition similar to

```
field(INP,"#C0 S0 @").
```

This would become something like

```
field(INP.VME_IO) {
  card(0)
  signal(0)
  parm("")
}
```

6.4 Benefits

Because the link definition specifies both the record type and a field name, there is no hidden dependency between two fields like there currently is between DTYP and INP or OUT. In addition, it is possible to have multiple link definition fields in a record.

Because the link specifies a structure name, arbitrary information can be entered for the field. Thus, the currently hard-coded information in link.h is replaced by a general mechanism. Support for new bus types and other configuration information is possible.

Because the link specifies an arbitrary interface definition rather than just a DSET, more general interfaces are possible.

7 References

The following web site contains a large collection of EPICS documentation:
http://www.aps.anl.gov/asd/controls/epics/EpicsDocumentation/WWWPages/EpicsDoc.html

Of particular interest for this paper are the Application Developer's Guide and the Record Reference Manual.