EXAMPLE for device support that uses VME, PCI, ..
on an operating system that allows memory-mapped register access

Assume an analog input board with 'short' (16 bit) registers:
    Register 0: Version code
    Register 1: Channel 0
    Register 2: Channel 1
These are typically memory-mapped to some base address
in memory.
Details depend on the hardware and OS.

We want to be able to attach one or more records
which each read from the board.
The INP link provides the register index.

```
record(ai, "$(user):version")
{
    field(DTYP, "hwreg")
    field(INP,  "0")
    field(SCAN, "1 second")
}

record(ai, "$(user):ch0")
{
    field(DTYP, "hwreg")
    field(INP,  "1")
    field(SCAN, "1 second")
}

record(ai, "$(user):ch1")
{
    field(DTYP, "hwreg")
    field(INP,  "2")
    field(SCAN, "1 second")
}
```

In this example, we support only one board!
If we want to support multiple boards, INP would have
to hold some board index plus the register index on that board
--> Use INST_IO string which we'd then parse instead of CONSTANT number.

Or use "board * 1000 + register", so
```
    field(INP, "3002")
```
would address board #3, register #2 (= channel 1 on that board)

```
#include <stddef.h>
#include <stdio.h>
#include <string.h>
#include "alarm.h"
#include "dbDefs.h"
#include "dbAccess.h"
#include "recGbl.h"
#include "recSup.h"
#include "devSup.h"
```

```
#include "devLib.h"
#include "link.h"
#include "aiRecord.h"
#include "epicsExport.h"

// Holds info that we need in rec->dpvt
struct MyDeviceInfo
{
    short *register;
};

static long init_record(aiRecord *rec)
{
    // Read register index from INP link
    short index;
    recGblInitConstantLink(&rec->inp, DBF_SHORT, &index);

    // Base address depends on hardware, jumper settings, OS, ...
    // The manual for the board would help figure this out.
    // For Linux, you might have to do this:
    //
    //   int fd = open("/dev/that_piece_of_hardware", O_RDWR);
    //   short *base = mmap(...., PROT_READ, ...., fd, ....);
    //
    // For vxWorks, you need to do something like this:
    //   short *base;
    //   sysBusToLocalAdrs(VME_AM_SUP_SHORT_IO, 0x100, &base);
    //
    // Assume we somehow know this is the base address:
    short *base = (short *) 0x12344532;

    // Turn index of register into memory address of register
    MyDeviceInfo *info = new MyDeviceInfo();
    info->register = &base[index];

    // Just in case, we use an EPICS call for vxWorks and RTEMS
    // that checks if there is actually anything at that address
    short test;
    if (devReadProbe(2, info->register, &test) != 0)
    {
        printf("Dude, there is no AI board at 0x%x\n", base);
        rec->pact = 1;
    }

    /* device private (dpvt) is where we can park our device data */
    rec->dpvt = info;
    return 0;
}

static long read_ai(aiRecord *rec)
{
    MyDeviceInfo *info = (MyDeviceInfo *) rec->dpvt;

    // Two basic options:
    // Just read the register.
    // FAST, but risks bus error (CRASH) when
    // board removed at runtime
    rec->rval = *(info->register);

    // Other option would be to always use
```

```
    //  if (devReadProbe(2, info->register, &rec->rval) != 0)..
    // which is SAFE, but slower

    rec->udf = FALSE;
    return 0;
}

/* Create the device support entry table  */
aidset devAiHW =
{
    6,            /* number          */
    NULL,         /* report          */
    NULL,         /* global init     */
    init_record,
    NULL,         /* get_ioint_info  */
    read_ai,
    NULL          /* special_linconv */
};
epicsExportAddress(dset, devAiHW);
```

```
# dbd file that links DTYP="hwreg" to devAiHW
device(ai, CONSTANT, devAiHW, "hwreg")
```
