The Baraka System

Juan David González Cobas

January 20, 2011

1 The `vmeio` kernel module

1.1 Installation of `vmeio.ko`

This module provides the basic kernel-space facility for VME bus raw I/O. It is mainly due to the efforts of Julian Lewis.

To install it, the only pre-requisite is a Linux system running the `vmebridge` driver. A typical installation is as follows:

```
# insmod vmeio.ko \
    vme1=0x800,0xC00  amd1=0x29,0x29 win1=0x400,0x400 dwd1=4,4 \
    vme2=0x8100,0x8200 amd2=0x29,0x29 win2=0x100,0x100 dwd2=2,2 \
    vecs=0x83,0x84 lvls=2,2 luns=0,2
```

This command installs the `vmeio` driver and instructs it to handle two modules, referred to later by unit numbers 0 and 2. In this particular example, we are dealing with two CVORG modules. The first one, assigned unit number 0, has two register spaces that are memory-mapped and an interrupt to be handled. The first mapping is described by
vme base address 0x800
address modifier 0x29 (A16 non-privileged)
range or size of the window mapped 0x400
data width 4 (32-bit data)
and the second by
vme base address 0x8100
address modifier 0x29 (A16 non-privileged)
range or size of the window mapped 0x100
data width 2 (16-bit data)
The interrupt is at level 2, vector 0x83.

The second module has a similar set of data to be declared. For that purpose, the insmod arguments are actually arrays, and the values in them are correlative. So, the second CVORG module, to which we assign the arbitrary unit number 2, has the same paramenters as CVORG number zero, except that its base addresses are now 0xC00 and 0x8200, and that it uses interrupt vector 0x84.

The complete list of arguments is the following

luns Array of logical unit numbers assigned to the modules. These are different non-negative integers inferior to 32. All the remaining array arguments must have the same number of elements and correspond to the modules indexed by these luns.

vecs Array of interrupt vectors corresponding to the modules enumerated in `luns`.

lvls Array of interrupt levels.

vme1, vme2 Arrays of base addresses for the first and second register map of each module

win1, win2 Arrays of sizes for the first and second register map of each module

amd1, amd2 Arrays of address modifiers for the first and second register map of each module

dwd1, dwd2 Arrays of data widths for the first and second register map of each module (1, 2 or 4 meaning 8, 16 or 32-bit access).

nmap Non-map flag. If `nmap=1`, no windows are mapped and accesses will be done via block transfers instead of MMIO.

isrc Vector of offsets of the interrupt source register in the first register map

To avoid repeating values that are always identical in a module, the convention is adopted that a single-valued array argument is repeated as many times as luns are declared. So, we could abbreviate the command above to

```
# insmod vmeio.ko    \
     luns=0,2 \
     vme1=0x800,0xC00   amd1=0x29 win1=0x400  dwd1=4  \
     vme2=0x8100,0x8200 amd2=0x29 win2=0x100  dwd2=2  \
     vecs=0x83,0x84     lvls=2
```

Note 1 *This feature is probably best deprecated, and the arguments that need not be vectors (amd*, win*, lvls, isrc) declared as non-array parameters*

Note 2 *Probably, a simple GUI generating the list of parameters can make people's life easier*

Note 3 *The argument names are admittedly confusing; I would advocate to use explicit names like*

```
base_address
address_modifier
data_width
window_size
vectors
level
```

Note 4 *Another improvement here should be the possibility to install the module without any parameter, and set mappings and IRQs dynamically. To be done from Python.*

Note 5 *The following section on making device nodes should not be necessary; the device nodes should be created via* `sysfs`*, which takes out a real burden from the installation.*

1.2 Creating device nodes

By default, accessing the declared cards is done through device nodes named `/dev/vmeio.`x, where x is the unit number associated in the `insmod` command line to the module. Device nodes are created manually by issuing a command like

```
# mknod /dev/vmeio.0 c $MAJOR 0
# mknod /dev/vmeio.2 c $MAJOR 2
```

where MAJOR is the major device number associated to `vmeio` at installation time. It can be found out in the /verb—/proc/devices— file

Note 6 *This needs to be automated.*

2 The `pyvme` interface

The access to a VME device is provided by the python module `pyvme`. It provides a simple `Vme` class definition documented the usual pythonic way. An HTML doc is provided in the install directory.

An object is instantiated with the unit number that refers to it:

```
    ctr = pyvme.Vme(2)
```

From now on, ctr refers to the second module described at `vmeio` installation time. Now, methods for raw I/O, DMA analogous of them and for waiting for an interrupt with timeout are provided. Look for `pyvme.html` in the install directory: the usage is pretty straightforward.