SPEC Software Support

July 2012

A driver for the SPEC card and its FMC modules

Alessandro Rubini for CERN

Table of Contents

Introduction

This is the manual for the SPEC device driver. SPEC is the *Simple PCI-Express Carrier* for FMC cards, developed at http://www.ohwr.org/projects/spec. This manual is part of the associated software project, hosted at http://www.ohwr.org/projects/spec-sw, whose *git* repository hosts the latest version.

1 History and Overview

This driver is pretty different from the initial implementation, such as the one used by *fine-delay-sw-v1.1*. If you use such version, please compile the manual you find in your source code repository.

The package currently includes both the *spec* driver and the *fmc-core* driver (which benefits from its own documentation file which you may want to read.

2 Compiling the Drivers

The kernel modules part of this package live in the *kernel* subdirectory. To compile them, you need to set the following variables in your environment:

`LINUX`

The top directory of the kernel sources for the version you are going to run the driver under. I'm testing mostly with 3.4, but this version compiles against Linux-2.6.30 and later ones.

`CROSS_COMPILE`

If you are cross-compiling, you need to set this variable. It is not usually needed for the PC, but if you are using the *Powec* board, you'll most likely need this. It is not needed if you compile for a different-sized PC (see below).

`ARCH`

If you are cross-compiling, set this variable. Use `powerpc` for the *Powec*, `x86-64` if you compile on a 32-bit PC to run on a 64-bit PC and `i386` if you compile on a 64-bit PC to run on a 32-bit PC.

To compile run "`make`" with the previous variables set. To install run "`make install` to install under `/lib/modules/3.2.0` (or other version-based directory). You can set `INSTALL_MOD_PATH` to set a prefix before `/lib/modules`. For example, if your target computer's filesystem is mounted under `/mnt/target` you can run

```
make install INSTALL_MOD_PATH=/mnt/target
```

The modules are installed under the subdirectory `extra`. In the previous case your driver will end up being installed (together with the other modules) as

```
/mnt/target/lib/modules/3.2.0/extra/spec.ko
```

3 Role of spec.ko

The `spec.ko` driver depends on `fmc-core.ko`, that must be loaded first (if you don't rely on automatic dependencies).

The driver `spec.ko` registers itself as a PCI driver, using both the "old" vendor and device ID (the Gennum identifiers) and the new ones (CERN vendor and SPEC device).

For each new SPEC device found on the system, the driver performs the following steps:

- It enables MSI interrupts, the only kind supported in this package.
- It loads the `spec-init.bin` "golden" gateware file.
- It checks that the content of the binary is as expected (using a minimal *sdb*-based verification).
- It reads the whole I2C EEPROM found on the mezzanine.
- It allocates an *fmc_device* structure and registers as a nee device in the *fmc* bus.

Failure of any of those steps is fatal.

Any mezzanine-specific action must be performed by the driver for the specific FMC card, including reprograming the FPGA with the final gateware file. Similarly, the *spec* driver is not concerned with programming the LM32 image, when it makes sense to. This is different from the role splitting in previous versions of the driver.

Note: the gateware binary is looked-for in */lib/firmware/fmc*, which is where all fmc-related external files are expected to live. That's because our own installations share firmware for COTS peripherals but mount a host-specific NFS subdirectory.

Please refer to the *fmc-bus* document for details about the overall design of the interactions of carriers and mezzanines.

4 The WR-NIC

The *wr-nic* driver is being worked on. It should be a model of how FMC drivers should behave, so it is hosted in this package.

The driver will register an Network Interface Card on the host computer, to allow exchanging data on the same fiber channel where White-Rabbit synchronization is taking place. The driver relies on the 5-channel digital I/O mezzanine to time-stamp incoming pulses and generate precisely timed output signals.

5 Tools

The *tools* subdirectory of this package includes a few host-side programs that may be useful while working with the SPEC device.

The tools currently available are:

specmem

> The program acts like *devmem* or *devmem2* but in a simplified way. It receives one or two command line arguments: one for reading and two for writing. It makes a single 32-bit access to BAR0 of the Gennum PCI bridge; the first argument is the address, and the second argument is the value to be written. The `VERBOSE` environment variable makes the tool slightly more verbose.

spec-cl

This is the *cpu loader*. It is not called *lm32-loader* to avoid confusion with other tools we have been using. It loads a program at offset 0x8000 in BAR0. This is where we usually have RAM memory for the soft-core running in the SPEC. The program also keeps the soft-core in reset state during the load operation, so the new program will be run after loading is over.

Currently, the tools only work with one SPEC card (i.e., if you have two of them installed in your system, they work with one of them only – the first one appearing in *lspci*). This will be fixed in later versions, using command-line options or environment variables.

Use of these programs is exemplified in Chapter 6 [Troubleshooting], page 3.

6 Troubleshooting

When loading several binary files, sometimes there's something that goes wrong. This section includes a few suggestion to identify what step may have gone wrong.

First of all, you may want to check whether the FPGA binary has been properly loaded and stuff is running inside it. One test that can be made is whether the LM32 soft-core is actually able to run programs.

To this aim, you can compile and run the `spec-test00-cpu` program in the `test-lm32` directory of this package.

The strange name of the program is due to my plan to allow "`insmod spec name=test00`" in order to run this test and later other ones.

Currently, the LM32 programs can be re-loaded to a running FPGA using *spec-cl* (cpu loader), described in Chapter 5 [Tools], page 2. Thus, to load the test program, you can run this command:

```
spec-cl spec-test00-cpu.bin
```

The loader still sometimes fails (usually the first time), with a message like "`programming error at 0 (expected 98000000, found 0000c56a)`". Please retry: the second time it works (this has to be fixed, I apologize meanwhile).

The test program si simply incrementing an integer variable over and over. The variable lives at address 0x8000 in LM32 space, which is address 0x88000 in SPEC space. Thus, you can test it like this:

```
# spec-cl spec-test00-cpu.bin; specmem 88000; sleep 1; specmem 88000
00037cda
0025bfa2
```

If you `export VERBOSE=1` beforehand you'll get a little more 83 information from both programs (please note that any value in VERBOSE will work, the program only checks the variable exists):

```
# spec-cl spec-test00-cpu.bin; specmem 88000; sleep 1; specmem 88000
spec-cl: Wrote 1700 bytes at offset 0x8000
00088000 == 0003872c
00088000 == 0025c8e0
```

As you see a running LM32 core increments its memory location slightly more than 2.2million times per second (0x220000). If you can verify this, it means the FPGA is properly programmed and the CPU core is running.

More troubleshooting tests will be added.