SVEC Software Support

A driver for the SVEC card and its FMC modules
Git revision `41b1269` (2013-12-08 19:17:06 +0100)

Luis F. Ruiz Gago, Tomasz Wostowski (CERN BE-CO-HT)

Table of Contents

em

Introduction

This is the manual for the SVEC device driver. SVEC (*Simple VME FMC Carrier*) is a two-slot FPGA Mezzanine Carrier in VME64x form factor, developed at http://www.ohwr.org/projects/svec. This manual is part of the associated software project, hosted at http://www.ohwr.org/projects/svec-sw. The latest version of this work is always available in the *git* repository at ohwr.org.

1 History and Overview

The driver is a client of the FMC software bus abstraction. The package *fmc-bus* is dowloaded as a *git* submodule of this package, and its latest version can be found at http://www.ohwr.org/projects/fmc-bus.

Currently supported features are:

- Handling any number of SVEC cards with or without FMC mezzanines installed. Mezzanines are controlled in a carrier-agnostic way by their FMC drivers.
- A24 and A32 addressing modes with 32-bit data width.
- Sysfs-triggered VME bus configuration and device startup.
- Application FPGA firmware loading.
- Handling VME interrupts.

2 Compilation and installation

The kernel module that is part of this package lives in the *kernel* subdirectory. To compile it, you need to set the *LINUX* variable in your environment with the top directory of the kernel sources for the version your are going to run the driver under. The driver compiles against Linux-2.6.24 and it should work in later ones, but I'm only testing it with 3.2.43-rt63.

To compile run "`make`" with the previous variable set or "`make LINUX=<top_directory_of_kernel_sources>`".

Up to now Makefile doesn't perform an automatic module installation under a standard Linux installation, since we're mainly using it on CERN machines with their particular way of doing things.

Please note that by default the package compiles the *fmc-bus* modules, too (the project is a *git* submodule).

3 The svec.ko driver

The `svec.ko` is the only kernel module produced during compilation. It depends on `fmc.ko`, that must be loaded first (unless you rely on automatic dependencies), and the Linux VME bus infrastructure. It won't detect any SVECs unless a VME bus master/bridge driver is loaded (such as the Tundra TSI148 `vmebridge.ko` driver used at CERN).

During load time, the driver must be supplied with the list of the slots occupied by SVEC cards and the LUNs that will identify them in the system. Despite the SVEC being a VME64x card, there is no autodetection mechanism provided as it may be unsafe for certain older VME devices. The VME bus configuration can be supplied either when loading the driver, through

module parameters or at any later time through a sysfs interface (see Section 3.2 [SVEC Module Parameters], page 2 and Chapter 5 [User-Space Tools], page 5) .

The example below shows how to load the driver on a system with two SVECs installed in slots 4 and 12:

```
modprobe fmc
modprobe svec slot=4,12 lun=0,1
```

Note: The driver verifies the presence of the SVEC cards at a given slot by accesing their CSR registers which are geographically addressed. However, providing an incorrect slot number pointing to a non-VME64x card may have unpredictable results.

3.1 SVEC Initialization

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

- Map a VME CR/CSR window for the particular slot.
- Check if the card is present through the AFPGA bootloader interface.
- Check if the driver has been supplied with VME window configuration via the module parameters.
- If true, load the `fmc/svec-golden.bin` "golden" bitstream file (or any other bitstream configured through module parameters) and check what FMCs are connected.
- Map the register access VME window (A24/A32).
- Create two *fmc_device* structures and register as new devices in the *fmc* bus.
- If there is no VME configuration supplied, wait until the user sets up the VME window via sysfs and enumerate/startup the FMCs afterwards.

Failure of any of the above steps is considered fatal.

The suggested `svec-golden.bin` gateware binary is always available from the *files* area of the *svec-sw* project on `ohwr.org`. The binary version to be used with this software version is at `http://www.ohwr.org/projects/svec-sw/files`.

Note: currently the SVEC driver does not re-write the golden binary file when the sub-driver releases control of the card. This allows a further driver to make use of an existing binary, which may be useful during development.

3.2 SVEC Module Parameters

The module can receive the following parameters to customize its operation. All of the parameters below are arrays where each entry corresponds to one SVEC card.

`slot`

Mandatory. The slot(s) in which the SVEC(s) reside(s).

`lun`

Mandatory. Logical Unit (LUN) value for the SVEC.

`vmebase`

Optional. VME base address of the Application FPGA window.

`vme_size`

Optional. VME Application FPGA window size, in bytes. Default is `0x100000`.

`vme_am`

Optional. VME Application FPGA window address modifier, in bytes. Default is `0x39`, that is A24.

`vector`

Optional. VME IRQ vector used by the card.

`level`

Optional. VME IRQ level (default is 2).

`fw_name`

Optional. String parameter indicating the golden bitstream name, (`fmc/svec-golden.bin` by default). In a near future a default golden will be loaded - hopefully - from an onboard Flash memory and this parameter might be used to override it.

`show_sdb`

Optional. If not zero, the SDB internal structure of the golden binary is reported through kernel messages. Disabled by default.

`use_fmc`

Optional. If set to non-zero, the driver will not register the FMCs. Provided for debugging purposes.

Any mezzanine-specific action must be performed by the driver for the specific FMC card, including reprograming the FPGA with the final gateware file.

Note: the driver looks for the gateware binary 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* manual for details about the overall design of the interactions of carriers and mezzanines.

3.3 Interrupt support

The SVEC driver provides two ways of handling interrupts:

- Shared VME interrupt, routed to all interested FMC drivers.
- Vectored Interrupt Controller (VIC) interrupts.

3.3.1 Shared interrupt mode

In shared interrupt mode, the SVEC driver calls all registered FMC IRQ handlers until one of them has handled the interrupt by returing `IRQ_HANDLED`. Requesting a shared IRQ is done by passing IRQF_SHARED flag to `fmc->op->irq_request()`. If the interrupts are further multiplexed inside the FMC gateware, the driver must dispatch them accordingly.

```
irqreturn_t my_handler (int irq, void *data) {
    if(irq_is_for_us(irq))
         return IRQ_HANDLED;
    return 0;
}

fmc->op->irq_request( fmc, my_handler, "myirq", IRQF_SHARED);
```

3.3.2 VIC interrupt mode

This mode provides a simple abstraction for the Vectored Interrupt Controller (VIC), the standard BE-CO-HT HDL module for multiplexing interrupts inside an FPGA. The advantage is plug and play enumeration of the interrupts and no sharing overhead. Requesting an IRQ in VIC mode is done by:

- passing the SDB base address of the core whose interrupt we want to via the `irq` field in `struct fmc_device`.
- leaving the `flags` parameter at 0.

```
fmc->irq = fmc->base_address;
fmc->op->irq_request( fmc, my_handler, "my_vic_irq", 0);
```

The first time the `fmc->irq_request` is called, the SVEC driver will detect the VIC and configure it accordingly. It therefore requires an SDB-enabled gateware with correctly initialized VIC vector table. For more details on VIC hardware setup, please refer to the `general-cores` VHDL library manual.

4 The sysfs interface

The driver allows userspace programs to (re)configure each SVEC's VME interface through `sysfs` files. Such configuration method is useful for systems containing multiple SVEC cards mixed with other VME devices (such as many of CERN VME frontends, where the configuration is stored in an external database).

The `sysfs` controls for a SVEC card (identified by a `LUN`) reside in `/sys/bus/vme/devices/svec.LUN/` directory:

```
# ls -l /sys/bus/vme/devices/svec.0/
-rw-r--r-- 1 root root 4096 Aug 30 16:26 bootloader_active
-rw-r--r-- 1 root root 4096 Aug 30 11:53 configured
-rw-r--r-- 1 root root 4096 Aug 30 16:26 firmware_name
-rw-r--r-- 1 root root 4096 Aug 30 11:53 interrupt_level
-rw-r--r-- 1 root root 4096 Aug 30 11:53 interrupt_vector
-r--r--r-- 1 root root 4096 Aug 30 11:53 slot
-rw-r--r-- 1 root root 4096 Aug 30 11:53 use_fmc
-rw-r--r-- 1 root root 4096 Aug 30 11:57 vme_addr
-rw-r--r-- 1 root root 4096 Aug 30 11:53 vme_am
-rw-r--r-- 1 root root 4096 Aug 30 11:53 vme_base
-rw-r--r-- 1 root root 4096 Aug 30 11:57 vme_data
-rw-r--r-- 1 root root 4096 Aug 30 11:53 vme_size
```

4.1 Configuring the VME interface

VME configuration is done by writing the VME window parameters to appropriate files and afterwards, committing the changes by writing 1 to `configured` file:

```
# cd /sys/bus/vme/devices/svec.0
# echo 0x39 > vme_am
# echo 0xa00000 > vme_base
# echo 0x100000 > vme_size
# echo 0x80 > interupt_vector
# echo 1 > configured
```

The example above will configure a 1 MiB A24 window at `0xa00000`, and a VME interrupt at vector `0x80`. Alternatively, you can use the `svec-config` tool, documented in the next section.

Reading the `configured` file lets you check if the card has been already configured by someone else.

Note 1: VME reconfiguration causes the card's firmware and FMC drivers to be reloaded.

Note 2: Readback values of the VME parameters are updated when the configuration is committed.

Warning 1: The driver performs some trivial checks on the VME window/interrupt parameters, but it is still possible to crash the system by assigning a configuration that conflicts with other VME cards.

Warning 2: If the driver is to be configured via `sysfs`, it will almost always load without errors (unless there is no SVEC in the specified slot). If there's something wrong (with the SVEC config or the attached FMC drivers), the errors will be triggered during userspace reconfiguration.

4.2 Raw access to the VME registers

This is handled via the `vme_addr` and `vme_data` attributes. In order to read something from a given address, put the address in `vme_addr` file and then read the `vme_data` file. Writes are done in the same way.

Note: Raw VME access through `sysfs` works only if the VME register window is correctly configured.

5 User-Space Tools

5.1 svec-config

`svec-config` is a simple command-line tool for configuring the VME bus through sysfs. The parameters are described below:

`-l`
: Prints the list of all SVECs in the system.

`-u lun`
: Logical Unit number of the card to be configured. The card must be selected either by this, or by `-s` parameter.

`-s slot`
: Slot number of the card to be configured. The card must be selected either by this, or by `-u` parameter.

`-a modifier`
: Address modifier to use (acceptable values are `A24` or `A32`). No hex values are allowed. Default is `A24`.

`-b address`
: Base address of our VME window. Mandatory.

`-w size`
: Size of our VME window, in bytes. Default is `0x100000`.

`-v vector`
: VME Interrupt vector, if the card uses interrupts. Default is `0x86`.

`-k level`
: VME Interrupt level, if the card uses interrupts. Default is `2`.

`-f enable`
: Enables/disables enumeration and loading of FMC drivers after reconfiguration. On by default.

For example, the command below configures the card in slot `12` to use an A24 window at `0xc00000`:

```
./svec-config -s 12 -a A24 -b 0xc00000
```

Note: `svec-config` requires a Python interpreter.

6 Bugs and Missing Features

- Userspace-triggered firmware loading (that doesn't conflict with the FMCs).
- Flash programming tool.