

Firmware Upgrade in xTCA Systems

Dariusz Makowski

Piotr Perek

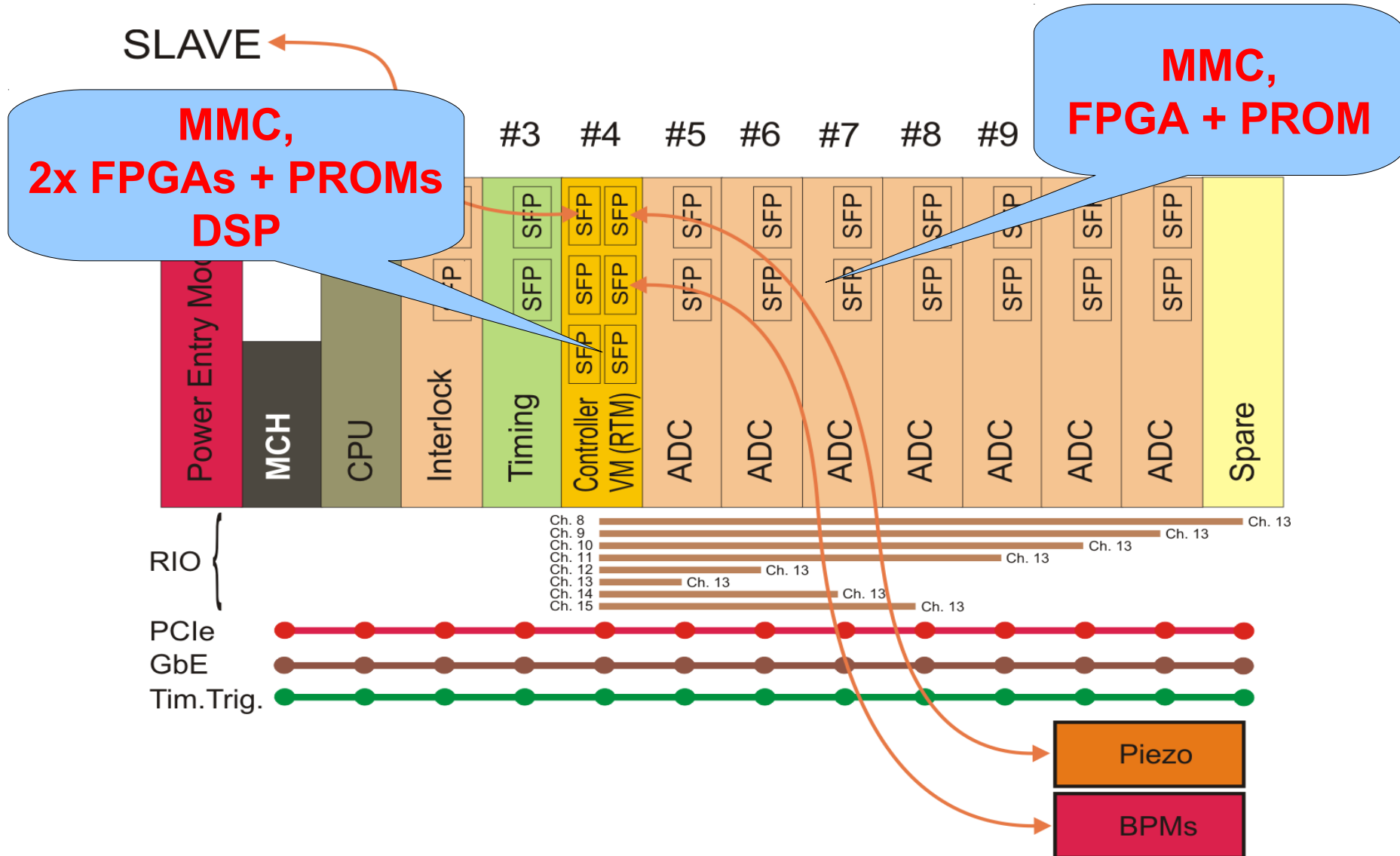
Grzegorz Jabłoński

xTCA and Programmable Devices (1)

Control and Data Acquisition Systems require programmable devices:

- Field Programmable Gate Array devices (~1 – 20 MB),
- Processors, DSP (~100 kB – 10 MB),
- IPMI controller: MMC/IPMC/RMC (~10 – 1 MB).

uTCA-based LLRF System



FPGAs and PROMs



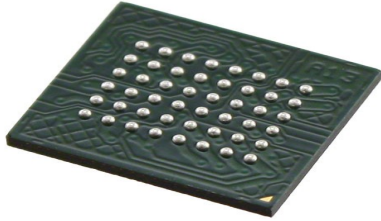
Xilinx Virtex 5

2x XCF32P

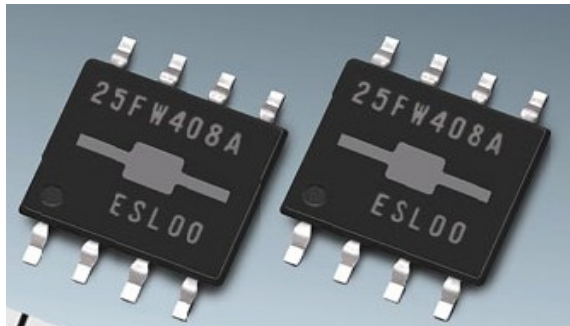
**Xilinx Spartan 6
SPI Platform FLASH**



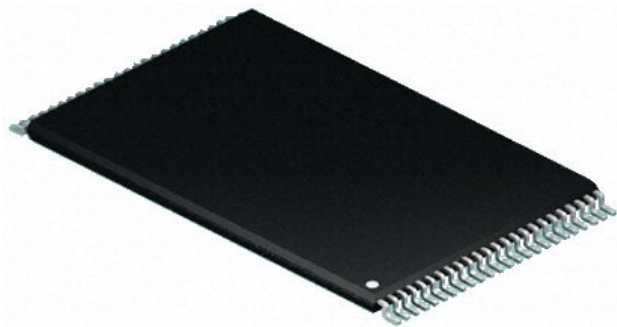
Xilinx PROMs



Platform FLASH with JTAG
(1 – 32 Mbit)



SPI FLASH memory
(1 kbit – 1 Gb)



BPI FLASH memory
with parallel bus
(64 kbit – 8 Gb)

Virtex 5/6 Family Requirements

Table 1-4: Virtex-5 FPGA Bitstream Length (Continued)

Device	Total Number of Configuration Bits ⁽¹⁾
XC5VLX20T	6,251,200
XC5VLX30T	9,371,136
XC5VLX50T	14,052,352
XC5VSX95T	35,716,096
XC5VSX240T	79,610,368

Table 6-4: Virtex-6 FPGA Bitstream Length

Device	Total Number of Configuration Bits ⁽¹⁾
XC6VHX250T	79,862,624
XC6VHX255T	79,862,624
XC6VHX380T	119,784,608
XC6VHX565T	160,655,264
XC6VLX550T	144,092,384
XC6VLX760	184,823,072

Firmware Upgrade Scenarios

Regular operation

- Updates of Flash memories and PROMs relatively rarely (upgrade to new firmware)

Development in labs and facilities

- Direct programming of FPGA
- Debugging over JTAG
- Frequent firmware updates (must be fast)

xTCA and Programmable Devices (2)

Two similar standards are available, however the proposed methods for firmware upgrade can differ significantly:

- ATCA hardware,
- uTCA hardware.

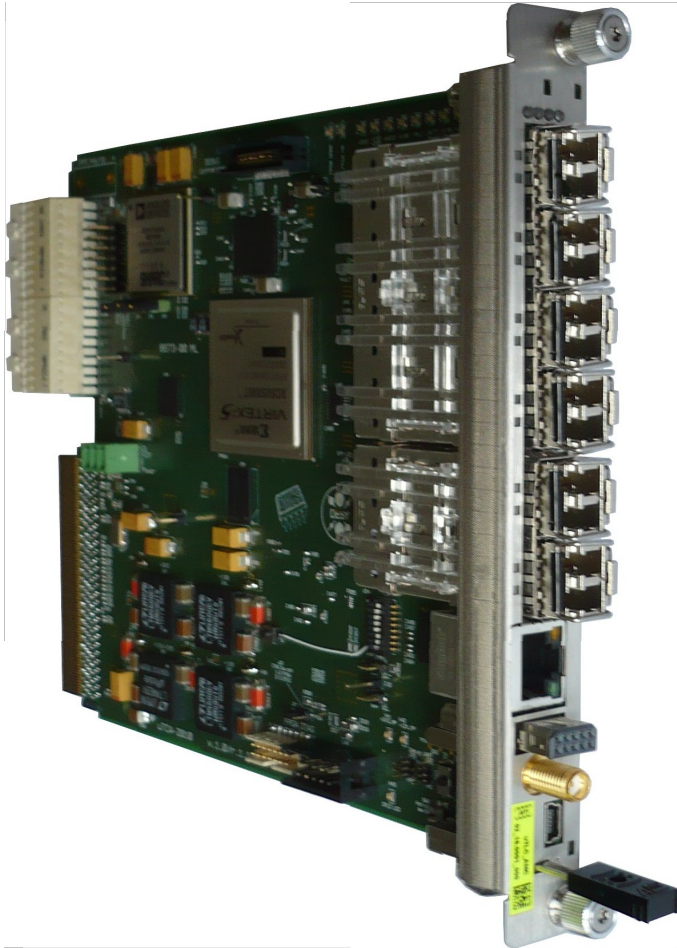
ATCA - Firmware Upgrade

- **IPMI** and **HPM.1** standards (IPMC),
- **I2C bus** – very slow programming (max. 100 kbit/s), large protocol overhead, no block transfer available (redundant bus),
 $36 \text{ Mbit} / 100 \text{ kbit/s} * 5 = 1800 \text{ s}$
- **IPMI-over-Ethernet** – (100 Mbit/s) requires access to base interface,
 $36 \text{ Mbit} / 100 \text{ Mbit/s} * 5 = 1,8 \text{ s}$
- **Custom solution** based on fabric interface (PCIe, GbE, etc.).

uTCA - Firmware Upgrade

- **IPMI** and **HPM.1** standards (MMC),
- **I2C bus** – very slow programming (max. 100 kbit/s), large protocol overhead, no block transfer available
- **IPMI-over-Ethernet** – not possible, only 1000Base-X on port 0/1 (requires large and complex processor for MMC),
- **Custom solution** based on main interface (PCIe, GbE, etc.).

Supported Boards

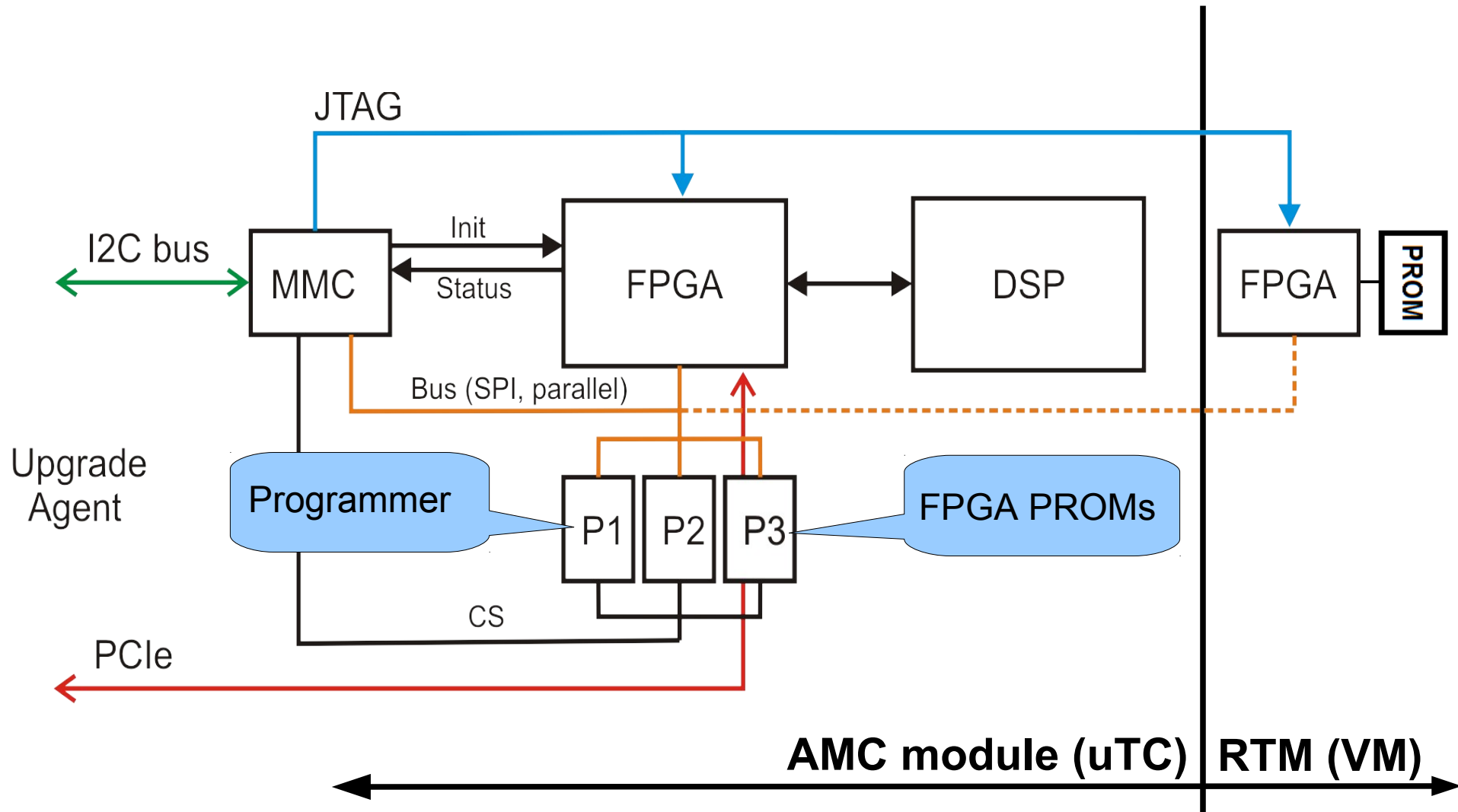


uTCA Controller for LLRF system

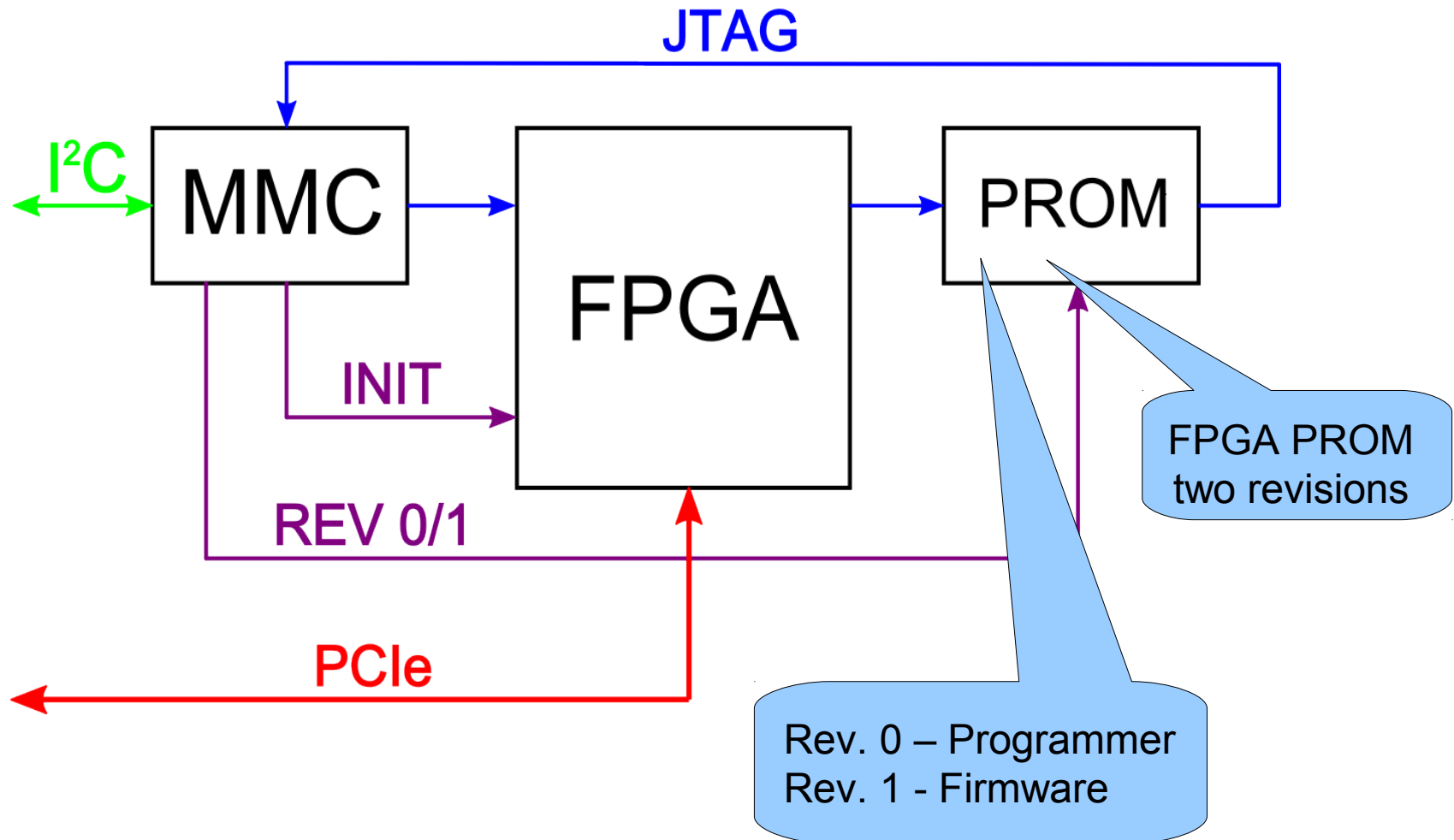


SIS8300 board

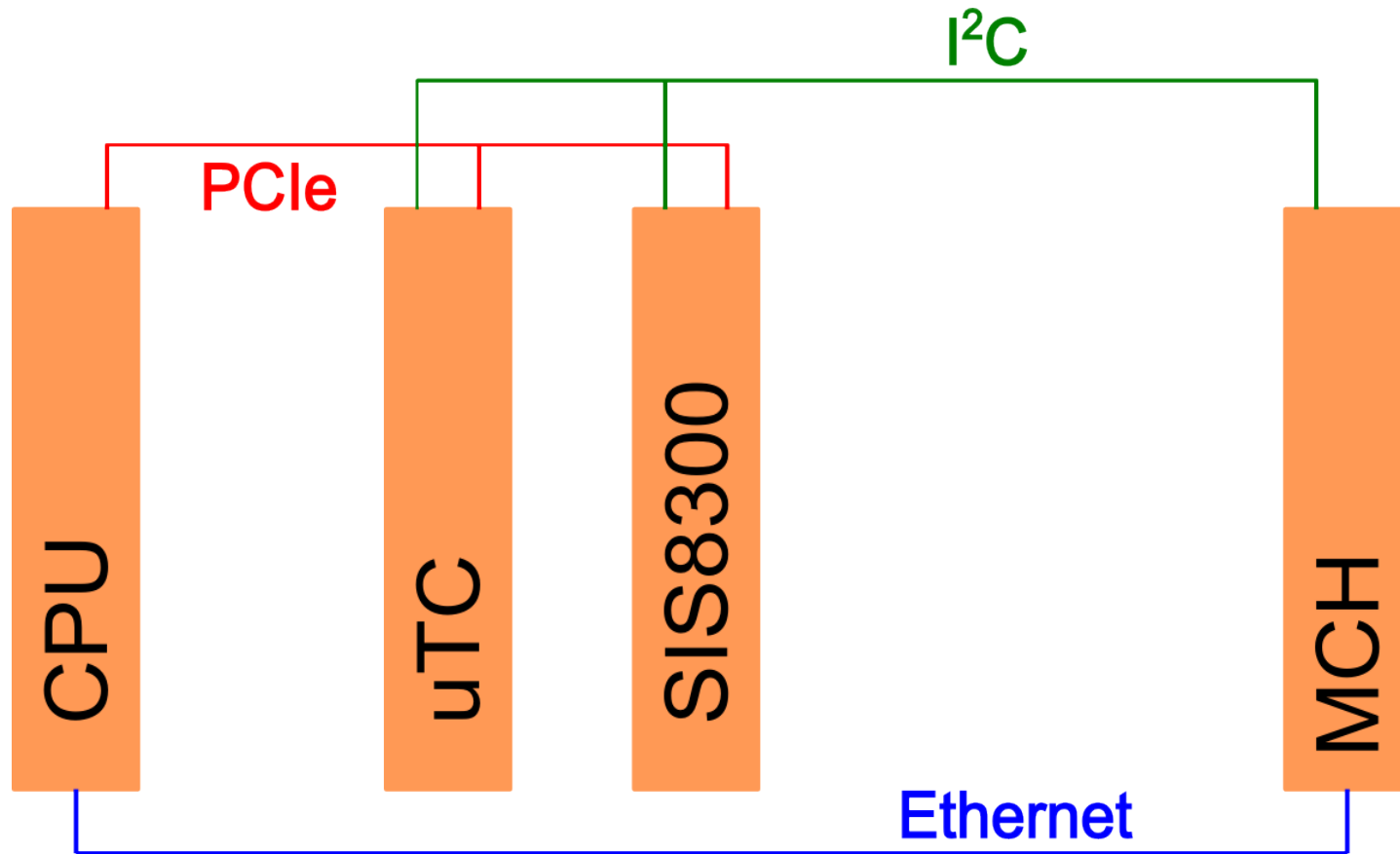
uTC – Programmable Devices



SIS8300 – Programmable Devices



Firmware Upgrade



FPGA Firmware Upgrade Procedure

1. Select memory to be programmed
2. Erase memory (JTAG, SPI, Parallel PROM)
3. Program memory (revision > 0, memory nr > 0)
4. Reload FPGA
5. Rescan PCIe bus
6. If failed back to PROM with programmer (revision 0 or memory 0)

Firmware Upgrade – IPMI Support

- Support from MMC required
 - New commands for PROMs and FPGA booting management:
 - Set PROM Number,
 - Get PROM Number,
 - Init FPGA (PICMG FRU Control),
 - Check FPGA Booting Status,
 - Configure Programming Method (JTAG, IPMI, PCIe, etc.)
 - Check Programming Method,
 - etc.
- Firmware Upgrade Agent
 - Protocol via PCIe, GbE, sRIO, etc.,
 - ipmitool for sending IPMI command.

Supported AMC modules

- uTC – 3 SPI PROMs
- SIS8300 – 1 Platform Flash with two revisions
- DAMC2 – 1 Platform Flash with two revisions
- ...

However, different IPMI commands – needs to be standardized.

Programming Time of PROMs

Comparison of memory erase and programming time by the Xilinx programmer and by llrf_prog	Xilinx programmer + impact	llrf_prog
SPI FLASH - uTC board (1 memory)	ca. 14 min	1 min 30 s
PlatformFlash-SIS8300 (2 revisions)	6 min 12s	6 min 25s

Bash Scripts for Firmware Upgrade

MTCA-based Controller uTC

Booting from different memory

./utc_mem_switch <Slot_number> <Memory with firmware>

./utc_mem_switch 4 2

Memory programming

./utc_programmer <Bitstream.bin> <Slot_number> <Memory with firmware>

./utc_programmer ./controler.bin 4 2

DAQ card SIS8300

./sis_mem_switch <Slot_number> <Memory bank with firmware>

./sis_programmer <Bitstream.xsvf> <Slot_number> <Memory bank with firmware>

Software for Firmware Upgrade

./uTCA_programmer [board_type] [options] [arguments]

where:

[board_type]

-U: uTC board

-S: SIS8300 board

[options]

-h <hostname>: IP address or name of MCH (e.g. mskmch2)

-p: program SPI PROM on the board

-r: recover default FPGA firmware on the board

-s: switch SPI PROM memories on the board

(0 or 1 in case of SIS8300 or 1 - 3 in case of uTC)

./uTCA_programmer -U -h mskmch2 -p 4 2 ./bin/uTC_firmware.bin

./uTCA_programmer -S -h mskmch2 -p 6 1 ./bin/SIS8300_rev1.xsvf

PCIe Bus Hot-Plug

- Scripts proposed by Adam Piotrowski:

`./down -b 0000:05:00.0`

`./up 0000:0a:09.0`

- Modified scripts with translation table :

`./down 4`

`./up 4`

`./rspci 4`

`PCIe_Down=(0000:09:00.0 0000:08:00.0 0000:07:00.0 0000:06:00.0 0000:05:00.0 0000:04:00.0
0000:11:00.0 0000:0f:00.0 0000:0e:00.0 0000:0d:00.0 0000:0c:00.0 0000:0b:00.0)`

`PCIe_Up =(0000:03:00.0 0000:03:01.0 0000:03:08.0 0000:03:09.0 0000:03:0a.0 0000:03:0b.0
0000:0a:01.0 0000:0a:02.0 0000:0a:08.0 0000:0a:09.0 0000:0a:0a.0 0000:0a:0b.0)`

Summary

- IPMI commands should be standardized
- Standardized reference design for hardware design (select supported memories)
- Firmware upgrade method (IPMI and PCIe)
- JTAG player should be included in each firmware
- In revision 0 and memory 0 should be always programmer (only programmer)
- Separated BAR for firmware upgrade?

PCIe Tree of NAT Switch

```

-[0000:00]--+-00.0
+-00.1
+-01.0
+-02.0-[0000:14]--+-00.0
|                \-00.1
+-03.0-[0000:11-13]--
+-1c.0-[0000:02-10]----00.0-[0000:03-10]--+-00.0-[0000:09-10]----00.0-[0000:0a-10]--+-01.0-[0000:10]--
|                |                +-02.0-[0000:0f]----00.0
|                |                +-08.0-[0000:0e]--
|                |                +-09.0-[0000:0d]--
|                |                +-0a.0-[0000:0c]--
|                |                \-0b.0-[0000:0b]--
|                |
|                +-01.0-[0000:08]--
|                +-08.0-[0000:07]--
|                +-09.0-[0000:06]--
|                +-0a.0-[0000:05]--
|                \-0b.0-[0000:04]--
+-1d.0
+-1d.1
+-1d.7
+-1e.0-[0000:01]----02.0
+-1f.0
+-1f.2
\ -1f.3

```

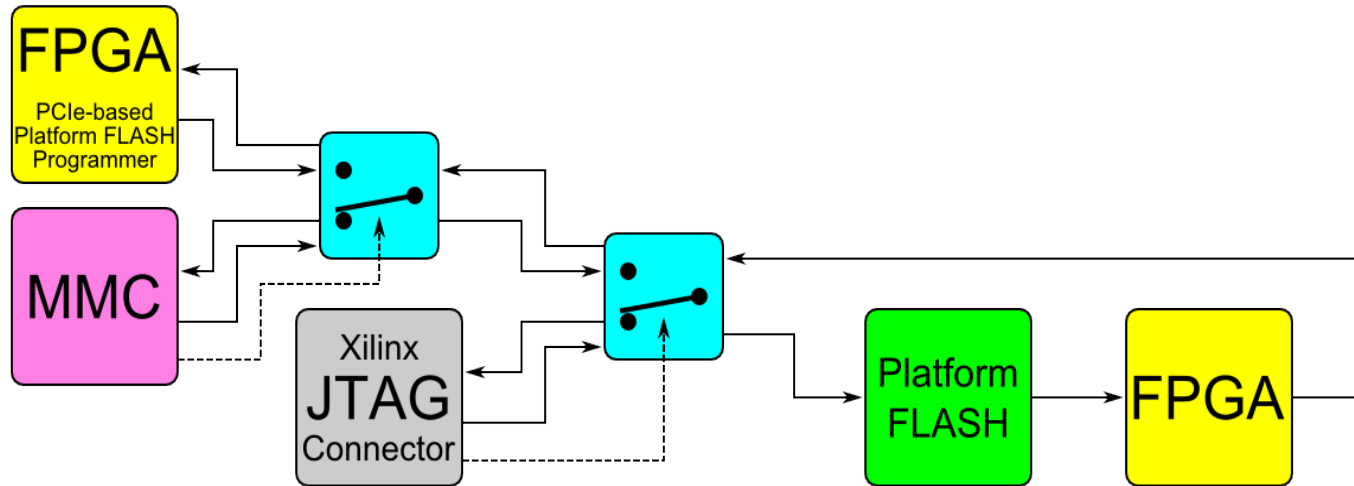
Script for XSVF File Generation

```
# SIS8300 - Script for impact
# Input files: firmware1.bit and firmware2.bit
# Output file: firmware.mcs.
setPreference -pref StartupClock:AUTO_CORRECTION
setMode -pff
setSubmode -pffversion
addPromDevice -position 1 -name xcf32p
addCollection -name firmware
addDesign -version 0 -name 0000
addDeviceChain -index 0
setCurrentDesign -version 0
addDevice -position 1 -file firmware1.bit
addDesign -version 1 -name 2000
addDeviceChain -index 0
setCurrentDesign -version 1
addDevice -position 1 -file firmware2.bit
generate -format mcs -fillvalue FF
setMode -bs
setCable -port xsvf -file "sis8300.xsvf"
addDevice -p 1 -sprom xcf32p -file firmware.mcs
addDevice -p 2 -part xc5vlx50t
Erase -p 1 -ver 0 -ver 1
Program -p 1 -parallel -ver 0 -ver 1 -defaultVersion 1
quit
```

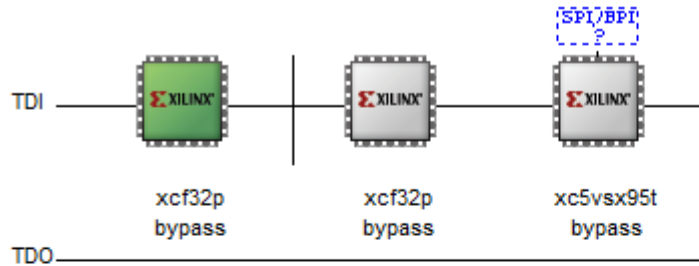

Registers of Programmer Core

Name	Offset	Description
write_data	0000h	Write buffer (1024 8-bit words). Put here bytes to be sent by SPI.
read_data	1000h	Read buffer (1024 8-bit words). Bytes read via SPI are available there.
spi_divider	2000h	Divider for the spi clock with respect to iclk. Default: 10
bytes_write	200ch	Number of bytes to write via SPI minus 1
bytes_read	2010h	Number of bytes to read via SPI minus 1
control	2014h	Bit 0: Write 1 to start transmission. When the SPI transfer is over, automatically reset to 0 by the firmware Bit 1: Write 1 if the SPI command should read bytes from SPI, 0 if only write is to be performed
tck	2018h	State of the TCK JTAG line
tms	201ch	State of the TMS JTAG line
tdi	2020h	State of the TDI JTAG line
tdo	2024h	State of the TDO JTAG line

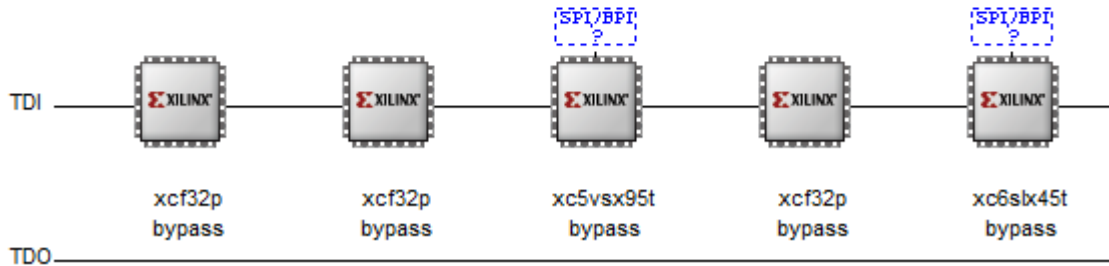
JTAG Chain of SIS8300



JTAG Chain for uTC



Chain without RTM (uVM)



Chain with RTM (uVM)