FPGA Common Documentation

Version 1.0

February 9, 2023

René Geißler
r.geissler@gsi.de

GSI Helmholtzzentrum für Schwerionenforschung GmbH

Contents

Resources 4

1 Introduction 5

2 Common monitoring and control features 6
2.1 Register bank 6
2.1.1 Status registers 6
2.1.2 Configuration Registers 6
2.2 Architecture information storage 6
2.2.1 Register information 6
2.2.2 Gateware information 7
2.2.3 Observer signal information 7
2.3 Observer 7
2.3.1 Observer configuration registers 8

3 FPGA Observer 9
3.1 Installation 9
3.1.1 Build from source 9
3.1.2 RPM 9
3.1.3 FPGA TCP server 9
3.1.4 PCIe driver 9
3.2 Usage 10
3.2.1 Register tab 10
3.2.2 Observer Trigger tab 11
3.2.3 Observer tab 12
3.2.4 Gateware tab 13
3.2.5 Project specific tabs 13

4 Build flow and simulation 14
4.1 Prerequisites 14
4.2 Build flow 14
4.2.1 GUI based build flow 14
4.2.2 Scripted build flow 14
4.3 Simulation 14
4.3.1 GUI based simulation 14
4.3.2 Scripted simulation 14
4.3.3 Peripherals simulation models 15

5 Helper scripts 16
5.1 PCIe access test script 16
5.2 VHDL beautification 16
5.3 Remote power cycling of a MicroTCA crate 16
5.4 Generation of a VHDL file for monitoring and control 16
5.5 Generation of documentation 16
5.5.1 PDF 16
5.5.2 DokuWiki 17

6 Continuous integration environment 18
6.1 Installation 18
6.2 Pipeline Stages 19
6.2.1 Documentation 20
6.2.2 Simulation 20

6.2.3 FPGA build . . . 20
6.2.4 Timing check . . . 21
6.3 Build results . . . 21
6.4 Settings . . . 21

7 Programming and hardware configuration **22**
7.1 Programming the gateware to the FPGA on an AFC board . . . 22
7.1.1 AFC version 3.1 and earlier . . . 22
7.1.2 AFC version 4 . . . 22
7.1.3 Storing a bitstream persistently in the SPI Flash . . . 22
7.1.4 JSM JTAG device numbering on PowerBrige 6 slot crate . . . 23
7.2 Configuration of the MCH . . . 24
7.2.1 Via the MCH's web interface . . . 24
7.2.2 Via USB . . . 24
7.3 Controlling a MicroTCA crate via the MCH . . . 24
7.3.1 SSH . . . 24
7.3.2 IPMI . . . 24
7.4 Enabling network boot on the FEC . . . 25
7.4.1 BIOS settings . . . 25
7.4.2 Defining the boot image . . . 25
7.5 MMC firmware . . . 26
7.5.1 Differences in MMC firmwares . . . 26
7.5.2 Building the openMMC firmware . . . 26
7.5.3 Programming the MMC firmware on AFC version 3.1 . . . 27
7.5.4 Programming the MMC firmware on AFC version 4 . . . 28
7.6 Configuration of an FTRN . . . 28

8 Hardware properties **29**
8.1 LEDs on the AFC front panels driven by the MMC . . . 29
8.1.1 Lighting patterns of the Hot Swap LED . . . 29
8.2 MCH PCIe status LEDs . . . 29
8.3 Differences between AFC versions . . . 30
8.4 Maximum achievable data rate to and from SDRAM . . . 30

9 FPGA board inventory list **31**
9.1 List of AFC boards . . . 31
9.1.1 AFC v2.0 . . . 31
9.1.2 AFC v3.1 . . . 31
9.1.3 AFC v4.0 . . . 31

References **32**

Resources

The code of this project and also the source of this documentation are under version control in a Git repository whose upstream is:
https://git.gsi.de/BEA_HDL/FPGA_Common
The relevant branch is *master*.

1 Introduction

This repository contains FPGA related code that is common to multiple projects. It is included as a Git submodule in the following projects:

- https://git.gsi.de/BEA_HDL/Cryring_BPM_Gateware
- https://git.gsi.de/BEA_HDL/UniMon_Gateware
- https://git.gsi.de/BEA_HDL/Rate_Divider_Gateware
- https://git.gsi.de/BEA_HDL/BLoFELD_Gateware
- https://git.gsi.de/BEA_HDL/BLoMQuIST_RTM_PCB
- https://git.gsi.de/BEA_HDL/Resonant_Transformer_Gateware
- https://git.gsi.de/BEA_HDL/Red_Pitaya_Gateware

2 Common monitoring and control features

Each gateware project that incorporates this repository uses the following common monitoring and control features:

2.1 Register bank

2.1.1 Status registers

There are 128 status registers, each of which has a width of 64 bits. They are mapped to a memory region starting from the address `0x00000000`, with an increment of `0x00000008` for every register.

Writing data to a status register does not have any effect.

2.1.2 Configuration Registers

There are 128 configuration registers, each of which has a width of 64 bits. They are mapped to a memory region starting from the address `0x00000400`, with an increment of `0x00000008` for every register.

Configuration registers can be written to and read from. Every write access has to write the full width of 64 bits. Unused bits can be set to any value. Write accesses with a width of e.g. 32 bits do not have any effect.

2.2 Architecture information storage

There is a Block RAM which is used to store information about the registers, the observers and the gateware version.

2.2.1 Register information

Information about the 128 status registers and the 128 configuration registers is stored in the first half of the Block RAM. Following information is stored for every register:

- name of register (31 bytes)
- number of bits (1 byte)

address	bytes 0 - 30	byte 31
`0x00004000`	name of status register 0	width of status register 0
`0x00004020`	name of status register 1	width of status register 1
...	...	...
`0x00004FE0`	name of status register 127	width of status register 127
`0x00005000`	name of configuration register 0	width of configuration register 0
`0x00005020`	name of configuration register 1	width of configuration register 1
...	...	...
`0x00005FE0`	name of configuration register 127	width of configuration register 127

Table 2.1: Register information storage formats

Table 2.1 shows the storage format of the 256 entries, each of which has a width of 32 bytes. The names are stored as ASCII strings. If a name is shorter than 31 bytes, the remaining bytes are filled with Null characters. If not all registers are in use, a width of 0 bits indicates that a register is not present.

The register information is used by the FPGA Observer software to display the registers in the Register Access tab (see chapter 3.2.1).

2.2.2 Gateware information

The address range from `0x00006000` to `0x00006FFF` is used to store information about the gateware version. The information is stored as an ASCII string of variable length (maximum 4 kiB), which is assembled from information from the Git repository. It contains the URL of the remote server of the Git repository, the latest commit hash and the latest commit date.

The gateware information is used by the FPGA Observer software to display the information in the Gateware Information tab (see chapter 3.2.4), except from the bitstream generation date, which is read from status register 124 'build timestamp'.

2.2.3 Observer signal information

The address range from `0x00007000` to `0x00007FFF` is used to store information about the signals connected to the 16 observer multiplexer inputs. Following 32 bytes wide information is stored for every signal connected to each of the 16 64 bits wide multiplexer inputs:

- name of signal (bytes 0 to 28)
- number of bits (the 6 LSBs of byte 29)
- bit offset inside the 64 bits wide observer input vector (the 6 LSBs of byte 30)
- display type of signal (the 4 LSBs of Byte 31)
- observer input number (the 4 MSBs of Byte 31)

The used storage size depends on the number of signals connected to the observer multiplexer inputs. If the field *number of bits* of one entry reads 0, it indicates that the corresponding 32 byte information is not used.

The coding of the display type is the following:

value	display type
0	unsigned analog
1	signed analog
2	unsigned int
3	signed int
4	hexadecimal

The names are stored as ASCII strings. If a name is shorter than 29 bytes, the remaining bytes are filled with Null characters.

The observer signal information is used by the FPGA Observer software to display the *Observer Trigger* tab and the *Observer* tab.

2.3 Observer

The observer is an intgrated logic analyzer with a simple two-stage trigger. Two signal vectors of 64 bits each can be observed in parallel, each of which can be chosen from a list of up to 16 different input vectors.

The observer stores a fixed number of 4096 samples after a configurable two-stage trigger condition has occurred in the signal.

2.3.1 Observer configuration registers

The observer uses the following configuration registers:

index	address	bits	radix	description	default value
112	0x00000780	1	binary	observer ignore valid	0
113	0x00000788	4	unsigned	observer multiplexer 0 select	0x0
114	0x00000790	4	unsigned	observer multiplexer 1 select	0x1
115	0x00000798	4	unsigned	observer trigger select	0x0
116	0x000007A0	64	none	observer trigger compare vector (t = 0)	0x0000000000000000
117	0x000007A8	64	none	observer trigger compare vector (t = 1)	0x0000000000000000
118	0x000007B0	64	none	observer trigger compare bit mask	0x0000000000000000
119	0x000007B8	1	binary	observer capture	0
120	0x000007C0	1	binary	observer cancel	0

Table 2.2: List of additional configuration registers

112: Observer ignore valid

If set to 1, the valid signal connected to the observer multiplexer input will be ignored and the data will be sampled at every clock cycle.

113, 114: Observer multiplexer {0, 1} select

The observer stores samples that are 128 bits wide, which consist of two concatenated 64 bits wide multiplexer outputs. Each multiplexer can choose between 16 different input vectors. Like this, each signal can be observed in parallel to any other signal.

115: Observer trigger select

Analog to register 113 and 114. Determines on which observer input vector the trigger listens.

116: Observer trigger compare vector (t = 0)

64 bit wide compare vector that is compared with the observer input vector determined by register 115 *observer trigger select*. If the two patterns match, the next sample will be compared to the compare vector determined by register 117 *trigger compare vector (t = 1)*.

117: Observer trigger compare vector (t = 1)

See above. If the patterns do not match, the next sample will be compared to the compare vector determined by register 116 *trigger compare vector (t = 0)*. If the patterns match the data acquisition is triggered.

118: Observer trigger compare bit mask

Determines which bits of the input vector shall be compared with that of the compare vectors. Valid for both trigger compare vectors (registers 116 and 117). For triggering, the patterns must match for all bits whose bit mask is set to 1.

119: Observer capture

Starts the comparing process. Data is captured if the patterns defined by the three previous registers match.

120: Observer cancel

Stops the comparing process.

3 FPGA Observer

FPGA Observer is an expert GUI implemented in C++ using the GTK 3 toolkit. It can be used for monitoring and controlling any gateware that incorporates the common monitoring and control features documented in chapter 2.

It can run on any Linux computer and also on a FEC itself if the GTK 3 toolkit and the corresponding C++ bindings are installed.

Three device access options are supported:

- TCP (requires the minimalistic server process *FPGA TCP server* to run on the FEC)
- AXI on Zynq based devices (Red Pitaya)
- PCIe if running on the FEC

3.1 Installation

3.1.1 Build from source

The sources and a Makefile can be found in `software/fpga_observer/`.

The following Makefile targets are usefull:

- *all*: builds the GUI and the *FPGA TCP server*
- *install*: install on the local computer
- *rpm*: creates a RPM package

3.1.2 RPM

You can find RPM packages (currently only for Fedora 36 and CentOS 7) here:

https://git.gsi.de/BEA_HDL/FPGA_Common/-/wikis/home

They can be installed either via the graphical installer or via: `rpm -i <package name>`

3.1.3 FPGA TCP server

If the GUI is not running on the FEC itself, `fpga-tcp-server` has to be running on the FEC.

This minimalistic server process listens on TCP port 35187 for connections and provides simple forwarding of data to and from the FPGA via either PCIe or AXI.

Setup examples for auto start can be found here: `software/fpga_observer/utils/FEC_autostart*`

3.1.4 PCIe driver

Gatewares using PCIe require the installation of the Xilinx XDMA driver on the FEC.

Install the PCIe driver:

```
cd software/fpga_observer/utils/pcie_driver
sudo ./install.sh
```

For the PCIe driver to work, the bitstreams of the FPGAs have to be loaded before booting the FEC. For MicroTCA systems, this can be achieved by the MCH's PCIe settings. For PCs, you might have to change the BIOS settings for the driver to work.

3.2 Usage

The GUI can either be started via its startmenu item or via the shell: `fpga-observer`

In the upper left corner you find a combo box named *FEC*. You can manage the list of predefined FECs via the menu button -> *manage FECs*. Provide the URL of the FEC, or in case the GUI is running on the FEC itself: `localhost`

If everything was set up correctly, the combo box named *FPGA* should contain at least one item and a number of tabs should appear:

3.2.1 Register tab

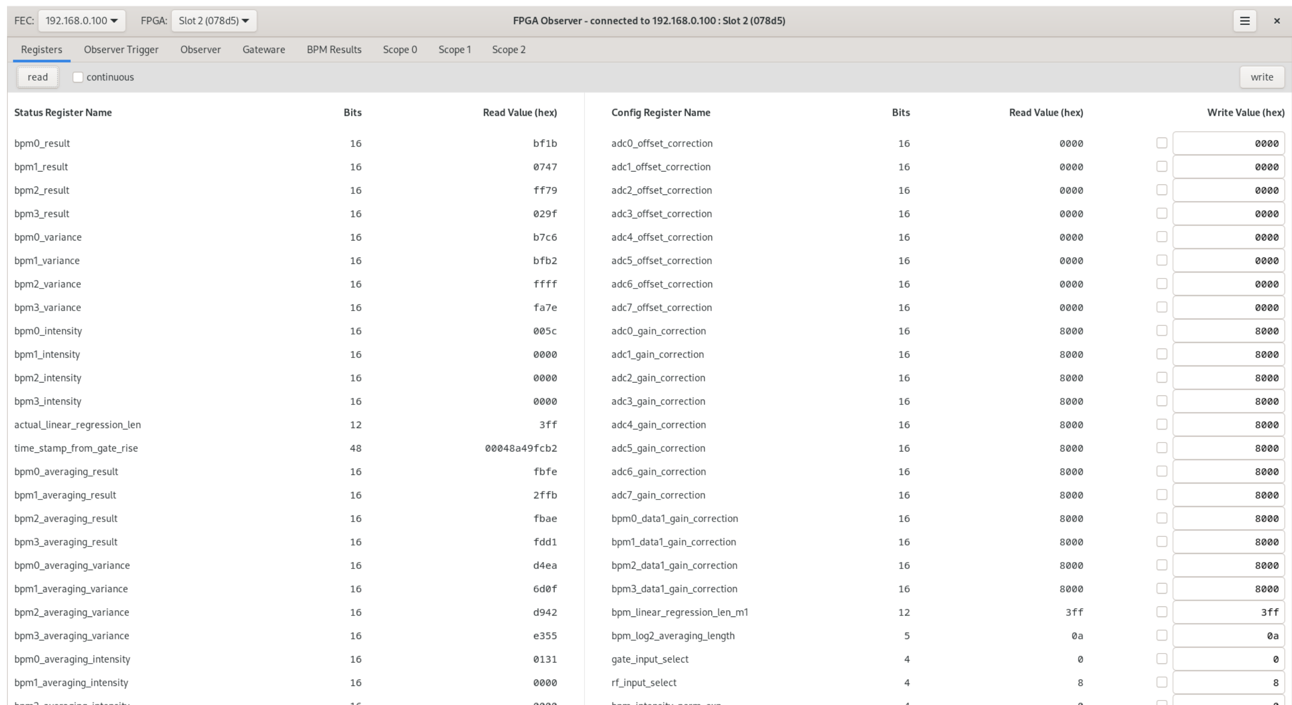

Figure 3.1: FPGA Observer - Register tab

The names and widths of the registers are read from an information memory region in the FPGA (see chapter 2.2.1). The status registers are displayed on the left and the configuration registers on the right.

The *read* button reads all the status registers either once or continuously if the *continuous* check button is checked. The *write* button writes all the configuration registers whose check buttons next to the write value are checked.

3.2.2 Observer Trigger tab

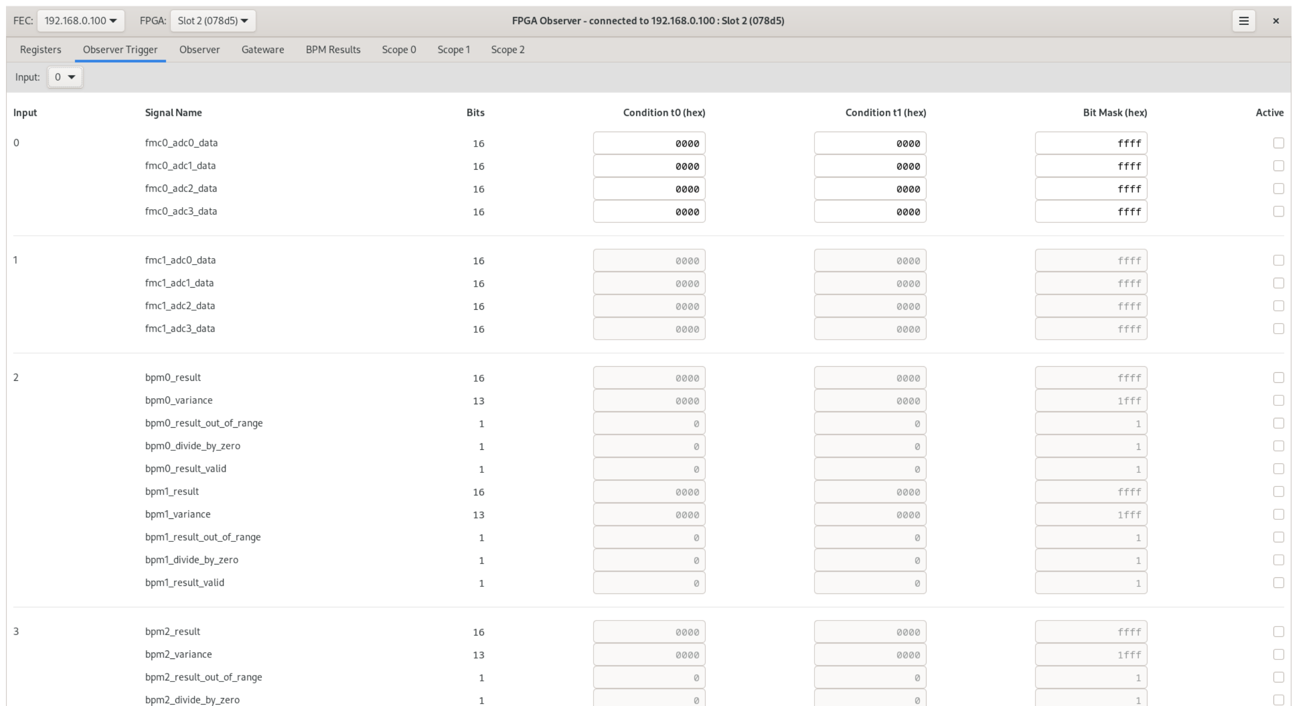

Figure 3.2: FPGA Observer - Observer Trigger tab

In this tab, the two stage trigger for the data displayed in the *Observer* tab can be configured (see chapter 2.3).

The trigger conditions for t = 0 and t = 1 contain the compare vectors of the two stage trigger which have to match in consecutive clock cycles.
The *Trigger Mask* defines on which bits of a signal the trigger will listen and the corresponding *Active* check buttons have to be checked for the trigger to become active.

All of the enabled conditions have to become true for a trigger event.

If nothing is configured, the Observer triggers at once.

3.2.3 Observer tab

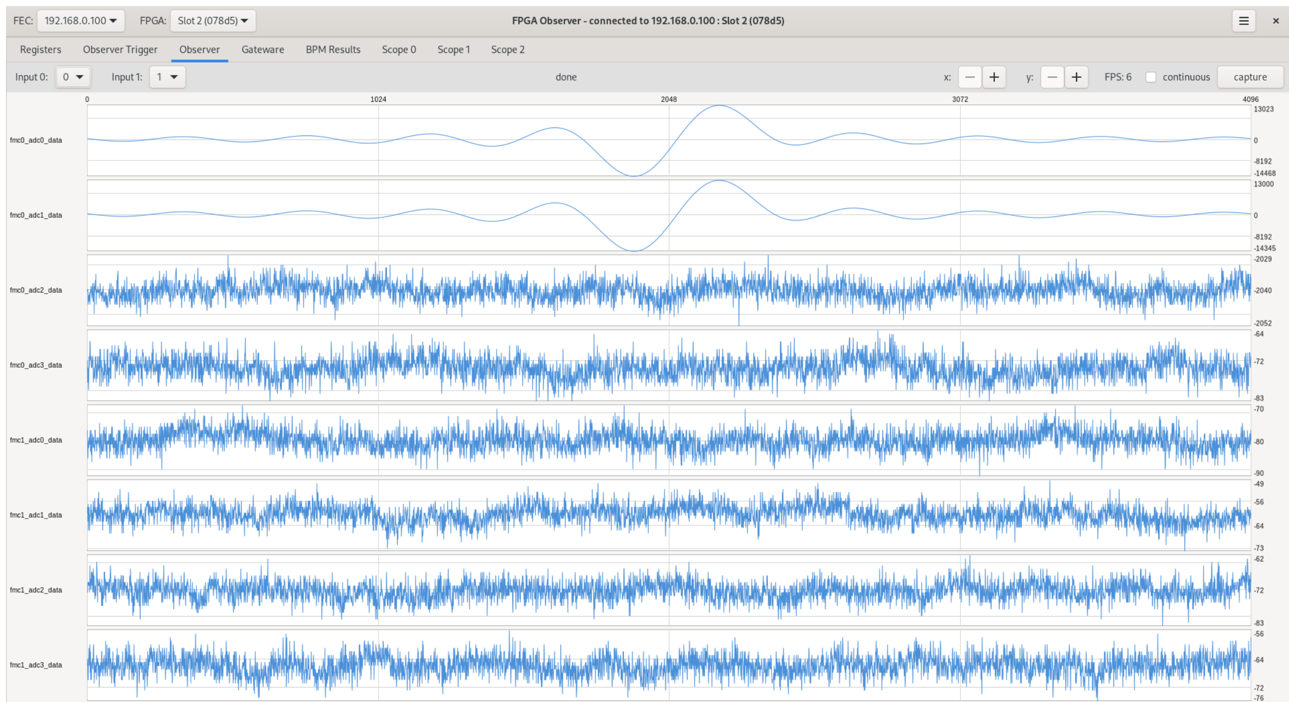

Figure 3.3: FPGA Observer - Observer tab

Data sampled using the gateware's *Observer* feature (see chapter 2.3) is displayed here. The list of available signals is project specific.

The sampling can be controlled by the *Observer Trigger* tab.

3.2.4 Gateware tab

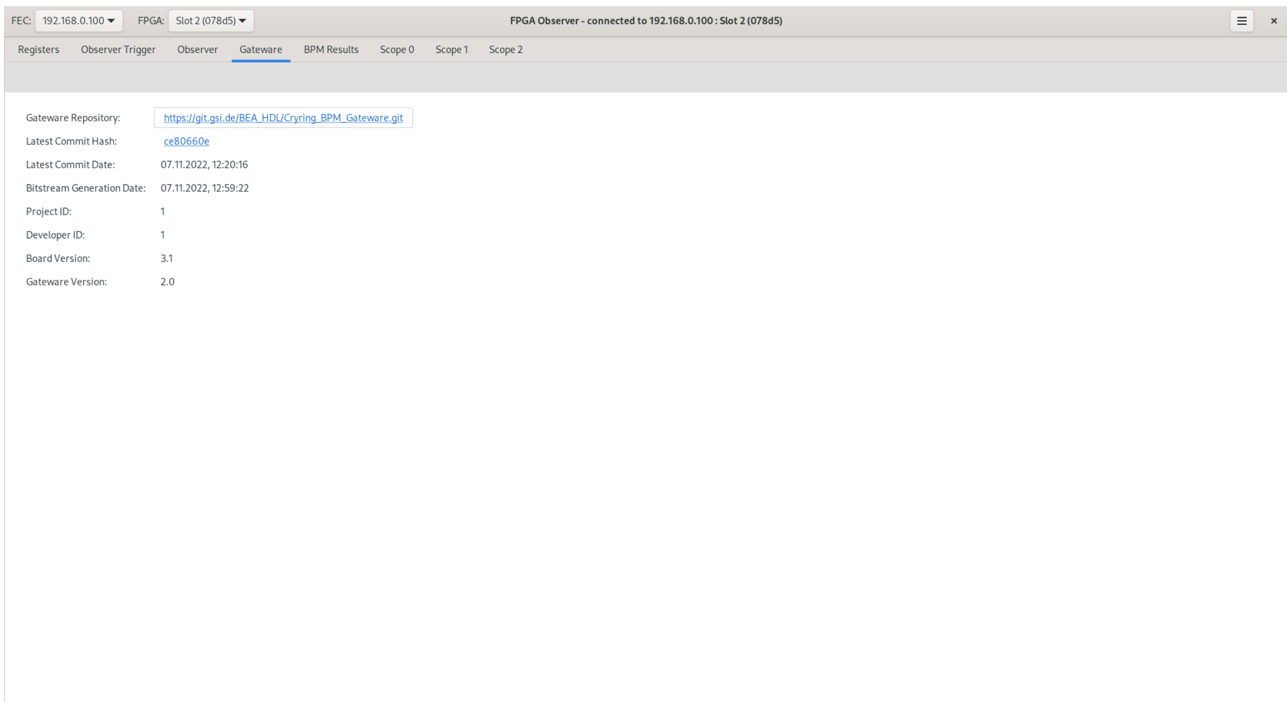

Figure 3.4: FPGA Observer - Gateware tab

The URL of the remote server of the Git repository, the latest commit hash and the latest commit date are read from an information memory region in the FPGA (see chapter 2.2.2) and are displayed in this tab.

The bitstream generation date is read from the status register `124` *build timestamp*.

3.2.5 Project specific tabs

Depending on the gateware, other project specific tabs like scope tabs or BPM results will appear.

4 Build flow and simulation

The build flow is designed and tested to be run on a Linux operation system.

The bitstream generation should also work on a Windows installation, but the depending Bash and Python scripts would have to be adapted for Windows. For example, the script `FPGA_Common/scripts/generate_monitoring_and_control.py` generates the VHDL file `FPGA_Common//src/vhdl/generated_mc_constant_package.vhd`, using the content of the configuration files in `config`.

4.1 Prerequisites

An installation of Xilinx Vivado is required. Currently the IP cores are built for version 2022.1 so that this version should be installed.

4.2 Build flow

4.2.1 GUI based build flow

- navigate to the root folder of the repository in a terminal
- type `FPGA_Common/run/create_project.sh`

This will open the Vivado GUI and set up a project, which can take some minutes. The project will be generated in the folder `output/project/project`.

4.2.2 Scripted build flow

For a completely automatic script based build flow without using the Vivado GUI proceed as follows:

- navigate to the root folder of the repository in a terminal
- type `FPGA_Common/run/run_build_flow.sh`

A project will be generated in the folder `output/build_flow/project`. The bitstream (if successful) will be generated in the subfolder `aft_top.runs/impl_1`.

4.3 Simulation

4.3.1 GUI based simulation

Prerequisite: an existing Vivado project see chapter 4.2.2.

Click on *Run Simulation -> Run Behavioral Simulation* in the Vivado GUI.

4.3.2 Scripted simulation

The scripted simulation checks that the simulation results match a predefined reference pattern.

- navigate to the root folder of the repository in a terminal
- type `FPGA_Common/run/run_simulation.sh <name of module (or leave empty for the toplevel simulation)>`
- you will find the output files of the simulation in the folder `output/simulation/<name of module>`

4.3.3 Peripherals simulation models

For gatewares that use the AFC's SDRAM, the toplevel simulation includes a Verilog simulation model from Micron, the manufacturer of the SDRAM.

The SDRAM interface IP needs an initial calibration process which finishes after about 120 us. If the communication to the SDRAM is of interest the simulation time should be chosen to be longer than that.

5 Helper scripts

5.1 PCIe access test script

There is a PCIe access test script fpga_observer/utils/pcie_driver/test_pcie_access.sh. It uses the tools provided together with the XDMA PCIe driver to test some basic reading and writing to different memories via the PCIe driver. The reading results are displayed via hexdump.

5.2 VHDL beautification

There is a script FPGA_Common/scripts/beautify_vhdl.py for autoformatting VHDL files using the open source software *Emacs*.

The script expects one parameter: <file that shall be formatted>, or all for formatting all VHDL files in the repository. The formatting is performed in place, overwriting the original source file.

The script applies several corrections and changes to the *Emacs* formatting result:

- correction of the handling of the comparison operator <=
- correction of the handling of initializations like (others => '0')
- enforcing of spaces around the operators +, -, *, /, &
- no indentation for closing brackets
- aligning of full comment lines to the indentation level of the following VHDL command
- indentation with tabs instead of spaces

5.3 Remote power cycling of a MicroTCA crate

The script FPGA_Common/scripts/powercycle_mtca_crate.py instructs an MCH via IPMI to power cycle the corresponding MicroTCA crate. The script expects one parameter: <name of MCH, e.g. sdmch021>.

5.4 Generation of a VHDL file for monitoring and control

The monitoring and control configuration of the gateware is defined by the configuration files status_registers.csv, config_registers.csv and observer_signals.csv in the folder config.

The script FPGA_Common/scripts/generate_monitoring_and_control.py is used to convert the configuration to a VHDL file stored as FPGA_Common/src/vhdl/generated_mc_constant_package.vhd, which contains the width of each register, the configuration register default values and a Block RAM initialization vector containing the architecture information.

The script is also executed by the gateware build flow documented in chapter 4.

5.5 Generation of documentation

5.5.1 PDF

The script FPGA_Common/doc/scripts/create_pdf.sh generating a PDF file from the content of Readme.md.

It uses the script `FPGA_Common/doc/scripts/create_tex.sh` to convert the Markdown syntax to LateX format first and afterwards calls the open source software *Pdflatex* twice to enable the generation of references inside the PDF file.

5.5.2 DokuWiki

The script `FPGA_Common/doc/scripts/create_dokuwiki.py` generates a DokuWiki file which can be used to populate a Wiki page on e.g. `https://www-bd.gsi.de/dokuwiki`.

The script converts the Markdown documentation to the DokuWiki format in three steps:

1. preprocessing of Markdown before the conversion to DokuWiki
2. calling *Pandoc* to convert Markdown to DokuWiki
3. postprocessing for correction, extension of functionality and a different style

The preprocessing actions are:

- removing the table of contents since it is automatically generated by DokuWiki

The postprocessing actions are:

- conversion or equations to images since DokuWiki can not render equations
- replacement of HTML tags, Latex color tags, etc. since DokuWiki can not handle them
- conversion of the format of references

6 Continuous integration environment

There is a continuous integration setup which is implemented as a so called Gitlab Runner that communicates with the remote of the Git repository, the Gitlab server *git.gsi.de*.

At the moment the Gitlab Runner is running on the Linux server *sdlx035* located in a server room in the basement.

The benefits of continuous integration are:

- every change will be tested automatically
- it is ensured that no files are missing in the repository
- build results like e.g. bitstreams are automatically generated and can be archived

6.1 Installation

There is an installation script `install.sh` in the *Gitlab_Runner_Setup_Centos_7* Git repository. It installs the Gitlab Runner as well as the software needed for simulation, generation of documentation and building an FPGA.

After the installation, the newly setup Gitlab Runner has to configured to connect to a remote repository on a Gitlab server. In the repository's web front end on the Gitlab server, go to *Settings CI/CD Runners* and copy the registration token which you will need in the following step.

On the newly installed Gitlab Runner server, open a terminal and type `sudo gitlab-runner register`.

Enter the following information:

- gitlab-ci coordinator URL: e.g. `https://git.gsi.de`
- gitlab-ci token: enter the registration token copied before
- gitlab-ci description: name of the server, e.g. `sdlx035`
- gitlab-ci tags: leave empty
- executor: `shell`

You can add multiple repositories with different tokens by running `sudo gitlab-runner register` multiple times.

6.2 Pipeline Stages

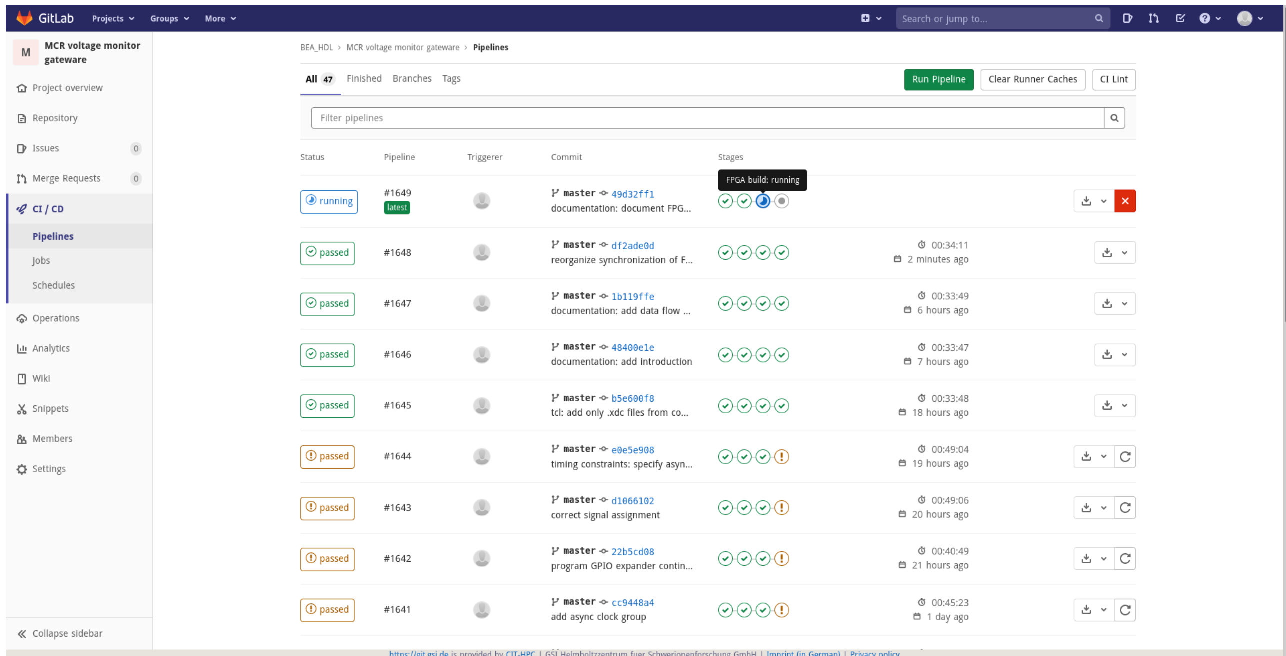

Figure 6.1: Gitlab: continuous integration pipelines

Each push to the Gitlab server will trigger a so called continuous integration / continuous delivery (CI/CD) pipeline. The pipeline setup is defined by the file `gitlab-ci.yml` in the root folder of the repository.

Typical pipeline stages are at least:

- documentation
- simulation
- FPGA build
- timing check

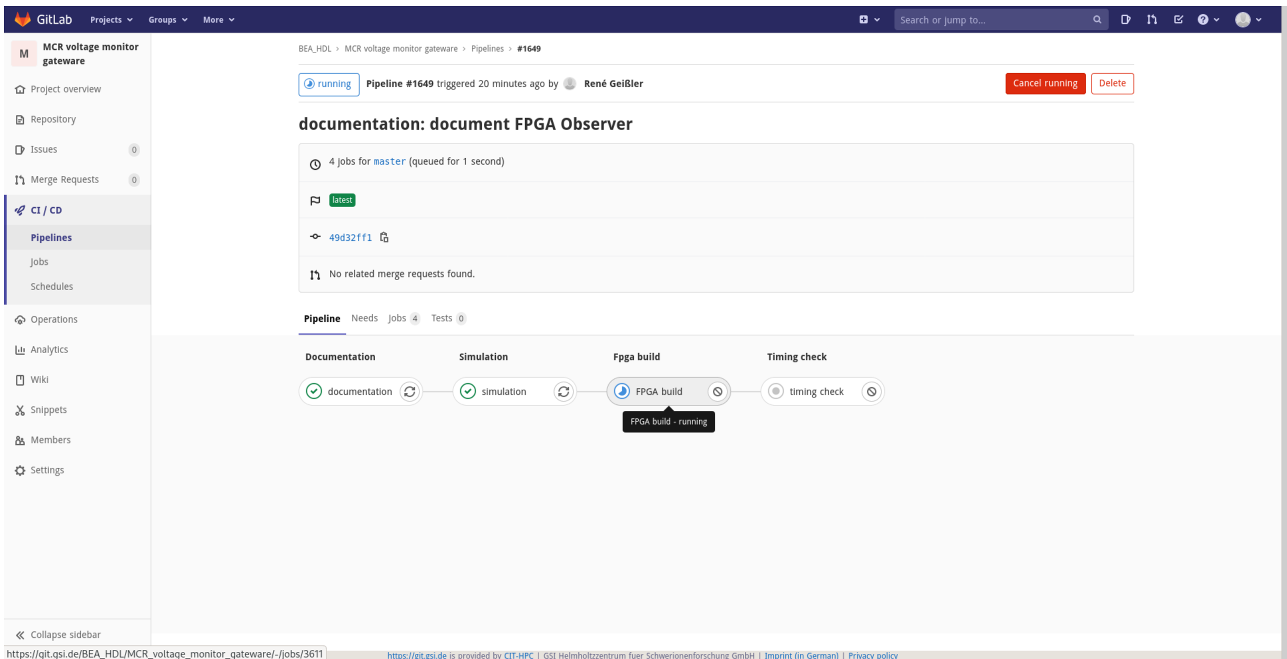

Figure 6.2: Gitlab: Pipeline stages

6.2.1 Documentation

The script `FPGA_Common/doc/scripts/create_pdf.sh` is run to generate this documentation from the Markdown file `README.md`. This pipeline stage succeeds if Pdflatex can generate the PDF without errors.

The log file of Pdflatex and - if successful - the PDF of the documentation are archived.

6.2.2 Simulation

The script `FPGA_Common/run/run_simulation.sh` is run which uses the Vivado command line interface to simulate the top level of the gateware. This pipeline stage succeeds if there is no error in simulation and if the output file matches the reference pattern.

The log file of the simulation and - if successful - a file with the output from the simulation are archived.

6.2.3 FPGA build

The script `FPGA_Common/run/run_build_flow.sh` is run which uses the Vivado command line interface to build the gateware. This pipeline stage succeeds if there is no error during the build process and if a bitstream file has been generated.

Different log files from synthesis and implementation, different reports like utilization and timing reports and - if successful - the bitstream file are archived.

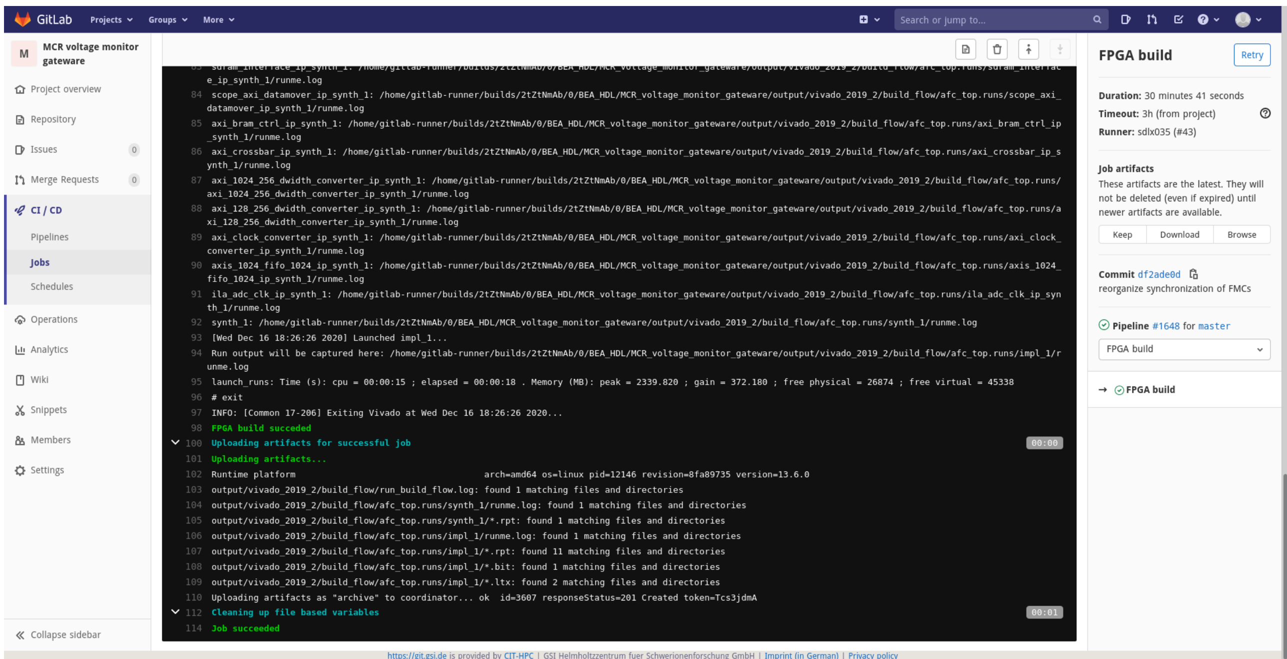

Figure 6.3: Gitlab: Pipeline progress console

6.2.4 Timing check

The script `FPGA_Common/scripts/check_fpga_timing.sh` is run which analyzes the timing summary report generated in the previous pipeline step. The script analyzes if any timing constraints were not met during the FPGA build process.

Since timing failures do not necessarily result in system malfunctions, this pipeline step is allowed to fail, but a warning will be displayed in the case of a failure.

6.3 Build results

For each of the pipeline stages the archiving of build results can be configured for an adjustable time period.
If the period has passed and the build results have been deleted, they can be generated again by restarting the pipeline.

The build results can be downloaded from the Gitlab web front end where they are called *job artifacts* (see figure 6.3).

> The CI/CD pipelines can also be used to generate FPGA bitstreams without having to set up a build environment.

6.4 Settings

You can define individual settings for the CI/CD section of each Git repository in the Gitlab web front end. The following settings should fit for most cases:

- Use git clone to get the recent application code, otherwise the pipelines might fail during git fetch: *Settings CI/CD General pipelines Git strategy for pipelines: git clone*
- Increase the timeout to allow FPGA build to finish in any case: *Settings CI/CD General pipelines Timeout: 6h*

7 Programming and hardware configuration

7.1 Programming the gateware to the FPGA on an AFC board

7.1.1 AFC version 3.1 and earlier

There is a JTAG switch on the AFC board that has to be programmed before being able to access the FPGA:

Using a JTAG programmer

Open the Vivado Hardware Manager software:

Tools -> *Run Tcl Script*: `FPGA_Common/scripts/program_scansta_jtag_switch.tcl`

You should now see a *xc7a200t_0* device. Right click on it and choose *Program Device*.

Choose the correct bitstream (.bit file) and press *OK*.

Using a JTAG Switch Module

If there is a JTAG Switch Module (JSM) in the MicroTCA crate, the bitstream can also be programmed remotely via a so called Xilinx Virtual Cable:

- upload `FPGA_Common/scripts/afc_scansta.nsfv` in *MCH GUI* -> *JSM* to the port of the JSM to which the AFC board you want to program is connected
- open Vivado Hardware Manager
- *Open Target* -> *New Target* -> *Next* -> *Local Server* -> *Add Xilinx Virtual Cable (XVC)*
- *Hostname*: `sdmch<xxx>.acc.gsi.de`
- Port: find correct port number in *MCH GUI* -> *JSM*
- *Finish*
- *Open target*
- you should see the FPGA now in Vivado Hardware Manager and can program it

7.1.2 AFC version 4

In version 4, the JTAG switch on the board is connected to the FPGA as a default, so it does not have to be programmed to access the FPGA.

The remaining steps to are identical to that of AFC version 3.1 and earlier.

7.1.3 Storing a bitstream persistently in the SPI Flash

There is a SPI Flash memory on the AFC board for persistent bitstream storage.
On AFC version 2.0 it has a size of 16 MiBytes. On the later versions its size was increased to 32 MiBytes.

File format conversion

First you have to convert the bitstream (.bit) file to a .mcs file using the script:
`FPGA_Common/scripts/convert_bit_to_mcs.sh <path to .bit file>`

The .mcs file will be generated in the same folder as the .bit file.

Programming

For AFC versions 2.0 and 3.1, program the JTAG switch on the AFC board as described in chapter 7.1.1. For later versions, you can skip this step.

You should now see a *xc7a200t_0* device.
Right click on it and choose *Add Configuration Memory Device* and choose *mt25ql256-spi-x1_x2_x4* (AFCv3.1) or *mt25ql128-spi-x1_x2_x4* (AFCv2.0)

You should now see a *mt25ql256-spi-x1_x2_x4* / *mt25ql128-spi-x1_x2_x4* device.

Right click on it and choose *Program Configuration Memory Device*.
Choose the .mcs file you created before and press *OK*.

7.1.4 JSM JTAG device numbering on PowerBrige 6 slot crate

JTAG Device

left slots	right slots
- (FEC)	AMC5
AMC1	AMC2 (MCH)
AMC3	
?	?

7.2 Configuration of the MCH

7.2.1 Via the MCH's web interface

Base configuration

MCH global parameter SSH access: enabled
This will trigger SSH key generation which takes some minutes to complete.

PCIe parameter Upstream slot power up delay: 25 sec
Delay before the FEC will power up on start up. For making sure that the bitstreams are loaded to the AMC's FPGAs from Flash memory before the FEC boots you might have to increase this value.

PCIe parameter PCIe hot plug delay for AMCs: 5 sec
Delay before the AMC boards will power up on start up.

Switch PCIe x80

Set the FEC as upstream AMC source in 'Virtual Switch 0':

PCIe Virtual Switches Upstream AMC: AMC1/4..7 (for FEC in AMC slot 1)

Make sure you enable *PCIe downstream '4..7'* for the AMC slots which contain your AFC boards of version 3.1 or later. For AFC version 2.0 boards, due to a different connection of the four FPGA's PCIe lanes to the eight AMC's PCIe lanes you need to enable *PCIe downstream '8..11'*.

7.2.2 Via USB

The most comfortable way of configuring the MCH is via its web interface. If you have accidentally disabled the webserver, set an invalid IP or DHCP configuration or reset the MCH settings to default, you can access the MCH via an USB connection to the micro USB port on the left side of the front panel.

On a Linux PC, connect a micro USB cable and check via `dmesg` that a *LUFA USB-RS232 Adapter* has been detected. The driver will be accessible at /dev/ttyACM<some number>, use e.g. Putty to connect to this serial port using the parameter speed = 19200.

Now typing `mch` will output information about the MCH. Typing `?` will display a list of available commands. Most of the settings of the web interface are also available on the command line interface. You can for example set the IP address or a DHCP name to be able to connect to the web interface.

7.3 Controlling a MicroTCA crate via the MCH

Here are some useful commands:

7.3.1 SSH

Connect to the MCH via ssh root@<URL of MCH>. The default password is 'nat'.

`show_fru` will show you the FRU numbers of the AMC slots.

shutdown <FRU number> will request an AMC board to power down.

fru_start <FRU number> will request an AMC board to power up.

7.3.2 IPMI

A powercycle of a MicroTCA crate can be performed using the following command:

```
ipmitool -H <URL of MCH> -A none chassis power reset
```

7.4 Enabling network boot on the FEC

7.4.1 BIOS settings

Shortly after powering the FEC, press F2 to enter the BIOS.

In the *Main* tab, go to *Boot Features* and select the following (using F6 for enabling and F5 for disabling):

- PXE BOOT: <Enabled>
- Front ETH0: <Enabled> or Front Panel ETH1: <Enabled>, depending on the version of the FEC and the BIOS
- Auto Retry PXE Boot: Enabled. The existence of this menu entry depends on the BIOS version.

In the *Advanced* tab, go to *Network Stack Configuration* and enable *Ipv4 PXE Support*. The availability of this menu entry also depends on the BIOS version.

In the *Boot* tab, go to *Legacy* and *Boot Type Order*. There should be an *Others* entry that has to be shifted to the top of the list using F6. In newer BIOS versions, there is a *Boot Option #1* entry in the *Boot* tab, which has to be chosen to *IBA GE Slot 1600 v1513*.

Save the settings by pressing F4.

The FEC should boot from network after the next reboot. For the loading of the correct network image, the MAC address of the desired Ethernet port of the FEC has to registered in the DHCP server responsible for distributing the locations from which to load the network images.

7.4.2 Defining the boot image

The boot image that will be loaded by the FEC can be defined by settings accessible from e.g. asl740:

- ssh <username>@asl740
- cd /common/tftp/lobi/pxe/pxelinux.cfg
- cat info.txt will display a configuration example

In the images with the naming syntax R<Rocky version>_R<major version>_<minor version>_UTCA, no ethernet device is defined for retrieving the DHCP settings, which leads to maximum flexibility but also to a very long boot time, since all the ethernet devices of the CPU will poll for DHCP settings.

To speed up the boot process, the ethernet device of the front panel ETH1 connector should be defined. Unfortunately, Linux numbers the ethernet devices dynamically and different PCIe switch settings in the MCH will lead to different numberings.

To enable booting only from e.g. enp15s0 on e.g. sddsc085:

- cp R<Rocky version>_R<major version>_<minor version>_UTCA R<Rocky version>_R<major version>_<minor version>_UTCA.enp15s0
- change ip=dhcp to ip=enp15s0:dhcp
- rm sddsc085
- ln -s R<Rocky version>_R<major version>_<minor version>_UTCA.enp15s0 sddsc085

Keep in mind that this configuration is only valid for one special PCIe switch setting. The boot process will fail if the PCIe switch settings are changed.

7.5 MMC firmware

The NXP LPC1764FBD100 microcontroller on the AFC board is responsible for:

- implementing the AMC protocol:
  - power control
  - provision of sensor data
  - driving the AMC standard LEDs
  - upstart procedure
- configuration of the onboard devices:
  - PLL configuration
  - clock switch configuration
- control of various pins on the FPGA and the FMC connectors:
  - FPGA reset pin
  - FMC power good pins

7.5.1 Differences in MMC firmwares

The AFC version 2.0 boards use the MMC firmware of Creotech. All newer boards use the openMMC firmware from LNLS.

Creotech's MMC firmware routes a 125 MHz clock to the PCIe reference clock input, whereas the openMMC firmware routes a 100 MHz clock to this pin.
The frequency of *sys_clk* is 125 MHz for both.

All of the current gatewares for AFC board version 3.1 and newer are only functional with the openMMC firmware. For running it together with Creotech's MMC firmware, the PCIe reference clock frequency setting in the Xilinx PCIe IP core has to be changed to 125 MHz.

7.5.2 Building the openMMC firmware

Getting the sources

The original source code of LNLS's openMMC code can be found here: `https://github.com/lnls-dig/openMMC`

A modified version suitable for all projects and with some adaptions needed for the UniMon_Gateware and the BLoFELD_-Gateware can be found here: `https://git.gsi.de/BEA_HDL/openMMC`

Installation of required software

An ARM cross compiler is needed to compile the code on a x86-64 architecture.

Centos

A prebuild GCC for ARM can be downloaded from e.g. `https://launchpad.net/gcc-arm-embedded/+download`.

Install the downloaded TAR archive:

```
cd /usr/local/bin && tar xjf <path to downloaded archive>.
```

Fedora 36

```
sudo dnf install arm-none-eabi-gcc-cs arm-none-eabi-binutils-cs.x86_64 arm-none-eabi-newlib.noarch
cmake
```

Building for AFC version 3.1

- Leave the repository's folder openMMC, `mkdir openMMC_output && cd openMMC_output`
- `cmake ../openMMC -DBOARD=afc-bpm -DVERSION=3.1`
- `make`

The firmware files will be created in the `openMMC_output/out` directory. There will be two different output files:

- `bootloader.bin`, intended to be loaded to the base address 0x0 in the microcontroller's flash memory
- `openMMC.bin`, intended to be loaded to the base address 0x2000 in the microcontroller's flash memory

Building for AFC version 4.0

- Leave the repository's folder openMMC, `mkdir openMMC_output && cd openMMC_output`
- `cmake ../openMMC -DBOARD=afc-v4`
- `make`

The output file is `openMMC_output/out/openMMC.bin`

In contrast to AFCv3.1, the output file `openMMC_output/out/bootloader.bin` is not needed with AFCv4. Just program `openMMC_output/out/openMMC.bin` to address `0x0`.

7.5.3 Programming the MMC firmware on AFC version 3.1

For programming the MMC firmware into the LPC microcontroller you need to install a proprietary software from NXP called LPCxpresso.

Installation of LPCxpresso on Linux

Download LPCxpresso from the NXP website [1]. You need to register for the download. Follow the instructions in `INSTALL.txt`.

On Fedora 36, the following works:

```
sudo dnf install gtk2.i686 glibc.i686 glibc-devel.i686 libstdc++.i686 zlib-devel.i686 ncurses-devel.i686 libX11-devel.i686 libXrender.i686 libXrandr.i686 libusb.i686 libXtst.i686 nss.i686
```

You have to run the IDE with a path variable `SWT_GTK3` set to zero: `SWT_GTK3=0 /usr/local/lpcxpresso_8.2.0/lpcxpresso/lpcxpresso`

You have to register the installation via *Help -> Activate -> Create serial number and register (Free Edition)*. Create a serial number in the dialog, copy it to the form in the website and afterwards paste the activation key you got from the website to *Help -> Activate -> Activate (Free Edition)*.

Programming

Disconnect the AFC board completely. The power for programming the microcontroller will come from the LPC-Link programmer. Connect and power the LPC-Link programmer via USB and connect the customized cable to the *CPU-JTAG* connector on the AFC board. Connect the plug so that the flat cable is pointing in the direction of the FMC connector.

Program the device via:

```
lpcxpresso/bin/dfu-util -d 0x0471:0xdf55 -c 0 -t 2048 -R -D lpcxpresso/bin/LPCXpressoWIN.enc
```

```
sudo lpcxpresso/bin/crt_emu_cm3_nxp -pLPC1768 -g -wire=winusb -load-base=0 -flash-load-exec=<path to firmware binary>
```

Or, in the case of two split binaries:

```
sudo lpcxpresso/bin/crt_emu_cm3_nxp -pLPC1768 -g -wire=winusb -load-base=0 -flash-load-exec=<path
to bootloader binary>

sudo lpcxpresso/bin/crt_emu_cm3_nxp -pLPC1768 -g -wire=winusb -load-base=0x2000 -flash-load-exec=<path
to openMMC binary>
```

7.5.4 Programming the MMC firmware on AFC version 4

On AFC version 4 boards the MMC firmware can be programmed using a micro USB cable and the open source software mxli.

Follow the instructions in the README of: https://git.gsi.de/BEA_HDL/mxli

7.6 Configuration of an FTRN

The GPIOs of an FTRN (FAIR Timing Receiver Node) can be controlled via command line commands available in the network boot image of the FEC:

- display all available GPIOs: `saft-io-ctl tr0 -i`
- enable the outputs to the backplane: `saft-io-ctl tr0 -n V_MTCA4B_EN -q 1`
- display the properties of a single GPIO: e.g. `saft-io-ctl tr0 -n MTCA4_IO1`
- enable the output of a single GPIO: e.g. `saft-io-ctl tr0 -n MTCA4_IO1 -o 1`
- drive the output of a single GPIO high: e.g. `saft-io-ctl tr0 -n MTCA4_IO1 -d 1`

The timing receiver is connected to the AFC boards via eight differential MLVDS lines via the backplane of the MicroTCA crate.

8 Hardware properties

8.1 LEDs on the AFC front panels driven by the MMC

- In Service (*L1*), green
- Alarm (*L2*), red
- Hot Swap (*HS*), blue

8.1.1 Lighting patterns of the Hot Swap LED

Insertion of an AFC board:

event	Hot Swap Handle	Hot Swap LED
AMC inserted into chassis with handle open	Open	On
AMC handle closed	Closed	Blinks
Activation granted and AMC powers up	Closed	Off

Source: [2]

Removal of an AFC board:

event	Hot Swap Handle	Hot Swap LED
AMC handle pulled open	Open	Blinks
Deactivation granted and AMC powers down (AMC can now be removed)	Open	On

Source: [2]

8.2 MCH PCIe status LEDs

The lighting patterns of the PCIe status LEDs on the MCH show the link status and the link speed of the PCIe connections:

LED state	meaning
off	no PCIe link
1 blink/sec	2.5 GBaud
2 blinks/sec	5 GBaud
on	8 GBaud

Source: [3]

8.3 Differences between AFC versions

In AFC version 2.0 the FPGA's four PCIe lanes are connected to the upper four of the eight AMC PCIe lanes, whereas they are connected to the lower four lanes in AFC version 3.1 and later.
This also affects the necessary settings of the MCH, see chapter 7.2.

Version 2.0 carries a Micron MT25QL128 128 MiBits Flash memory as a bitstream storage.
Version 3.1 and later use a Micron MT25QL256 256 MiBits Flash memory.

Both boards carry 2 GiBytes of DDR3-SDRAM, divided in four modules of 512 MiBytes each. The SDRAM model can be determined via the FBGA code printed on the modules using the Micron part decoder webpage [4].

There are major differences in the conntections of FPGA pins to the FMC connectors between version 2.0 and version 3.1. Version 4 brings only minimal changes.

Version 4 brings two major differences:

- the JTAG switch does not have to be programmed any more to access the FPGA
- the board's I2C infrastructure has been redesigned

8.3.1 AFC version 2.0

- FBGA code: D9PBC, translates to Micron MT41J512M8RA-125:D
- operates at 1.5 V

8.3.2 AFC version 3.1

- FBGA code: D9QBV, translates to Micron MT41K512M8RH-125 IT:E
- compatible to older MT41J family, operates at 1.5 V or 1.35 V

8.4 Maximum achievable data rate to and from SDRAM

The gross data rate of the SDRAM interface is 800 MT/s with 32 bits/transfer, resulting in a theoretical gross data rate of 3.2 GiBytes/s. The maximum achievable data rate is limited by concurrent read and write accesses and by SDRAM refresh cycles.

The storage of the samples of all eight ADCs in parallel at a sampling rate of 125 MHz results in a write data rate of:

$125 \text{ Mtransfer/s} \cdot 8 \cdot 16 \text{ bits/transfer} = 2 \text{ GBytes/s} = 1.863 \text{ GiBytes/s}$

The SDRAM capacity of 2 GiBytes would be sufficient to store the stream data of all eight ADCs for 1.07 seconds.

9 FPGA board inventory list

9.1 List of AFC boards

9.1.1 AFC v2.0

In AFC v2.0, the FPGA is connected to the PCIe lanes *8 .. 11* instead of the lanes *4 .. 7* in the newer versions.

The AFC v2.0 boards use the MMC firmware from Creotech, so that the frequency of the PCIe reference clock is 125 MHz instead of the 100 MHz of the newer versions which are using the OpenMMC firmware.

FPGA serial number	location	project	FEC	MCH	AMC slot number	comments
`0x00408c8522c3004`	ELR	Rate Divider	sddsc079	sdmch019	2	SDRAM broken, FMC connector
`0x06408c8522c300c`	ELR	Rate Divider	sddsc080	sdmch018	2	

9.1.2 AFC v3.1

AFC serial number	FPGA serial number	location	project	FEC	MCH	AMC slot number
`001011`	`0x048A82110D1B05C`	Cryring container	Cryring BPM	sddsc085	sdmch026	3
`004069`	`0x008182110D1B05C`	Cryring container	Cryring BPM	sddsc085	sdmch026	4
`013227`	`0x010A82110D1B05C`	Cryring container	Cryring BPM	sddsc085	sdmch026	5
`032211`	`0x068B48160E47054`	Cryring container	Cryring BPM	sddsc085	sdmch026	6
`035233`	`0x054B48160E47054`	Cryring FCT	Rate Divider	TBD	TBD	TBD
`111154`	`0x004ACC24235885C`	ask Harald / Rene	LNLS RT DAQ	?	?	?
`191087`	`0x004D5C242358854`	ask René	-			
`240030`	`0x078D5C24235885C`	ask René	-			
`256118`	?	ask René	-			
`260046`	`0x018D5C24235885C`	Cryring container	Cryring BPM	sddsc085	sdmch026	7
`261056`	`0x068D5C24235885C`	ask René				
`290148`	`0x058D5C24235885C`	ELR	LNLS RT DAQ	sddsc045	sdmch013	2

9.1.3 AFC v4.0

AFC serial number	FPGA serial number	location	project	FEC	MCH	AMC slot number	comments
?	?	?	LNLS RT DAQ	?	?	?	
?	?	?	LNLS RT DAQ	?	?	?	

References

[1] NXP: LPCxpresso download web page, https://www.nxp.com/design/microcontrollers-developer-resources/lpc-microcontroller-utilities/lpcxpresso-ide-v8-2-2:LPCXPRESSO

[2] NXP: AMC documentation, https://www.nxp.com/docs/en/reference-manual/MSC8156AMCUM.pdf

[3] NAT GmbH: MCH technical reference manual, https://www.nateurope.com/manuals/nat_mch_pciex48_v2x_man_hw.pdf

[4] Micron: FBGA and Component Marking Decoder, https://www.micron.com/support/tools-and-utilities/fbga