UniMon Gateware Documentation

Version 0.4

August 21, 2022

René Geißler
r.geissler@gsi.de

GSI Helmholtzzentrum für Schwerionenforschung GmbH

Contents

List of non-common abbreviations

CI/CD	=	**C**ontinuous **I**ntegration / **C**ontinuous **D**elivery, a design strategy which includes automated tests
MMCM	=	**M**ixed **M**ode **C**lock **M**anager, a PLL with extended functionality inside a FPGA

Resources

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

Installation scripts to set up a Gitlab runner for continuous integration including all necessary software to build the gateware can be found in a Git repository whose upstream is: https://git.gsi.de/BEA_HDL/Gitlab_Runner_Setup_Centos_7. The relevant branch is *master*.

1 Introduction

This document describes the gateware (= FPGA firmware) implementation of the UNIMON Main Control Room Voltage Monitor.

The gateware is intended to be used together with a hardware upgrade of the existing system. The current design is implemented for the OpenHardware AFC v3.1 FMC carrier.

The system uses a total of 4 IOxOS ADC_3117 FMC cards. Each of them provides 20 ADCs with a maximum sampling rate of 5 MHz and a resolution of 16 bits.

There are two AMC FMC carriers which carry respectively two FMC cards. The gateware of each AMC card stores the data stream of respectively 40 ADCs to a ring buffer, from which it can be read via PCIe.

The sampling rate of the ADCs is adjustable between 1 kHz and 5 MHz. Due to the parallel storage of the data streams in a ring buffer, the sampling rates of the all the ADCs mounted on a single AMC card must be equal.

Each ADC input is individually configurable as differential (two wires) or as single ended relative to the FMC's ground (one wire).

2 Peripheral devices

The gateware communicates with the following devices on each of the IOxOS ADC_3117 FMC cards:

- 10 * Analog Devices LTC2323-16 dual ADC
- 10 * EXAR XRA1405 GPIO expander

2.1 Analog Devices LTC2323-16 ADC

Each LTC2323-16 consists of two ADCs that are sampled in parallel.

The sampling frequency is adjustable in the range between 1 kHz and 5 MHz.

Each ADC provides a differential input pair and is directly controlled by the FPGA without any external clock.

The FPGA has to read the samples at a clock speed of 105 MHz. The sampling rate of the ADCs is controlled by the distance between the so called conversion pulses that have to be generated by the FPGA [1].

Due to this driving logic of the ADCs, the sampling frequency can be controlled in the steps described in chapter 5.2.2.

2.2 EXAR XRA1405 GPIO expander

The GPIO expanders are used for multiple purposes. Among these are the control of the inputs to each ADC.

2.2.1 ADC input selection

There are dedicated switch devices in front of each of the differential ADC inputs, which can be controlled independently.

The following input settings are available:

- differential p / n input pin
- ground
- configurable calibration voltage (not implemented yet)
- fixed calibration voltage of + 4.128 V

2.2.2 Gain selection

The gain of each differential ADC input pair can be controlled by a dedicated fixed gain amplifier device which can be programmed to an amplifications of 1, 2, 5 or 10.

In front of each ADC input there is an attenuator with a fixed gain of 0.4.

In total the following gains are available:

amplifier gain	input range	resolution per LSB
1	+/- 10.24 V	312.5 μV
2	+/- 5.12 V	156.25 μV
5	+/- 2.048 V	62.5 μV
10	+/- 1.024 V	31.25 μV

3 Gateware implementation

3.1 Clocking

The gateware uses only one primary clock.

3.1.1 PCIe reference clock

The PCIe reference clock has a nominal frequency of 100 MHz and feeds the reference clock input of the PCIe IP core by Xilinx, which contains a PLL producing a 125 MHz output clock for the AXI interface named *clk_125_pcie_axi*.

clk_125_pcie_axi drives a MMCM to generate:

- *clk_100*, 100 MHz, this is the main processing clock of the design
- *clk_200*, 200 MHz, used for the SDRAM interface IP core by Xilinx
- *clk_50*, 50 MHz, used for generating a 210 MHz clock for the ADC interface

The SDRAM interface IP core contains a MMCM which generates a 100 MHz clock named *clk_100_sdram* for the AXI interface.

3.2 Resets

3.2.1 PLL not in lock

As long as the PLL in the MMCM producing the main processing clock *clk_100* is not yet in lock, the design is held in reset. After this the lock should be stable until the next power cycle.

3.2.2 Reset button

There is a push button labeled *RST* at the center of the AFC front panel which is connected to the microcontroller for the MMC firmware. The OpenMMC firmware forwards a button press with a duration of at least two seconds to the FPGA pin AG26 as an active low signal to initiate a reset of the gateware.

Resetting the FPGA leads to the loss of the PCIe connection. To re-enable the connection, the CPU unit has to be power cycled.

3.3 Data flow diagram

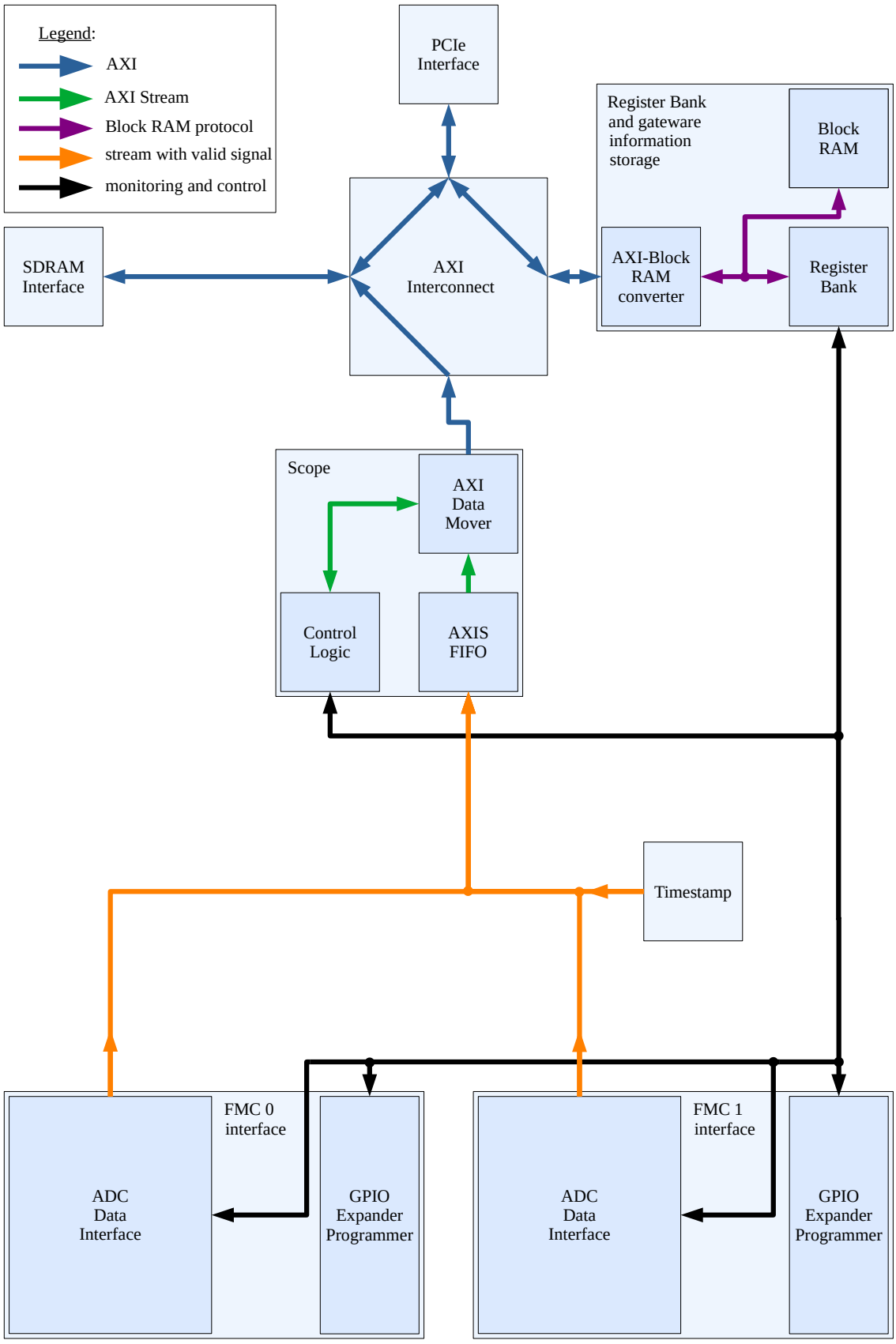

Figure 3.1: Simplified data flow diagram

Figure 3.1 shows a simplified data flow diagram. For simplicity, some features are not included in the diagram:

- processing clocks and clock domain crossings
- resets
- LEDs
- read out logics of FPGA serial number and build time stamp
- data width conversions of AXI connections

3.3.1 SPI Interface

3.4 PCIe Interface

This gateware uses the Xilinx IP core *DMA/Bridge Subsystem for PCI Express* with the following configuration:

- PCIe speed: 5 GTransfers/s
- AXI clock frequency: 125 MHz
- reference clock frequency: 100 MHz

The PCIe reference clock is routed to the FPGA via the MMC firmware and is configured to be driven by the 100 MHz *FCLKA* clock coming from the AMC connector.

3.5 SDRAM interface

For the communication with the SDRAM, an IP core by Xilinx is used. The clock frequency of the SDRAM interface's AXI bus is 100 MHz, but has a different source so that an AXI clock converter is used to connect it to the rest of the AXI infrastructure which is clocked at the main processing clock of 100 MHz.

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, there is one generated VHDL file `src/vhdl/generated_constant_package.vhd` which is generated by the script `src/scripts/generate_monitoring_and_control.py` using the content of the configuration files in `src/config`.

4.1 Prerequisites

An installation of Xilinx Vivado is required. Currently the IP cores are built for version 2019.2 so that this version should be installed.
You can set your preferred Vivado version in the script `src/config/project_config.sh`.

4.2 Build flow

4.2.1 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 `run/run_build_flow.sh`

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

4.2.2 Vivado GUI based build flow

Via a Bash script

- navigate to the root folder of the repository in a terminal
- type `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/<Vivado version>/project`.

Via the Vivado GUI

If you intend to use the Vivado GUI itself to set up the project proceed as follows:

- open Vivado
- use the TCL console in the bottom of the GUI to navigate to `src/scripts` using the commands `pwd` and `cd`
- type `source create_project.tcl` in the TCL console. This will set up the Vivado project.

4.3 Simulation

4.3.1 Vivado GUI based simulation

Prerequisite: an existing Vivado project → see chapter 4.2.2.
Click on *Run 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 `run/run_simulation.sh <name of module (or none for the toplevel simulation)>`
- you will find the output files of the simulation in the folder `output/<Vivado version>/simulation`.

4.3.3 Peripherals simulation models

The toplevel simulation includes a Verilog simulation model from Micron, the manufacturer of the AFC's SDRAM, which allows the simulation of the behavior of the external 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 Gateware software interface

The communication between the gateware inside the FPGA and the software running on the CPU unit takes place via a PCIe driver by Xilinx called XDMA. There is only one PCIe Bar in use in the gateware which maps the memory space to different physical memories on the AMC board.

The following mapping is applied:

address	size	memory type	description
0x00000000	2 kiB	Flip Flops	inside FPGA, for registers
0x00004000	16 kiB	Block RAM	inside FPGA, for device information
0x80000000	2 GiB	SDRAM	external, for scope data

Table 5.1: Memory mapping

5.1 Scope memory

The complete SDRAM is used for the scope memory. The ADC data is stored in the following format:

address	bits	radix	description
0x80000000	16	signed	ADC 0 data (time = 0)
0x80000002	16	signed	ADC 1 data (time = 0)
...	...	...	...
0x8000004E	16	signed	ADC 39 data (time = 0)
0x80000050	64	unsigned	timestamp (time = 0)
0x80000058	8	none	state of the MLVDS lines (time = 0)
0x80000059	312	none	unused
0x80000080	16	signed	ADC 0 data (time = 1)
...	...	...	...

Table 5.2: ADC data storage format

The scope is a free running ring buffer. The write pointer is incremented with each incoming ADC sample vector by the vector size of 128 bytes.

The scope memory can hold up to 2^{24} samples. At the maximum sampling frequency of 5 MHz this corresponds to a maximum capture duration of 3.36 seconds.

5.2 Register map

5.2.1 Status registers

The following status registers can be read by software:

index	address	bits	radix	description
0	0x00000000	16	signed	latest ADC 0 data
1	0x00000008	16	signed	latest ADC 1 data
...	...	...	...	...
39	0x00000138	16	signed	latest ADC 39 data
48	0x00000180	32	unsigned	scope next write address
124	0x00000eE0	32	unsigned	build timestamp
125	0x000003E8	57	unsigned	FPGA serial number
126	0x000003F0	64	unsigned	module ID
127	0x000003F8	64	unsigned	magic number

Table 5.3: List of status registers

0 - 39: latest ADC {0 - 39} data

This register holds the latest ADC data sample.

48: Scope next write address

Address where the next data sample will be stored during the scope's capturing process. The recently stored ADC data can be found below this address, or - in case of a ringbuffer wraparound - also below the end of the memory region.

124: build timestamp

Time when the bitstream was created. This information can be used to identify the gateware version (together with the Git commit information documented in chapter 6.2.2).

Format:

bits	information
0 - 5	seconds
6 - 11	minutes
12 - 16	hours
17 - 22	last two decimal digits of the year
23 - 26	month
27 - 31	day

125: FPGA serial number

The XDMA PCIe driver by Xilinx numbers the devices randomly and is not able to identify the slot number of an AMC board. This register holds the FPGA's unique serial number and can be used to identify an AMC board.

126: module ID

The module ID can be used to identify the type of the current bitstream. The module ID of the UniMon gateware is `0x0001010300010002`.

127: magic number

The magic number can be used to identify the position of the following three registers in the register bank. This register contains the value `0xBADEAFFEDEADC0DE`.

5.2.2 Configuration registers

The following registers can be written by software:

index	address	bits	radix	description	default value
0	0x00000400	18	unsigned	ADC conversion pulse length	0x00000
1	0x00000408	5	unsigned	MLVDS calibration delay	0x00
15	0x00000478	1	binary	partial reset	0
16	0x00000480	40	none	FMC0 ADC gain	0x0000000000
17	0x00000488	40	none	FMC0 ADC input mux select p	0x0000000000
18	0x00000490	40	none	FMC0 ADC input mux select n	0x0000000000
24	0x000004C0	40	none	FMC1 ADC gain	0x0000000000
25	0x000004C8	40	none	FMC1 ADC input mux select p	0x0000000000
26	0x000004D0	40	none	FMC1 ADC input mux select n	0x0000000000
32	0x00000500	16	signed	ADC 0 offset correction summand	0x0000
...	...	...	...	...	...
71	0x00000638	16	signed	ADC 39 offset correction summand	0x0000
72	0x00000640	16	unsigned	ADC 0 gain correction factor	0x8000
...	...	...	...	...	...
111	0x00000778	16	unsigned	ADC 39 gain correction factor	0x8000

Table 5.4: List of configuration registers

0 : ADC conversion pulse length

This parameter p defines the sampling frequency of all 40 ADCs in parallel.

The sampling frequency is given by:

$f_s = 210\text{MHz} \cdot \frac{1}{42+p}$

The allowed range for the sampling frequency is: $[1\text{kHz}, 5\text{MHz}]$

The default setting of 0 corresponds to a sampling frequency of 5 MHz.

The allowed range for p is: $[0, 209958]$

1 : MLVDS calibration delay

The signals from the timing receiver enter the FPGA via the MLVDS lines with virtually no delay, while the corresponding ADC samples will be delayed due to the sampling process.

This register can be used to add an additional delay to the signals from the timing receiver in order to calibrate them to the delays of the ADCs.

The delay can be chosen from the range $[10\text{ns}, 320\text{ns}]$. Each LSB corresponds to an additional delay of 10 ns.

15: Partial reset

Writing a 1 to this register triggers a partial reset on the gateware, which affects the scope and the FMC interfaces. The PCIe interface, the SDRAM interface, the AXI infrastructure and the register bank will not be resetted.

The reset will be automatically lifted so that the register does not have to be written to 0 after initiating a reset.

16, 24: FMC {0, 1} ADC gain

There are programmable amplifiers in front of each ADC input, which provide four discrete gains.

The following gains are available:

value	gain	input range	resolution per LSB
0	1	+/- 10.24 V	312.5 μV
1	2	+/- 5.12 V	156.25 μV
2	5	+/- 2.048 V	62.5 μV
3	10	+/- 1.024 V	31.25 μV

This register holds the concatenation of the gain settings of the FMC's 20 ADCs:

bits	gain of ADC
1 .. 0	0
3 .. 2	1
...	...
39 .. 38	19

17, 18, 25, 26: FMC {0, 1} ADC input mux select p / n

The ADCs have differential inputs. The two lanes p and n can be configured separately and individually for each ADC.

The following input settings are available:

value	input
0	differential p / n input pin
1	ground
2	not implemented
3	calibration voltage of +4.128 V

This register holds the concatenation of the input settings of the FMC's 20 ADCs:

bits	settings of input p / n of ADC
1 .. 0	0
3 .. 2	1
...	...
39 .. 38	19

32 - 71: ADC {0 - 39} offset correction summand

Correction summand for a possible offset deviation of the ADC. The offset correction precedes the gain correction.

72 - 111: ADC {0 - 39} gain correction factor

Correction factor for a possible gain deviation of the ADC. The default value 0x8000 corresponds to a multiplication by 1. The possible correction range is $[0, 2[$.

6 Extended gateware software interface

Besides the interface documented in chapter 5 which is meant for productive use, there is an extended interface for development and debugging purposes. The extended interface is also present in the bitstream by default.

While the productive interface is intended to be kept as downward compatible as possible, the extended interface may be subject to major changes during the development process.

6.1 Extended register map

6.1.1 Additional status registers

The following additional status registers can be read by software:

index	address	bits	radix	description
111	0x00000788	1	binary	SDRAM initial calibration complete

Table 6.1: List of additional status registers

111: SDRAM initial calibration complete

The communication to the SDRAM is controlled by an IP core by Xilinx which performs a timing calibration at start up. The value of this register will be 1 after completion of the initial calibration.

6.2 Architecture information storage

The Block RAM (see memory mapping, table 5.1) is used to store information about the registers, the observers and the gateware version.

6.2.1 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 6.2: Register information storage format

Table 6.2 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 7.1.2).

6.2.2 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 7.1.4), except from the bitstream generation date, which is read from status register 124 'build timestamp'.

7 Test software

7.1 FPGA Observer

There is a graphical test software intended to be run on the CPU unit. It is implemented in Python using the GTK 3 GUI toolkit.

7.1.1 Installation and usage

The sources and an installation script can be found under `src/software/fpga_observer/` in the *UniMon_Gateware* Git repository.

Installation

Connect to the CPU unit e.g. via ssh. Clone the *UniMon_Gateware* Git repository:

```
git clone git@git.gsi.de:BEA_HDL/UniMon_Gateware.git
```

Install the PCIe driver:

```
cd src/software/pcie_driver
sudo ./install.sh
```

Install the FPGA Observer software:

```
cd ../fpga_observer
sudo ./install.sh
```

Usage

For the PCIe driver to work, the bitstreams of the FPGAs have to be loaded before powering the CPU unit. If that is not the case, power cycle the CPU unit by pulling out the Hot Swap Handle and pushing it in again. A software reboot does not work.

Connect to the CPU unit e.g. via `ssh -X` in order to allow a graphical connection. Start the FPGA Observer software by:

```
sudo fpga_observer
```

A GUI should open and a choice of FPGA serial numbers should be displayed on the upper left corner. If the list is empty, either the loading of the FPGAs finished after powering the CPU unit or the PCIe driver did not install correctly. The FPGA serial numbers can be used to identify the AMC board you like to access. Choose a serial number and click *connect*.

7.1.2 Register Access tab

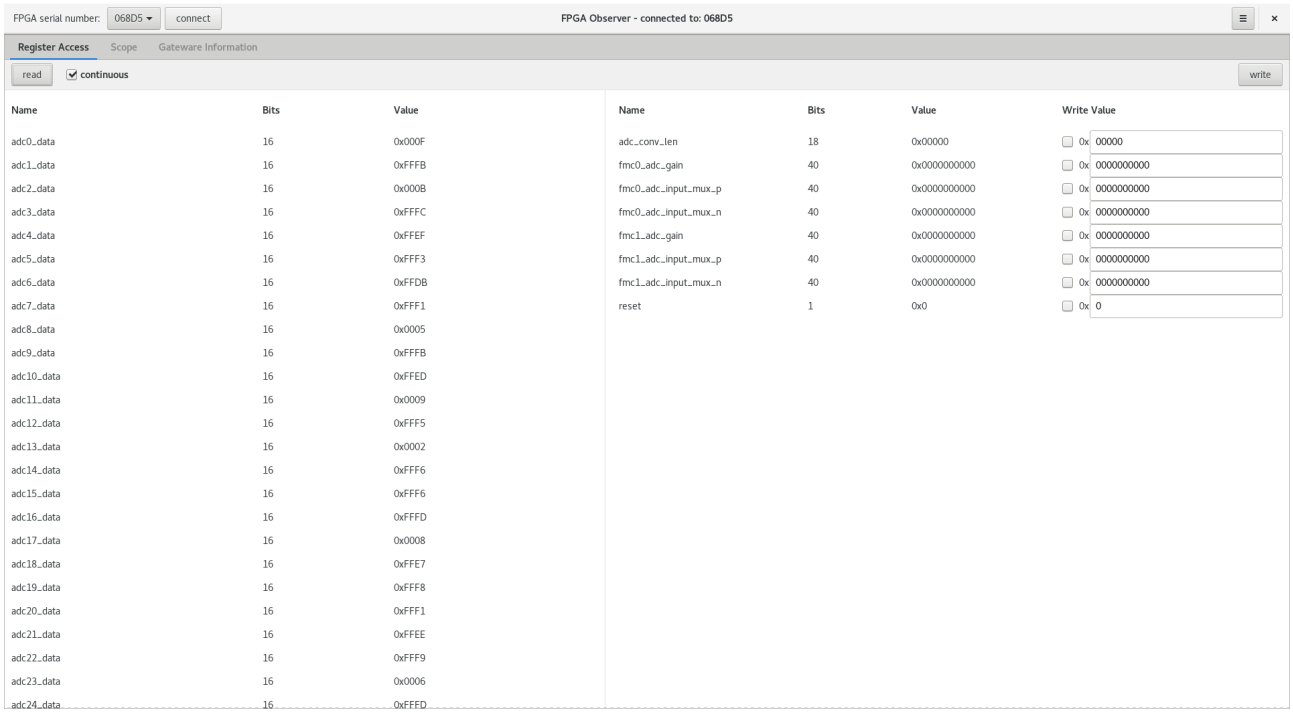

Figure 7.1: FPGA Observer - Register Access tab

The names and widths of the registers are read from an information memory region in the FPGA (see chapter 6.2.1). 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.

7.1.3 Scope tab

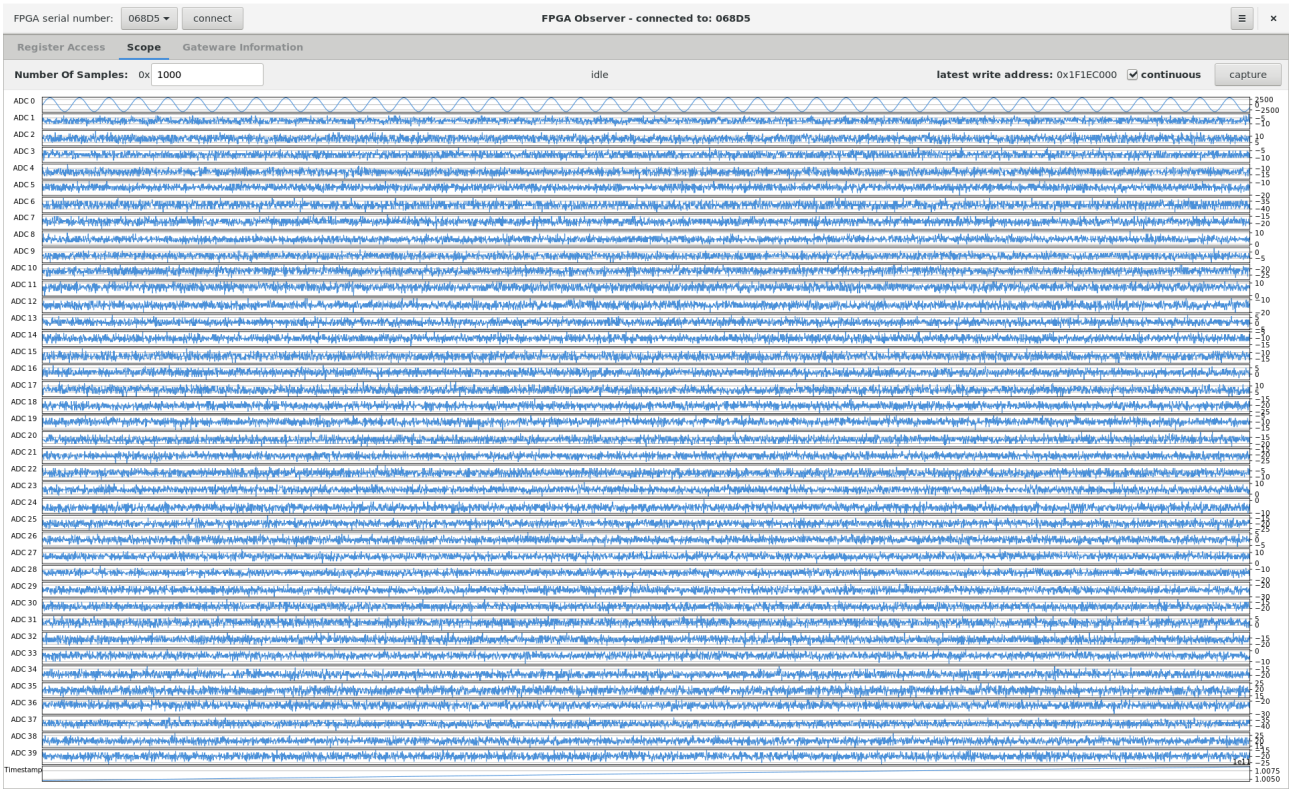

Figure 7.2: FPGA Observer - Scope tab

The scope tab displays the signals of all the ADCs that are mounted on a single AFC carrier.

For this purpose the write pointer of the ring buffer is polled and the required number of samples are read from positions before this pointer.

7.1.4 Gateware Information tab

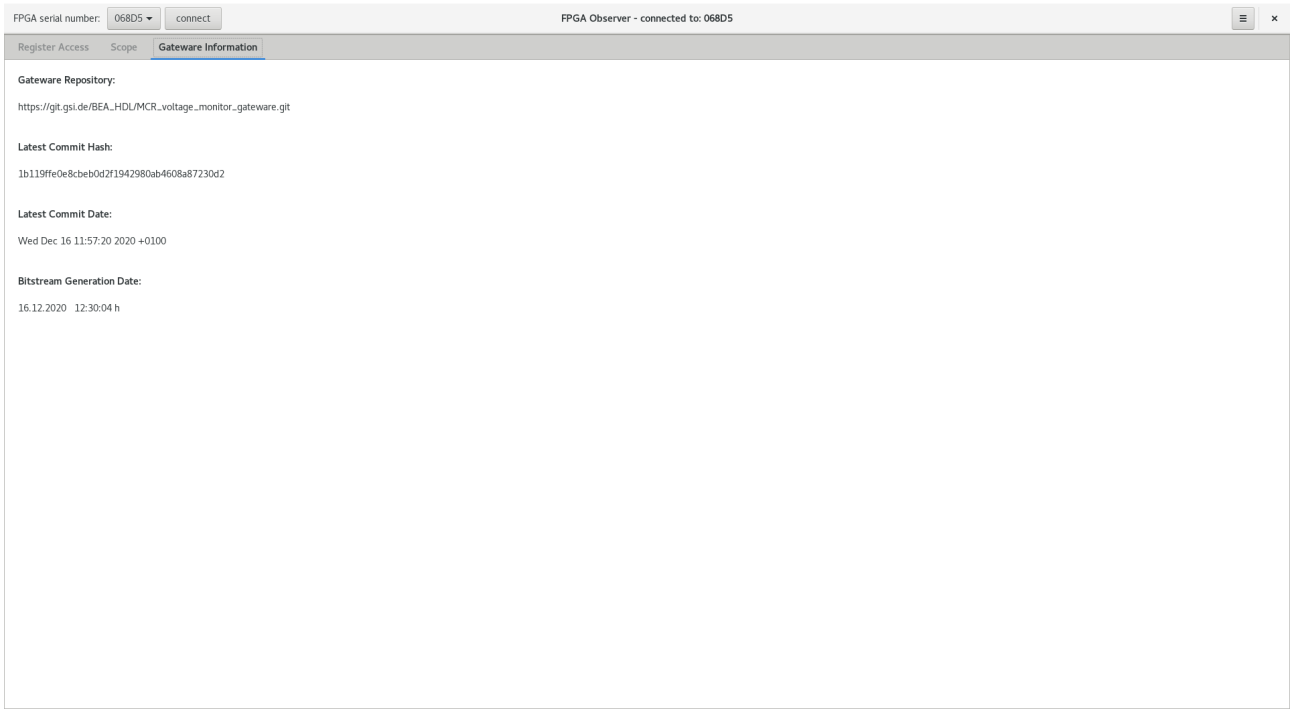

Figure 7.3: FPGA Observer - Gateware Information tab

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 6.2.2) and are displayed in this tab.

The bitstream generation date is read from status register 126 'build timestamp'.

7.2 Test scripts

7.2.1 PCIe access test script

There is a PCIe access test script under `src/software/pcie_driver/test_pcie_access.sh` in the *UniMon_Gateware* Git repository. 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.

8 Helper scripts

8.1 VHDL beautification

There is a script `src/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

8.2 Remote power cycling of the CPU unit

Whenever the bitstream of an FPGA is reloaded, the CPU unit has to be rebooted via its Hot Swap Handle in order to establish a PCIe connection. A software reboot does not work.

An alternative possibility of remote power cycling the CPU unit is via the MCH.

The script `src/scripts/powercycle_cpu_unit.py` instructs the MCH via SSH to power down and repower the CPU unit. The script expects one parameter: `<name of MCH, e.g. sdmch023>`.

The script takes about 60 seconds to complete.

8.3 Generation of a VHDL file for monitoring and control

The monitoring and control configuration of the gateware is defined by the configuration files `read_registers.csv` and `write_registers.csv` in the folder `src/config`.

The script `src/scripts/generate_monitoring_and_control.py` is used to convert the configuration to a VHDL file stored as `src/vhdl/generated_constant_package.vhd`. It contains the register default values and a Block RAM initialization vector containing the register configuration.

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

8.4 Generation of documentation

8.4.1 PDF

There is a script `doc/scripts/create_pdf.sh` for generating a PDF file from the Latex sources in `doc/tex`. It calls the open source software *Pdflatex* twice on the top level documentation file `UniMon_Gateware_Documentation.tex` to enable the generation of references inside the PDF file.

8.4.2 Markdown

There is a script `doc/scripts/create_markdown.py` for generating the Markdown file `README.md`, which is displayed on the repository's start page in Gitlab. The script uses the open source software *Pandoc* for an initial conversion of the Latex sources in `doc/tex`.

The result of *Pandoc* is postprocessed for multiple reasons:

- conversion of the math syntax to Gitlab's .md format
- corrections of the bibliography, HTML syntax and Latex labels
- corrections of the references to figures, tables and equations
- enumeration of chapters, sections and subsections
- adding of captions for figures and equations
- enumeration of figures, tables and equations
- generation of a table of contents
- implementation of citations

Additional documentation which is not included in the Latex sources is appended from the file `doc/markdown/epilog.md`.

8.4.3 DokuWiki

There is a script `doc/scripts/create_dokuwiki.py` for generating 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 Latex documentation sources to the DokuWiki format in four steps:

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

The preprocessing actions are:

- removing 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 reference format

9 Continuous integration environment

There is a continuous integration environment setup for the *UniMon_Gateware* Git repository. It 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
- the master branch can be kept functional at any time
- build results like e.g. bitstreams are automatically generated and can be archived

9.1 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.

9.2 Pipeline Stages

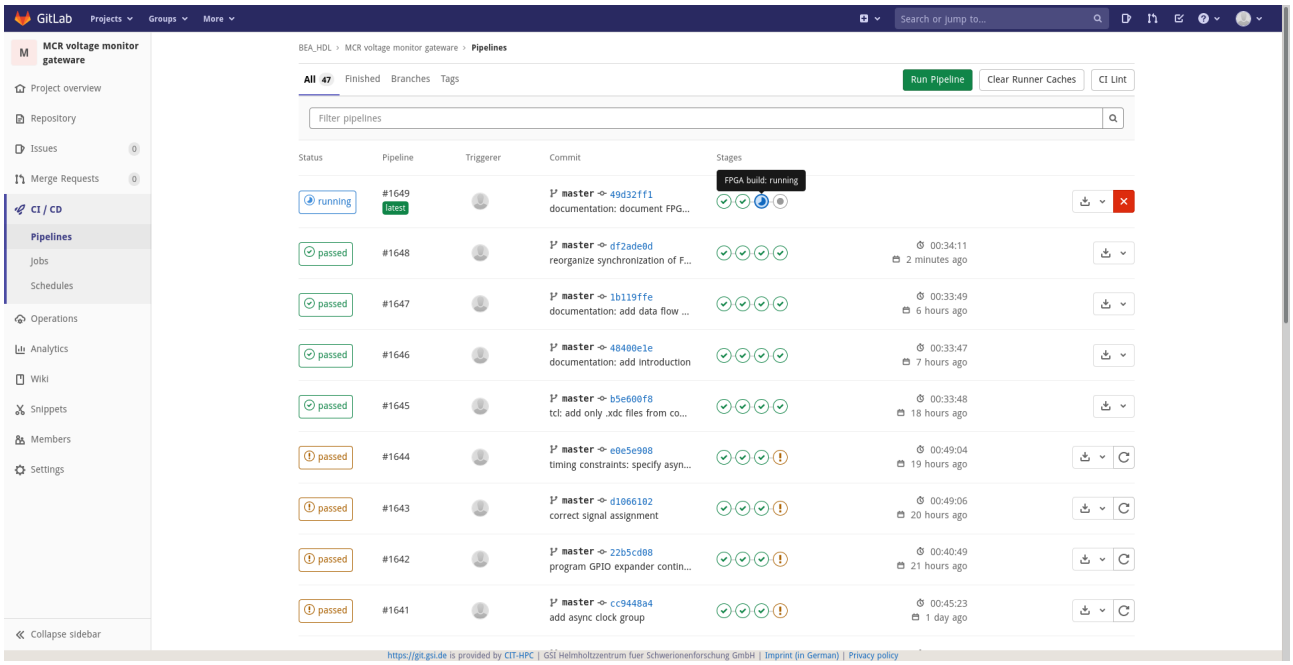

Figure 9.1: Gitlab: continuous integration pipelines

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.

The following pipeline stages are defined:

- documentation
- simulation
- FPGA build

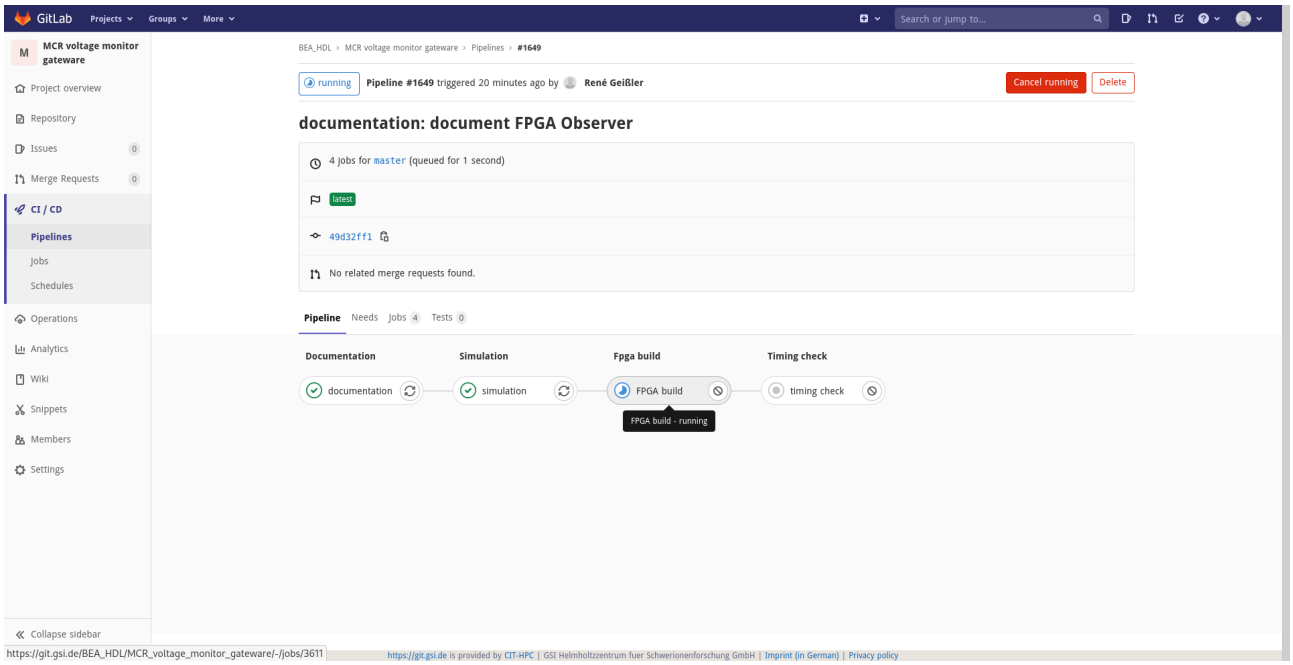

Figure 9.2: Gitlab: Pipeline stages

9.2.1 Documentation

The script `create_documentation.sh` in `doc/tex` is run to generate this documentation from the Latex source files. Pdflatex is run twice by the script to allow the generation of references inside the document. 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.

9.2.2 Simulation

The script `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.

9.2.3 FPGA build

The script `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.

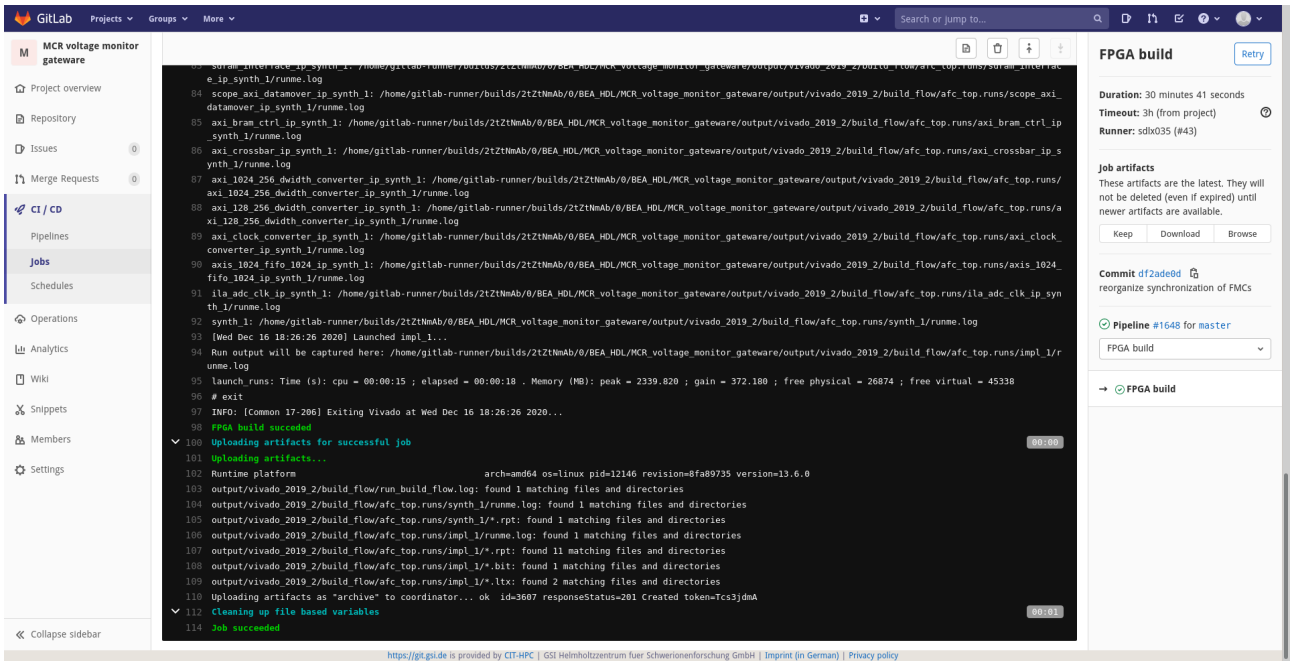

Figure 9.3: Gitlab: Pipeline progress console

9.2.4 Timing check

The script `src/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.

9.3 Build results

For each of the pipeline stages the archiving of build results can be configured for an adjustable time period, which is set to one week. 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 9.3).

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

9.4 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

10 Programming and hardware configuration

10.1 Programming the gateware

10.1.1 Using a JTAG Switch Module

10.2 Configuration of the MCH

10.2.1 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: 5 sec
Delay before the CPU unit will power up on start up. For making sure that the bitstreams are loaded to the AMC's FPGAs from Flash memory before the CPU unit boots you might have to increase this value.

PCIe parameter → PCIe hot plug delay for AMCs: 0 sec
Delay before the AMC boards will power up on start up.

Switch PCIe x80

Set the CPU-Unit as upstream AMC source in 'Virtual Switch 0':

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

10.2.2 Via USB

The most comfortable way of configuring the MCH is via its web interface. If you have accidentally disabled the web-server, 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.

10.3 Enabling network boot on the CPU unit

Shortly after powering the CPU unit, 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 CPU unit 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 CPU unit 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 CPU unit has to registered in the DHCP server responsible for distributing the locations from which to load the network images

10.4 Configuration of the timing receiver

The GPIOs of the timing receiver can be controlled via command line commands available in the network boot image of the CPU unit:

- 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`

11 System limitations

11.1 CPU unit reboot

The FPGA bitstreams have to be loaded before the CPU unit boots, otherwise the PCIe driver will not detect the FPGAs. This can be ensured by setting suitable power up delays in the MCH (see chapter 10.2.1).

Whenever a new bitstream is loaded to a FPGA, the CPU unit has to be rebooted either via its Hot Swap Handle or remotely via the MCH (see chapter 8.2). An operating system reboot does not work.

References

[1] IOxOS ADC_3117 datasheet, https://git.gsi.de/BEA_HDL/UniMon_Gateware/-/blob/master/doc/datasheets/ADC_3117_UG.pdf

[2] Analog Devices LTC2323-16 datasheet, https://git.gsi.de/BEA_HDL/UniMon_Gateware/-/blob/master/doc/datasheets/AD_LTC2323_ADC_datasheet.pdf