FTU Firmware Specifications

Quirin Weitzel[1], Patrick Vogler

v2 - October 2010

[1]Contact for questions and suggestions concerning this document: qweitzel@phys.ethz.ch

Contents

Chapter 1

Introduction

The FACT Trigger Unit (FTU) is a Mezzanine Card attached to the FACT Pre-Amplifier (FPA) board. On the FPA, the analog signals of nine adjacent pixels are summed up, and the total signal is compared to a threshold. Four such trigger patches are hosted by one FPA. The corresponding four digital trigger signals are processed further by the FTU, generating a single trigger primitive out of them. A total of 40 FTU boards exists in the FACT camera. Their trigger primitives are collected at one central point, the FACT Trigger Master (FTM)[1]. The FTM serves also as slow control master for the FTUs, which are connected in groups of ten to RS485 data buses for this purpose. These buses are realized on the midplanes of the four crates inside the camera.

The main component on each FTU is a FPGA[2], fulfilling different tasks within the board. The purpose of this document is to describe the main features of the firmware of this device, which is identical for all 40 boards. After a brief summary of the FTU functionality and its digital components, a general overview of the firmware design is given. In the following, all FPGA registers available for reading and writing are listed. Afterwards the communication with the FTM is detailed, and the most important finite state machines implemented are explained.

[1]For more information about the FACT trigger system see: P. Vogler, *Development of a trigger system for a Cherenkov Telescope Camera based on Geiger-mode avalanche photodiodes*, Master Thesis, ETH Zurich, 2010.

[2]Xilinx Spartan-3AN family (XC3S400AN-4FGG400C); for programming information see: `http://www.xilinx.com/support/documentation/user_guides/ug331.pdf` (Spartan-3 Generation FPGA User Guide).

Chapter 2

FTU Tasks and Digital Components

In order to define the thresholds for the individual trigger patches, four channels of an octal 12-bit Digital-to-Analog Converter (DAC) are used. This chip[1] is accessed by the FPGA through a Serial Peripheral Interface (SPI). A fifth channel of the same DAC is employed to control a n-out-of-4 majority coincidence logic, generating the trigger primitive out of the patch signals. The FTU can furthermore switch off single pixels within the trigger patches by disabling the corresponding input buffers just before the summation stage on the FPA. However, the decision to switch off a pixel for the trigger or to change the threshold for a patch is not taken by the FTU itself, but has to come in form of a command from the FTM.

For each of the four trigger patches, the FTU counts the number of triggers within a certain time period. Also the number of trigger primitives after the n-out-of-4 majority coincidence is counted. In this way, the rates per patch and per board are known for each FTU. The counters are implemented inside the FPGA with a range of 30 bit. In case the number of triggers exceeds this limit, an overflow flag is set. The counting period is changeable from outside between 500 ms and 128 s with a resolution of 8 bit.

Each FTU board has one RS485 communication interface to the FTM. Ten boards are connected to one bus, where they are operated in slave-mode. Only in case a read or write command for a specific FTU arrives from the FTM, this board will react and answer. Broadcast commands are not supported by the current firmware version. To avoid data collisions on the buses, the FTM has to address the FTUs one by one to read out the rates for example. A RS485 interface is implemented inside the FPGA, including frame receiving, data buffering and instruction decoding. It is minimal in the sense that no stacked commands, command buffering or interrupts are supported. Outside the FPGA, a RS485 driver/receiver chip translates the signal levels between the differential bus lines and the FPGA logic levels. This chip is enabled for data transmission or data receiving, respectively.

[1] Details concerning specific electronics components as well as schematics can be found at the FACT construction page: `http://ihp-pc1.ethz.ch/FACT` (password protected).

Chapter 3

Firmware Organization

The FTU firmware is written in VHDL (VHSIC Hardware Description Language). For some of its components the Xilinx core generator tools[1] have been used. In this chapter, an overview of the firmware content is given, followed by a listing of the files containing the source code. The complete project is available from the FACT repository[2].

3.1 Design Overview

The highest level entity in the firmware is called `FTU_top`. Its ports are the physical connections of the FPGA on the FTU board. Inside `FTU_top` other entities are instantiated, representing different functional modules. They are discussed in the following subsections. For important numbers and constants the package `ftu_constants` inside the library `ftu_definitions` has been created. A second package `ftu_array_types` contains customized array types.

3.1.1 `FTU_clk_gen`

This is an interface to the Digital Clock Managers (DCM) of the FPGA. At the moment only one DCM is used, providing the central 50 MHz clock. Once the DCM has locked and is providing a stable frequency `FTU_clk_gen` sends a ready signal. This is a prerequisite for the board to enter the `RUNNING` state. `FTU_clk_gen` also generates a central 1 MHz clock for the rate counters (by division, not using a second DCM). In addition, some modules within the FTU design have their own built-in clock dividers to generate custom frequencies.

3.1.2 `FTU_dual_port_ram64`

All FTU registers which can be set from outside during operation are stored in a dual-port block RAM (Random Access Memory). The FPGA provides specific resources for this purpose, and therefore the RAM has been created using the Xilinx core generator tools. The entity `FTU_dual_port_ram64` serves as an interface to the RAM, the actual gate level description is

[1] Xilinx ISE Design Suite, release 11.5, homepage: `http://www.xilinx.com` (downloads, documentation).
[2] Project page: `https://fact.isdc.unige.ch/trac` (password protected).

stored directly in the net-list file `FTU_dual_port_ram64.ngc`. Thus this file is part of the design, although not available as source code. The RAM has a size of 64 bytes and two fully featured ports to access its content. One port is based on 1-byte words, the other one on 2-byte words. The corresponding address space is presented together with the register tables in chapter 4. In addition to the control registers, also the current counter readings are stored in the RAM.

3.1.3 `FTU_rate_counter`

Here the trigger counting is done. A rate counter has a range of 30 bit and counts until a defined period is finished. This period is derived from an 8-bit prescaling value y as $T = \frac{y+1}{2}$ s. In total five such counters are instantiated which are running and set up synchronously. If the FTU settings are changed during operation all counters are reset. Only in case a full period has been finished without interruption, the number of counts from each counter is stored in the RAM. An overflow flag is set by the counters if necessary.

3.1.4 `FTU_spi_interface`

The octal DAC defining the trigger thresholds is controlled by means of a SPI. As soon as the `FTU_spi_interface` entity receives a start signal, it will clock out the data pending at one of its input ports to the DAC. These data are provided in form of a customized array. The generation of the serial clock, the distribution of the DAC values to the right addresses and the actual transmission of the data to the chip are performed by three more entities, which are instantiated inside `FTU_spi_interface`. Once the transmission has started or finished, respectively, a signal is pulled.

3.1.5 `FTU_rs485_control`

The communication between the FTU and the FTM is handled by a RS485 interface. The top level entity of this module is called `FTU_rs485_control`. It contains a state machine and further sub-entities for frame receiving or transmitting, byte buffering and instruction decoding. The details of the underlying protocol and the possible instructions are detailed in chapter 5. In case an instruction has been decoded successfully, the main FTU control is informed and the corresponding data (like new DAC values) are provided. After the command has been executed an answer is send to the FTM. `FTU_rs485_control` has direct control of the involved transmitter/receiver chip outside the FPGA and takes care that it is only transmitting if this particular FTU has been contacted by the FTM. In this way also the rates are send on request. The baud rate is adjustable and defined in the library `ftu_definitions`.

3.1.6 `FTU_control`

This entity contains the main state machine of the FTU firmware. It receives control, ready, start, etc. flags from all modules and interfaces and reacts accordingly by changing to a new state. This may be done with some delay, depending on what the board is doing at the moment. `FTU_control` is furthermore the only place in the design from where the RAM is read or written. The state machine is described in more detail in chapter 6.

3.1.7 FTU_dna_gen

In order to be able to unambiguously identify each FTU during operation, the device identifier[3] (DNA) of its FPGA is used. After power-up this DNA is read-out once by `FTU_dna_gen` and stored for later usage inside `FTU_top` as a permanent signal.

3.2 File Structure

Table 3.1 specifies all source files necessary to compile the firmware for the FTU boards[4]. For each file its location path inside the directory `firmware` of the FACT repository is stated. Furthermore it is indicated whether a certain file is needed for the simulation and/or the hardware implementation. The design entities described in section 3.1 are contained in those files which have the corresponding prefix. The file `ucrc_par.vhd` has been downloaded from OpenCores[5].

file name	location	simulation	implement	comment
FTU_top.vhd	FTU	yes	yes	top level entity
FTU_top_tb.vhd	FTU	yes	no	test bench
ftu_definitions.vhd	FTU	yes	yes	library
ftu_board.ucf	FTU	no	yes	pin constraints
FTU_control.vhd	FTU	yes	yes	top state machine
FTU_clk_gen.vhd	FTU/clock	yes	yes	clock interface
FTU_dcm_50M_to_50M.vhd	FTU/clock	yes	yes	clock manager
FTU_rate_counter.vhd	FTU/counter	yes	yes	trigger counter
FTU_spi_interface.vhd	FTU/dac_spi	yes	yes	SPI top entity
FTU_spi_clock_gen.vhd	FTU/dac_spi	yes	yes	serial clock
FTU_distributor.vhd	FTU/dac_spi	yes	yes	DAC loop
FTU_controller.vhd	FTU/dac_spi	yes	yes	low level SPI
FTU_dna_gen.vhd	FTU/dna	yes	yes	DNA readout
FTU_dual_port_ram64.vhd	FTU/ram64	yes	yes	RAM interface
FTU_dual_port_ram64.ngc	FTU/ram64	no	yes	RAM netlist
FTU_rs485_control.vhd	FTU/rs485	yes	yes	RS485 top entity
FTU_rs485_interpreter.vhd	FTU/rs485	yes	yes	data decoding
FTU_rs485_receiver.vhd	FTU/rs485	yes	yes	28-byte buffer
FTU_rs485_interface.vhd	FTU/rs485	yes	yes	low level RS485
ucrc_par.vhd	FTU/rs485	yes	yes	check sum

Table 3.1: List of all source files needed to compile the FTU firmware.

[3]This unique identifier has 57 bit and is built-in for each FPGA of the Spartan-3A series. For more information see: `http://www.xilinx.com/support/documentation/user_guides/ug332.pdf` (Spartan-3 Generation Configuration User Guide).

[4]As of October 2010; the file structure might still be changed.

[5]Ultimate CRC project: `http://www.opencores.org/cores/ultimate_crc` (free software under the terms of the GNU General Public License).

Chapter 4

Register Tables

There are 41 accessible control and rate registers implemented in the FTU firmware, each with a size of 8 bit. They are organized as a 64-byte RAM (see also section 3.1.2), the last 23 bytes of which are empty and serve as spares. Table 4.1 presents an overview of the address space inside the RAM, more details can be found in tables 4.2 - 4.5. Registers marked as read-only cannot be written by the FTM, but are updated by the FTU itself.

RAM address	register block	comment
00 . . . 07	enable patterns	read/write
08 . . . 27	rate counters	read-only
28 . . . 37	DAC settings	read/write
38	prescaling y (see section 3.1.3)	read/write
39	overflow bits	read-only
40	check sum error counter	read-only
41 . . . 63	empty	spare

Table 4.1: Overview of the register mapping inside the RAM.

4.1 Enable Patterns

address	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
00	En_A7	En_A6	En_A5	En_A4	En_A3	En_A2	En_A1	En_A0
01								En_A8
02	En_B7	En_B6	En_B5	En_B4	En_B3	En_B2	En_B1	En_B0
03								En_B8
04	En_C7	En_C6	En_C5	En_C4	En_C3	En_C2	En_C1	En_C0
05								En_C8
06	En_D7	En_D6	En_D5	En_D4	En_D3	En_D2	En_D1	En_D0
07								En_D8

Table 4.2: Mapping of the 4×9 enable bits inside the RAM.

4.2 Rate Counters

address	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
08	Ct_A7	Ct_A6	Ct_A5	Ct_A4	Ct_A3	Ct_A2	Ct_A1	Ct_A0
09	Ct_A15	Ct_A14	Ct_A13	Ct_A12	Ct_A11	Ct_A10	Ct_A9	Ct_A8
10	Ct_A23	Ct_A22	Ct_A21	Ct_A20	Ct_A19	Ct_A18	Ct_A17	Ct_A16
11	0	0	Ct_A29	Ct_A28	Ct_A27	Ct_A26	Ct_A25	Ct_A24
...	...	...	...	...	...	...	...	...
20	Ct_D7	Ct_D6	Ct_D5	Ct_D4	Ct_D3	Ct_D2	Ct_D1	Ct_D0
21	Ct_D15	Ct_D14	Ct_D13	Ct_D12	Ct_D11	Ct_D10	Ct_D9	Ct_D8
22	Ct_D23	Ct_D22	Ct_D21	Ct_D20	Ct_D19	Ct_D18	Ct_D17	Ct_D16
23	0	0	Ct_D29	Ct_D28	Ct_D27	Ct_D26	Ct_D25	Ct_D24
24	Ct_T7	Ct_T6	Ct_T5	Ct_T4	Ct_T3	Ct_T2	Ct_T1	Ct_T0
25	Ct_T15	Ct_T14	Ct_T13	Ct_T12	Ct_T11	Ct_T10	Ct_T9	Ct_T8
26	Ct_T23	Ct_T22	Ct_T21	Ct_T20	Ct_T19	Ct_T18	Ct_T17	Ct_T16
27	0	0	Ct_T29	Ct_T28	Ct_T27	Ct_T26	Ct_T25	Ct_T24

Table 4.3: Mapping of the four patch counter (A–D) and the trigger counter reading (T) inside the RAM. The two most significant bits of the 32 bits per counter are always set to 0.

4.3 DAC Settings

address	data[7 . . . 0]
28	DAC_A_[7 . . . 0]
29	DAC_A_[15 . . . 8]
...	...
34	DAC_D_[7 . . . 0]
35	DAC_D_[15 . . . 8]
36	DAC_H_[7 . . . 0]
37	DAC_H_[15 . . . 8]

Table 4.4: Mapping of the DAC values (12 bit) for the thresholds (DAC_A – DAC_D) and the n-out-of-4 logic (DAC_H) inside the RAM; the bits 15 . . . 12 are filled up with zeros.

4.4 Overflow Bits

bit	7 . . . 5	4	3	2	1	0
0	not used	overflow_T	overflow_D	overflow_C	overflow_B	overflow_A

Table 4.5: Bit mapping inside the RAM overflow register (address 39).

Chapter 5

Communication with FTM

The slow control system between the FTU boards (slaves) and the FTM (master) is based on two transmission sequences: Either the FTM is sending data to a particular FTU, or a particular FTU is answering to the FTM. Broadcast commands are not supported. Each board has a unique 1-byte address for identification on the RS485 buses, which is 0–63 for the FTUs[1] and 192 for the FTM. The transmission sequences are of fixed length (28 byte) and, if necessary, filled up with arbitrary data. In case the data transmission is disturbed or not complete, a time-out system ensures that the communication doesn't get stuck[2]. In the following, the slow control protocol, the instruction codes and the check sum error-detection are discussed.

5.1 Transmission Protocol

Table 5.1 summarizes the structure of the data sequences sent between the FTM and the FTUs. A FTU only replies if contacted by the FTM. The answer is a copy of the received data package with swapped source/destination address and eventually the requested data. Byte 26 is used to transmit the number of CRC errors counted by a FTU until a valid sequence arrived. In that case the number of errors is communicated and the error counter is set to 0.

byte	content	comment
00	start delimiter	ASCI @ (binary "01000000")
01	destination address	192 (FTM) or slot position 0–63 (FTUs)
02	source address	192 (FTM) or slot position 0–63 (FTUs)
03	firmware ID	firmware version of source FPGA
04	instruction / info	see section 5.2
05 . . . 25	21 byte data	DACs, rates, etc.
26	CRC error counter	number of CRC errors on FTU
27	check sum	CRC-8-CCITT, see section 5.3

Table 5.1: Composition of the FTM-FTU slow control data packages.

[1] Two bits are used to specify the crate, four bits to indicate the slot position within a crate. This 6-bit address is different from the 57-bit DNA which is FPGA-bound (see section 3.1.7).

[2] At a baud rate of 250 kHz, for example, this time-out is set to 2 ms on the FTU side.

5.2 Instruction Table

A set of eight instructions has been foreseen for the communication between the FTM and the FTUs. They are listed in table 5.2 including a short description. In case a FTU receives the ping-pong command, it returns also the DNA of its FPGA (see section 3.1.7). Combined with the 6-bit address, which is related to the geographical position insided the camera crates, it is therefore possible to identify each FTU.

code	instruction	description
00	set DAC	write new values into DAC registers
01	read DAC	read back content of DAC registers
02	read rates	read out rates and overflow bits
03	set enable	write new patterns into enable registers
04	read enable	read back content of enable registers
05	ping-pong	ping a FTU to check communication (see text)
06	set counter mode	write into the prescaling register
07	read counter mode	read back prescaling and overflow registers

Table 5.2: Instruction set for the FTM-FTU slow control communication.

5.3 CRC Calculation

The integrity of the 28-byte data packages is evaluated by means of a Cyclic Redundancy Check (CRC). An 8-CCITT CRC has been chosen which is based on the polynomial $x^8 + x^2 + x + 1$ (100000111). Bytes 0–26 of table 5.1 constitute the input vector for the CRC calculation, the resulting 1-byte check sum being compared with the one transmitted by the FTM (byte 27 in table 5.1). If the check sum turns out to be wrong, the FTU doesn't answer and increases the number of error counts in its corresponding register (RAM address 40). The FTM will consequently run into a time-out and repeat its command.

Chapter 6

Finite State Machines

There are several finite state machines (FSM) used in the FTU firmware design, distributed over several files. They are in principal all running in parallel, some of them are, however, only waking up if triggered by the main control. This is for example the case for the SPI interface controlling the DAC settings. Since the most complicated FSMs are inside `FTU_control` and `FTU_rs485_control` (see section 3.1), they are explained in more detail in this chapter.

6.1 Main Control FSM

This state machine has full control over the FTU board during operation. After power-up or reboot it is in an `IDLE` state, waiting for the DCMs to lock. Afterwards it passes through two `INIT` sequences, where default values for all registers are written to the RAM and the DNA is read out. The defaults are all defined in the library `ftu_definitions`. When the initialization has finished, the `RUNNING` state is entered. This is the principal state during which the board is counting triggers. `RUNNING` is left only if a counting period has finished and the number of counts is stored in the RAM, or if a command has arrived via RS485 and is communicated by the responsible FSM (see next section). A dedicated state has been implemented for each possible command and, if appropriate, also for the subsequent change of settings (e.g. `CONFIG_DAC`). In any case the board goes back to `RUNNING`.

6.2 RS485 Control FSM

The main and default state of this FSM is `RECEIVE`. This means that the RS485 receiver is enabled and the board is waiting for commands from the FTM. If a full 28-byte package has arrived and correctly been decoded[1], the main control FSM is informed about the instruction (e.g. new DACs). The RS485 FSM then enters a wait state (e.g. `SET_DAC_WAIT`) until it gets an internal ready signal. It subsequently sends the answer to the FTM (e.g. `SET_DAC_TRANSMIT`) and goes back to `RECEIVE`. While during `RECEIVE` the RS485 FSM and all processes below are running in parallel to the main control FSM, the sequence of states in case a command has arrived is prescribed.

[1] This involves a further state machine which is inside the file `FTU_rs485_interpreter.vhd`