Conv-TTL-Blo Production Test Suite
HDL Developer's Guide

Theodor-Adrian Stana
BE-CO-HT

May 30, 2013

Contents

1 Introduction **2**

2 Memory Map **3**

3 PTS General-Purpose Registers **4**

4 One-Wire Master **6**

5 SPI Master **7**

6 Clock Information Counters **8**

7 I2C Master **9**

8 Pulse Counters **10**

9 Folder Structure **12**

10 Getting Around the Code **14**

List of Figures

1 Memory-mapping several on-board peripherals in FPGA firmware 3
2 PTS CSR 5
3 Declarative part of VHDL architecture 14
4 Body of VHDL architecture 15

List of Tables

1 Memory map of the PTS firmware 4
2 General-purpose registers for PTS 4
3 CSR fields 5
4 Clock information counter registers 8
5 Pulse counter registers 11

List of abbreviations

DAC	Digital-to-Analog Converter
FPGA	Field-Programmable Gate Array
HDL	High-level Description Language
PTS	Production Test Suite
RTM	Rear-Transition Module
PLL	Phase-Locked Loop
SFP	Small Form-factor Pluggable (in the context of SFP connectors)
SPI	Serial Peripheral Interface

1 Introduction

This document presents a high-level view of the firmware implemented on the FPGA for the Production Test Suite (PTS) project for the Conv-TTL-Blo board. The document starts by presenting the memory map, followed by sections describing the logic implemented to run the tests comprising PTS. Then, the folder structure is described, along with hints on where the developer should look for specific files. This is followed by a presentation on the structure of the top-level HDL together with hints on where the developer should look in case changes are to be made to the code.

All the logic implemented on the FPGA is written in VHDL and can be obtained freely by cloning the OHWR git repository for the Conv-TTL-Blo PTS project at the following link:

link ohwr repo

REFERENCE THE OTHER DOCUMENTS

2 Memory Map

Various peripherals on the Conv-TTL-Blo board can be accessed by sending telnet commands to the VME crate; these commands are sent to the converter board by means of the serial I2C interface on the VME P1 connector. Each board peripheral is accessible via a memory-mapped Wishbone interface. A *vme64x_i2c* component (reference) is used alongside a Wishbone crossbar component (reference) to translate the I2C transfers into Wishbone transfers. Fig. 1 shows how these two components are connected to various other memory-mapped Wishbone slaves.

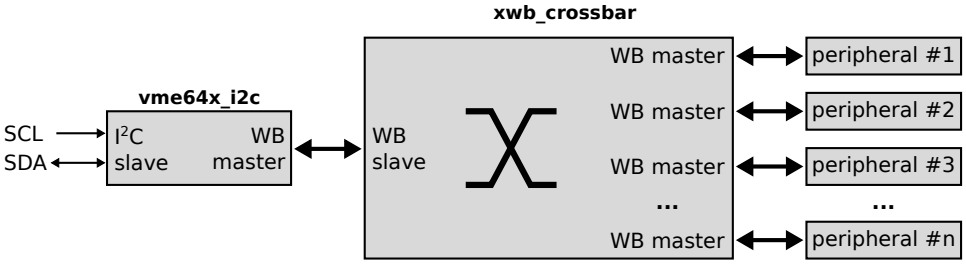

Figure 1: Memory-mapping several on-board peripherals in FPGA firmware

The PTS firmware accesses all peripheral devices on the Conv-TTL-Blo board and sends device-specific commands to test their functionality. Table 1 shows a simplified version of the memory map, with the base addresses of each peripheral. Details about the memory map of each device can be found by reading the description of the peripheral devices in the following sections.

Table 1: Memory map of the PTS firmware

Address	Name	Description
0x000	*pts_regs*	General-purpose registers
0x010	*onewire_mst*	One-wire master for thermal sensor
0x020	*dac_spi_125*	SPI master to control the 125 MHz DAC
0x080	*dac_spi_20*	SPI master to control the 20 MHz DAC
0x100	*clk_info_125*	Counters used to observe changes in the 125 MHz clock
0x120	*clk_info_125*	Counters used to observe changes in the 20 MHz clock
0x140	*sfp_i2c*	I2C master interface for communicating to SFP EEPROM
0x200	*endpoint*	Endpoint for communicating via the SFP
0x400	*minic*	MiniNIC for communicating via the SFP
0x800	*dpram*	Dual-port RAM for the *endpoint* and *minic* components
0xC00	*pulse_cntrs*	Pulse counters for TTL and blocking pulse repetition tests

3 PTS General-Purpose Registers

Two registers are implemented as general-purpose registers for the PTS. The first is a control and status register (CSR), at offset 0x0, followed by a board ID register at offset 0x4 (see Table 2). Both registers are 32 bits wide.

Table 2: General-purpose registers for PTS

Offset	Name	Access	Description
0x0	*csr*	R/W	Control and Status Register
0x4	*board_id*	R/W	Board ID register

The board ID register contains 32 read-and-writable bits which can be used to identify the board as the Conv-TTL-Blo. It is by default set to read as the ASCII string **BLO2**, the hex value 0x424C4F32.

Fig. 2 shows the control and status register of PTS. It consists of 16 bits of control data and 16 bits of status data. The first four bits are used to enable various test functionality. The reset bit can be used to reset all of the logic inside the FPGA when set (logic high). Caution should therefore be taken when writing the CSR, as an erroneous write might result in the whole logic resetting itself. When the logic is reset via a write to this bit, the *writereg* telnet command will return a *Not acknowledged!*, as the reset bit also resets the *vme64x_i2c* module.

The RTM field in the CSR can be used to read the status of the RTM detection lines and is relevant within the context of the blocking pulse and

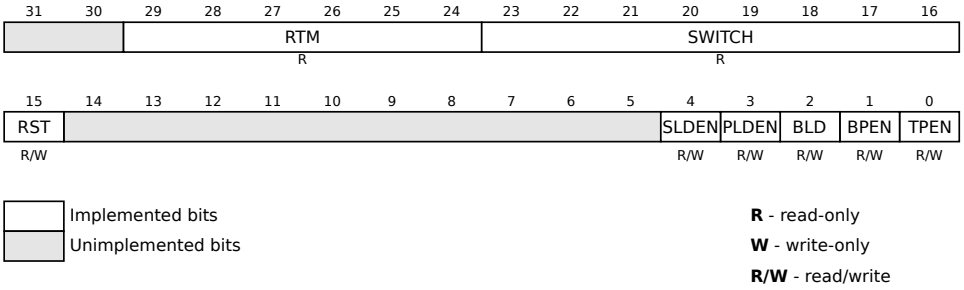

Figure 2: PTS CSR

Table 3: CSR fields

Name	Access	Description
TPEN	R/W	**1** – enable TTL pulse generation **0** – disable TTL pulse generation
BPEN	R/W	**1** – enable blocking pulse generation **0** – disable blocking pulse generation
BLD	R/W	Rear pulse LED line value
PLDEN	R/W	**1** – enable pulse LED sequencing **0** – disable pulse LED sequencing
SLDEN	R/W	**1** – enable status LED sequencing **0** – disable status LED sequencing
RST	R/W	**1** – Reset FPGA logic **Note:** Will reset *all* FPGA logic; therefore, it also resets itself.

RTM interface test.

4 One-Wire Master

This one-wire master module can be used to communicate to the DS18B20U+ thermometer (IC12). Two registers are implemented as part of this module. These registers and the functionality of the one-wire master module are described in the module's documentation [1].

5 SPI Master

The two Analog Devices DACs (IC17, IC18) on the Conv-TTL-Blo board can be controlled via a 3-wire SPI interface. The OpenCores SPI master module is used to implement this interface. More information can be found in the SPI master core's documentation [2].

6 Clock Information Counters

Two clock information counter modules are used to test that the clocks are reacting to changes in DAC and PLL values. The counters can be controlled and checked by means of six registers, shown in Table 4. An extra register at offset 0x1C can be used to check correct communication with the clock info counter module. This register is a read-only register which should return the hex value 0xC000FFEE when read.

Table 4: Clock information counter registers

Offset	**Name**	**Access**	**Description**
0x00	Counter full	R	Bit 0 set signals a full counter
0x04	Clock error	R	Bit 0 set signals a clock error occured
0x08	*Unused*	–	Unused in PTS
0x0C	Max value	R/W	Maximum value of the counter
0x10	Current value	R/W	Current value of the counter
0x14	Counter reset	R/W	Setting bit 0 clears the counter to 0
0x18	Counter enable	R/W	bit 0 set enables up-counting bit 0 cleared disables up-counting
0x1C	Module check	R	Should return 0xC000FFEE when read

7 I2C Master

An I2C master interface is implemented to send commands to the SFP EEPROM. The OpenCores I2C master core is used to implement the interface. More details about the module and the access registers can be found via its online documentation [3].

8 Pulse Counters

This module is used to count the number of sent and received pulses. It is useful in the context of the TTL and blocking pulse repetition test. On the Conv-TTL-Blo board, there are six TTL channels, four INV-TTL channels and six blocking channels, which are seen as sixteen pulse channels in the pulse counter module.

Two counters are implemented per each channel, one to count the number of input pulses, and another to count the output pulses. A counter counts up whenever a rising edge occurs on the channel it is associated to. The pulse counter Wishbone module implements 32 registers to store the current values of the counters for all these sixteen channels. The registers are stacked up starting from offset zero, with the TTL CH1 output pulse counter occupying address offset 0x00, the TTL CH1 input counter offset 0x04, followed by TTL CH2 output at 0x08 and TTL CH2 input at 0x0C, and so on, up to blocking CH6 input counter, which is located at address offset 0x64. Table 5 shows a the pulse counter address map.

Table 5: Pulse counter registers

Offset	Name	Access	Description
0x00	*ch1_out*	R/W	TTL channel 1 output counter
0x04	*ch1_in*	R/W	TTL channel 1 input counter
0x08	*ch2_out*	R/W	TTL channel 2 output counter
0x0C	*ch2_in*	R/W	TTL channel 2 input counter
0x10	*ch3_out*	R/W	TTL channel 3 output counter
0x14	*ch3_in*	R/W	TTL channel 3 input counter
0x18	*ch4_out*	R/W	TTL channel 4 output counter
0x1C	*ch4_in*	R/W	TTL channel 4 input counter
0x20	*ch5_out*	R/W	TTL channel 5 output counter
0x24	*ch5_in*	R/W	TTL channel 5 input counter
0x28	*ch6_out*	R/W	TTL channel 6 output counter
0x2C	*ch6_in*	R/W	TTL channel 6 input counter
0x30	*ch7_out*	R/W	INV-TTL channel A output counter
0x34	*ch7_in*	R/W	INV-TTL channel A input counter
0x38	*ch8_out*	R/W	INV-TTL channel B output counter
0x3C	*ch8_in*	R/W	INV-TTL channel B input counter
0x40	*ch9_out*	R/W	INV-TTL channel C output counter
0x44	*ch9_in*	R/W	INV-TTL channel C input counter
0x48	*ch10_out*	R/W	INV-TTL channel D output counter
0x4C	*ch10_in*	R/W	INV-TTL channel D input counter
0x50	*ch11_out*	R/W	Blocking channel 1 output counter
0x54	*ch11_in*	R/W	Blocking channel 1 input counter
0x58	*ch12_out*	R/W	Blocking channel 2 output counter
0x5C	*ch12_in*	R/W	Blocking channel 2 input counter
0x60	*ch13_out*	R/W	Blocking channel 3 output counter
0x64	*ch13_in*	R/W	Blocking channel 3 input counter
0x68	*ch14_out*	R/W	Blocking channel 4 output counter
0x6C	*ch14_in*	R/W	Blocking channel 4 input counter
0x70	*ch15_out*	R/W	Blocking channel 5 output counter
0x74	*ch15_in*	R/W	Blocking channel 5 input counter
0x78	*ch16_out*	R/W	Blocking channel 6 output counter
0x7C	*ch16_in*	R/W	Blocking channel 6 input counter

9 Folder Structure

The folder structure used within the PTS firmware is presented below

- → ip_cores/
- → Conv-TTL-Blo/hdl/
  - → bicolor_led_ctrl/
    - → *bicolor_led_ctrl.vhd*
    - → *bicolor_led_ctrl_pkg.vhd*
  - → glitch_filt/
    - → rtl/
      - → *glitch_filt.vhd*
  - → **pts/**
    - → **rtl/**
      - → ***clk_info_wb_slave.vhd***
      - → ***incr_counter.vhd***
      - → ***pts_regs.vhd***
      - → ***pulse_cnt_wb.vhd***
    - → **top/**
      - → ***conv_ttl_blo_v2.vhd***
      - → ***conv_ttl_blo_v2.ucf***
  - → ctb_pulse_gen/
    - → rtl/
      - → *ctb_pulse_gen.vhd*
  - → ctb_pulse_gen_gp/
    - → rtl/
      - → *ctb_pulse_gen_gp.vhd*
  - → reset_gen/
    - → rtl/
      - → *reset_gen.vhd*
  - → vme64x_i2c/
    - → rtl/
      - → *i2c_slave.vhd*
      - → *vme64x_i2c.vhd*

The ip_cores folder contains repository files that the firmware uses, such as the OpenCores SPI (see 5)and one-wire masters (see 4). The modules that have been developed as part of the Conv-TTL-Blo project and can be used in both PTS and other firmware are present in their own folders as sub-nodes of the *conv-ttl-blo/hdl/* folder. In general, the module files are present under an *rtl/* sub-folder. The *pts/* folder is the main folder in the case of the PTS suite, as can be seen from the fact that it is bolded in the folder structure above. It contains top-level files in the *top/* folder (HDL and UCF file for pin definitions) and other PTS-specific modules in the *rtl/* folder. The *ctb_pulse_gen_gp* module implements the general-purpose fixed-width, fixed-frequency-and-delay pulse generator used to generate pulses on CH10 in the TTL pulse repetition test and the pulses on the blocking output channels for the blocking pulse test. The *ctb_pulse_gen* module is the module used to generate fixed-width pulses when a trigger is received; it is the same pulse generator used in the release version of the firmware.

10 Getting Around the Code

All of the PTS-specific firmware can be found in the *conv-ttl-blo/hdl/pts/* folder. The top-level file, *conv-ttl-blo.vhd*, is the main-part of the firmware and thus shall be the topic of this short section, where its structure and guidelines for making changes are given. Most of the top-level ports of the file have been named according to the schematic file netlist names. The exceptions from this are due to either net names that could not be syntactically represented in VHDL, or net names that have been made clearer in VHDL code. Input ports are assigned to architecture signals and signals are assigned to output ports in each code section, as appropriate. Ports and signals usually follow the coding guidelines at [4].

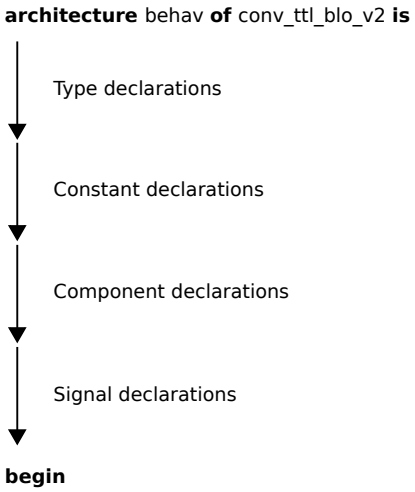

Figure 3: Declarative part of VHDL architecture

The top module architecture is divided into sections, delimited by visible comments. For example, code pertaining to a certain test, code pertaining to more than one test, or general top-level code can go into a code section. The declarative part of the architecture is organized as shown in Fig. 3. Types are declared right after the architecture declaration, followed by constant declarations, followed by component declarations, after which the various signals are declared.

The body of the architecture is organised as showin in Fig. 4. It begins by instantiating a differential buffer for the 125 MHz system clock and instantiating the *reset_gen* component. Then, the *vme64x_i2c* bridge module is instantiated along with the Wishbone crossbar that offers access to the rest of the Wishbone modules in the design. Next, the general-purpose PTS register (see Sec. 3) module is instantiated, followed by logic necessary for each of the tests comprising PTS.

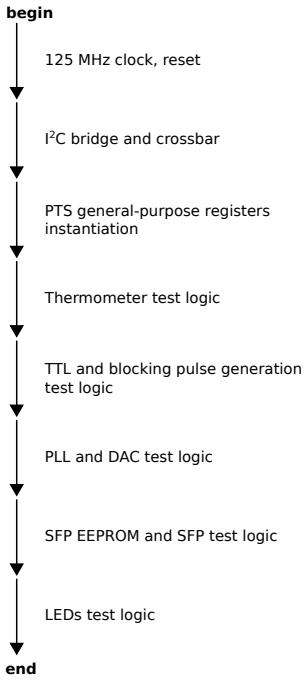

Figure 4: Body of VHDL architecture

References

[1] I. Jeras, "sockit_owm, 1-wire (onewire) master," 2011. http://opencores.org/websvn,filedetails?repname=sockit_owm&path=%2Fsockit_owm%2Ftrunk%2Fdoc%2Fsockit_owr.pdf.

[2] S. Srot, "SPI Master Core Specification," 2004. http://opencores.org/websvn,filedetails?repname=spi&path=%2Fspi%2Ftrunk%2Fdoc%2Fspi.pdf.

[3] R. Herveille, "I2C Master Core Specification," 2003. http://opencores.org/websvn,filedetails?repname=i2c&path=%2Fi2c%2Ftrunk%2Fdoc%2Fi2c_specs.pdf.

[4] P. Loschmidt, N. Simanić, C. Prados, P. Alvarez, and J. Serrano, "Guidelines for VHDL Coding," 04 2011. http://www.ohwr.org/documents/24.