CONV-TTL-BLO Production Test Suite
HDL Developer's Guide

Theodor-Adrian Stana
CERN, BE-CO-HT

May 3, 2013

Contents

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 with a description of the folder structure and where the developer should look for specific files. In the following section, the structure of the top-level HDL file is given along with hints on where the developer should look in case changes are to be made to the code. After that, the memory map is presented, followed by sections describing the logic implemented to run the tests comprising PTS.

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 I^2C 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 I^2C transfers into Wishbone transfers. Fig. 1 shows how these two components are connnected to various other memory-mapped Wishbone slaves.

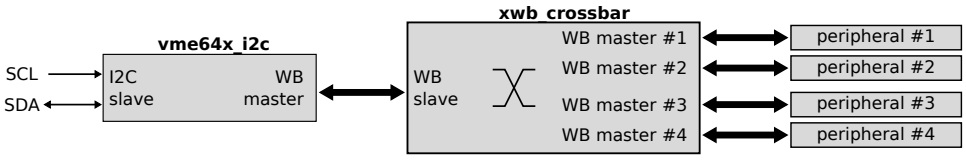

Figure 1: Memory-mapping several on-board peripherals in FPGA firmware

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*	Onewire master to control 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_20*	Counters used to observe changes in the 20 MHz clock
0x140	*sfp_i2c*	I^2C master interface for communicating with SFP EEPROM
0x200	*endpoint*	Endpoint for communicating via the SFP
0x400	*minic*	MiniNIC for communicating via the SFP
0x800	*dpram*	RAM for the *endpoint* and *minic* components
0xC00	*pulse_cntrs*	Pulse counters for TTL and blocking pulse repetition

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 in width.

Table 2: PTS general-purpose registers, base address 0x000

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 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 control bits (bits 3..0) should be written with the value of the current test. The test number is a fractional integer number with a fractional part of one bit. In this format, each test can consist of up to two parts, up to a maximum number of seven tests. Fig. 3 shows how CSR test number values can be obtained. This test number is used by the main PTS state machine implemented in the FPGA firmware to assert various control signals, based on the currently running test.

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

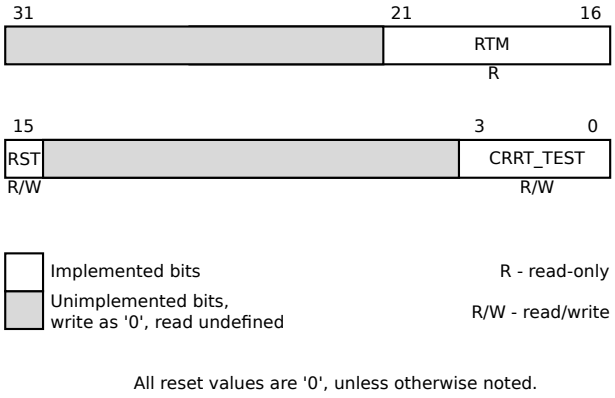

Figure 2: PTS CSR

the logic is reset via a write to this bit, the *writereg* telnet command will return a *Not acknowledged!*, as the reset also resets the *vme64x_i2c* module.

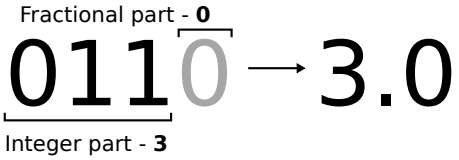

Figure 3: CSR test number format

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 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 along with 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 counters 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 3. 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 3: Clock information counter registers

Offset	Name	Access	Description
0x00	Counter full	R	Bit 0 high means counter is full
0x04	Clock error	R	Bit 0 high means a clock error occured
0x0C	Max value	R	Maximum value of the counter
0x10	Current value	R/W	Current value of the counter
0x14	Counter reset	R/W	Setting bit 0 resets the counter to 0
0x18	Counter enable	R/W	Setting bit 0 starts the counter; clearing bit 0 disables the counter
0x1C	Module check	R	Should return 0xC000FFEE when read

7 I^2C Master

An I^2C master interface is implemented to send commands to the SFP EEPROM. The OpenCores I^2C master core is used to implement the interface; more details about the interface 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. 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 whenenver a rising edge occurs on the channel it is associated to.

On the CONV-TTL-BLO board, there are six TTL channels, four INV-TTL channels and six blocking channels. 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. Fig. 4 shows a simplified version of the pulse counter address map.

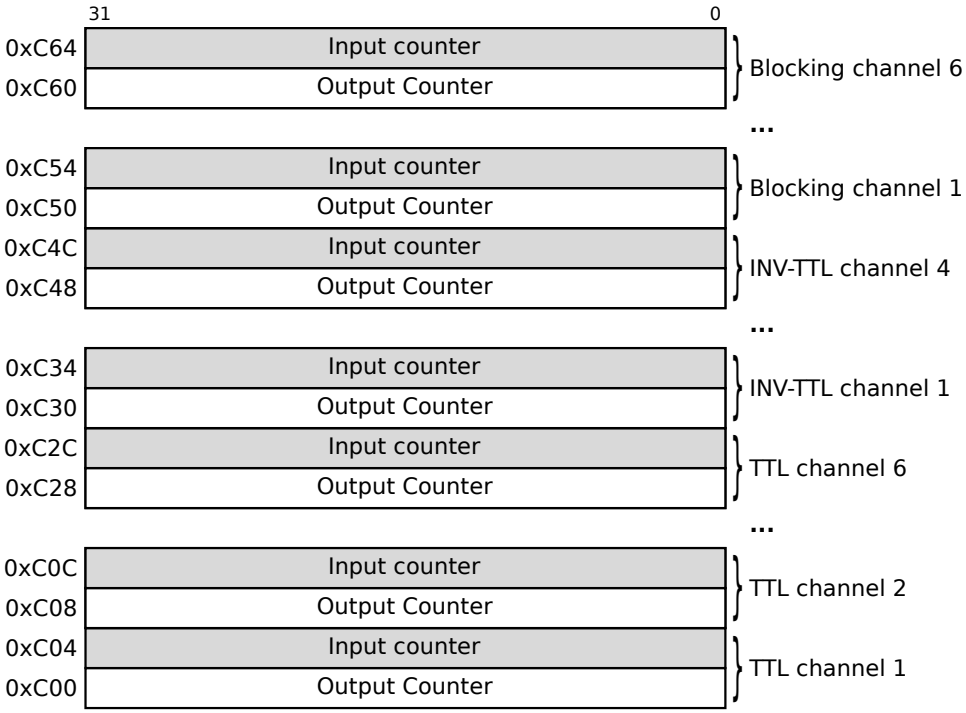

Figure 4: Pulse counters memory map

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***
  - → pulse_gen
    - → rtl/
      - → *pulse_gen.vhd*
  - → pulse_generator
    - → rtl/
      - → *pulse_generator.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 and one-wire masters. 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 *pulse_gen* module implements the fixed-width, frequency and delay pulse generator used to generate pulses on CH10 in the TTL pulse and the pulses on the blocking output channels. The *pulse_generator* 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 from a VHDL standpoint. Input ports are assigned to architecture signals and signals are assigned to output ports in each code section, as appropriate.

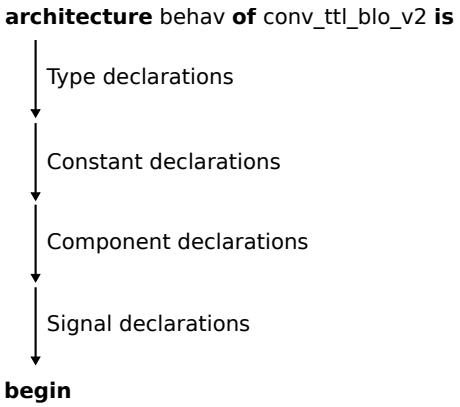

Figure 5: Declarative part of the VHDL architecture

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. 5. Types are declared right after the architecture declaration, followed by constant declarations, followed by component declarations, after which the various signals are declared.

The architecture body starts with instantiation of components useful in all the design, such as the reset generator and the I^2C bridge component. Then, the main state machine of the PTS firmware is described. It is used to assign the enable signals for the pulse generators and control the sequencing of LEDs. The state machine is controlled by the value of the *CRRT_TEST* field in the PTS CSR (see Sec. 3). The test numbers can be changed by means of the constants at the top of the architecture declarative part.

References

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

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

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