CONV-TTL-BLO HDL Guide

July 3, 2013

Theodor-Adrian Stana (CERN/BE-CO-HT)

Revision history

Date	Version	Change
04-07-2013	0.1	First draft

Contents

List of Figures

List of Tables

List of Abbreviations

DAC	Digital-to-Analog Converter
FPGA	Field-Programmable Gate Array
FSM	Finite-State Machine
IC	Integrated Circuit
I^2C	Inter-Intergrated Circuit (bus)
PLL	Phase-Locked Loop
SPI	Serial Peripheral Interface
SysMon	(ELMA) System Montior
VCXO	Voltage-controlled oscillator

1 Introduction

This document details the HDL implemented on the Spartan-6 FPGA on the CONV-TTL-BLO board. The HDL (mostly implemented in VHDL) handles the following aspects of the CONV-TTL-BLO capabilities:

- pulse detection (on pulse rising edge)
- fixed-width pulse generation
- status retrieval via I^2C and the ELMA protocol

Figure 1 shows a simplified block diagram of the HDL firmware. Each of the blocks in the figure is presented in following sections.

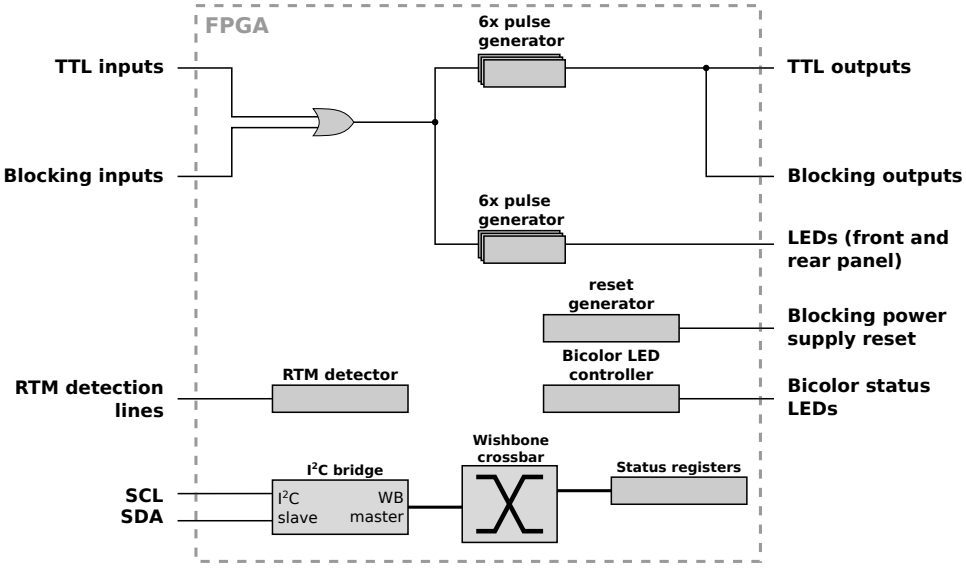

Figure 1: Block diagram of FPGA firmware

Additional documentation

!!!

- CONV-TTL-BLO User Guide [1]

2 FPGA Clocks

There are two clock signals input to the FPGA (Figure 2). The first is a 20 MHz signal from a VCXO. The second clock signal with a frequency of 125 MHz is generated on-board via a Texas Instruments PLL IC from a 25 MHz VCXO.

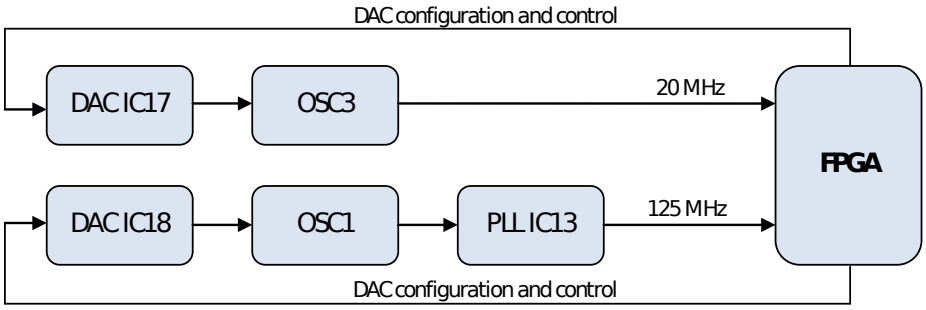

Figure 2: FPGA clock inputs

Two DACs are provided on-board for controlling the two VCXOs. The DACs can be controlled via SPI, but this feature is not yet implemented.

Table 1 lists the clock domains in the firmware.

Table 1: Clock domains

Clock domain	Frequency	Comments
clk125	125 MHz	Global clock input to all sequential logic

3 Reset generator

Entity	*reset_gen*	
Generics	*g_reset_time*	Reset time in *clk_i* cycles
Ports	*clk_i*	Clock signal
	rst_i	Active-high reset input
	rst_n_o	Active-low reset output
Usage	Global reset generation	96 *ms* reset

The reset generator module (*reset_gen*) implemented inside the FPGA generates a predefined-width reset signal when power is applied to the FPGA, or when an external reset is triggered via the *rst_i* pin.

When a power-on reset occurs on the Xilinx FPGA, a counter inside the *reset_gen* module starts counting up. While this counter is counting up, the active-low reset signal is kept low, resetting synchronous logic inside the FPGA. When the counter reaches the value of the reset width (specified via the *g_reset_time* generic at synthesis time), the reset signal is de-asserted, the counter is disabled and the *reset_gen* module remains inactive.

The module reactivates on the power-on reset, or when a reset is triggered externally, via the *rst_i* pin.

Note that the VHDL of this module is Xilinx and XST-specific and porting to a different FPGA architecture is not guaranteed to provide the same

results. The *reset_gen* module has an initial value set for the counter signal after power-up, which is guaranteed by XST to be set after the FPGA's GSR signal is de-asserted.

By default, the reset time is set to 96 *ms*.

4 RTM detection

Entity	*rtm_detector*	
Ports	*rtmm_i(2..0)*	RTM mainboard detection lines
	rtmp_i(2..0)	RTM piggyback detection lines
	rtmm_ok_o	RTM mainboard present
	rtmp_ok_o	RTM piggyback present
Usage	Light ERR status LED	

RTM detection is described in [2]. Since an RTMM/P missing would mean all *rtmm_i*/*rtmp_i* lines are all-ones, the *rtm_detector* module sets the *rtmm_ok* and *rtmp_ok* signals low if the *rtmm_i* and *rtmp_i* input signals are respectively all-ones.

The *rtmm_ok* and *rtmp_ok* signals are NANDed together to light the ERR status LED on the CONV-TTL-BLO.

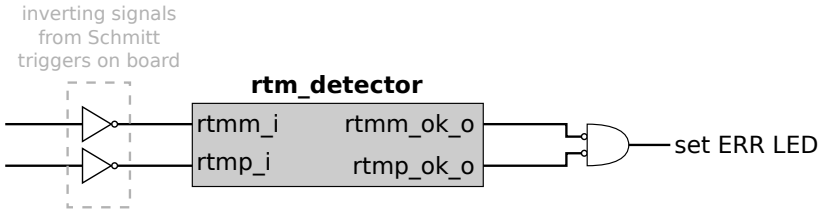

Figure 3: *rtm_detector* block in CONV-TTL-BLO firmware

The status of the RTM detection lines can also be read via their respective fields in the CONV board status register (Section 7.2).

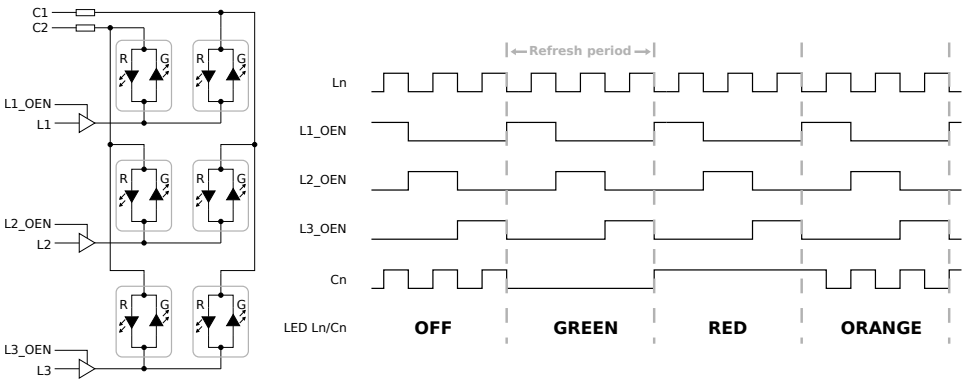

Figure 4: 3x2 bicolor LED matrix control

5 Bicolor LED controller

Entity	*bicolor_led_ctrl*	
Generics	*g_NB_COLUMN*	Number of columns
	g_NB_LINE	Number of lines
	g_CLK_FREQ	Frequency (in Hz) of *clk_i* signal
	g_REFRESH_RATE	LED refresh rate (in Hz)
Ports	*rst_n_i*	Active-low reset input
	clk_i	Clock signal input
	led_intensity_i(6..0)	7-bit LED intensity vector
	led_state_i(..)	LED state vector, two bits per LED
	column_o(..)	LED column vector, one bit per column
	line_o(..)	LED line vector, one bit per line
	line_oen_o(..)	LED line enable vector, one bit per line
Usage	Light bicolor LEDS	

The *bicolor_led_ctrl* block controls the lighting of a bicolor LED matrix. Based on the refresh rate given via the *g_REFRESH_RATE* generic, the clock frequency (*g_CLK_FREQ* generic) and the number of lines and columns, the module calculates controls lighting of the LED matrix.

Figure 4 shows an example of controlling a three-line, two-column red-and-green LED matrix. The FPGA ouputs for the columns (C) are connected to buffers and serial resistors and then to the LEDs. The FPGA outputs for lines (L) are connected to tri-state buffers and the to the LEDs. The FPGA outputs for line output enables (L_OEN) are connected to the output enable of the tri-state buffers.

The two-bit *led_state_i* vector can be used to control the color of each LED. Table 2 lists the values that should be input on *led_state_i* to get the needed color. Definitions are provided in the *bicolor_led_ctrl_pkg.vhd* file for setting the color of the LED via *led_state_i*; the constants are also listed in Table 2.

Table 2: LED state input

State	Constant	Value
Off	c_LED_OFF	00
Green	c_LED_GREEN	01
Red	c_LED_RED	10
Orange	c_LED_RED_ORANGE	11

Each LED's two-bit state is connected to *led_state_i* on a column-first, line-second basis. Table 3 shows the *led_state_i* connections for the example in Figure 4.

Table 3: Connecting the LED state vector to the 2x3 matrix

Line	Column	LED state bits
1	1	*1..0*
1	2	*3..2*
2	1	*5..4*
2	2	*7..6*
3	1	*9..8*
3	2	*11..10*

6 Pulse generators

Entity	*ctb_pulse_gen*	
Generics	*g_pulse_width*	Width of the output pulse in *clk_i* cycles
	g_glitch_filt_len	Length of glitch filter
Ports	*clk_i*	Clock signal
	rst_n_i	Active-low reset signal
	glitch_filt_en_n	Active-low glitch filter enable
	en_i	Pulse generator enable
	trig_i	Pulse trigger
	pulse_o	Pulse output
Usage	Output pulse	1.2 μs pulses
	Flash pulse LEDs	96 ms pulses

The *ctb_pulse_gen* blocks are twice used in the CONV-TTL-BLO firmware. First, they are used for generating the output pulses based on the trigger input. In this case, they are configured for 1.2 μs pulses ($g\_pulse\_width = 150$, considering the 8 ns clock input).

Second, they are used for blinking the front and rear-panel pulse LEDs when a pulse is generated. In this second case, the pulse generator blocks are configured to generate 96 ms pulses ($g\_pulse\_width = 12 * 10^6$), enough

to be visible to the human eye.

In both cases, the logic associated to the blocks is multiplied by six, since there are six replication channels.

6.1 Implementation

Figure 5 shows the implementation of the *ctb_pulse_gen* block. It employs a simple counter finite-state machine (FSM) that is used to generate a fixed-width pulse at the output.

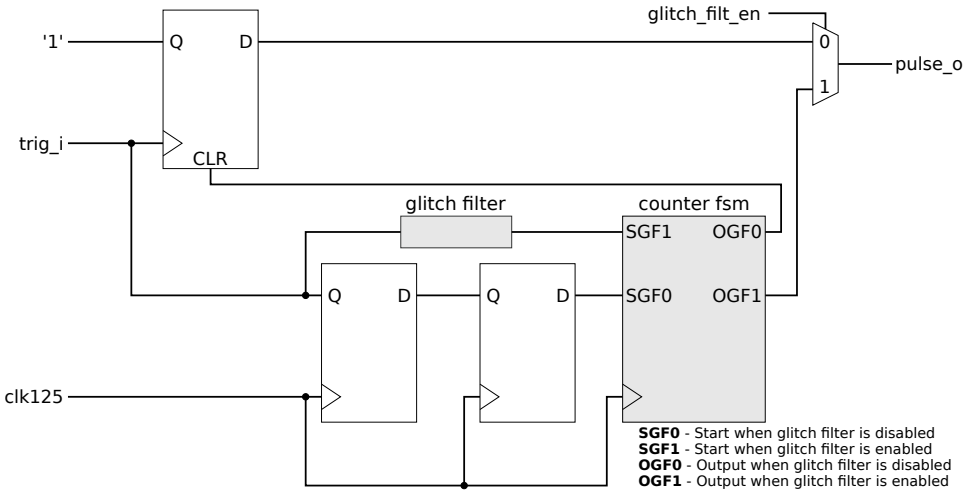

Figure 5: Pulse generator block

The block contains a glitch filter that can be used to decrease sensitivity to glitches in noisy environments. The glitch filter length can be enabled via the *glitch_filt_en_n* input (connected to SW1.1 on the CONV-TTL-BLO). The length of the filter can be set via the *g_glitch_filt_len* generic.

Enabling the glitch filter will lead to the trigger being sampled using *clk125* and introduces leading-edge jitter on the *pulse_o* output. To avoid this leading-edge pulse jitter, the glitch filter can be disabled.

Regardless of whether the glitch filter is enabled or not, the FSM reacts to the rising edge of one of its two start inputs. A rising edge on an input starts the internal counter, which counts up to a maximum value of *g_pulse_width*. The behavior of the outputs are different, depending on the state of the glitch filter.

With the glitch filter disabled, the input pulse enables the input flip-flop, which starts pulse generation. The pulse signal is then synchronized in the *clk125* domain and input to the synchronous counter FSM. The rising edge on *SGF0* triggers the counter, and when the counter reaches the maximum value it sets the *OGF0* output for one clock cycle, which will reset the input flip-flop, thus ending the pulse.

With the glitch filter enabled, the rising edge on *SGF1* sets *OGF1*, and this will be kept high until the counter reaches the maximum value.

6.2 Board-level view

The use of the pulse generator module is put into perspective in this section. Figure 6 shows the pulse replication mechanism on the CONV-TTL-BLO, where the *PG* block is the *ctb_pulse_gen* block.

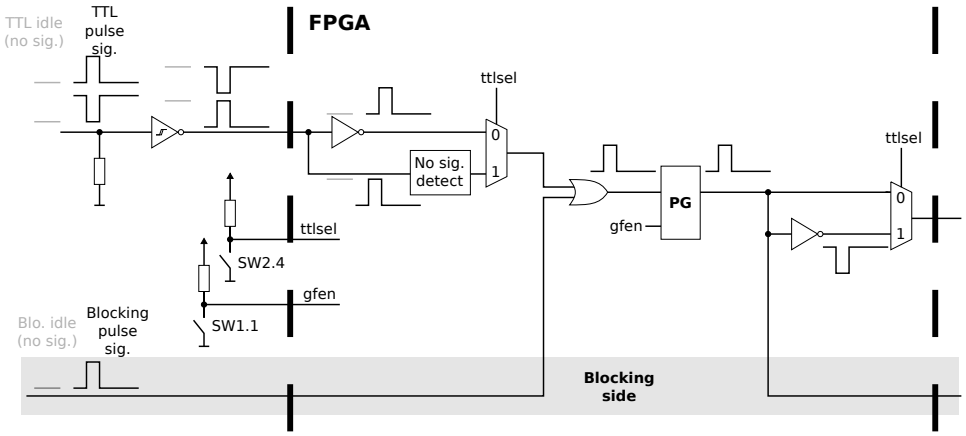

Figure 6: Board-level view of pulse replication mechanism

Considering that the counter FSM in the *ctb_pulse_gen* reacts to rising edges on its inputs, it can now be understood why the *PG* block in Section 4.3 of [1] expects TTL type pulses at its inputs.

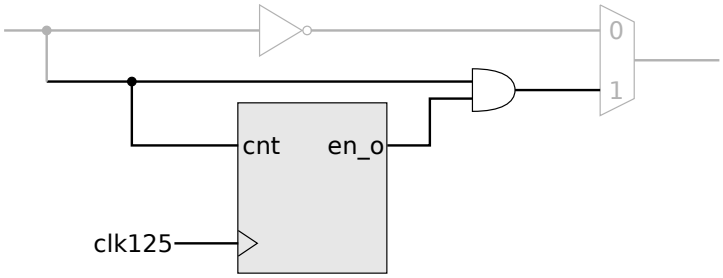

Figure 7: No signal detect block

The implementation of the *no sig. detect* block in Figure 6 is shown in Figure 7. The block is implemented as a counter which keeps the *en_o* signal high as long as it does not reach its maximum value. The counter counts up when the *cnt* input is high. By setting the maximum value of the counter to 12499, it disables the line to the multiplexer if this stays high for 100 μs, thus allowing for blocking pulses at the input of the OR gate. The line is

re-enabled as soon as it goes back low, i.e., when a wire has been plugged in to the channel.

7 Memory-mapped peripherals

This section details the various peripherals mapped on the internal Wishbone bus. Access to these peripherals is made through the two serial lines on the VME P1 connector (*SERCLK*, *SERDAT*). The I^2C-based protocol proposed by ELMA [3] for their VME crates is used to access these peripherals. A bridge module (Section 7.1) translates I^2C transfers into Wishbone transfers.

The complete memory map of the firmware can be found in Appendix A.

7.1 I^2C to Wishbone bridge

The *elma_i2c* module **REFERENCE** implements a bridge between the serial lines on the VME P1 connector using the ELMA I^2C-based protocol [3], and the Wishbone interconnect. The module provides one I^2C slave interface for connecting to an ELMA SysMon and one Wishbone master interface.

Details about the module's implementation can be found in its documentation **REFERENCE**.

7.2 Control and status registers

The status registers implemented in the firmware contain the current firmware version, the position of the on-board switches and the values on RTM detection lines.

No control registers are currently implemented.

See Appendix B for more information.

8 Folder Structure

The folder structure used within the firmware is presented below.

- → ip_cores/
- → conv-ttl-blo/hdl/
  - → bicolor_led_ctrl/
    - → *bicolor_led_ctrl.vhd*
    - → *bicolor_led_ctrl_pkg.vhd*
  - → glitch_filt/
    - → doc/
    - → rtl/
      - → *glitch_filt.vhd*
  - → **release/**
    - → **rtl/**
      - → ***conv_regs.vhd***
    - → **top/**
      - → ***conv_ttl_blo_v2.vhd***
      - → ***conv_ttl_blo_v2.ucf***
  - → ctb_pulse_gen/
    - → rtl/
      - → *ctb_pulse_gen.vhd*
  - → reset_gen/
    - → rtl/
      - → *reset_gen.vhd*
  - → elma_i2c/
    - → doc/
      - → elma_i2c/
      - → i2c_slave/
    - → rtl/
      - → *i2c_slave.vhd*
      - → *elma_i2c.vhd*

The *ip_cores/* folder contains repository files that the firmware uses, such as the Wishbone crossbar (*xwb_crossbar*). The modules that have been developed as part of the CONV-TTL-BLO project 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; documentation files (if any) for

the modules appear under a *doc/* sub-folder. The I^2C bridge module folder also contains the instantiated *i2c_slave.vhd* file (see **REFERENCE TO ELMA_I2C**) and the documentation for it.

The *release/* folder is the main folder in the firmware pack, 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 specific modules in the *rtl/* folder.

9 Getting Around the Code

As described above, the main part of the release firmware can be found in the *conv-ttl-blo/hdl/pts/* folder. The top-level file is *conv_ttl_blo_v2.vhd*.

Ports and signals usually follow the coding guideline at [4]. Most of the top-level ports of the firmware are lower-case versions of their schematics counterparts. 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.

The declarative part of the module architectures is as shown in Figure 8. It starts

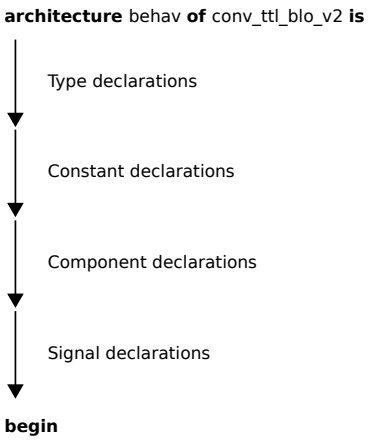

Figure 8: Declarative part of VHDL architecture

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 Figure 8. Types are declared right after the architecture declaration, followed by constant declarations, followed by component declarations, after which the various signals are declared.

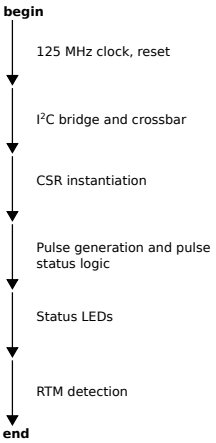

Figure 9: Body of VHDL architecture

The body of the architecture is organised as showin in Figure 9. It begins by instantiating a differential buffer for the 125 MHz system clock and instantiating the *reset_gen* component. Then, the *elma_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. **??**) module is instantiated, followed by logic necessary for each of the tests comprising PTS.

Appendices

A Memory map

Table 4 shows the complete memory map of the firmware. The following sections list the memory map of each peripheral.

Table 4: CONV-TTL-BLO memory map

Periph.	Address		Description
	Base	End	
CSR	0x000	0x010	Control and status register

B Control and status registers

Base address: 0x000

Offset	ELMA reg	Description
0x0	1	Board ID register
0x4	2	Status register
0x8	3	Reserved
0xC	4	Reserved

Reserved addresses read undefined and should be written as 0x00000000.

B.1 Board ID register

Bits	Field	Access	Default	Description
31..0	*id*	R/O	0x424c4f32	Board ID

Field	Description
id	Board ID (ASCII string **BLO2**)

B.2 Status register

Bits	Field	Access	Default	Description
15..0	*fwvers*	R/O	X	Firmware version
23..16	*switches*	R/O	X	Switch status
29..24	*rtm*	R/O	X	RTM detection lines
31..30	*reserved*	R/O	X	

Field	**Description**
fwvers	Firmware version, hex value containing the firmware version – leftmost byte *hex value* is major release *decimal value* – rightmost byte *hex value* is minor release *decimal value* e.g. 0x0101 – v1.01 0x0107 – v1.07 0x0274 – v2.74 etc.
switches	Current switch status bit 0 – SW1.1 bit 1 – SW1.2 ... bit 7 – SW2.4 **1** – switch is **OFF** **0** – switch is **ON**
rtm	RTM detection lines status **0** – line active **1** – line inactive
reserved	Write as '0'; read undefined

References

[1] T.-A. Stana, "CONV-TTL-BLO User Guide." http://www.ohwr.org/documents/263, 06 2013.

[2] "Rear Transition Module detection." http://www.ohwr.org/projects/conv-ttl-blo/wiki/RTM_board_detection.

[3] ELMA, "Access to board data using SNMP and I2C." http://www.ohwr.org/documents/227.

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