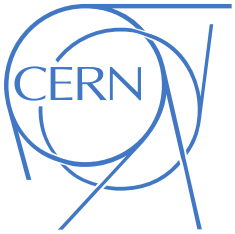
CONV-TTL-BLO HDL Guide

Gateware v2.00
October 29, 2013



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

Revision history

Date	Version	Change
04-07-2013	0.1	First draft
26-07-2013	0.2	Second draft
07-08-2013	1.02	Added pulse rejection to *ctb_pulse_gen*
14-08-2013	1.02	Changed name of *elma_i2c* to *vbcp_wb*
28-10-2013	2.00	Added MultiBoot support to firmware

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 VBCP
- remote reprogramming via I^2C and VBCP

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

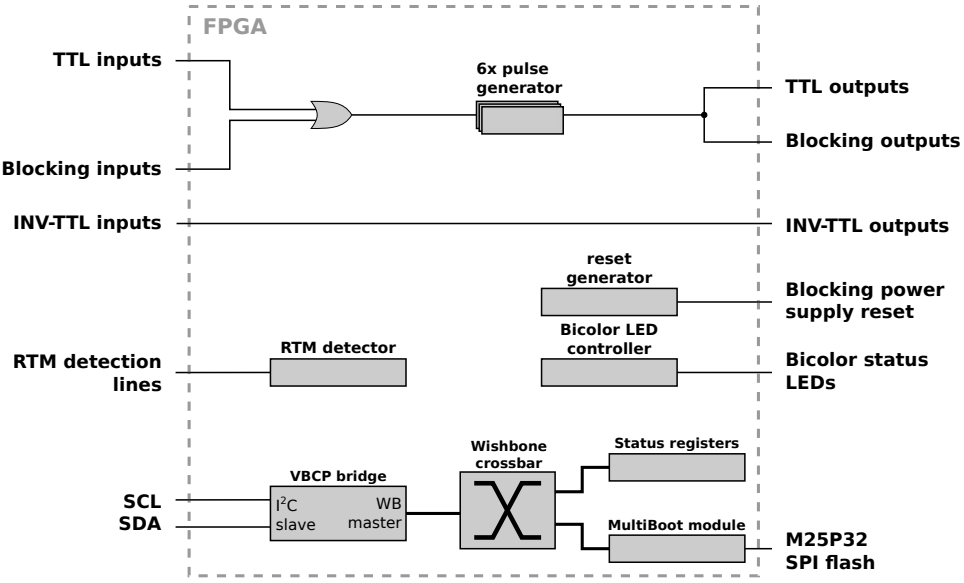


Figure 1: Block diagram of FPGA firmware

Additional documentation

- CONV-TTL-BLO User Guide [1]
- CONV-TTL-BLO Hardware Guide [2]

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

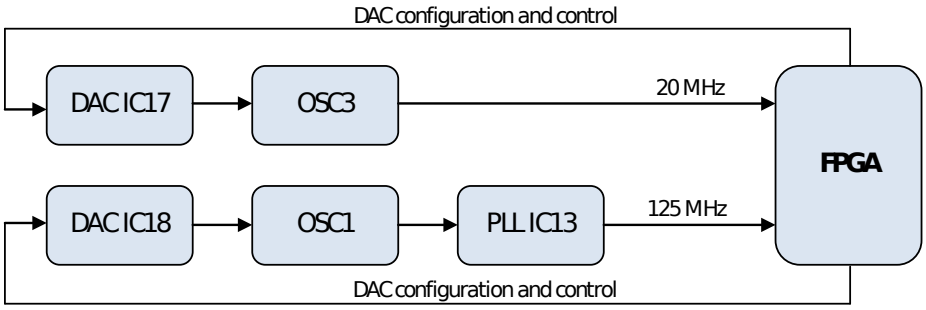


Figure 2: FPGA clock inputs

125 MHz is generated on-board via a Texas Instruments PLL IC from a 25 MHz VCXO.

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 [3]. 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.

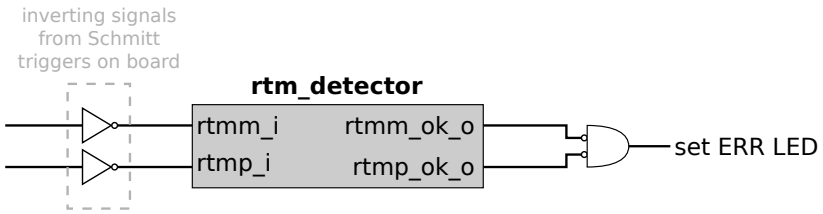


Figure 3: *rtm_detector* block in CONV-TTL-BLO firmware

The status of the RTM detection lines can also be read via their respective fields in the CONV board status register (Appendix A.1).

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 controls lighting each LED in 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.

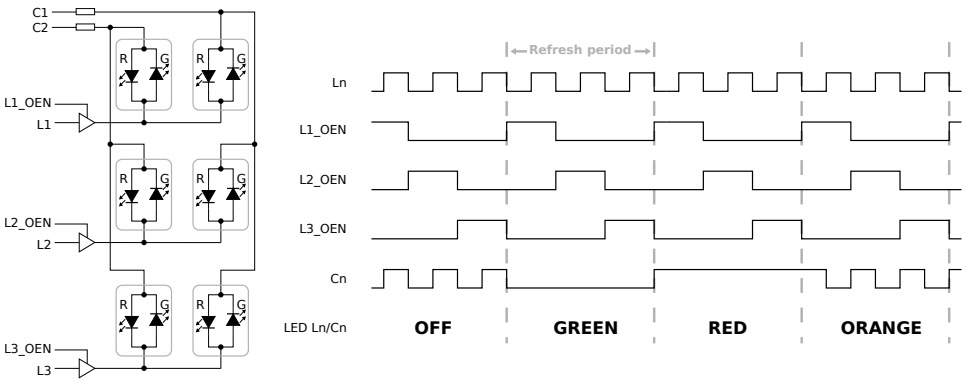


Figure 4: 3x2 bicolor LED matrix control

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*; these 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.

5.1 Board-level view

There are twelve bicolor LEDs on the CONV-TTL-BLO; they are connected in a two-line, six-column pattern controlled by a *bicolor_led_ctrl* block. Table 3 shows the *led_state_i* connections for the bicolor status LEDs in the CONV-TTL-BLO firmware.

Table 3: LED state vector connections in the firmware

Line	Column	LED	LED state bits	Setting
1	1	WHITE_RABBIT_ADDR	*1..0*	
1	2	WHITE_RABBIT_GMT	*3..2*	
1	3	WHITE_RABBIT_LINK	*5..4*	
1	4	WHITE_RABBIT_OK	*7..6*	
1	5	MULTICAST_ADDR_1	*9..8*	
1	6	MULTICAST_ADDR_2	*11..10*	
2	1	I2C	*13..12*	
2	2	TTL	*15..14*	
2	3	ERR	*17..16*	
2	4	PW	*19..18*	
2	5	MULTICAST_ADDR_4	*21..20*	
2	6	MULTICAST_ADDR_8	*23..22*	

The states of the used LEDs can be found in Table 1 of [1]. They are controlled by combinatorial multiplexers. The selection signals to these multiplexers are set throughout the logic.

6 Pulse generator

Entity	*ctb_pulse_gen*	
Generics	*g_pwidth*	Width of the output pulse in *clk_i* cycles
	g_gf_len	Length of glitch filter
Ports	*clk_i*	Clock signal
	rst_n_i	Active-low reset signal
	en_i	Pulse generator enable
	gf_en_n_i	Active-low glitch filter enable
	trig_i	Pulse trigger
	pulse_o	Pulse output
Usage	Output pulse	1.2 μs pulses with min. period of 6 μs

The *ctb_pulse_gen* block generates pulses on the rising edge of the *trig_i* input. The pulse width is configurable via the *g_pwidth* generic. The block also incorporates a glitch filter with a configurable length (*g_gf_len*) that can be used to avoid pulses generated because of glitches at the *trig_i* input.

Pulse widths at the output are limited internally to 1/5 duty cycle, to safeguard the blocking output transformers.

Six *ctb_pulse_gen* blocks (one per channel) are used for generating blocking and TTL pulses at the outputs, based on trigger inputs arriving on the channels. The *ctb_pulse_gen* blocks are configured for 1.2 μs pulses (*g_pwidth = 150*, considering the 8 ns clock input).

6.1 Implementation

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

The glitch filter can be used to decrease sensitivity to glitches in noisy environments. It can be enabled via the *gf_en_n_i* input (connected to SW1.1 on the CONV-TTL-BLO). The length of the filter can be set via the *g_gf_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 left 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_pwidth*. 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 FSM, which extends the pulse

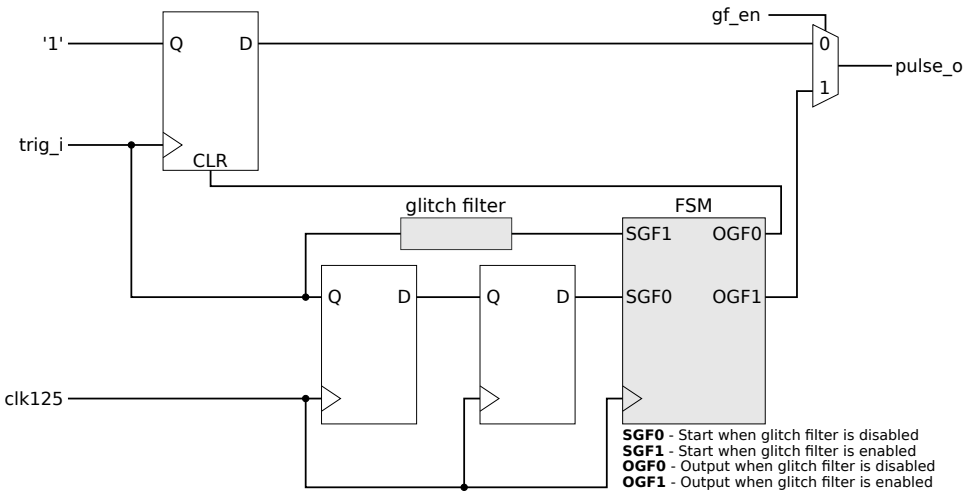


Figure 5: Pulse generator block

to *g_pwidth*. The rising edge on *SGF0* triggers the counter, and when the counter reaches the value corresponding to the selected pulse width, it sets the *OGF0* output, 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 value corresponding to the pulse width.

After the pulse generation period, the FSM goes into a pulse rejection state, where the pulse reset is kept high. If any pulses arrive on the input while the FSM is in this rejection state, they are not replicated at the output. The pulse rejection phase lasts for 4*g_pwidth*, yielding a maximum duty cycle of 1/5 for input pulses.

6.2 Board-level view

Figure 6 shows the pulse replication mechanism on the CONV-TTL-BLO. Here, the *PG* block is the *ctb_pulse_gen* block with the necessary settings. Since the *ctb_pulse_gen* block expects a rising edge at its *trig_i* input in order to generate a pulse at the output, logic external to the block caters for the different types of signals that arrive on CONV-TTL-BLO inputs.

Most of this external logic is on the TTL pulse side, where both TTL and TTL-BAR pulses may arrive. As described in Section 4.3 of [1], if a wire is not plugged in when TTL-BAR pulses are input, a continuous logic high level on the line would inhibit pulses arriving on the blocking side from triggering a pulse generation. This is why the *no sig. detect* block has been implemented.

The block's implementation is shown in Figure 7. It is implemented as a counter which keeps the *en_o* signal high as long as it does not reach its

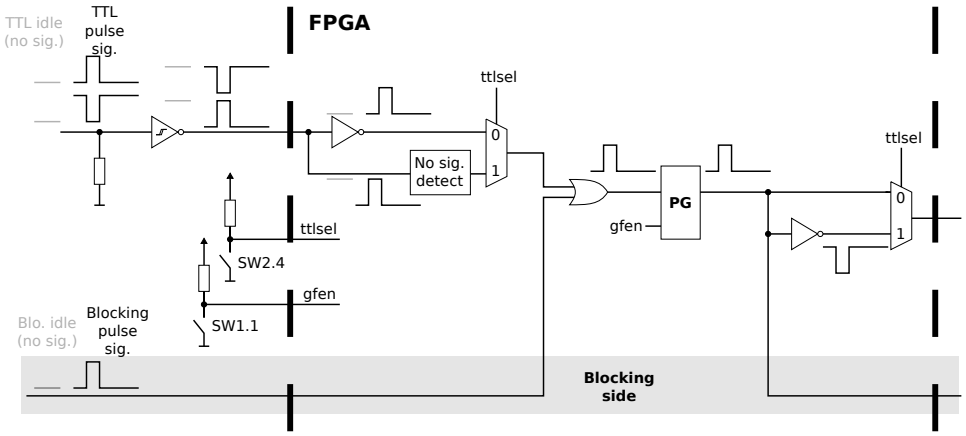


Figure 6: Board-level view of pulse replication mechanism

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.

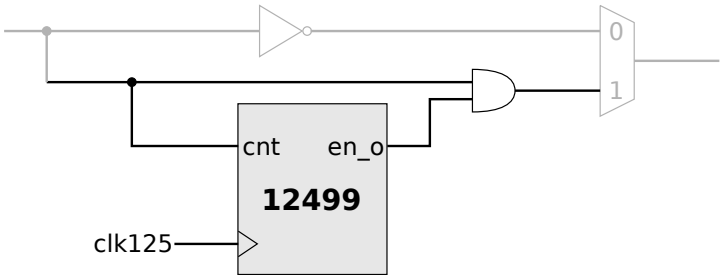


Figure 7: No signal detect block

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 VBCP protocol is used to access these peripherals. A bridge module translates VBCP transfers into Wishbone transfers.

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

7.1 VBCP to Wishbone bridge

The *vbcp_wb* module implements a bridge between the serial lines on the VME P1 connector using VBCP 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, in the *conv_ttl_blo_gw/doc/vbcp_wb/* folder (see Section 8).

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 A.1 for more information.

7.3 MultiBoot control

The MultiBoot module offers the remote reprogramming capabilities for the CONV-TTL-BLO board. It offers a set of registers for controlling writing a bitstream to the M25P32 flash chip and for issuing the remote reprogramming command.

For information on the module, refer to its documentation under the *conv_ttl_blo_gw/doc/multiboot/* folder (see Section 8). The memory map of the module is also present in this manual, for quick reference (see Appendix A.2).

8 Folder Structure

The folder structure for the project is presented below.

- → ../ip_cores/
- → conv-ttl-blo-gw/doc/
  - → hdlguide/
  - → i2c_slave/
  - → multiboot/
  - → vbcp_wb/
- → conv-ttl-blo-gw/hdl/
  - → bicolor_led_ctrl/
    - → *bicolor_led_ctrl.vhd*

- → *bicolor_led_ctrl_pkg.vhd*
- → ctb_pulse_gen/
  - → rtl/
    - → *ctb_pulse_gen.vhd*
- → glitch_filt/
  - → rtl/
    - → *glitch_filt.vhd*
- → multiboot/
  - → rtl/
    - → *multiboot_fsm.vhd*
    - → *multiboot_regs.vhd*
    - → *spi_master.vhd*
    - → *xil_multiboot.vhd*
- → **release/**
  - → **rtl/**
    - → ***conv_regs.vhd***
  - → **top/**
    - → ***conv_ttl_blo.vhd***
    - → ***conv_ttl_blo.ucf***
- → reset_gen/
  - → rtl/
    - → *reset_gen.vhd*
- → vbcp_wb/
  - → rtl/
    - → *i2c_slave.vhd*
    - → *vbcp_wb.vhd*

The *ip_cores/* folder contains repository files that the firmware uses, such as the Wishbone crossbar (*xwb_crossbar*).

Documentation such as this HDL guide and some HDL modules developed as part of the CONV-TTL-BLO project can be found in the *conv-ttl-blo/doc/* folder.

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. The I^2C bridge module folder also contains the instantiated *i2c_slave* module.

The *release/* folder is the main folder in the firmware package, 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/release/* folder. The top-level file is *conv_ttl_blo.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.

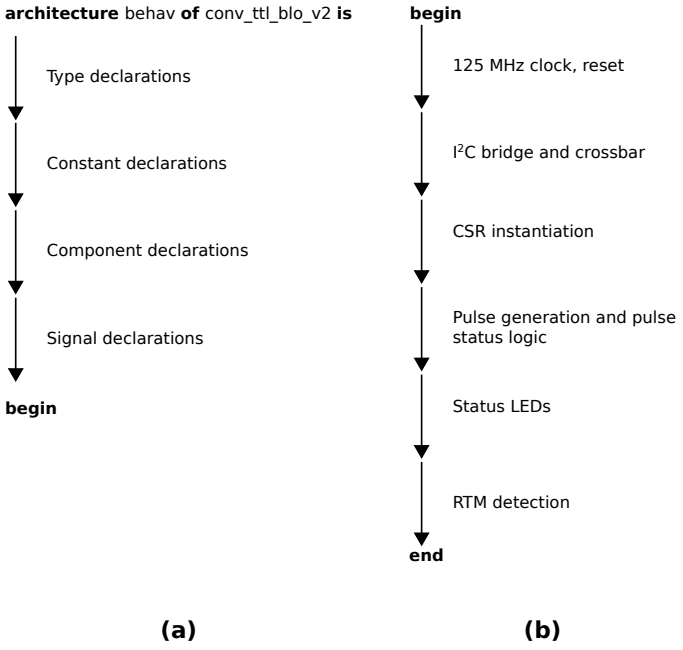


Figure 8: VHDL architecture

The declarative part of the architecture is organized as shown in Figure 8 (a). Types are declared right after the architecture declaration, followed by con- stant declarations, followed by component declarations, after which the var- ious signals are declared.

The body of the architecture is organised as shown in in Figure 8 (b). It begins by instantiating a differential buffer for the 125 MHz system clock and instantiating the *reset_gen* component. Then, the *vbcp_wb* bridge module is instantiated along with the Wishbone crossbar that offers access to the rest of the Wishbone modules in the design. Next, the CONV board CSR module is instantiated, followed by the instantiation of twelve pulse generator modules, six for pulse repetition and six for the pulse LEDs. This is followed by the logic for the status LEDs and the file ends with the RTM detection modules.

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	0x0f	Control and status register
MultiBoot	0x040	0x5f	MultiBoot module

A.1 Control and status registers

Base address: 0x000

Offset	Name	Description
0x0	BID	Board ID register
0x4	SR	Status register
0x8	*Reserved*	Read undefined; write as 0
0xc	*Reserved*	Read undefined; write as 0

A.1.1 Board ID register

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

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

A.1.2 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 – 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

A.2 MultiBoot module

Base address: 0x040

Offset	Name	Description
0x00	CR	Control Register
0x04	SR	Status register
0x08	GBBAR	Golden Bitstream Base Address Register
0x0c	MBBAR	Multiboot Bitstream Base Address Register
0x10	FAR	Flash access register
0x14	*Reserved*	Read undefined; write as 0
0x18	*Reserved*	Read undefined; write as 0
0x1c	*Reserved*	Read undefined; write as 0

A.2.1 CR – Control Register

Bits	Field	Access	Default	Description
31..18	*Reserved*	–	X	
17	IPROG	R/W	0	IPROG bit
16	IPROG_UNL	R/W	0	IPROG unlock bit
15..7	*Reserved*	–	X	
6	RDCFGREG	R/W	0	Read config register
5..0	CFGREGADR	R/W	0	Config register address

Field	Description
Reserved	Write as '0'; read undefined
IPROG	When 1, it triggers the FSM to send the IPROG command to the ICAP controller This bit needs to be unlocked by setting the IPROG_UNL bit in a previous cycle
IPROG_UNL	Unlock bit for the IPROG command. This bit needs to be set to 1 prior to writing the IPROG bit
RDCFGREG	Initiate a read from the FPGA configuration register at address CFGREGADR This bit is automatically cleared by hardware
CFGREGADR	The address of the FPGA configuration register to read (see Configuration Registers section in [5])

A.2.2 IMGR – Image Register

Bits	Field	Access	Default	Description
31..17	*Reserved*	–	X	
16	VALID	R/O	0	Image register is valid
15..0	CFGREGIMG	R/O	0	Config. register image

Field	Description
Reserved	Write as '0'; read undefined
VALID	A read has been performed from the FPGA configuration register at address CR.CFGREGADR, and its value is present in CFGREGIMG
CFGREGIMG	Contains the value of the FPGA configuration register; validated by the VALID bit (see Configuration Registers section in [5])

A.2.3 GBBAR – Golden Bitstream Base Address Register

Bits	Field	Access	Default	Description
31..24	OPCODE	R/W	0	Flash chip read op-code
23..0	GBA	R/W	0	Golden Bitstream Address

Field	Description
OPCODE	Op-code for the flash chip read (or fast-read) command. Get this value from the flash chip datasheet
GBA	Start address of the Golden bitstream on the flash chip

A.2.4 MBBAR – MultiBoot Bitstream Base Address Register

Bits	Field	Access	Default	Description
31..24	OPCODE	R/W	0	Flash chip read op-code
23..0	MBA	R/W	0	MultiBoot Bitstream Address

Field	Description
OPCODE	Op-code for the flash chip read (or fast-read) command. Get this value from the flash chip datasheet
MBA	Start address of the MultiBoot bitstream on the flash chip

A.2.5 FAR – Flash Access Register

Bits	Field	Access	Default	Description
31..29	*Reserved*	–	0	Flash chip read op-code
28	READY	R	1	SPI access status
27	CS	R/W	0	SPI chip select
26	XFER	R/W	0	Start SPI transfer
25..24	NBYTES	R/W	0	Number of bytes to send
23..16	DATA[2]	R/W	0	Data at offset 2
15..8	DATA[1]	R/W	0	Data at offset 1
7..0	DATA[0]	R/W	0	Data at offset 0

Field	Description
Reserved	Write as '0'; read undefined
READY	SPI transfer ready; NBYTES have been sent to the flash chip, and NBYTES read from the chip present in DATA fields
CS	SPI chip select. Note that this pin has opposite polarity than the normal SPI chip select pin: '1' – flash chip is selected (CS pin = 0) '0' – flash chip is not selected (CS pin = 1)
XFER	'1' – starts SPI transfer This bit is automatically cleared by hardware
NBYTES	Number of DATA fields to send in one transfer 0 – send 1 byte (DATA[0]) 1 – send 2 bytes (DATA[0], DATA[1]) 2 – send 3 bytes (DATA[0], DATA[1], DATA[2]) 3 – *Reserved*
DATA[2]	Write this register with the value of data byte 2 After an SPI transfer, this register contains the value of data byte 2 read from the flash
DATA[1]	Write this register with the value of data byte 1 After an SPI transfer, this register contains the value of data byte 1 read from the flash
DATA[0]	Write this register with the value of data byte 0 After an SPI transfer, this register contains the value of data byte 0 read from the flash

References

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

[2] T.-A. Stana, "CONV-TTL-BLO Hardware Guide." http://www.ohwr.org/documents/282, 07 2013.

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

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

[5] Xilinx, "UG380 - Spartan-6 Configuration Guide." http://www.xilinx.com/support/documentation/user_guides/ug380.pdf, Jan. 2013. v2.5.