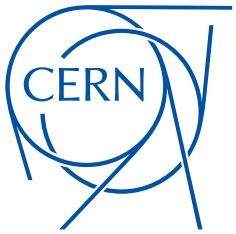
# CONV-TTL-BLO HDL Guide

Gateware v1.0
January 5, 2014




**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* |
| 29-10-2013 | 1.03 | Added MultiBoot support to firmware |
| 20-11-2013 | 1.04 | Changed system clock to 20 MHz |
| 05-01-2014 | 1.05 | Updated folder structure and renamed *vbcp_wb* to *i2c_bridge* |

# Contents

**1 Introduction** **1**

**2 Folder Structure** **2**

**3 Getting Around the Code** **3**

**4 FPGA Clocks** **5**

**5 Reset generator** **5**

**6 RTM detection** **6**

**7 Bicolor LED controller** **7**
7.1 Board-level view . . . . . 8

**8 Pulse generator** **9**
8.1 Implementation . . . . . 9
8.2 Board-level view . . . . . 10

**9 Memory-mapped peripherals** **11**
9.1 $I^2C$ to Wishbone bridge . . . . . 12
9.2 Control and status registers . . . . . 12
9.3 MultiBoot control . . . . . 12

**Appendices** **13**

**A Memory map** **13**
A.1 Control and status registers . . . . . 13
A.1.1 Board ID register . . . . . 13
A.1.2 Status register . . . . . 13
A.1.3 Control register . . . . . 14
A.2 MultiBoot module . . . . . 15
A.2.1 CR – Control Register . . . . . 16
A.2.2 SR – Status Register . . . . . 17
A.2.3 GBBAR – Golden Bitstream Base Address Register . 17
A.2.4 MBBAR – MultiBoot Bitstream Base Address Register 17
A.2.5 FAR – Flash Access Register . . . . . 18

# List of Figures

1 Block diagram of FPGA firmware . . . . . 1
2 VHDL architecture of the release firmware . . . . . 4
3 FPGA clock inputs . . . . . 5

4 *rtm_detector* block in CONV-TTL-BLO firmware . . . . . 6
5 3x2 bicolor LED matrix control . . . . . 7
6 Pulse generator block . . . . . 10
7 Board-level view of pulse replication mechanism . . . . . 11
8 No signal detect block . . . . . 11

## List of Tables

1 Gateware projects in the repository . . . . . 3
2 Clock domains . . . . . 5
3 LED state input . . . . . 8
4 LED state vector connections in the firmware . . . . . 8
5 CONV-TTL-BLO memory map . . . . . 13

## List of Abbreviations

| | |
|---|---|
| DAC | Digital-to-Analog Converter |
| FPGA | Field-Programmable Gate Array |
| FSM | Finite-State Machine |
| IC | Integrated Circuit |
| $I^2C$ | Inter-Integrated 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 $I^2C$
- remote reprogramming via $I^2C$

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

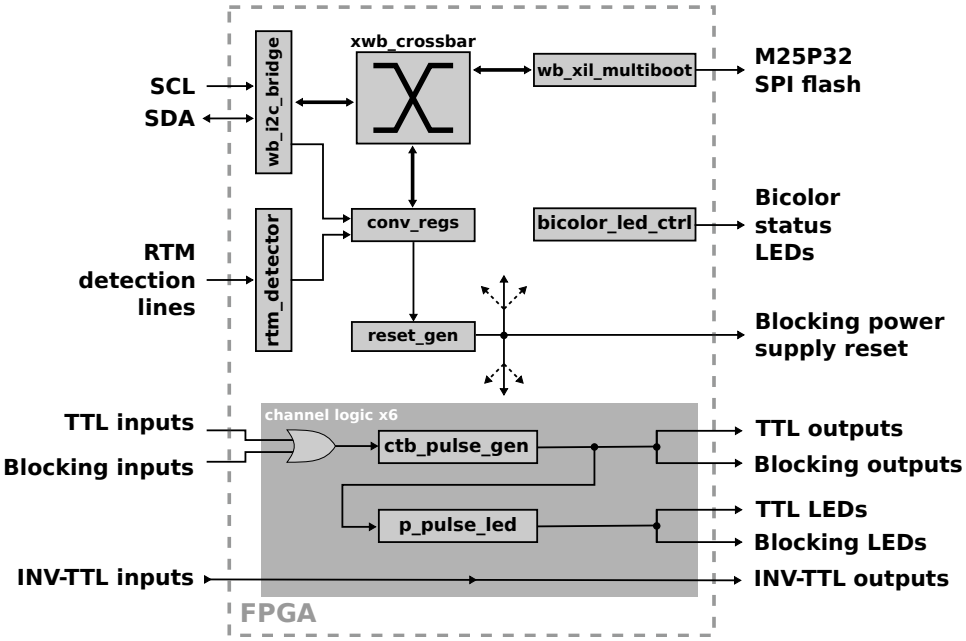



Figure 1: Block diagram of FPGA firmware

## Additional documentation

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

# 2 Folder Structure

The folder structure for the project is presented below.

- → conv-ttl-blo-gw/
  - → doc/
    - → hdlguide/
  - → ip_cores/
    - → general-cores/
  - → modules/
    - → Release/
    - → pulsetest/
    - → ctb_pulse_gen.vhd
    - → reset_gen.vhd
    - → rtm_detector.vhd
  - → sim/
  - → syn/
    - → Release/
    - → pulsetest/
    - → regtest/
  - → top/
    - → Release/
      - → conv_ttl_blo.ucf
      - → conv_ttl_blo.vhd
    - → pulsetest/
      - → pulsetest.ucf
      - → pulsetest.vhd
    - → regtest/
      - → pulsetest.ucf
      - → pulsetest.vhd

Gateware files are organized on a per type-of-project basis. There are two different types of projects for CONV-TTL-BLO gateware: the *release project* and *test projects*. The release project is the latest production firmware version, that goes on the CONV-TTL-BLO board used in the field. Test projects are meant to be downloaded to a CONV-TTL-BLO for testing the CONV-TTL-BLO system under long-term test conditions. The projects present in the repository at the time of writing of this document are presented in Table 1.

Table 1: Gateware projects in the repository

| Project | Description |
|---|---|
| *conv_ttl_blo* | Design-wide release project to be used in the field |
| *regtest* | Long-term test for testing the $I^2C$ communication by writing to a RAM on the FPGA |
| *pulsetest* | Long-term test for testing pulse repetition on the CONV-TTL-BLO |

As can be seen from the folder structure above, gateware files are organized in the *modules/*, *syn/* and *top/* folders following this project convention. Files in these folders (where relevant) are organized in the *Release/*, *pulsetest/* and *regtest/* folders, where the *Release/* folder of course represents the release firmware and the other two are test projects, as their names suggest.

HDL files are organized into modules and top-level files. Modules are blocks used within the design, while top-level files combine modules together into a design. Modules relevant for the whole design are stored directly under the *modules/* folder, while modules specific to a certain project are stored within the project's sub-folder in the *modules/* folder. Apart from the top folder of the design, the *top/* folder for each project also contains the .ucf constraints file for synthesis.

One place where the project structure is not necessarily enforced is the *sim/* folder. This folder is meant to contain files relevant for simulation of various modules within the design and as such can be composed of folders named after the component to be simulated.

The *syn/* folder holds the actual project files for Xilinx ISE, as well as other various output files from ISE. Each ISE project, be it for release or test project, together with its output files, is contained within its own sub-folder in the *syn/* folder.

# 3 Getting Around the Code

Code in the top-level files is organized in code sections. A code section is a piece of code pertaining to a certain part of the design, where component instantiations and input and output port assignments are made. For example, there is a section pertaining to pulse repetition, where there is a generate block to generate the logic necessary for pulse repetition on each channel, including the pulse status LEDs.

Ports and signals usually follow the coding guideline at [3]. 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

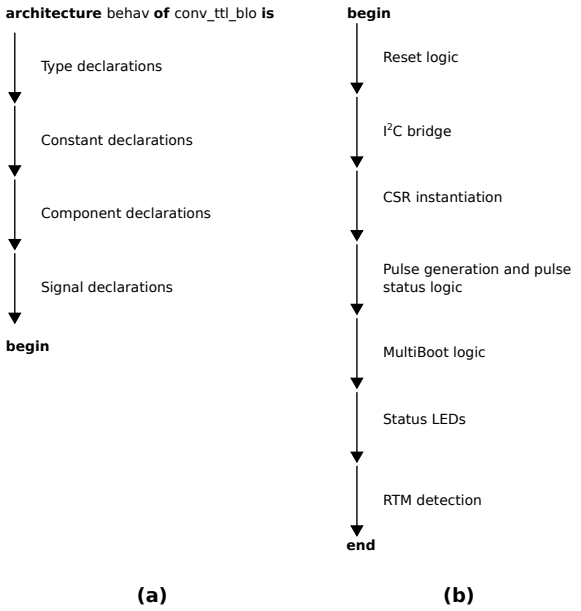



Figure 2: VHDL architecture of the release firmware

could not be syntactically represented in VHDL, or net names that have been made clearer in VHDL code.

The declarative part of the architecture is organized as shown in Figure 2 (a). 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 in code sections as shown in Figure 2 (b). It starts with the instantiation of the *reset_gen* component which generates the board-wide reset.

Then, in the $I^2C$ section, the $I^2C$ bridge component is instantiated, the logic for lighting the I2C front panel LED is defined, as well as the CWDTO bit in the SR (see Appendix A.1). Before the end of this section, an *xwb_crossbar* is instantiated to communicate to the various peripherals.

The $I^2C$ section is followed by the main code section of the design, the pulse generation section. Here, a generate block is used to generate the logic for each channel, including the instantiation of a *ctb_pulse_gen* block, the implementation of the no signal detect block (see Section 8.2), the output pulse assignments and the logic for flashing the pulse LED for 262 ms.

After the pulse generation section, the MultiBoot component is instantiated and connected to the SPI pins to and from the on-board flash chip.

Two more short code sections remain, that in which the *bicolor_led_ctrl* component is connected to the line and column outputs to the bicolor LED matrix, and the connection of the RTM detection inputs to the *rtm_detector* component.

# 4 FPGA Clocks

There are two clock signals input to the FPGA (Figure 3). 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.

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 2 lists the clock domains used in the firmware.

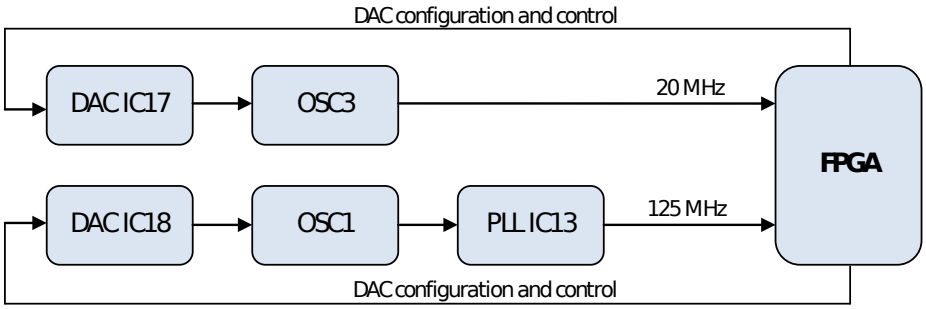



Figure 3: FPGA clock inputs

Table 2: Clock domains

| Clock domain | Frequency | Comments |
|---|---|---|
| *clk20_vcxo_i* | 20 MHz | Global clock input to all sequential logic |

# 5 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 | 100 *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), 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. The *rst_i* pin is tied in the design to the first bit in the control register (CR, see Appendix A.1), which has to be first unlocked by writing the RST_UNLOCK bit. Both these registers are implemented in the top-level file of the design.

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 100 *ms*.

# 6 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 [4]. 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. The status of the RTM detection lines can also be read via their respective fields in the CONV board status register (see Appendix A.1).

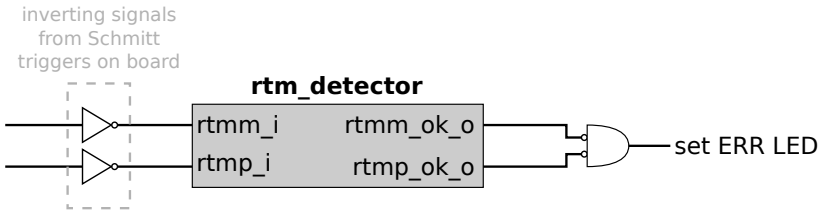



Figure 4: *rtm_detector* block in CONV-TTL-BLO firmware

# 7 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 lights each LED in the LED matrix sequentially at the refresh rate given by the user.

Figure 5 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 then to the LEDs. The FPGA outputs for line output enables (L_OEN) are connected to the output enable of the tri-state buffers.

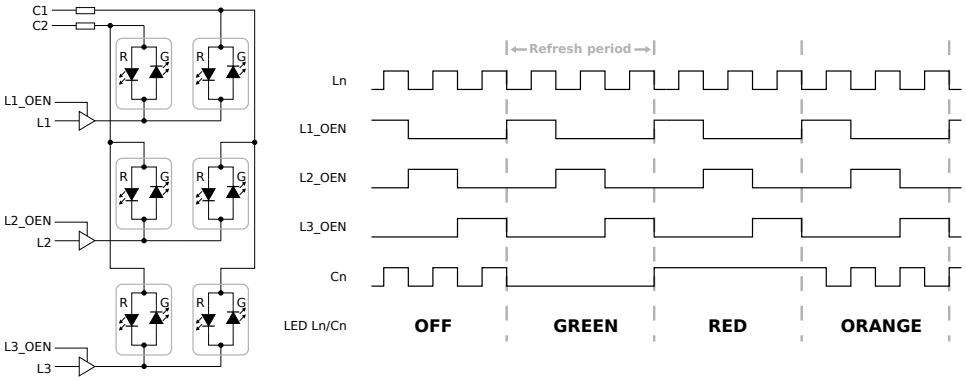



Figure 5: 3x2 bicolor LED matrix control

The two-bit *led_state_i* vector can be used to control the color of each LED. Table 3 lists the values that should be input on *led_state_i* to get the needed color, as well as constant definitions provided in the *bicolor_led_ctrl_pkg.vhd* file for setting the color of the LED via *led_state_i*.

Table 3: 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.

## 7.1 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 4 shows the *led_state_i* connections for the bicolor status LEDs in the CONV-TTL-BLO firmware.

Table 4: LED state vector connections in the firmware

| Line | Column | LED | LED state bits |
|---|---|---|---|
| 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 the CONV-TTL-BLO User Guide [1]. They are controlled by combinatorial multiplexers. The selection signals to these multiplexers are set throughout the logic.

# 8 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 in *clk_i* cycles |
| **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 $\mu$s pulses with min. period of 6 $\mu$s |

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 $\mu$s pulses (*g_pwidth = 24*, considering the 50 ns clock input).

## 8.1 Implementation

Figure 6 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 *clk20_vcxo_i* 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 in order to assure a pulse with the length *g_pwidth*. The behavior of the outputs is 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 *clk20_vcxo_i* domain and input to the synchronous FSM, which extends the

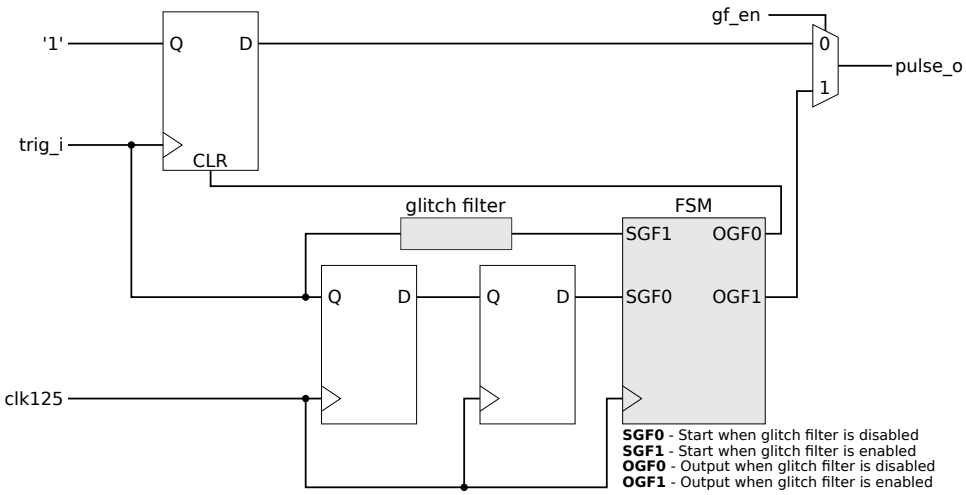



Figure 6: Pulse generator block

pulse 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.

Note that due to the fact that the counter starts counting up from zero and delays in the glitch filter when it is enabled, the maximum value of the internal counter is not *g_pwidth*. Instead, the counter counts up to a pair of VHDL constants defined in the code. These constants assure the pulse at the output is kept high for a number of *g_pwidth* cycles of the *clk_i* signal.

## 8.2 Board-level view

Figure 7 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

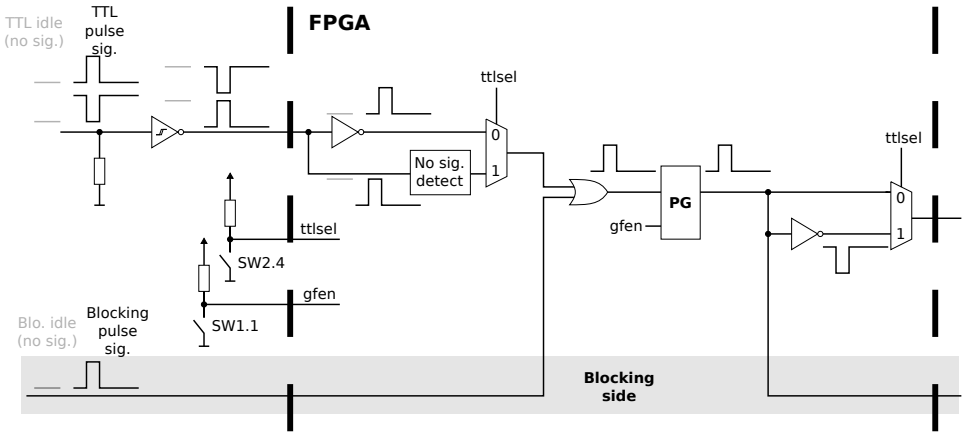



Figure 7: Board-level view of pulse replication mechanism

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 signal detect* block has been implemented.

The block's implementation is shown in Figure 8. It 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 1999, it disables the line to the multiplexer if this stays high for 100 $\mu s$, 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 into the channel.

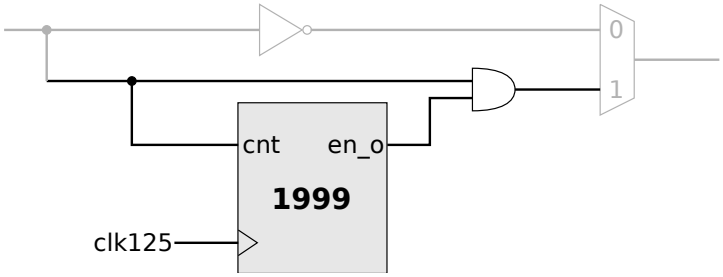



Figure 8: No signal detect block

# 9 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). A protocol based on $I^2C$

is used to access these peripherals. The protocol, as well as the bridge component translating $I^2C$ accesses into Wishbone accesses, are defined in the bridge component's documentation.

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

## 9.1 $I^2C$ to Wishbone bridge

The *i2c_bridge* module implements a bridge to translate $I^2C$ accesses on the VME P1 connector into Wishbone accesses on the FPGA. 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.

## 9.2 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.

## 9.3 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 the module's documentation. The memory map of the module is also present in this manual, for quick reference (see Appendix A.2).

# Appendices

## A Memory map

Table 5 shows the complete memory map of the firmware. The following sections list the memory map of each peripheral.

Table 5: CONV-TTL-BLO memory map

| Periph. | Address | | Description |
|---|---|---|---|
| | Base | End | |
| CSR | 0x000 | 0x00f | Control and status register |
| MultiBoot | 0x040 | 0x05f | MultiBoot module |

### A.1 Control and status registers

Base address: 0x000

| Offset | Name | Description |
|---|---|---|
| 0x0 | BID | Board ID register |
| 0x4 | SR | Status register |
| 0x8 | CR | Control register |

#### A.1.1 Board ID register

| Bits | Field | Access | Default | Description |
|---|---|---|---|---|
| 31..0 | ID | R/O | 0x54424c4f | Board ID |

| Field | Description |
|---|---|
| ID | Board ID (ASCII string **TBLO**) |

#### A.1.2 Status register

| Bits | Field | Access | Default | Description |
|---|---|---|---|---|
| 7..0 | FWVERS | R/O | X | Firmware version |
| 15..8 | SWITCHES | R/O | X | Switch status |
| 21..16 | RTM | R/O | X | RTM detection lines |
| 22 | CWDTO | R/W | 0 | Communication watchdog timeout |
| 31..23 | *Reserved* | – | X | |

| Field | Description |
|---|---|
| FWVERS | Firmware version<br>– leftmost nibble *hex value* is major release *decimal value*<br>– rightmost nibble *hex value* is minor release *decimal value*<br>e.g.<br>0x11 – v1.1<br>0x1e – v1.15<br>0x20 – v2.0<br>etc. |
| SWITCHES | Current switch status<br>bit 0 – SW1.1<br>bit 1 – SW1.2<br>...<br>bit 7 – SW2.4<br>**1** – switch is **OFF**<br>**0** – switch is **ON** |
| RTM | RTM detection lines status<br>**0** – line active<br>**1** – line inactive |
| CWDTO | Communication watchdog timeout status<br>**0** – watchdog idle<br>**1** – communication error has occured and watchdog timer fired<br>This bit is cleared by writing a '1' to it |
| *Reserved* | Write as '0'; read undefined |

### A.1.3 Control register

| Bits | Field | Access | Default | Description |
|---|---|---|---|---|
| 0 | RST_UNLOCK | R/W | 0 | Reset bit unlock |
| 1 | RST | R/W | 0 | Reset bit |
| 31..2 | *Reserved* | – | X | |

| Field | Description |
|---|---|
| RST_UNLOCK | Reset bit unlock<br>**0** – RST bit locked, cannot be written<br>**1** – RST bit unlocked, can be written |
| RST | Reset bit<br>**0** – Idle<br>**1** – Initiate a system reset<br>This bit needs to be unlocked by writing a '1' to the RST_UNLOCK bit in a previous cycle. A write to this bit while RST_UNLOCK = '0' has no effect.<br>Writing this bit to 1 with RST_UNLOCK = '1' will issue a system reset and the communication to the board will be lost for approx. 100 ms |
| *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 |

### A.2.1 CR – Control Register

| Bits | Field | Access | Default | Description |
|---|---|---|---|---|
| 31..18 | *Reserved* | – | X | |
| 17 | IPROG | R/W | 0 | IPROG bit |
| 16 | IPROG_UNLOCK | 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 | Start IPROG sequence<br>**0** – Idle<br>**1** – Start the IPROG sequence<br>This bit needs to be unlocked by setting the IPROG_UNLOCK bit in a previous cycle. Any write to this bit with IPROG_UNLOCK = '0' has no effect. Writing this bit to '1' with IPROG_UNLOCK = '1' will issue the IPROG sequence and communication to the board will be lost until reprogramming is completed |
| IPROG_UNLOCK | Unlock bit for the IPROG command<br>**0** – IPROG bit locked, cannot be written<br>**1** – IPROG bit unlocked, can be written |
| RDCFGREG | Read FPGA configuration register<br>**0** – Idle<br>**1** – Initiate read from configuration register at address CFGREGADR<br>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 SR – Status Register

| Bits | Field | Access | Default | Description |
|---|---|---|---|---|
| 31..18 | *Reserved* | – | X | |
| 17 | MWDTO | R/W | 0 | Multiboot watchdog timeout |
| 16 | IMGVALID | R/O | 0 | Image register is valid |
| 15..0 | CFGREGIMG | R/O | 0 | Config. register image |

| Field | Description |
|---|---|
| *Reserved* | Write as '0'; read undefined. |
| MWDTO | The watchdog of the MultiBoot FSM has timed out This bit is cleared by writing a '1' to it |
| IMGVALID | 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 (see Configuration Registers section in [5]); validated by the IMGVALID bit |

### 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:<br>**0** – Flash chip is not selected (CS pin = 1)<br>**1** – Flash chip is selected (CS pin = 0) |
| XFER | Start SPI transfer<br>**1** – Idle<br>**1** – Start SPI transfer<br>This bit is automatically cleared by hardware |
| NBYTES | Number of DATA fields to send in one transfer<br>**0** – Send 1 byte (DATA[0])<br>**1** – Send 2 bytes (DATA[0], DATA[1])<br>**2** – Send 3 bytes (DATA[0], DATA[1], DATA[2])<br>**3** – *Reserved* |
| DATA[2] | Write this register with the value of data byte 2<br>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<br>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<br>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] P. Loschmidt, N. Simanić, C. Prados, P. Alvarez, and J. Serrano, "Guidelines for VHDL Coding," 04 2011. http://www.ohwr.org/documents/24.

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

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