CONV-TTL-BLO
User Guide

Carlos Gil Soriano
BE-CO-HT
carlos.gil.soriano@cern.ch

February 23, 2012

Abstract

This *User's Guide* covers the following topics:

- HDL global structure.
- Golden image memory mapping.
- Register mapping of all the HDL cores.
- Instructions for accesing to the functionalities.

Contents

1 HDL global structure

The following schema is used as a reference for the HDL development:

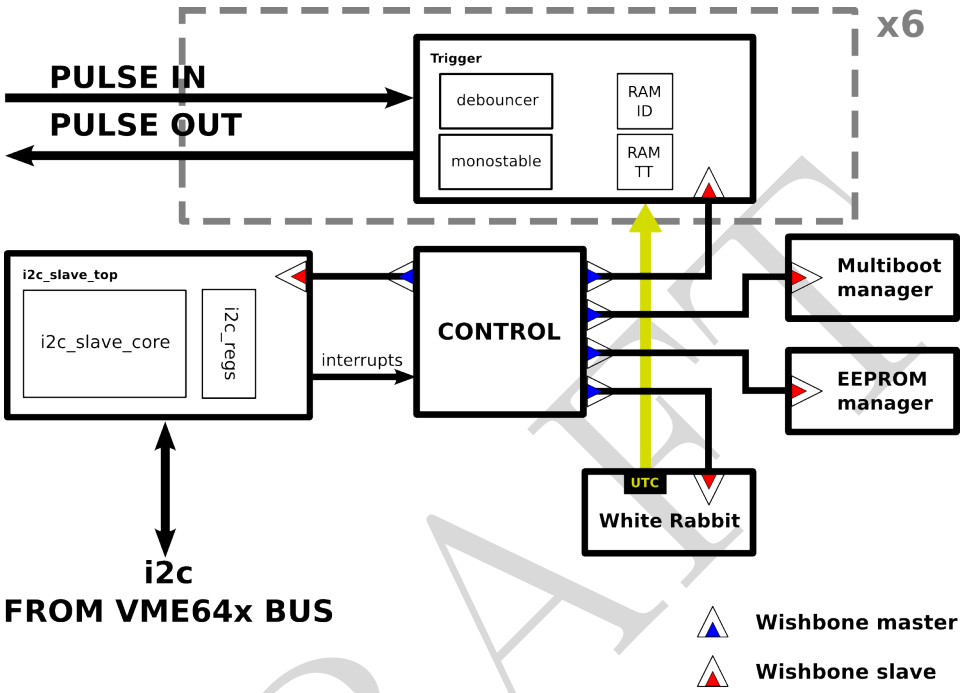

Figure 1: CONV-TTL-BLO HDL structure

1.1 Control

It is the part that bridges the I2C frames to the correct wishbone module. The tasks it is responsible of are:

- Correctly power-up the rest of the modules.
- Provide connectivity of all the wishbone registers via I2C. It manages control access to the registers.

1.2 I2C slave

An I2C slave is needed to receive the frames from the VME64x SERA and SERB pins in P1 connector. This module will communicates with *control hdl module* in the following fashion:

- It connects as a wishbone **slave** to control hdl core and provides interrupt lines for data reception and transmission.

The main reason of implementing a wishbone slave is that by dividing data reception -*i2c slave*- from control access -*control*- the development is more reliable and clear.
This module offers configuration registers to ease the task of data assembly –for instance, the communication schema in ELMA crate.

1.3 Trigger

The trigger manages the pulse repetition. The parameters handled that affect the pulse repetition are:

- **Debouncing** stages of the input pulse.
- **Pulse length** of the output pulse according to Standard Blocking definition.
- **Minimun spacing between pulses** to let the magnetizing current of the transformer be drained off. It is also refered in other documents as *Inactivity timeout upon output pulse is outputted*.

The pulses must be time-tagged. It can be achieved either by a lossy time-tagging via i2c, or with a precise one via White Rabbit. Every time-tag has appended event identifiers (metadata).

1.3.1 Time-tagging Format

The format of the time-tags should be defined. At this moment, an implementation with 96 bits for timestamping with 32 bits of metadata is the default. A record of the last 256 time-tags per channel is hold in the FPGA.

1.4 Multiboot manager

The task of the *Multiboot manager* is to manage a golden bitstream and another one to update the FPGA from. From within this module a FPGA reprogramming command is issued.

1.5 EEPROM manager

The module responsible to write into the EEPROM, read it back and reprogramming the memory module. It should be targeted to interface directly with a MICRON M25P32-VMF6P memory. It will be able to write the MAC address that will be used by White Rabbit and block memory parts of the EEPROM.

1.6 White Rabbit core

Provides precise timestamping.

2 Golden image memory mapping

To access the devices thanks to *control module* the following memory map applies:

NUMBER	DEVICE	FIRST WISHBONE ADDRESS	LAST WISHBONE ADDRESS
0	Control	0x0100	0x01FF
1	I2C slave	0x0200	0x02FF
2	Trigger 1	0x0300	0x03FF
3	Trigger 2	0x0400	0x04FF
4	Trigger 3	0x0500	0x05FF
5	Trigger 4	0x0600	0x06FF
6	Trigger 5	0x0700	0x07FF
7	Trigger 6	0x0800	0x08FF
8	Multiboot manager	0x0900	0x09FF
9	EEPROM manager	0x0A00	0x0AFF
10	White Rabbit core	0x0B00	0x0BFF
11	EEPROM memory	0x1000	0x1FFF

3 Control module

4 I2C slave module

4.1 Structure

The i2c module contains several blocks related the following way:

– i2c_slave_top.vhd
—— i2c_regs.vhd
—— i2c_slave_core.vhd
——— FIFO_dispatcher.vhd
——— FIFO_stack.vhd
——— gc_counter.vhd
——— gc_ff.vhd
——— i2c_bit.vhd

4.2 Interrupting lines offered

4.2.1 ind_wb_addr

This interrupting signal issues when a *indirect wishbone address* has been received. It is notified right after the first CTR0[BIA] + 1 bytes upon the reception of the I2C byte address packet. This signal is vital for correctly prefetching when a I2C read operation is requested.

4.2.2 inst_rd

This signal is issued when a read operation directed by an external master over the HDL slave core is finished. That means it will be generated after the last data byte has been sent by the HDL core.

4.2.3 inst_wr

This signal is issued when a write operation directed by an external master over the HDL slave core is finished. That means it will be generated after the last data byte has been received.

4.3 Registers

4.3.1 STA

The STA register is a read-only register. It control the general enable and reset of the module. It also contains the current value of the finite state machine of the i2c modue (useful for easy debugging).

Bits	Field	Meaning
0	EN	General ENable
1	RST	General ReSeT
2	RD_WRN_INST	Reserved
3	A_RX	
4	A_TX	
8-5	x	Reserved
15-9	i2c_sla_fsm	i2c fsm
31-16	Not used	

4.3.2 PRE

The PRE register is a write-read register. Right now it is not used.

Bits	Field	Meaning
15-0	PRE	PREscaler value
31-16	-	Not used

4.3.3 CTR0

The CTR0 register is a write-read register. It controls the indirect addres and holds the I2C address (which in the case of *CONV-TTL-BLO* will be connected to VME64x geographical address pins).

Bits	Field	Meaning
0	EN	general ENable
1	RST	general ReSeT
2	PEN	Prescaler ENable
5-3	x	Reserved
7-6	BIA	Bytes Indirect Addressing
14-8	A[6:0]	I2C address
15	x	Reserved
31-16	-	Not used

4.3.4 CTR1

The CTR1 register is a write-read register. It shows the fsm of the separate *read* and *write* fsms.

Bits	Field	Meaning
7-0	RDS	fsm status: ReaD Status
15-8	WDS	fsm status: WRite Status
31-16	-	Not used

4.3.5 DRXA

The DRXA register is a read-only register. It holds the last four received bytes through the I2C. DRX0 has the most recent byte received from the serial interface. DRX3 has the oldest byte received.

Bits	Field	Meaning
7-0	DRX0	Data RX register 0
15-8	DRX1	Data RX register 1
23-16	DRX2	Data RX register 2
31-24	DRX3	Data RX register 3

4.3.6 DRXB

The DRXB register is a read-only register. It holds the fifth and sixth latest received bytes, respectively.

Bits	Field	Meaning
7-0	DRX4	Data RX register 0
15-8	DRX5	Data RX register 1
31-16	-	Not used

4.3.7 DTX

The DTX register is a write-read register. It shows the fsm of the separate *read* and *write* fsms.

Bits	Field	Meaning
7-0	RDS	fsm status: ReaD Status
15-8	WDS	fsm status: WRite Status
31-16	-	Not used

4.4 Internal Memory Mapping

The internal registers map is as follow:

Address	Register	Access
0x0	*STA*	Read-only
0x1	*PRE*	Write-read
0x2	*CTR0*	Write-read
0x3	*CTR1*	Write-read
0x4	*DRXA*	Read-only
0x5	*DRXB*	Read-only
0x6	*DTX*	Write-read

4.5 How to use it

4.5.1 Initialization

1. Perform a reset of the module while module is not enabled:
   CTR0: write 0 to EN and 1 to RST.

2. Load the prescaler:
   PRE: set a new value.

3. Set the I2C address of the slave module:
   CTR0[A]: set the I2C address.

4. Set the rest of bits of CTR0, including EN:
   CTR0: set rest of bits.

4.5.2 Indirect Write from Master to Slave

It is a one-phase transaction: one indirect writing is achieved by signaling only one I2C start condition by the master.

1. The Master I2C device starts an I2C transaction. In the first byte it specifies the type of transaction issued as a write.

2. Then, two bytes are received in the slave. At the end of the reception of the last bit of this second byte (third since the I2C start condition), the finite state machine in *i2c_slave_core.vhd* launches the interrupt *inst_wb_addr*.

   Address prefetching: at this point, the Wishbone Address can be stored. It is found in *DRX0* and *DRX1* registers:

   - *DRX0*: holds the Wishbone Address Lowest Byte
   - *DRX1*: holds the Wishbone Address Highest Byte

3. Following the reception of the two bytes corresponding to the Wishbone Address, four more bytes will be received. They are the data bytes. Once the last bit of this fourth byte is received (seventh byte since the I2C start condition), the finite state machine in *i2c_slave_core.vhd* launches the interrupt *inst_wr*.

   Address and Data fetching: at this point, the *Wishbone Address* and the *Data* to be written in that address can be both fetched through the *DRX* registers:

   - *DRX0*: holds the Data Lowest Byte
   - *DRX1*: holds the Data 2^{nd} Lowest Byte
   - *DRX2*: holds the Data 2^{nd} Highest Byte

- *DRX3*: holds the Data Highest Byte
- *DRX4*: holds the Wishbone Address Lowest Byte
- *DRX5*: holds the Wishbone Address Highest Byte

4. The Master I2C device stops the I2C transaction.

4.5.3 Indirect Read from Master to Slave

It is a two-phases transaction: one indirect read is achieved by signaling only two I2C start conditions by the master.

FIRST PHASE

1. The Master I2C device starts an I2C transaction. In the first byte it specifies the type of transaction issued as a write.

2. Then, two bytes are received in the slave. At the end of the reception of the last bit of this second byte (third since the I2C start condition), the finite state machine in *i2c_slave_core.vhd* launches the interrupt *inst_wb_addr*.

   Address Prefetching: at this point, the Wishbone Address can be stored. It is found in *DRX0* and *DRX1* registers:

   - *DRX0*: holds the Wishbone Address Lowest Byte
   - *DRX1*: holds the Wishbone Address Highest Byte

   Data Prefetching: it is a good practice to do the *data prefetching* of the Wishbone Address (in case this is accessible). The control logic attached to the *i2c_slave_wb* module should perform a wishbone write to the four *DTX[X]* registers:

   - *DTX0*: holds the Data Lowest Byte
   - *DTX1*: holds the Data 2^{nd} Lowest Byte
   - *DTX2*: holds the Data 2^{nd} Highest Byte
   - *DTX3*: holds the Data Highest Byte

   so that the data in the transmission registers is up-to-date, in order to be sent through I2C.

SECOND PHASE

1. The Master I2C device (re)starts an I2C transaction. In the first byte it specifies the type of transaction issued as a read. At the end of the reception of the last bit on the first byte of this *second phase* (third byte sinc the I2C start condition from the *first phase*), the finite state machine in *i2c_slave_core.vhd* launches the interrupt *inst_wb_addr*.

2. The *i2c_slave_wb* module sends the four data bytes in the following order:

   (a) Data Lowest Byte
   (b) Data 2^{nd} Lowest Byte
   (c) Data 2^{nd} Highest Byte
   (d) Data Highest Byte

3. Once the last byte has been already send, the finite state machine in *i2c_slave_core.vhd* launches the interrupt *inst_rd_addr*.

5 Trigger module

5.1 Structure

The trigger module contains several blocks related the following way:

– ***trigger_top.vhd***
—— **trigger_regs.vhd**
—— **trigger_core.vhd**
——— debouncer.vhd
——— monostable.vhd
—— **TT_RAMhandler.vhd**
——— gc_RAM.vhd (for IDs)
——— gc_RAM.vhd (for TTs)

5.1.1 *trigger_top.vhd*

The top module interconnects the three basic building blocks: registers, core and generic RAM. Each *trigger_top.vhd* module will control one output Blocking driver.

Inside *trigger_top.vhd* there are some constant that are used as *generic* in both the registers and generic RAMs. The constants are explained later in the section 'Parameters' and are the following:

- *c_RAM_SIZE*
- *c_MAX_GLITCH_STAGES*
- *c_DEFAULT_GLITCH_MASK*
- *c_MIN_PULSE_LENGTH*
- *c_MAX_PULSE_LENGTH*
- *c_DEFAULT_PULSE_LENGTH*
- *c_TAGS_DATA_WIDTH*

5.1.2 trigger_regs.vhd

It consist of a core that can be accessed via Wishbone and that contains all the registers which control this trigger core. The minimum and maximum values and proceeding for configuring them will be explained in the two next following sections.

5.1.3 TT_RAMhandler.vhd

The TT_RAMhandler is the component that control reads and writes into the RAM space used for time-tagging pulses. It is subdivided into two separated blocks of RAM: one for identification of the input and output pulse shape (so called "ID Block") and the other one for time-tagging ("TT Block").

The idea behind seperating the block memories lies in ease the task of updating the code in case different widths for the ID and TT fields are decided.

5.2 Behaviour

Once a pulse has been deglitched, which translates into a delay of *wb_clk * cycles to match c_DEFAULT_GLITCH_MASK* , a monostable will reproduce a pulse with a length determined by the CPL field in CTR0 register. The duration of the output pulse will be hence, *CTR0[CPL] * wb_clk*. After this time, a preventive action has been taken to not damage the coupled inductors in **CONV-TTL-BLO**. A timeout will be run in which no input pulse will be replicated. The value of this timeout corresponds to *CTR0[CPL] * wb_clk*, which is the same as the outputted pulse.

Min. Deglitch Mask delay *wb_clk * 1*

Max. Deglitch Mask delay *wb_clk * bits of c_DEFAULT_GLITCH_MASK*

Min. input pulse length *c_DEFAULT_GLITCH_MASK * wb_clk*

Output pulse length *CTR0[CPL] * wb_clk*

Inactivity timeout upon output pulse is done *CTR0[CPL] * wb_clk*

5.3 Parameters

5.3.1 *g_MAX_GLITCH_STAGES*

It specifies the maximum stages used for debouncing an input signal. The input signal is validated by bit 0 in CTR0. If CRT0[0] is set to 0. Inputs won't be replicated (it won't even be deglitched).

The value of this parameters express the length in bits of the parameter that holds the deglitching mask. The value of the Current deGlitching Mask can be found in CGM in CTR0 register.

5.3.2 *g_DEFAULT_GLITCH_MASK*

It specifies the default value that it is used for the Current deGlitching Mask for the CGM field in CTR0 register. Only the lower CGM bits specified by

the values of *g_MAX_GLITCH_STAGES* will be used as deglitching mask. It should be remarked that the input will be validated against a mask, so values not monotonical can be accepted. Examples are shown at the end of this document.

5.3.3 *g_MIN_PULSE_LENGTH*

It specifies the default value that it is used for the Minimum Pulse Length for the MinPL field in CTR1 register.

This field overrides the value of CPL in CTR0 in case the user tries to configure an output pulse with a width lower than MinPL. Therefore CTR0 will be MinPL.

BOUNDING MUST BE IMPLEMENTED

5.3.4 *g_MAX_PULSE_LENGTH*

It specifies the default value that it is used for the Maximum Pulse Length for the MaxPL field in CTR1 register.

This field overrides the value of CPL in CTR0 in case the user tries to configure an output pulse with a width longer than MaxPL. Therefore CTR0 will be MaxPL.

BOUNDING MUST BE IMPLEMENTED

5.4 *g_DEFAULT_PULSE_LENGTH*

It specifies the default value that it is used for the Current Pulse Length for the CPL field in CTR0 register.

It should be noted that its value is bounded by MinPL and MaxPL fields of CTR1 register.

BOUNDING MUST BE IMPLEMENTED

5.5 Registers

5.5.1 STATUS

The STATUS register is a read-only register. It shows the basic configuration of the trigger HDL core and the status of the trigger core RAM blocks.

Bits	Field	Meaning
0	EN	General ENable
1	CLR	General CLeaR
3-2	x	Reserved
4	EN_TT	Enable Time-Tagging
5	CLR_TT	CLeaR Time-Tagging
7-6	x	Reserved
8	EMPTY	RAM empty flag
9	FULL	RAM full flag
10	WA	RAM wrapped around
15-11	x	Reserved
31-16	CPL	Current Pulse Length

5.5.2 CTR0

The CTR0 register is a read-write register. It allows setting up the basic configuration of the trigger HDL core, its RAM blocks and both the deglitching mask and output pulse length to be used.

Bits	Field	Meaning
0	EN	General ENable
1	CLR	General CLeaR
3-2	x	Reserved
4	EN_TT	Enable Time-Tagging
5	CLR_TT	Clear Time-Tagging
7-6	RDM	time-tagging ReaD Mode
15-8	CGM	Current Glitch Mask
31-16	CPL	Current Pulse Length

5.5.3 CTR1

The CTR1 register is a read-write register. It allows setting up the boundaries that override invalid values of CPL field in CTR0.

Bits	Field	Meaning
15-0	MinPL	Minimum Pulse Length
31-16	MaxPL	Maximum Pulse Length

5.5.4 RAM0

The RAM0 register is a read-write register. It allows setting up the RAM and the read request to it.

Bits	Field	Meaning
0	EN_TT	Enable Time-Tagging
1	CLR_TT	Clear Time-Tagging
3-2	RDM	time-tagging ReaD mode
4	EMPTY	RAM empty
5	FULL	RAM full
6	WA	RAM Wrapped Around
7	RQT	ReQuesT read
31-8	x	Reserved

5.5.5 RAM1

The RAM1 register is a read-only register. It shows the current read and write address configured to be accessed.

Bits	Field	Meaning
15-0	CRA	Current Read Address
31-16	CWA	Current Write Address

5.5.6 RAM2

The RAM2 register is a read-write register. It allows setting up the RAM adress range to be read.

Bits	Field	Meaning
15-0	SA	Starting read Address
31-16	EA	Ending read Address

5.6 Internal Memory Mapping

The internal registers map is as follow:

Address	Register	Access
0x0	*STATUS*	Read-only
0x1	*CTR0*	Read-write
0x2	*CTR1*	Read-write
0x3	Not used	
0x4	Not used	
0x5	*RAM0*	Read-write
0x6	*RAM1*	Read-only
0x7	*RAM2*	Read-write

5.7 How to use it

5.7.1 Initialization

1. Disable trigger core before configuring:
   CTR0: write 0 to EN and EN_TT

2. Set the minimum and maximum pulse lengths:
   CTR1: writes into MinPL and MaxPL

3. Set the deglitching mask and pulse length to be used:
   CTR0: set CGM and CPL

4. Clearing RAM block up before starting:
   RAM0: write 1 to CLR_TT

5. Disable RAM clearup and enable module:
   CTR0: write 1 to EN and EN_TT, write 0 to CLR_TT

5.7.2 Changing the deglitch mask and stages length

Three examples are given.

Example A

- ***g_MAX_GLITCH_STAGES*** : 6

- **CTR0[CGM]** : 0x0BAA

In this case, only the six less significant bits of CGM field will be used as mask: "11 1010". This mask signal is checked against the sampled input signal. If it matches, it is considered as a pulse and it will be replicated.

Example B The common use will be in the form:

- ***g_MAX_GLITCH_STAGES*** : 6
- **CTR0[CGM]** : 0xFFFF

Which translates into *pulses of length **6 * wb_clk** or greater should be replicated.*

Example C Reducing the Deglitch Mask delay is achieved by configuring CGM properly:

- ***g_MAX_GLITCH_STAGES*** : 6
- **CTR0[CGM]** : 0x0007

which means *pulses of length **3 * wb_clk** or greater should be replicated.*

5.7.3 Register writes step-by-step

1. Disable trigger core before configuring:
   CTR0: write 0 to EN and EN_TT

2. Set the minimum and maximum pulse lengths:
   CTR1: writes into MinPL and MaxPL

3. Set the deglitching mask and pulse length to be used:
   CTR0: set CGM and CPL

4. Reenable module:
   CTR0: write 1 to EN and EN_TT

5.7.4 Changing the pulse width

1. Disable trigger core before configuring:
   CTR0: write 0 to EN and EN_TT

2. Check/set the minimum and maximum pulse lengths:
   CTR1: read/write into MinPL and MaxPL

3. Set the pulse length to be used:
   CTR0: set CPL

4. Reenable module:
   CTR0: write 1 to EN and EN_TT

6 Multiboot manager

6.1 Structure

NOTE1: this module is platform specific. It only works with Spartan 6
NOTE2: in case the EEPROM memory is replaced, SPI opcode will change. User should notice this issue.

The trigger module contains sever blocks related the following way:

– ***multiboot_top.vhd***
—— **multiboot_regs.vhd**
—— **multiboot_core.vhd**
——— ICAP_SPARTAN6 (*Xilinx primitive*)

6.1.1 *multiboot_top.vhd*

The top file of the module. It interconnects the Wishbone to internal register module, *multiboot_regs.vhd*, to the core logic in *multiboot_core.vhd*.

No *generics* are implemented in this HDL module.

6.1.2 multiboot_regs.vhd

In this module the registers neeeded for specifiying the memory addresses in which the FPGA must boot to are defined.

An internal register is defined for selectively controlling operations to be performed by this module (full ICAP reprogramming process, issuing ICAP commands, refreshing ICAP registers). The set of operations that can be issued is restricted for security reasons. The allowed operations are further listed in the *Register subsection*.

6.1.3 multiboot_core.vhd

It is responsible of accessing ICAP port through the internal *ICAP_SPARTAN6 Xilinx primitive*. A finite state machine is implemented in accordance to Chapter 7 of [1].

6.1.4 Behaviour

Following the instructions of [1] strictly leads to correct multiboot of the FPGA. Firstly, registers *GENERAL1*, *GENERAL2*, *GENERAL3* and *GENERAL4* must be programmed with valid values. It should be keept in mind that the *SPI opcode* in *GENERAL4* register depends on the *EEPROM chip* mounted on the board.
Then, a *full multiboot* command must be performed via ICAP interface through a write in *CTRL* register in *multiboot module*.

6.2 Parameters

No *generic* parameters are offered in this module.

6.3 Registers

6.3.1 CTRL

The *CTRL* register is a read-write register for *OP* field and a read-only for *PEND* bit. It specifies the operations that can be controlled by an user.

Bits	Field	Meaning
3-0	OP	OPeration to be performed
4	PEND	operation PENDing

Whenever an operation is specified by the user, it is passed to ICAP Xilinx primitive through *multiboot_core.vhd* and the bit flag *PEND* is set to '1' until it is completely finished.

Operations

The valid operations that can be requested are the following:

OP byte	Operation
0x0	**Full multiboot process** as specified in [1]
0x1	**Write GENERAL1** register from *multiboot_regs.vhd* into FPGA
0x2	**Write GENERAL2** register from *multiboot_regs.vhd* into FPGA
0x3	**Write GENERAL3** register from *multiboot_regs.vhd* into FPGA
0x4	**Write GENERAL4** register from *multiboot_regs.vhd* into FPGA
0x7	Perform **IPROG command**
0xD	**Refresh STAT** register into *multiboot_regs.vhd*

Full multiboot process, OP = 0x0, comprises commamnds:

1. OP = 0x1
2. OP = 0x2
3. OP = 0x3
4. OP = 0x4
5. OP = 0x7

6.3.2 STAT

The *STAT* register is a read-only register. A *refresh operation* should be completed before retrieving correct *STAT* information.

Bits	Field	Meaning
0	CRC_ERROR	CRC ERROR detected in bitstream
1	ID_ERROR	IDCODE not validated
2	DCM_LOCK	DCMs and PLL are locked
3	GTS_CFG_B	Global tristate
4	GWE	Global Write Enable
5	GHIGH_B	GHIGH
6	DEC_ERROR	DEC_ERROR
7	PART_SECURED	Decryption is set
8	HSWAPEN	HWSAPEN
11-9	MODE	MODE pins
12	INIT_B	INIT_B
13	DONE	DONE input pins
14	IN_PWRDWN	suspend status
15	SWWD_STRIKEOUT	config error because of invalid sync

6.3.3 GENERAL1

Bit scrambling is done in VHDL code. Bit order must be as specified below:

Bits	Field	Meaning
15-0	MBT_ADDR_L	MultiBoot image ADDRess Lower half

6.3.4 GENERAL2

Bit scrambling is done in VHDL code. Bit order must be as specified below:

Bits	Field	Meaning
7-0	MBT_ADDR_L	Multiboot image ADDRess Lower Half
15-8	SPIO	SPI Opcode

6.3.5 GENERAL3

Bit scrambling is done in VHDL code. Bit order must be as specified below:

Bits	Field	Meaning
15-0	GLD_ADDR_L	GolDen image ADDRess Lower half

6.3.6 GENERAL4

Bit scrambling is done in VHDL code. Bit order must be as specified below:

Bits	Field	Meaning
7-0	GLD_ADDR_H	GoLDen image ADDRess Higher Half
15-8	SPIO	SPI Opcode

6.4 Internal Memory Mapping

The Internal Memory Mapping is as follows:

Address	Register	Access
0x0	*CTRL*	Read-only
0x1	*STAT*	See *STAT* description
0x2	Not used	
0x3	Not used	
0x4	*GENERAL1*	Read-write
0x5	*GENERAL2*	Read-write
0x6	*GENERAL3*	Read-only
0x7	*GENERAL4*	Read-write

6.5 How to use it

It requieres three parameters to be specified:

- Address of the Golden Image
- Address of the Multiboot Image
- SPI Opcode of the EEPROM serial interface

Bad specifications of addresses will not reprogram the FPGA.

6.5.1 Submitting ICAP instructions

It can be either a two-step or a single-step process. Two-step processes are related with changes in *GENERAL[X]* register. Submitting an ICAP command is a single-step-process (*IPROG* instruction, for instance).

Example A: Full Multiboot Configuration

This is a scenario is useful when the EEPROM memory map has changed for the allocation of the two FPGA bitstreams.

1. Write *GENERAL1* register.
2. Write *GENERAL2* register.

3. Write *GENERAL3* register.
4. Write *GENERAL4* register.
5. Write *CTRL* register.
   CTRL should issue a **Full multiboot process** operation code (0x0).

Example B: Change an individual Boot Look Up Address
This is a scenario is useful when the EEPROM memory map has changed for the allocation of only one of the FPGA bitstreams.

1. Write *GENERAL[X]* register. Where X=1,3
2. Write *GENERAL[X+1]* register.
3. Write *CTRL* register.
   CTRL should issue a **Write GENERAL[X]** operation code.
4. Write *CTRL* register.
   CTRL should issue a **Write GENERAL[X+1]** operation code.

Example C: Reprogram FPGA without change in Bitstream Location
If the EEPROM memory map has not changed but we want to reload one of the images, we just issue an *IPROG* instruction through the *ICAP* interface.

1. Write *CTRL* register.
   CTRL should issue an **IPROG** operation code(0x7).

7 EEPROM manager

References

[1] Spartan-6 FPGA Configuration User Guide. Technical Report UG380 v2.3, Xilinx Inc., July 2011. http://www.xilinx.com/support/documentation/user_guides/ug380.pdf.