EUROPEAN ORGANIZATION FOR NUCLEAR RESEARCH

CERN – AB DEPARTMENT

Details and specifications of the $\mathbf{I^2C}$ controller core (FPGA) in the combiner card

Erik Verhagen

Abstract

Achieved during technical student period May 2006 - May 2007
Under supervision of ***Bernd Dehning*** - AB–BI

The combiner card needed to compute the emergency and control signals of the BLMTC cards is based on a powerful Altera Stratix© FPGA. Unfortunately the number IO's of this chip is not sufficient to use dedicated parallel buses for each peripheral component. An $\mathbf{I^2C}$ bus has to be implemented. This document is giving details and describing the specifications of the core needed to be implemented in the FPGA to communicate with the peripherals.

Geneva, Switzerland

February 27, 2007

Contents

1 Introduction

In the combiner card, the Stratix© FPGA acts as a central processing unit. To limit the number of used I/O pins of this CPU dedicated to the peripherals (memory, I/O extension, digital potentiometers), the chosen communication standard is **I^{2}C**.

To ease the use and the implementation of this component in the future, an effort has been made to simplify the interface and the manipulation of this block. For example, the inner global constants are fixed to suite the Philips **I^{2}C** protocol, and to control the specific components present on the combiner card. This block is provided as is, and is usable in this form only in the BLM combiner card. Nevertheless, it is extensible to other environments providing minor changes in the VHDL code.

The combiner card is planned to perform several test and decision functions, based on the input signals provided by the BLM detectors. These functions already occupy a large amount of space in the main FPGA. For this reason, the space constraints applicable to additional blocks the is essential. According to this, it has been decided not to use a generic out-of-the-box **I^{2}C** core, but to build a home made core instead. This one should fulfill only the basic needs of the combiner card, gaining this way the non essential functions of the generic **I^{2}C** protocol.

This document is aimed to give details of the needs of our core, describing its composition and finally listing the features to ease the use of it. Doing so, aside the important information regarding the implementation, it will also provide useful information in the case of transmission problems, or strange behaviors of the combiner card.

2 Specifications

2.1 General

The **I^{2}C** protocol (standing for Inter Integrated Circuit) was designed by Philips begin eighties. It's aim is to link together different chips with a simple and inexpensive communication channel. After numerous revisions, it has become a complete serial bus, with addressable targets and fast data transmission abilities, over 2 single lines. The protocol is strictly defined in the **I^{2}C** specifications, but this is only concerning upper level characteristics, like timing and synchronization. Data rates or voltage levels are left user definable, which leaves a great flexibility of application fields and working environments.

The **I^{2}C** bus is bi-directional. This means that every peripheral is able to initiate a transfer to one of its neighbors. This typical multi-mastering feature has been disabled in our application. The reason is that our peripherals do not need to initiate a transfer, they are always slaves. This characteristic simplifies a lot our design, causing thus a significant gain in terms of logic elements.

The component is a monolithic **I^{2}C** controller, build to perform safe communication between the FPGA and the modulation signal generation components needed by the self-test functionality. These components are 8 digital 8 bit potentiometers with **I^{2}C** interface. Additionally, the front panel status LEDs are also driven by an **I^{2}C** bus extension component. Thanks to a complete IEEE compliant VHDL description, this **I^{2}C** controller can be used by simply dropping it anyever in the design. According to the specifications, the *SDA* and *SCL* outputs are already tristates, enabling them to be tied directly to I/O pin of the FPGA. This is to comply with the pulled up condition of both lines when they are inactive.

Here is the functional block view of the component :

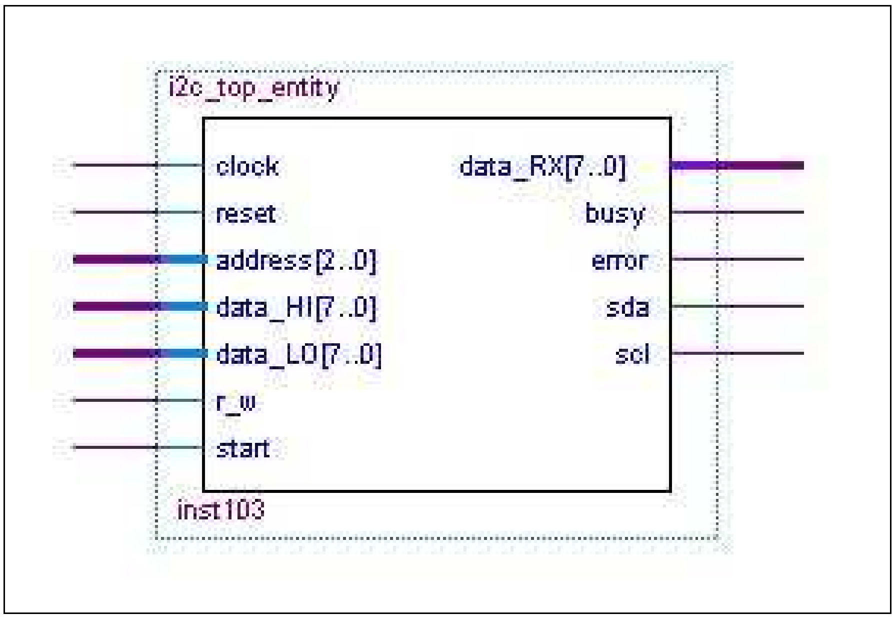

Figure 2.1: ***I*^{2}*C* *top entity***

Every kind of transfers is initiated with a rising edge on the *start* bit. This signal is synchronous, so should be applied during at least a clock cycle length. At that time the *busy* flag switches to high and stay in this position during the whole transfer phase. This is useful for interfacing the controller with upper level processing units. The *start* bit is not effective when the *busy* flag is high, enabling thus the possibility to perform synchronized transfers. Additionally, an *error* flag is present. This sets itself to high during one clock cycle when an error occurs (acknowledgment timeout during transmit or receive timeout). The device falls back in IDLE mode (usable) after such an error, and the data should be send again.

The *address* input is a three bit long bus, addressing the specific **I^{2}C** components on the combiner card. Details are given in the next section. The two data buses (*data_HI* and *data_LO*) correspond to the data bytes sent one after the other on the bus, starting with *data_HI*. In the case of the **I^{2}C** port extender used to control the LED displays, both bytes are used as data to enable or disable the 16 LEDs. No read capabilities are included for this peripheral.

Concerning the digital potentiometers, the first sent byte (*data_HI*) corresponds to the command byte, addressing one of the four specific devices inside the component. These inner 8 bits digital potentiometers are individually addressed with the 2 most significant bits inside the command byte (see datasheet). The next sent byte (*data_LO*) corresponds to the 8 bits data setting the position of the sweeper. Read capabilities are included for this component, enabling auto check in case of doubt on the output amplitude of the modulation signal. For this, the $r/\bar{w}$ bit must be set to high.

$r/\bar{w}$	Direction
'0'	Write
'1'	Read

In Read situation, an eight bit byte is outputted on *data_RX* and stays until the come of new value. Synchronization can however be achieved with the drop back of the *busy* flag.

2.2 Specific features

2.2.1 Tristated pins

As mentioned above, the two **I^{2}C** output lines (*sda* and *SCL*) are tristated. This means that they are both to be connected to a Vcc line through a pull up resistors. Here is the schematic of the bidirectional *SDA* pin logic, implemented as this inside the output of the controller :

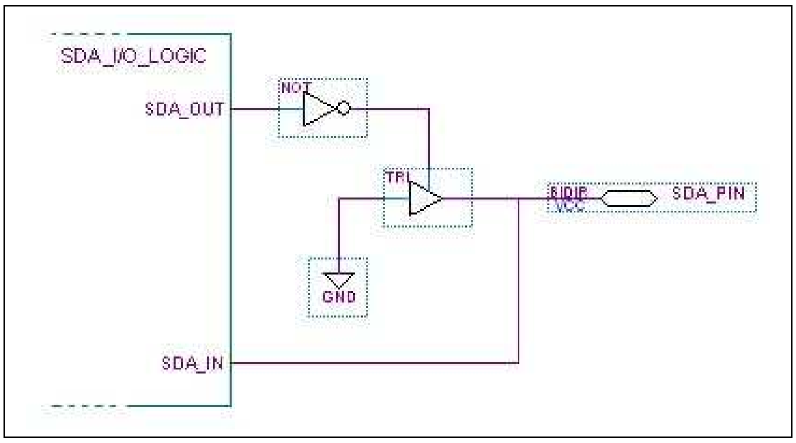

Figure 2.2: *internal SDA tristated logic*

The corresponding truth tables are shown on the next figures :

Table 2.1: DATA OUT

SDA_OUT	SDA_PIN
'0'	'0' (pulled down)
'1'	'Z' (pulled up)

Table 2.2: DATA IN

SDA_OUT	SDA_PIN	SDA_IN
'1'	**'0'** (pulled down)	'0'
'1'	**'Z'** (pulled up)	'1'

2.2.2 Data rate and clock requirements

Here we will discuss the clock requirements to ensure a proper data transfer. First of all, for synchronization reasons, the whole design needs to be feeded with the same clock signal. This unique clock is defined by the transmitting unit (framer) clock. Because the **I^{2}C** bus specification limits the data rate to 100 kbits/sec in normal mode, the clock frequency must be calculated regarding the available clock signals. This is for efficiency purpose, to avoid counter cascades. The drawback is that the frequency division must be calculated for each design.

To transmit one bit, the framer state machine needs 3 clock cycles. Knowing the upper limit data rate of 100 kbits/sec, the maximum clock frequency should thus not exceed **300 KHz**. For the specific BLECS combiner card, this means that a minimum of 2 bits counter should be implemented to create a local clock tree for the **I^{2}C** function. For security purpose however, a 4-5 bit counter would be preferable. A watchdog timer has been implemented to monitor transmission failures. In such a case, the first thing should be to decrease the clock tree frequency by increasing the number of bits in the counter.

2.2.3 Data format and addresses

The typical **I^{2}C** frame for our components contains one 8-bit address preamble, and two data bytes (also 8-bit). Between each byte, and according to the **I^{2}C** specifications, the component is ought to acknowledge during a clock cycle provided by the controller (this component).

For the **I^{2}C** digital potentiometers, the first data byte is a command byte. This command byte is automatically calculated for each individual potentiometer. This is mandatory, because the address present in the address byte is pointing to an **I^{2}C** component. In each component are 4 individual potentiometers, which have to be addressed individually (report to datasheet). The second data byte represents the position of the sweeper, out of 256 values. For the digital **I^{2}C** bus expander (used for the LEDs display), both data bytes are affected to outputs. No command byte is needed, the two bytes represent the value to be displayed.

The data inputs consists in two 8-bit port, named DATA_HI and DATA_LO. According to the **I^{2}C** specifications and the components datasheets, the first sent byte is DATA_LO. DATA_HI is sent afterward. So in the case of a frame aimed to the digital potentiometers, DATA_LO will contain the command byte.

Concerning the addresses, the use of this controller has been made as easy as possible. The address bus is a set of 3 bits. This amount comes from an earlier development stage, in fact only two bits are used :

Table 2.3: ADDRESSES TABLE

ADDRESS	CORRESPONDING **I^{2}C** COMPONENT
001	**I^{2}C** bus expander (LEDs)
010	First digital potentiometer (U304)
011	Second digital potentiometer (U308)
—	Nothing (available)

3 Implementation

3.1 Structure

The **I^{2}C** top entity (controller) is in fact a combination of a low level framer (only able to transmit and receive frames) and a higher level control unit (arbitration). Both communicate with a dedicated data bus, made of a busy flag, an error flag and a start strobe.

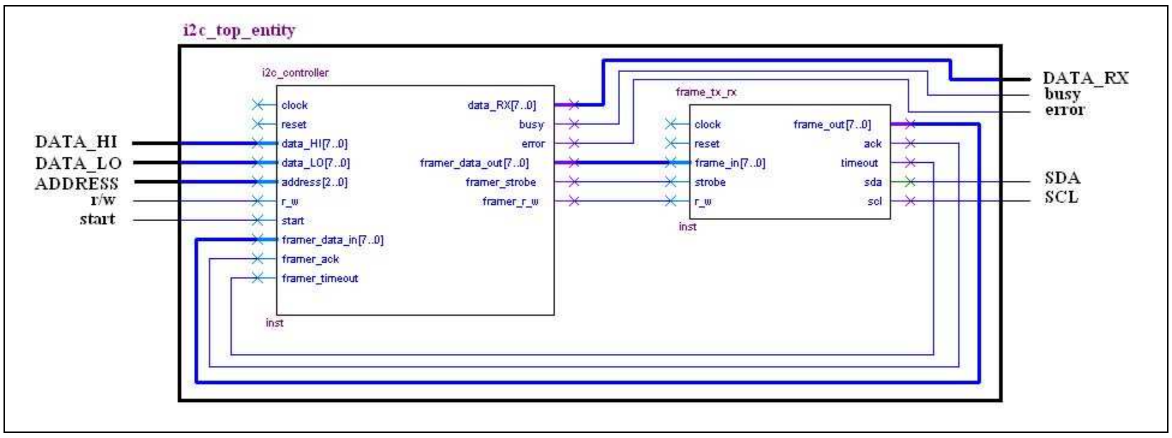

Figure 3.1: Detailed ***I^{2}C** top entity structure*

The control unit is of course a finite state machine, described in genuine VHDL. Please report to the source code for the details on the sequence, and the possibilities. Both transmitting and receiving are implemented.

The framer is a bit more complex. This entity is described in structural VHDL. It brings together a shift register associated to a counter (to load or send 8 bits on *SDA*), tristates, and a three 8 bit register bank to freeze the data bytes and address at the beginning of a transmit cycle.

3.2 Files and Hierarchy

At the framer side, four files are involved to describe the entity :

- *reg_8.vhd*: 8-bit register bank (8 D flip-flops with enable)
- *shift.vhd*: 8-bit long shift register, bidirectional with enable
- *upcnt3.vhd*: 3-bit counter with enable to ensure the good frame length
- *frame_tx_rx.vhd*: Entity and structural architecture wiring the three above mentioned components together.

The three first entities are generic RTL descriptions of common components, and can be reused for other designs.

Concerning the controller, a single file called *i2c_controller.vhd* is describing the Finite State Machine. Several generic parameters (constants) are present, and can be modified in relevant cases. These parameters are base ans physical addresses. This can be useful if components from another supplier are planned to be used.

At the highest level, the framer and the controller are wired together in a file called *i2c_top_entity.vhd*. Nothing special in this file except that this would be the entity to synthesize at last and from which a graphical schematics will be generated in some evaluated synthesis tools (like Altera Quartus ©).

4 Add-on interface

4.1 Description

To add a step in decreasing the complexity of implementation of this **I^{2}C** block, an interface component is designed. This component is aimed to initiate a transfer every time a change occurs on the inputs. This is especially convenient for the LEDs display.

This interface will be placed above the **I^{2}C** block. This place is also ideal to add simplicity to the data transfers. Every component (**I^{2}C** bus expander and all the potentiometers) will have a distinct input, and this block calculates the addresses and formats the data to be send, according to the specifications detailed in the previous chapter.

Here is a block view of the component :

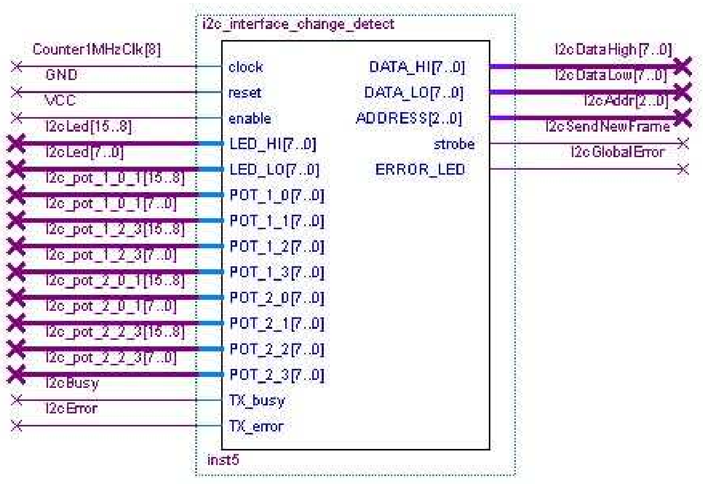

Figure 4.1: ***I^{2}C*** add-on interface

4.2 Features

This circuit is made of 10 home made change detect blocks for each input (LEDs and Potentiometers). These are simply made of 8 bit register banks with exclusive Or's between input and output. This combinatory signal gives a strobe when the input data changed.

This 10-bit wide bus is wired to a *change_detect_register* (10 clocked RS flip-flops), and is scanned sequentially in a circular way by a finite state machine. If a flag appears to be set (change occurred on the input), the FSM generate the correct **I^{2}C** address (first byte of the frame) and the correct first data byte (command byte of the right potentiometer, or low data byte for LEDs). It the MUXes then the right data on the bus between this interface and the **I^{2}C** controller for sending.

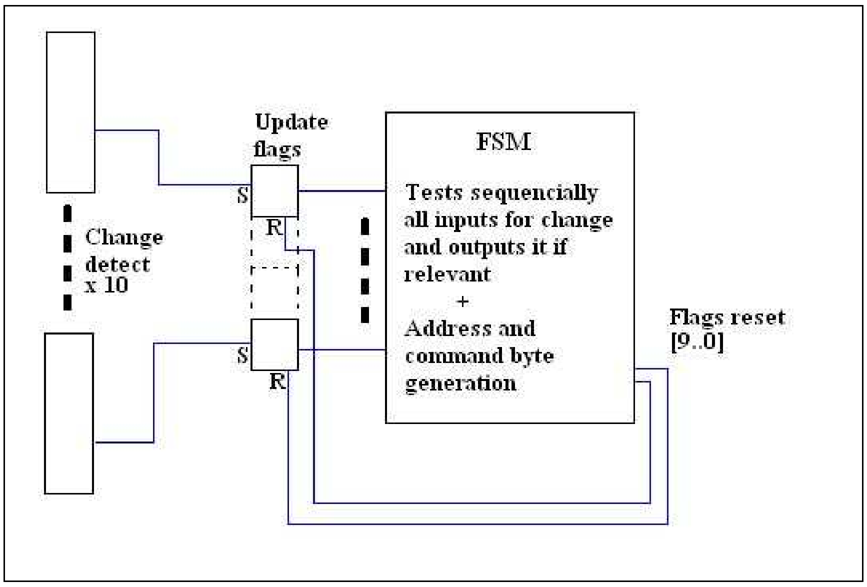
Update flags
S
R
S
R
Change detect x 10
FSM
Tests sequencially all inputs for change and outputs it if relevant
+
Address and command byte generation
Flags reset [9..0]

Figure 4.2: Structure of the ***I^{2}C*** add-on interface