

COLLEGE OF ENGINEERING

Department of Computer Engineering and Computer Science

**I2C Protocol On FPGA**

By

Chanartip Soonthornwan (014353883)

On November 29, 2018

Intentionally left blank.

Table of Contents

[1 Introduction 5](#_Toc529840075)

[2 Features 5](#_Toc529840076)

[3 General I2C Operation 5](#_Toc529840077)

[4 Data Validity 6](#_Toc529840078)

[5 Design Methodology 7](#_Toc529840079)

[6 Verilog 7](#_Toc529840080)

[7 I2C for Garbage Collector 8](#_Toc529840081)

[8 I2C With a TramelBlaze 9](#_Toc529840082)

[9 External Document 9](#_Toc529840083)

[9.1 Nexy4 Datasheet 9](#_Toc529840084)

[9.1.1 I/O Connection 10](#_Toc529840085)

[9.1.2 Seven Segment Display 10](#_Toc529840086)

[9.1.3 Pmod Connectors 11](#_Toc529840087)

[9.2 PicoBlaze 12](#_Toc529840088)

[9.2.1 Architecture 12](#_Toc529840089)

[9.2.2 Instruction Set 13](#_Toc529840090)

[9.3 Artix-7 Libraries Guide for HDL Design 16](#_Toc529840091)

[10 Requirements 16](#_Toc529840092)

[10.1 Interface Requirements 16](#_Toc529840093)

[10.2 Physical Requirements 16](#_Toc529840094)

[11 Top Level Design 17](#_Toc529840095)

[11.1 Description 17](#_Toc529840096)

[11.2 Block Diagram 17](#_Toc529840097)

[11.3 Data Flow Description 17](#_Toc529840098)

[12 Externally Developed Blocks 18](#_Toc529840099)

[12.1 TramelBlaze 18](#_Toc529840100)

[13 Internally Developed Blocks 19](#_Toc529840101)

[13.1 I2C Top Level Design 19](#_Toc529840102)

[13.2 I2C\_Core 22](#_Toc529840103)

[13.3 I2C Interface 25](#_Toc529840104)

[13.4 AISO\_RST 27](#_Toc529840105)

Table of Appendices

[Appendix a: TOP\_LEVEL Source Code 28](#_Toc529840106)

[Appendix b: I2C Top Level Source Code 30](#_Toc529840107)

[Appendix c: I2C\_Core Source Code 33](#_Toc529840108)

[Appendix d: I2C\_Interface Source Code 41](#_Toc529840109)

[Appendix e: TSI Source Code 44](#_Toc529840110)

[Appendix f: I2C\_Top\_Level Testbench 46](#_Toc529840111)

[Appendix g: I2C\_Core Testbench 48](#_Toc529840112)

[Appendix h: I2C\_Interface Testbench 50](#_Toc529840113)

Table of Figures

[Figure 1: Example of I2C Bus 5](#_Toc529840114)

[Figure 2: START and STOP condition 5](https://d.docs.live.net/95182daea981e71a/2018Fall_CSULB/CECS490B/Current_Project/Document/I2C_Document.docx#_Toc529840115)

[Figure 3: I2C master to slave communication 6](#_Toc529840116)

[Figure 4: Acknowledge from a slave 6](https://d.docs.live.net/95182daea981e71a/2018Fall_CSULB/CECS490B/Current_Project/Document/I2C_Document.docx#_Toc529840117)

[Figure 5: SDA is set while SCL is LOW and remain stable while SCL is HIGH. 6](https://d.docs.live.net/95182daea981e71a/2018Fall_CSULB/CECS490B/Current_Project/Document/I2C_Document.docx#_Toc529840118)

[Figure 6: I2C simple state diagram 7](https://d.docs.live.net/95182daea981e71a/2018Fall_CSULB/CECS490B/Current_Project/Document/I2C_Document.docx#_Toc529840119)

[Figure 7: I2C Timing Diagram with States 8](https://d.docs.live.net/95182daea981e71a/2018Fall_CSULB/CECS490B/Current_Project/Document/I2C_Document.docx#_Toc529840120)

[Figure 8: Nexys4 I/O interface 10](#_Toc529840121)

[Figure 9: Common anode circuit node 11](#_Toc529840122)

[Figure 10: PMOD connectors and pin assignments 11](https://d.docs.live.net/95182daea981e71a/2018Fall_CSULB/CECS490B/Current_Project/Document/I2C_Document.docx#_Toc529840123)

[Figure 11: PicoBlaze Block Diagram 12](#_Toc529840124)

[Figure 12: Top Level Block Diagram 17](#_Toc529840125)

[Figure 13: Top Level Detail Block Diagram 17](#_Toc529840126)

[Figure 14: TramelBlaze Block Diagram 18](https://d.docs.live.net/95182daea981e71a/2018Fall_CSULB/CECS490B/Current_Project/Document/I2C_Document.docx#_Toc529840127)

[Figure 15: I2C Top Level Block Diagram 19](https://d.docs.live.net/95182daea981e71a/2018Fall_CSULB/CECS490B/Current_Project/Document/I2C_Document.docx#_Toc529840128)

[Figure 16: I2C Top Level Detail Block Diagram 19](https://d.docs.live.net/95182daea981e71a/2018Fall_CSULB/CECS490B/Current_Project/Document/I2C_Document.docx#_Toc529840129)

[Figure 17: Top Level Verification 20](#_Toc529840130)

[Figure 18: Top Level Complete Transmission 21](#_Toc529840131)

[Figure 19: Top Level Complete Transmission Explanation 21](https://d.docs.live.net/95182daea981e71a/2018Fall_CSULB/CECS490B/Current_Project/Document/I2C_Document.docx#_Toc529840132)

[Figure 20: Top Level Incomplete Transmission 21](#_Toc529840133)

[Figure 21: I2C Core Block Diagram 22](https://d.docs.live.net/95182daea981e71a/2018Fall_CSULB/CECS490B/Current_Project/Document/I2C_Document.docx#_Toc529840134)

[Figure 22: I2C Core State Diagram 22](https://d.docs.live.net/95182daea981e71a/2018Fall_CSULB/CECS490B/Current_Project/Document/I2C_Document.docx#_Toc529840135)

[Figure 23: I2C\_Core Verification with Simulated Acknowledgement 23](https://d.docs.live.net/95182daea981e71a/2018Fall_CSULB/CECS490B/Current_Project/Document/I2C_Document.docx#_Toc529840136)

[Figure 24: I2C\_Core Timing waveform 24](#_Toc529840137)

[Figure 25: I2C\_Interface Block Diagram 25](https://d.docs.live.net/95182daea981e71a/2018Fall_CSULB/CECS490B/Current_Project/Document/I2C_Document.docx#_Toc529840138)

[Figure 26: I2C\_Interface Reading and Writing phrase 26](#_Toc529840139)

[Figure 27: I2C\_Interface debouncing button input and clearing registers 26](https://d.docs.live.net/95182daea981e71a/2018Fall_CSULB/CECS490B/Current_Project/Document/I2C_Document.docx#_Toc529840140)

[Figure 28: AISO\_Reset Block Diagram 27](https://d.docs.live.net/95182daea981e71a/2018Fall_CSULB/CECS490B/Current_Project/Document/I2C_Document.docx#_Toc529840141)

[Figure 29: AISO verification 27](#_Toc529840142)

Table of Tables

[Table 1: Bidirectional Verilog code and Diagram 8](#_Toc529840143)

[Table 2: Picoblaze Instruction Set 15](https://d.docs.live.net/95182daea981e71a/2018Fall_CSULB/CECS490B/Current_Project/Document/I2C_Document.docx#_Toc529840144)

[Table 3: Buffers used in TSI 16](#_Toc529840145)

[Table 4: Designed buttons interface 16](#_Toc529840146)

[Table 5: Designed switches part 1 16](#_Toc529840147)

[Table 6: Designed switches part 2 16](#_Toc529840148)

[Table 7: TramelBlaze I/O 18](#_Toc529840149)

[Table 8: I2C Top Level I/O Table 20](#_Toc529840150)

[Table 9: I2C Core I/O table 23](#_Toc529840151)

[Table 10: I2C\_Interface I/O Table 25](#_Toc529840152)

[Table 11: AISO I/O 27](#_Toc529840153)

**I2C Protocol**

# Introduction

I2C or Two-wire interface (TWI) is very popular and power protocol for communication between multiple masters and slaves which only require two wires, SCL and SDA, connected on each device. A master initiates all communication on the wires which means to supply the clock to slaves, and the slave will never initiate a communication. The I2C interface can operate up to 400Kbits/second, but for this project there would be a master and it will operate to the standard of 100Kbits/ second.

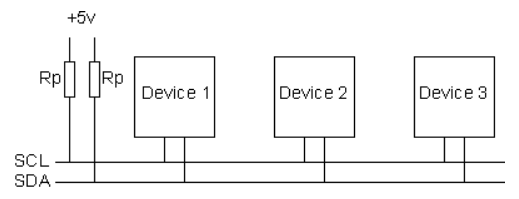


Figure 1: Example of I2C Bus

# Features

* A master and multiple slave operation
* Requires only two pins to interface I2C bus (SCL and SDA)
* Support standard rate 100 Kbps
* Auto reset to Idle state if no selected slave address is on the bus

# General I2C Operation

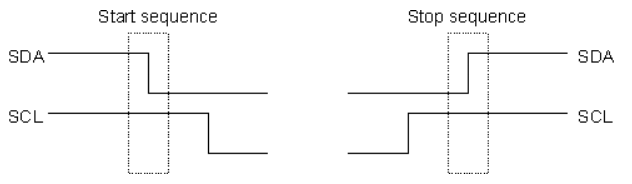
 When the I2C master demands to communicate to a slave, the master initiates transmission by pulling the SDA bus LOW while SCL is still HIGH to indicating the START condition (Figure 2). Then, shifting one bit at the time from the most significant bit (MSB) to the least significant bit (LSB) of the address of the slave that the master wants to communicate following by a read or write bit. For this I2C version, the master will be only writing. After shifting the address and the write bit, the master will wait for an acknowledge from the slave by releasing control of the SDA line. On the other hands, the slave receiving the address and check if it is its address, then the slave will seize the SDA control and output LOW as an acknowledge. Once the master receives the acknowledge, the master transmits an 8-bit data then waits for another acknowledge before ending the communication by sending a STOP condition. The writing communication is shown in Figure 3.

Figure 2: START and STOP condition

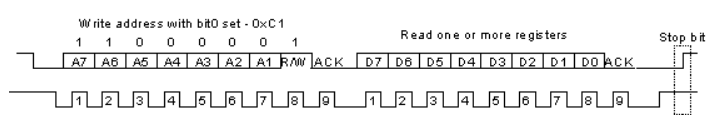
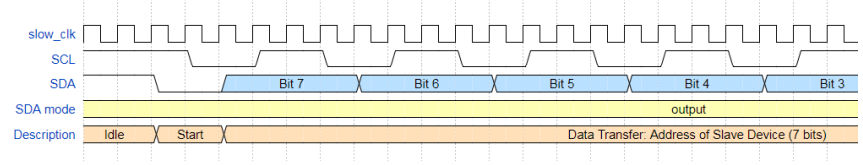


Figure 3: I2C master to slave communication

# Data Validity

 A bit of data (either address or data) is transmitting during a SCL clock pulse. Therefore, the data to be transmitted must be ready while SCL is LOW and has to be stable during HIGH time (Figure 5). In other words, the SDA line should be set when SCL is LOW because if SDA is LOW when SCL is HIGH, it will trigger a START or STOP condition instead. After sending 8-bit data, an acknowledge should be sent from

a slave under the same condition, the acknowledge should be ready before the rising edge of SCL and stable while SCL is HIGH (Figure 4).

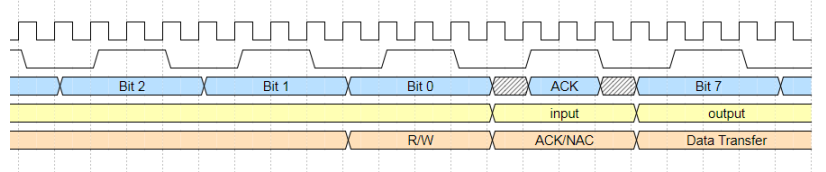


Figure 4: Acknowledge from a slave

Figure 5: SDA is set while SCL is LOW and remain stable while SCL is HIGH.

# Design Methodology

I2C module implements a Moore State Machine to control the inputs, outputs, and changes at the right event. The intermediate module interprets and translates the controls or commands into bit-level data for sending SCL and SDA lines. The state diagram in figure 6 below explains an understandable chart while the state is more complex behind the scene.

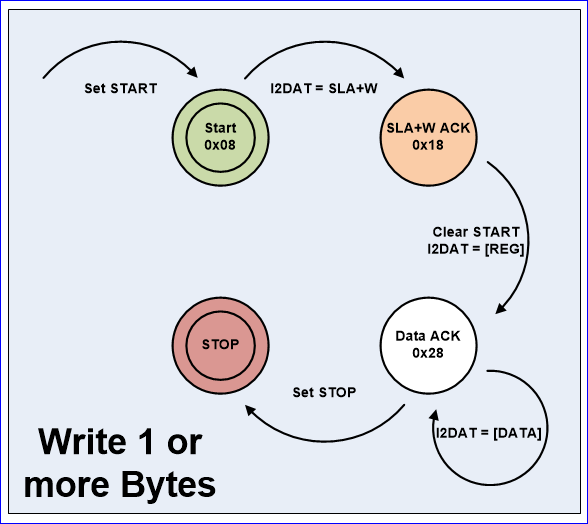


Figure 6: I2C simple state diagram

# Verilog

I2C wires traditionally are bidirectional pins, but only SCL are commonly used as an output pin since multi-master device case is rare, such that SDA is the only bidirectional pin used for outputting and receiving data. Since SDA is operating serially (one-bit at the time), the FPGA must be able to switch the characteristic of the pin between output and input state. In Verilog, it could be done by utilizing an output enable variable. The output-enable variable acts like a tri-state buffer which takes control of the SDA pin when it is enable or releases the control the pin by setting high impedance to the pin to allow another device to take control over the SDA line.

|  |  |
| --- | --- |
| **Sample snippet code**  input sda\_pad\_oen; inout SDA;  reg sda\_pad\_o; assign SDA = (sda\_pad\_oen) ? sda\_pad\_o : 1'bz; assign sda\_pad\_i = (~sda\_pad\_oen) ? SDA : 1'bz; |  |

Table 1: Bidirectional Verilog code and Diagram

SDA is an output when sda\_pad\_oen is set. Meaning the program takes controls the SDA line and transmit data from sda\_pad\_o out. Otherwise, the program sets SDA on high impedance state. Meanwhile, when sda\_pad\_oen is disable and the SDA is at high impedance state, the program allows any change on SDA line to store to sda\_pad\_i, so a user could utilize the value on sda\_pad\_i (i.e. the value on SDA).

# I2C for Garbage Collector

The I2C for Garbage Collector is customized specifically for this project as the requirement of the project is to use the I2C master on a FPGA to be a half-duplex protocol which only transmitting data to a destination slave that does not have a special register on it. In other words, whatever data input to the slave will be the data that it uses on their process right away. The SCL clock provides 100KHz or 10 us period times while transmitting the data. Since the FPGA has on-board a 100 MHz oscillator, a clock divider is implemented to create a slow clock which is four times faster than the required SCL clock or 400KHz slowed clock. The reason that slowed clock has to be four times faster than a period of SCL is that a state on the state diagram above could be divided into four states, therefore, the state machine has a full control over data path and control path for the I2C device (Figure 7).

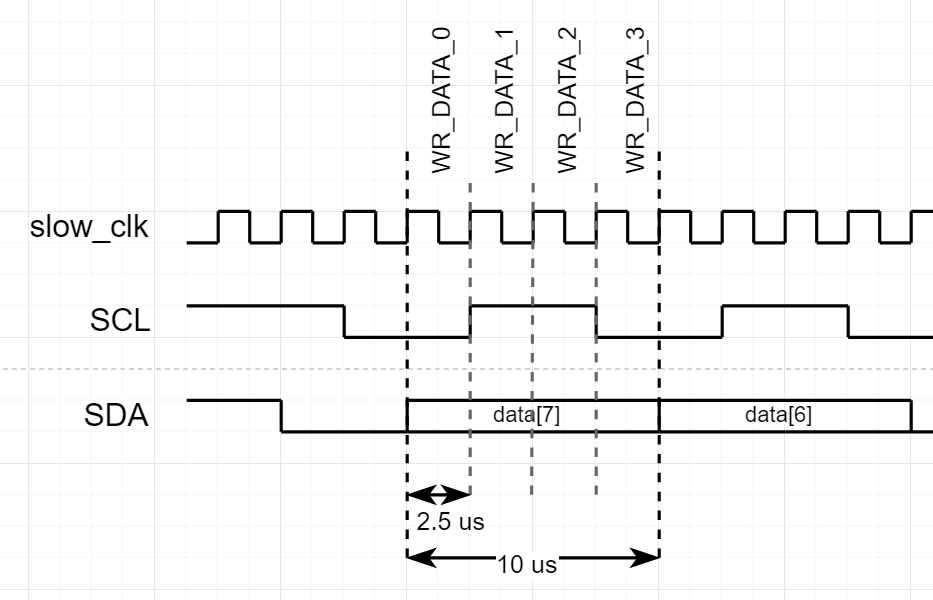


Figure 7: I2C Timing Diagram with States

# I2C With a TramelBlaze

TramelBlaze is a 16-bit emulator of an 8-bit software microprocessor, PicoBlaze. The TramelBlaze will collect 7-MSB-bit on a FPGA, Nexys4, switches as an address and 8-LSB-bit of the switches as an 8-bit data for the I2C to transmit to a slave device. Since the switches are inputs from the FPGA, the inputs go through the microprocessor before transmitting out to a slave through I2C. The switches will be stored in the microprocessor’s scratchpad RAM in order which are the address and then the data, and the microprocessor will output an address, a data, and a write command respectively.

To make the design less complicated, I2C\_Interface is implemented to reduce the intermediate connection wires between FPGA’s inputs, microprocessor’s I/O, I2C’s I/O, and other small modules. The interface contains registers that hold immediate values of a slave’s address, data, and i2C start bit. Moreover, it sets and reset interrupt request for the microprocessor, and selects an input for the microcontroller to read.

# External Document

These documents are used in developing this chip design as to assist the understanding of how to utilize the microcontroller (TramelBlaze) interface, and to create and actual interface on top of it for the later asserted Assembly program on a Programmable Read-Only Memory (PROM).

## Nexy4 Datasheet

The datasheet provides essential information of Nexys4 (Artix-7) including basic I/O interfaces and 100MHz Crystal Oscillator.

### I/O Connection

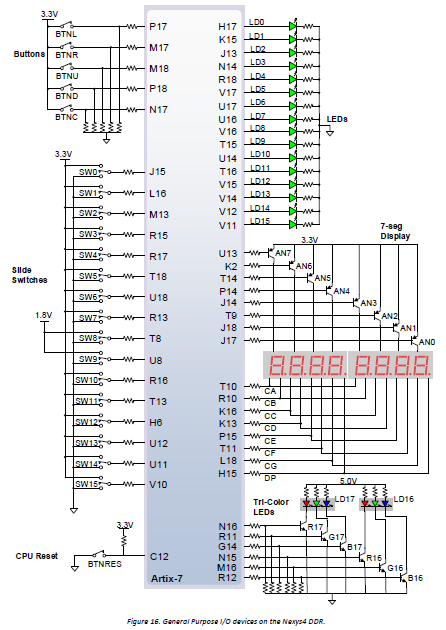


Figure 8: Nexys4 I/O interface

The figure above shows the input and output pins of buttons, switches, LEDs, and 7-segment displays, and provides the voltages requirement for logic 1 and 0 for the buttons and switches.

### Seven Segment Display

There are two of four-digit common anode of seven-segment LED displays on Nexys4. The segment LED can be individually illuminated in a “figure 8” pattern. In order to illuminate a segment, the segment’s anode should be driven high while individually driving low on the cathode. In other word, the segments are driven low when active.

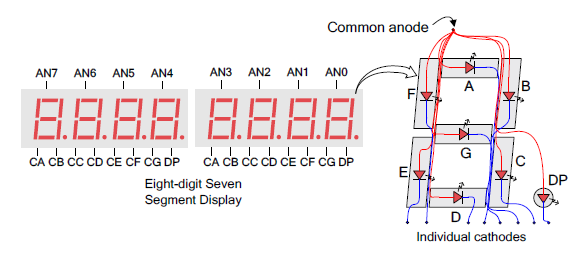
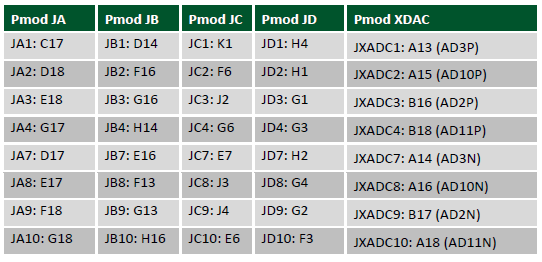
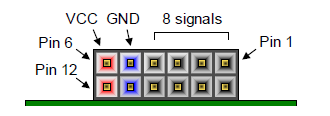


Figure 9: Common anode circuit node

### Pmod Connectors

Pmod Connectors are standard 2x6 pin headers which included two pins of VCC, two pins of GND, and eight signal pins as shown in figure below. The VCC and GND pins can be deliver up to 1 ampere. Pmod pins on JA group are used for I2C pins since the pins can be programmed as open-drain pins which are needed for bidirectional SDA bus of the I2C.

Figure 10: PMOD connectors and pin assignments



## PicoBlaze

PicoBlaze is an 8-bit RISC microcontroller core optimized for Spartan-3, Virtex-II, and Virtex-II Pro families which is later developed to be a 16-bit RISC microcontroller called TramelBlaze, delveloped by John Tramel, which is capable to apply on Spartan-6 (Nexys3) and Artix-7 (Nexys 4). PicoBlaze provides the basis of instruction set and architecture, therefore PicoBlaze user’s guide clarifies and developing a program on the TramelBlaze’s instruction ROM.

### Architecture

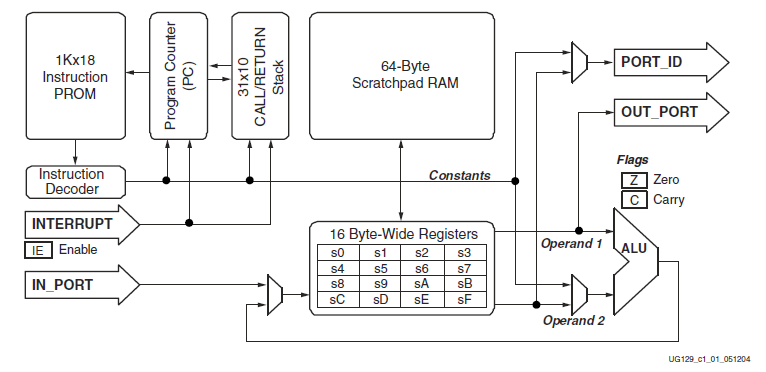
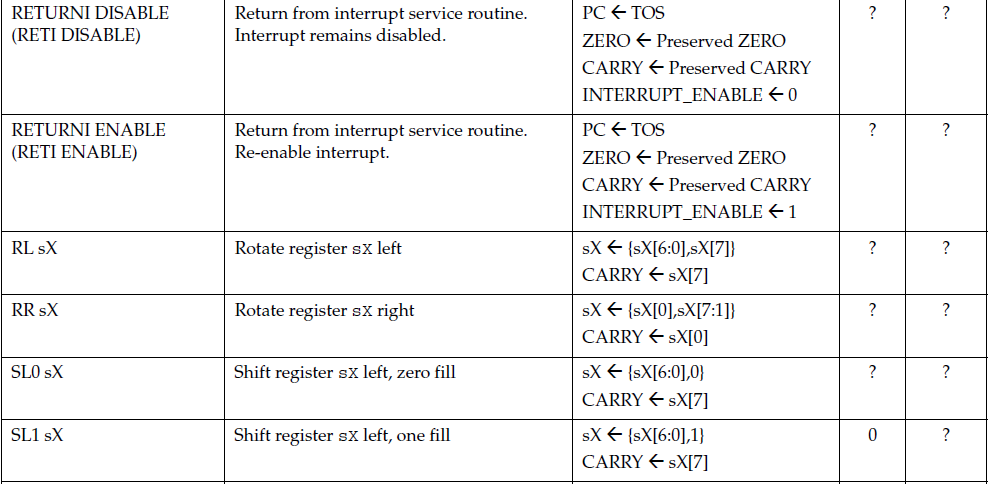
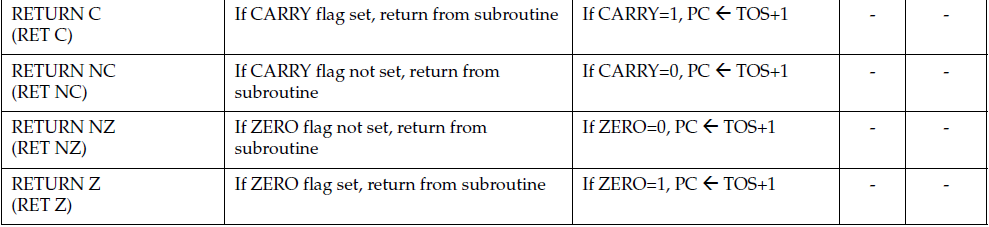
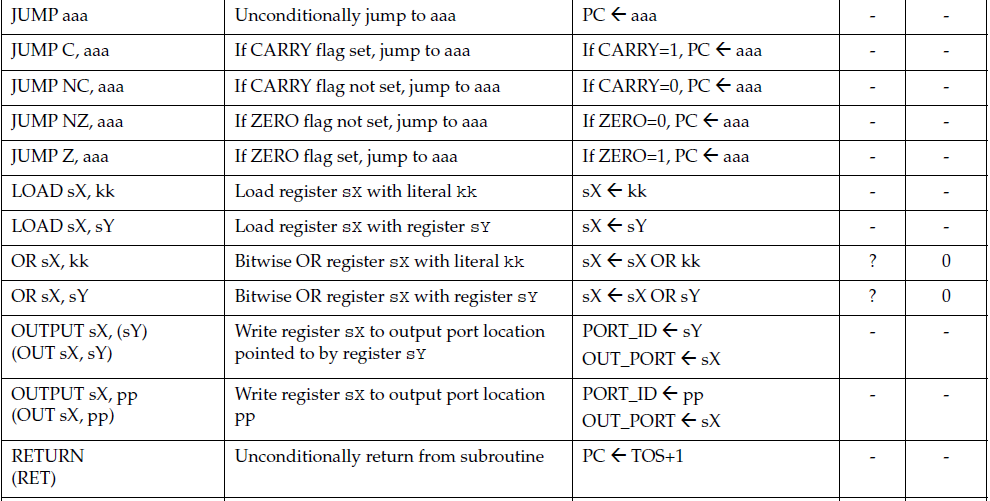
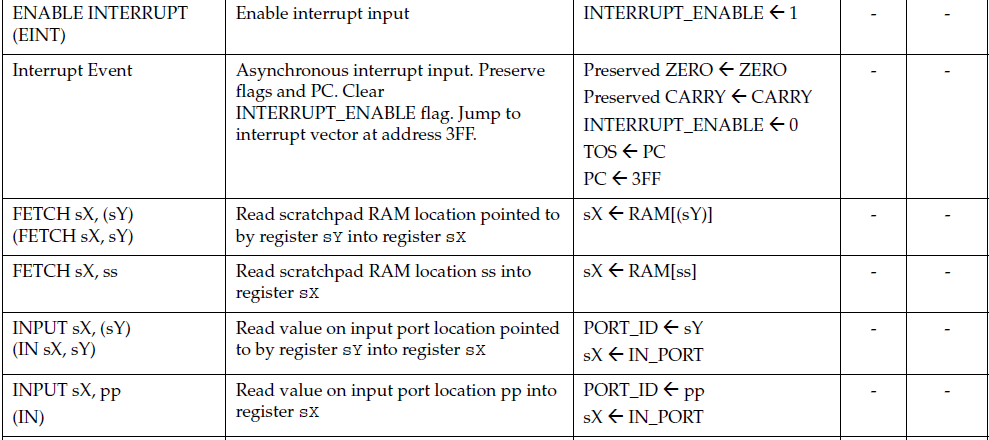
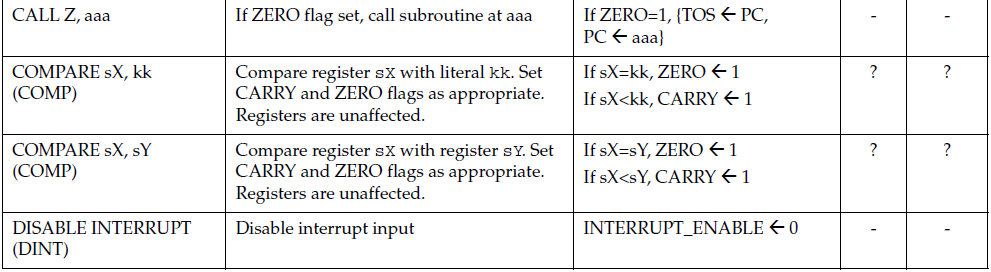
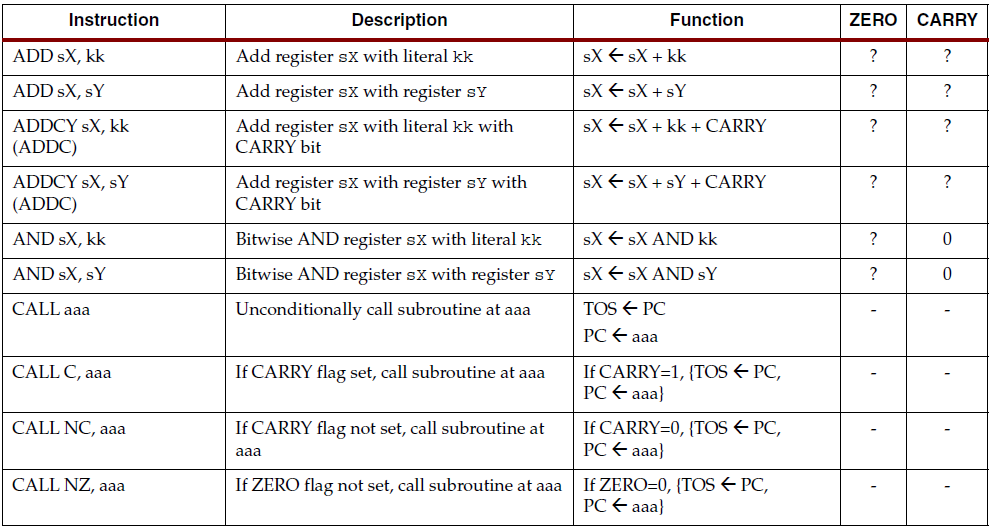


Figure 11: PicoBlaze Block Diagram

PicoBlaze and TramelBlaze share the similar architecture. The differences are that TramelBlaze consists of a 4Kx16 instruction ROM, 128x16 stack RAM, 512x16 Scratchpad RAM, and all bus lines are 16 bits.

### Instruction Set



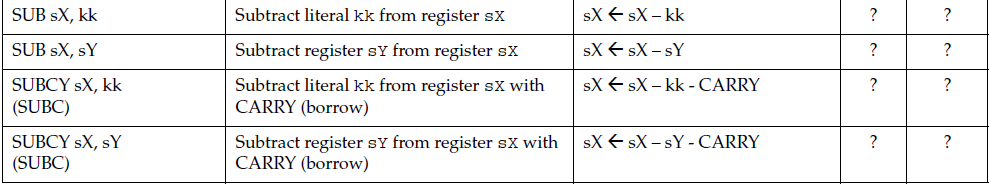
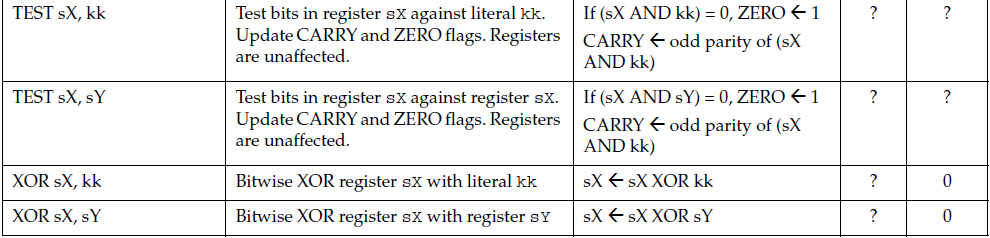


Table 2: Picoblaze Instruction Set

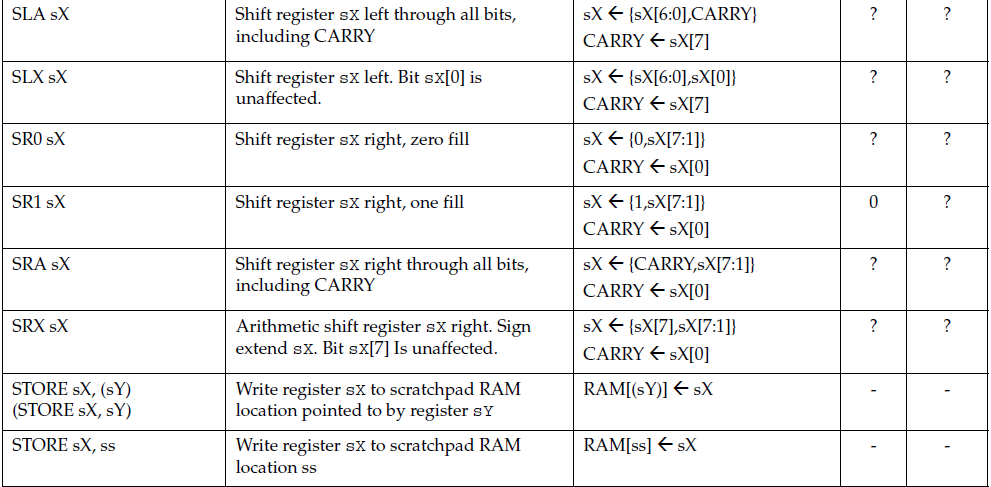


Table 1 lists all PicoBlaze instructions that are applicable to use on TramelBlaze.

## Artix-7 Libraries Guide for HDL Design

The Artix-7 Libraries Guide provides essential primitive information to complete the Technology Specific Instantiations (TSI).

|  |  |
| --- | --- |
| **Buffer** | **Description** |
|  | IBUF/IBUFG are an input buffer which IBUFG is an input buffer for a clock signal. |
|  | OBUF is an output buffer driving signals from Artix-7 to external output pads. |

Table 3: Buffers used in TSI

# Requirements

## Interface Requirements

This design has three major components which are I2C Core, I2C Interface, and TramelBlaze altogether created a SOPC core that has inputs and outputs from and to Nexys4 board through a Technology Specific Instantiation (TSI). The design uses 15 on-board switches for 7-bit slave address and 8-bit slave data, two mechanical buttons for reset and fire command to initiate I2C transmission from on-board inputs, and seven-segment display to illuminate switches inputs.

## Physical Requirements

As mention above, a user can use switches and buttons as inputs to interact to slave devices. Switches [15:9] are for slave address, switches [7:0] are for slave data, button up is for reset, and button down is for firing command. Also, seven-segment display are implemented for the user to be able to visualize the inputs.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| BTNU | BTND | BTNC | BTNL | BTNR |
| Global Reset | Firing Command | N/A | N/A | N/A |

Table 4: Designed buttons interface

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| SW15 | SW14 | SW13 | SW12 | SW11 | SW10 | SW9 | SW8 |
| Address[6] | Address[5] | Address[4] | Address[3] | Address[2] | Address[1] | Address[0] | N/A |

Table 5: Designed switches part 1

Table 6: Designed switches part 2

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| SW7 | SW6 | SW5 | SW4 | SW3 | SW2 | SW1 | SW0 |
| Data[7] | Data[6] | Data[5] | Data[4] | Data[3] | Data[2] | Data[1] | Data[0] |

# Top Level Design

## Description

Top level demonstrates interconnections between I2C\_Top\_Level and TSI where the I2C\_Top\_Level consists of the entire digital design of I2C device and a Microprocessor, while TSI contains references to the Artix-7 libraries. Any I/O for the I2C\_Top\_Level must pass through the TSI before interacting with the I2C\_Top\_Level. The I2C utilizes seven-segment on the Nexys4 to display switches input.

## Block Diagram

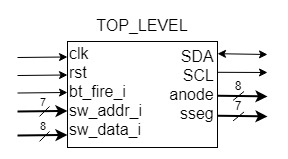


Figure 12: Top Level Block Diagram

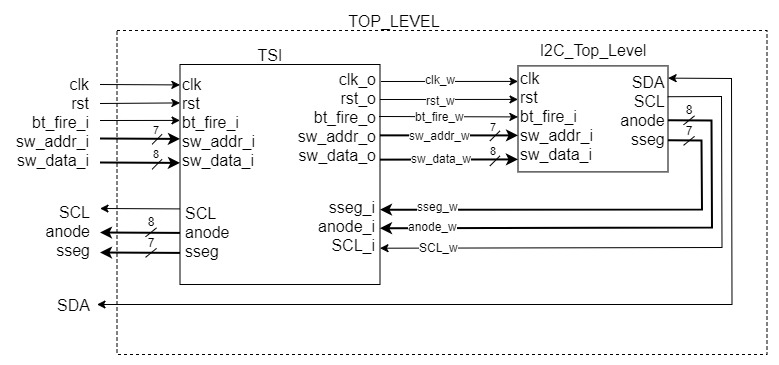


Figure 13: Top Level Detail Block Diagram

**Source Code:** Appendix a: TOP\_LEVEL Source Code

## Data Flow Description

First, a C program will be compiled to be an assembly program and stored in TramelBlaze(TB)’s instruction memory, or Programable Read Only Memory (PROM), on Nexys4. Then TB will fetch each instructions of the program and execute. Whenever there is an INPUT or OUTPUT instruction, it will setup I2C registers. Once OUTPUT instruction is executed with a port id of I2C\_CMD\_PORT, the I2C device changes from IDLE state to START state and operates the communication.

# Externally Developed Blocks

## TramelBlaze

**Description**

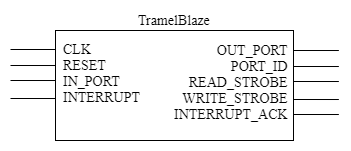


Figure 14: TramelBlaze Block Diagram

A 16-bit microcontroller that emulates the 8-bit PicoBlaze utilizing 4Kx16 bit ROM as instruction memory where the processor reads and performs the assembly program. In this application, TramelBlaze utilizes UART engine to communicate with a Serial Terminal as to display and receive an ASCII value according to which port\_id the processor preferred.

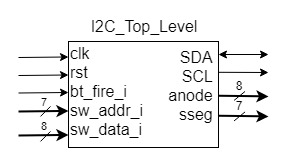
**I/O**

|  |  |  |  |
| --- | --- | --- | --- |
| **Signal** | **Size (bit)** | **I/O** | **Connected to** |
| CLK | 1 | I | 100MHz Crystal Oscillator |
| RESET | 1 | I | AISO\_RST |
| INTERRUPT | 1 | I | RS\_FLOP |
| IN\_PORT | 16 | I | UART\_TOP |
| OUT\_PORT | 16 | O | UART\_TOP |
| PORT\_ID | 16 | O | Address Decoder |
| INTERRUPT\_ACK | 1 | O | RS\_FLOP |
| READ\_STROBE | 1 | O | UART\_TOP |
| WRITE\_STROBE | 1 | O | UART\_TOP |

Table 7: TramelBlaze I/O

# Internally Developed Blocks

## I2C Top Level Design

****

**Description**

Figure 15: I2C Top Level Block Diagram

The top level demonstrates interconnections of controls and data paths through a I2C core, a microprocessor (TramelBlaze), a I2C interface, and an Asynchronized-In-Synchronized-Out Reset signal module (AISO\_RST). The switches input will be sent through the I2C interface before sending to the microprocessor and then sending to the I2C core through the interface.

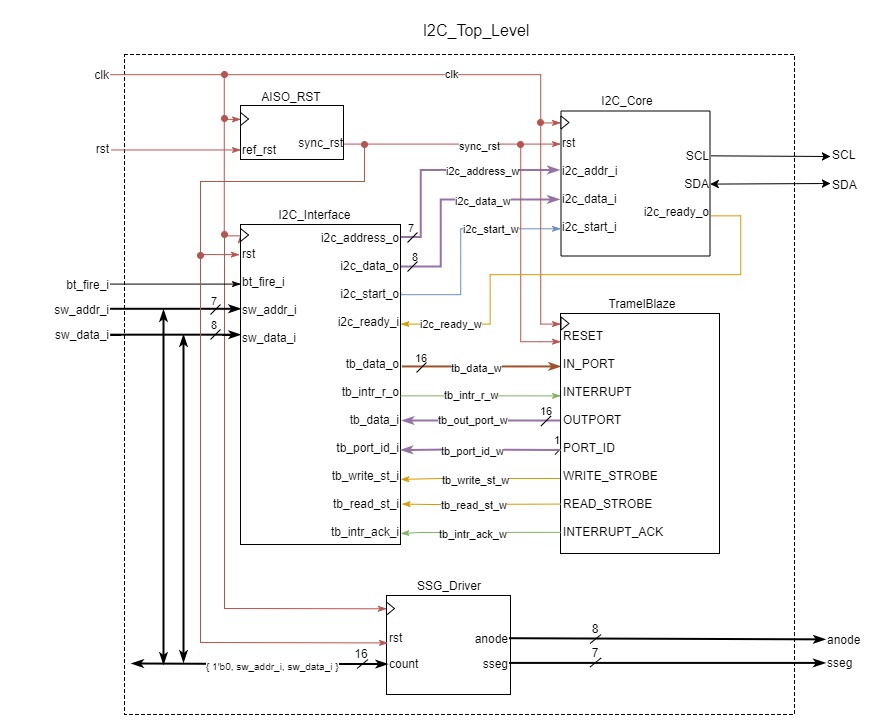
****Block Diagrams**

Figure 16: I2C Top Level Detail Block Diagram

**I/O**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Signals** | **I/O** | **Size (bits)** | **Type** | **Description** | **Operation** |
| clk | I | 1 | Digital | Provide 100 MHz clock frequency to the design | Generate 100 MHz clock |
| rst | I | 1 | Digital | Reset signal | Press button up to active |
| bt\_fire\_i | I | 1 | Analog | Start I2C transmission | Press button down to active |
| sw\_addr\_i | I | 7 | Analog | Slave address input | Toggle switches [15:9] |
| sw\_data\_i | I | 8 | Analog | Slave data input | Toggle switches [7:0] |
| SCL | O | 1 | Digital | Serial Clock Line | A bus line to slaves |
| SDA | IO | 1 | Digital | Serial Data Line | A bus line to slaves |
| anode | O | 4 | Digital | Anodes of 7-segment display | Control anodes on 7-segment display |
| sseg | O | 7 | Digital | LED segments | Display a number or a character |

Table 8: I2C Top Level I/O Table

**Source Code:** Appendix b: I2C Top Level Source Code

**Verification**

To verify, the top level will be provided different switches input for the master to communicate on multiple slave devices. Then, the top level will receive bt\_fire\_i signal to simulate a physical push button on a FPGA, and the signal will be stabilized by a debounced module to create a digital pulse signal to operate the I2C Core. The slave addresses are, 7’h01, 7’h72, 7’h55, and 7’h12. The slave devices were programmed by other designer which could successfully verify the communication because the slaves could response to the protocol as expected.

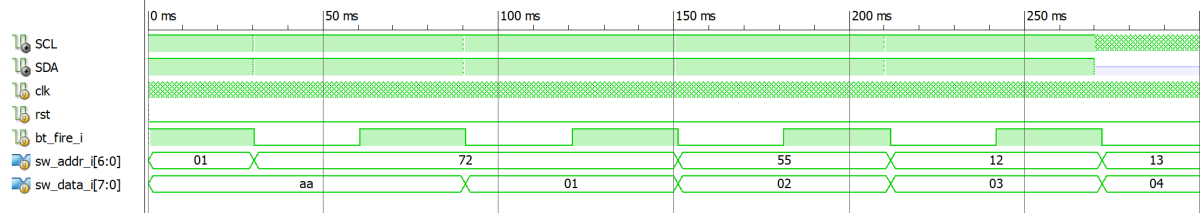


Figure 17: Top Level Verification

The bt\_fire\_i button is pushed for 30 ms before generating a one-pulse shot signal to The Tramelblaze to start the I2C transmission. Each time a transmission is done, there is a change on switches for address and data to simulate that the I2C master can communicate to multiple slaves, but there is a limit of this module which is the transmission would fall into a state of reading an acknowledge forever if the address that the I2C is transmitted to an unknown slave device around the end of the simulation.

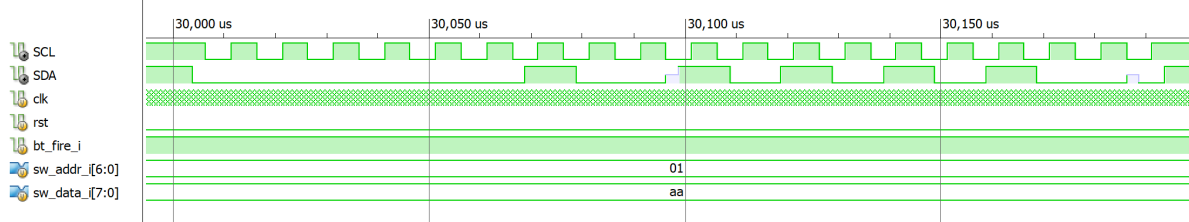


Figure 18: Top Level Complete Transmission

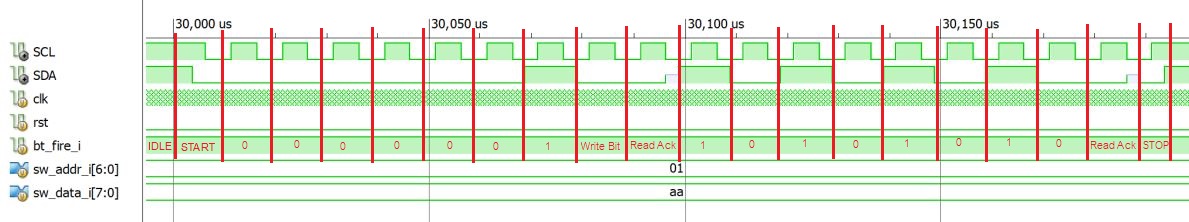
According to the state diagram (Figure 6), a complete transmission begins with a START condition which SDA line is pulled LOW following by SCL line, then transmits 7-bit data with a write bit (LOW) before waiting for an acknowledge from a slave. Since the slave is not created for specifically for this project, then there is a little red spike in the diagram, but the acknowledge bit from the slave has been read while the SCL is on HIGH time. Therefore, the Top Level could move to the next steps which are sending 8-bit of data, reading for another acknowledge, send a STOP bit, and come back to idle state as expected (show in Figure 19 ).

Figure 19: Top Level Complete Transmission Explanation

On the other hands, when the master fails by sending an address to a known slave, the master will be waiting for an acknowledge forever which could be fixed by adding a function to scope the reading for acknowledge duration in the next version.

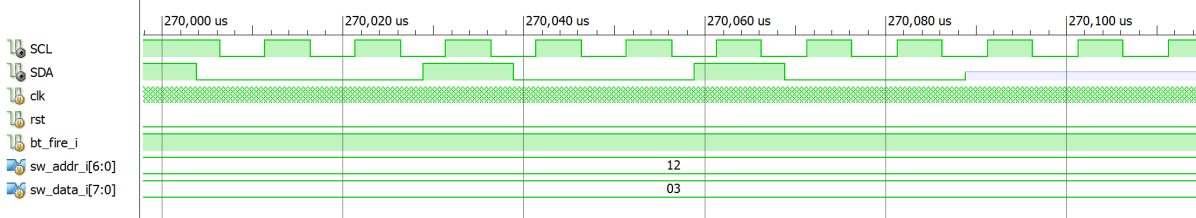


Figure 20: Top Level Incomplete Transmission

**Verification Source Code:** Appendix f: I2C\_Top\_Level Testbench

## I2C\_Core

**Description**

A master module of I2C protocol which can initiate a communication between itself to other devices on the Two Wire Interface (TWI) buses. It contains a clock divider which generates a slow clock from on-board oscillator. The slow clock is used for driving states of a state machine that controls the input and output of the I2C Core since the SCL and SDA line must be setup in sequences of each other.

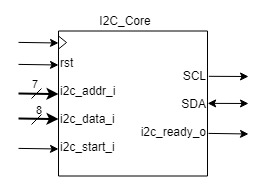
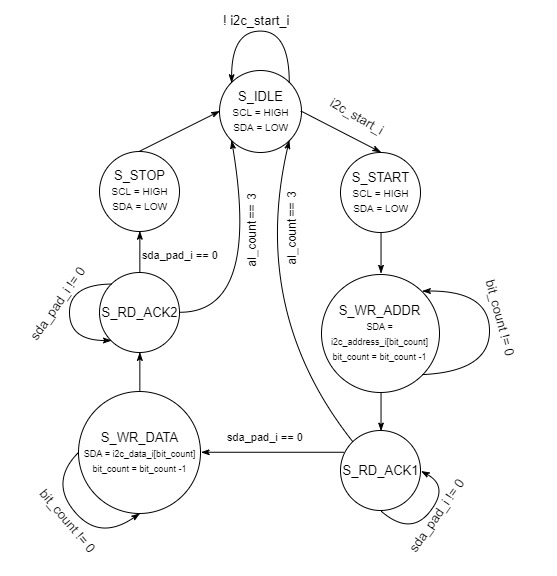


Figure 21: I2C Core Block Diagram

**Block Diagrams**

Figure 22: I2C Core State Diagram



**I/O**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Signals** | **I/O** | **Size (bits)** | **Type** | **Description** | **Operation** |
| clk | I | 1 | Digital | Provide 100 MHz clock frequency to the design | Generate 100 MHz clock |
| rst | I | 1 | Digital | Reset signal | Receive synchronous reset signal from AISO\_RST |
| i2c\_addr\_i | I | 7 | Digital | Address to transmit | Receive Address from I2C\_Interface |
| i2c\_data\_i | I | 8 | Digital | Data to transmit | Receive Data from I2C\_Interface |
| i2c\_start\_i | I | 1 | Digital | I2C\_Core Start signal | Receive Start signal from I2C\_Interface |
| i2c\_ready\_o | O | 1 | Digital | I2C\_Core Ready signal | Send Ready to I2C\_Interface  1: ready / 0: busy |

Table 9: I2C Core I/O table

**Source Code:** Appendix c: I2C\_Core Source Code

**Verification**

The I2C\_Core will be provided an address, data, and a start signal to start the communication, and it will be provided a simulated acknowledgement from a slave at the READ\_ACK state, such that the state machine, SCL and SDA signals transition, and the clock divider could be observed and verified.

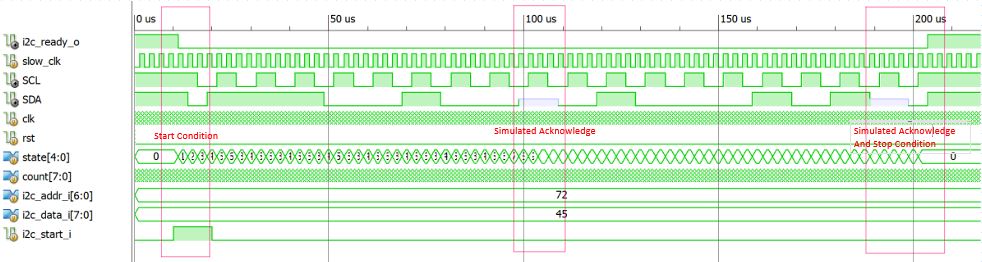


Figure 23: I2C\_Core Verification with Simulated Acknowledgement

As shown in Figure 23, the I2C\_Core completes a communication to a slave that has the address of 7’h72 by sending a start condition, sending the 7’h72 address, receiving simulated acknowledge, sending data, receiving another simulated acknowledge, and then sending a stop condition. This could verify that I2C\_Core’s state machine is successfully transitioning from START to STOP states.

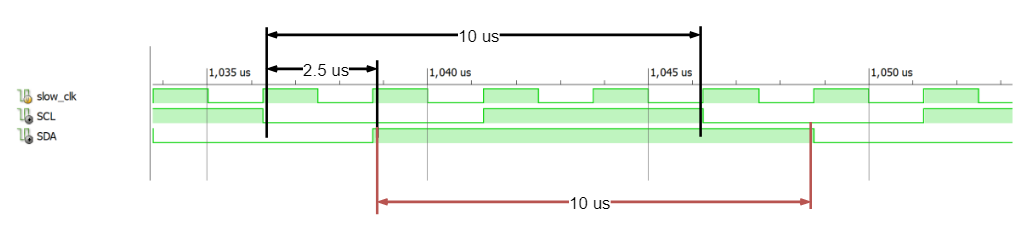


Figure 24: I2C\_Core Timing waveform

According to the timing calculation, slow clock should have 2.5 us period as this equation:

Also, I2C timing standard requirement is 100Kbit per second or 10 us that can be calculated by this equation:

Then, the slow clock, SCL, and SDL timings are verified as they are 2.5 us, 10 us, and 10 us period times respectively according to the simulation.

**Verification Source Code:** Appendix g: I2C\_Core Testbench

## I2C Interface

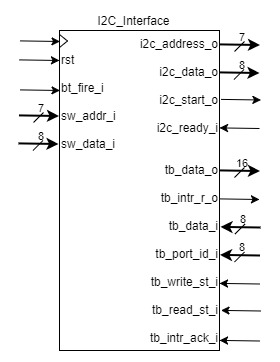


Figure 25: I2C\_Interface Block Diagram

**Description**

The module which is an interface of on-board inputs, outputs, and interconnection wires between I2C\_Core and TramelBlaze. I2C Interface holds immediate I2C address, data, and start signal for I2C\_Core. The signals are reset when I2C\_Core is ready for the next operation. The interface also debounce (or stabilize) a physical on-board push button’s signal and select which source of input for the TramelBlaze when it reads an input.

**I/O**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Signals** | **I/O** | **Size (bits)** | **Type** | **Description** | **Operation** |
| clk | I | 1 | Digital | Provide 100 MHz clock frequency to the design | Generate 100 MHz clock |
| rst | I | 1 | Digital | Reset signal | Press button up to active |
| bt\_fire\_i | I | 1 | Analog | Start I2C transmission | Press button down to active |
| sw\_addr\_i | I | 7 | Analog | Slave address input | Toggle switches [15:9] |
| sw\_data\_i | I | 8 | Analog | Slave data input | Toggle switches [7:0] |
| i2c\_addr\_o | O | 8 | Digital | Address to transmit | Send Address to I2C\_Core |
| i2c\_data\_o | O | 8 | Digital | Data to transmit | Send Data to I2C\_Core |
| i2c\_start\_o | O | 1 | Digital | I2C\_Core Start signal | Send Start signal to I2C\_Core |
| i2c\_ready\_i | I | 1 | Digital | I2C\_Core Ready signal | Receive Ready from I2C\_Core  1: ready / 0: busy |
| tb\_port\_id\_i | I | 16 | Digital | Port ID input | Receive Port ID from TB |
| tb\_data\_i | I | 16 | Digital | Data input | Receive Data from TB |
| tb\_write\_st\_i | I | 1 | Digital | Write Strobe input | Receive Write Strobe from TB |
| tb\_read\_st\_i | I | 1 | Digital | Read Strobe input | Receive Read Strobe from TB |
| tb\_intr\_ack\_i | I | 1 | Digital | Interrupt acknowledge input | Receive interrupt acknowledge from TB |
| tb\_intr\_r\_o | O | 1 | Digital | Interrupt request output | Send interrupt request to TB |
| tb\_data\_o | O | 16 | Digital | Data output | Send data to TB |

Table 10: I2C\_Interface I/O Table

**Source Code:** Appendix d: I2C\_Interface Source Code

**Verification**

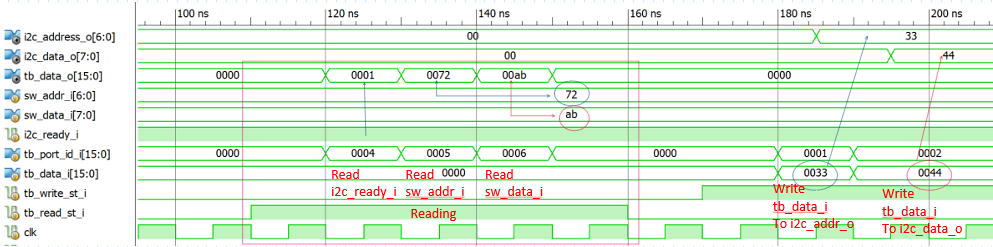
I2C\_Interface typically operates by combinational logic from inputs and outputs from TrambelBlaze(TB), I2C\_Core, and the FPGA. It is difficult to simulate the output without them, but I2C\_Interface could be breakdown to three parts; reading to TB, writing from TB, and clearing registers.

Figure 26: I2C\_Interface Reading and Writing phrase

Reading to TB, when TramelBlaze’s read strobe (tb\_read\_st\_i) is HIGH, it is indicating that the TB wants to read the address and data from switch inputs or reading the I2C\_Core status through the port ID of the input it wants to read. As shown in Figure 26, when Read Strobe is HIGH and the port ID state 16’h0004, it means the TB is reading i2c\_ready\_i. Since the ready signal is HIGH, then data output to TB (tb\_data\_o) is HIGH at LSB (16’h0001). Then, port id is 16’h0005 means that TB is reading sw\_addr\_i, so tb\_data\_o gets 16’h0072. After that, tb\_data\_o gets 16’h00ab when port id is 16’h0006.

Writing from TB, similar fashion to the reading part, if port id is 16’h0001, I2C\_Interface will pass the input from TB (tb\_data\_i) to I2C\_Core address input (i2c\_addr\_i). If port id is 16’h0002, it will pass tb\_data\_i to I2C\_Core data input (i2c\_data\_i). Lastly, if port id is 16’h 0003, it will write a start signal to the I2C\_Core (i2c\_start\_i).

Clearing registers, the data in i2c\_addr\_i and i2c\_data\_i will be cleared at the rising edge of the Ready signal. In other word, the data is cleared when I2C\_Core finishes a transmission. In order to set the I2C\_Core busy, the write button (bt\_fire\_i) will be HIGH for some amount of time in digital circuit, but it is roughly 30 milliseconds. During the times, there is a debounce module to stabilize the button input and pulse a clock-period long signal indicating that the button is pressed. When the pulse is set, it will trigger an interrupt request (tb\_intr\_r\_o) to TB to call its software interrupt service routine.

Since the switch inputs are sent to TB and registers read data from TB regarding on the port\_id successfully, the interrupt request is corresponded to the push button and the debounce module correctly, and the registers are reset when I2C\_Core is become ready, then I2C\_Interface is verified.

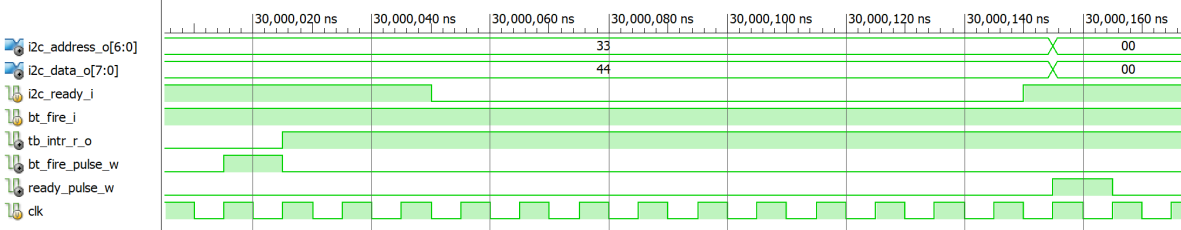


Figure 27: I2C\_Interface debouncing button input and clearing registers

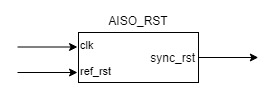
**Verification Source Code:** Appendix h: I2C\_Interface Testbench

## AISO\_RST

**Description**

A device to synchronous a digital circuit design’s system reset signal to occur on all components at the same edge of the clock instead of asynchronous reset signal that could make the system behave differently than expected.

Figure 28: AISO\_Reset Block Diagram



**I/O**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Signals** | **I/O** | **Size (bits)** | **Type** | **Description** | **Operation** |
| clk | I | 1 | Digital | Provide 100 MHz clock frequency to the design | Generate 100 MHz clock |
| ref\_rst | I | 1 | Digital | Reset signal | A reset signal from on-board reset signal |
| sync\_rst | O | 1 | Digital | Synchronized reset signal | Provide system a synchronized reset signal to the system |

Table 11: AISO I/O

**Source Code:** Appendix b: I2C Top Level Source Code

**Verification**

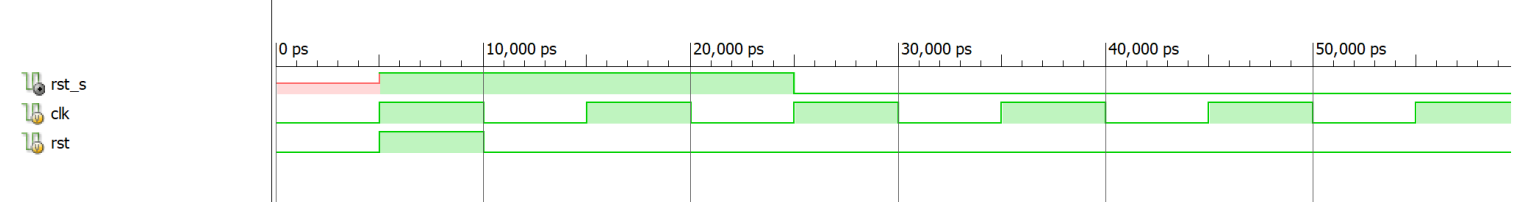


Figure 29: AISO verification

AISO\_reset is verified as it generates a synchronous signal reset output when it receives asynchronous input.

Appendix a: TOP\_LEVEL Source Code

`timescale 1ns / 1ps

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

// Class: CECS490B Senior Projects //

// Project name: Garbage Collector //

// File name: TOP\_LEVEL.v //

// //

// Created by Chanartip Soonthornwan on November 12, 2018. //

// Copyright @ 2018 Chanartip Soonthornwan. All rights reserved. //

// //

// Abstract: Overview of the system on chip shows instantiates //

// TSI and I2C\_Top\_Level. //

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

module TOP\_LEVEL(

input wire clk, // ON-BOARD clock

input wire rst, // BUTTON UP for reset signal

input wire bt\_fire\_i, // BUTTON DOWN for fire i2c transmission

input wire [6:0] sw\_addr\_i, // SWITCHES[15:9]

input wire [7:0] sw\_data\_i, // SWITCHES[ 7:0]

inout wire SDA, // Bidirectional on JA2

output wire SCL, // Output on JA1

output wire [7:0] anode, // Seven Segment Anodes

output wire [6:0] sseg // Seven Segment display

);

wire clk\_w;

wire rst\_w;

wire bt\_fire\_w;

wire [6:0] sw\_addr\_w;

wire [7:0] sw\_data\_w;

wire [7:0] anode\_w;

wire [6:0] sseg\_w;

wire SCL\_w;

// I2C Instantiate

I2C\_Top\_Level i2c\_sopc\_inst (

.clk(clk\_w), // clock from TSI

.rst(rst\_w), // clock from TSI

.SCL(SCL\_w), // SCL to TSI

.SDA(SDA), // SDA to/from JA2

.bt\_fire\_i(bt\_fire\_w), // fire input from TSI

.sw\_addr\_i(sw\_addr\_w), // address input from TSI

.sw\_data\_i(sw\_data\_w), // data input from TSI

.anode(anode\_w), // anode output to TSI

.sseg(sseg\_w) // sseg output to TSI

);

// TSI Instantiate

TSI tsi\_inst (

.clk(clk), // GLOBAL CLOCK

.rst(rst), // GLOBAL RESET

.bt\_fire\_i(bt\_fire\_i), // ON-BOARD BUTTON DOWN

.sw\_addr\_i(sw\_addr\_i), // ON-BOARD SWITCH[15:9]

.sw\_data\_i(sw\_data\_i), // ON-BOARD SWITCH[ 7:0]

.clk\_o(clk\_w), // Clock from TSI to SOPC

.rst\_o(rst\_w), // Reset from TSI to SOPC

.bt\_fire\_o(bt\_fire\_w), // Fire from TSI to SOPC

.sw\_addr\_o(sw\_addr\_w), // Address from TSI to SOPC

.sw\_data\_o(sw\_data\_w), // Data from TSI to SOPC

.SCL (SCL), // SCL from TSI to OUTSIDE

.anode (anode), // anode from TSI to OUTSIDE

.sseg (sseg), // sseg from TSI to OUTSIDE

.SCL\_i (SCL\_w), // SCL from SOPC to TSI

.anode\_i(anode\_w), // anode from SOPC to TSI

.sseg\_i (sseg\_w) // sseg from SOPC to TSI

);

endmodule

Appendix b: I2C Top Level Source Code

`timescale 1ns **/** 1ps

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

// Class: CECS490B Senior Projects //

// Project name: Garbage Collector //

// File name: I2C\_Top\_Level.v //

// //

// Created by Chanartip Soonthornwan on October 3, 2018. //

// Copyright @ 2018 Chanartip Soonthornwan. All rights reserved. //

// //

// Abstract: Display structural of Microprocessor (Tramel Blaze) //

// and Communication Protocol Device (I2C Core) //

// utilizing Interface for complex design. //

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

// Version 1.2 (November 11, 2018)

// - Added anode and sseg for displaying switches current value

//

// Version 1.1 (October 12, 2018)

// - Added sw\_addr\_i and sw\_data\_i for interacting to a user input

// to select slave address and data to send

// - Added tb\_port\_i to allow input into INPORT of TramelBlaze

//

// Version 1.0 (October 3, 2018)

//

**module** I2C\_Top\_Level**(**

**input** **wire** clk**,** // System clock (100MHz)

**input** **wire** rst**,** // System reset (from button up)

**output** **wire** SCL**,** // I2C Clock wire

**inout** **wire** SDA**,** // I2C Data wire

**input** **wire** bt\_fire\_i**,** // button down to fire a transmission

**input** **wire** **[**6**:**0**]** sw\_addr\_i**,** // switches[15:9] for 7-bit address input

**input** **wire** **[**7**:**0**]** sw\_data\_i**,** // switches[7:0] for 8-bit data input

**output** **wire** **[**7**:**0**]** anode**,** // Anodes of 7-segment display

**output** **wire** **[**6**:**0**]** sseg // 7 segments

**);**

**wire** sync\_rst**;** // Synchronized reset signal wire

**wire** tb\_intr\_r\_w**;** // Interrupt Request from I2C\_Interface to TB

**wire** tb\_intr\_ack\_w**;** // Interrupt Acknowledge from TB to I2C\_Interface

**wire** **[**15**:**0**]** tb\_out\_port\_w**;** // Output from TB to I2C\_Interface

**wire** **[**15**:**0**]** tb\_port\_id\_w**;** // PortID from TB to I2C\_Interface

**wire** tb\_wr\_st\_w**;** // Write Strobe from TB to I2C\_Interface

**wire** tb\_rd\_st\_w**;** // Read Strobe from TB to I2C\_Interface

**wire** **[**15**:**0**]** tb\_data\_w**;** // Data output from I2C\_Interface to TB

**wire** **[** 6**:**0**]** i2c\_addr\_w**;** // Address from I2C\_Interface to I2C\_Core

**wire** **[** 7**:**0**]** i2c\_data\_w**;** // Data from I2C\_Interface to I2C\_Core

**wire** i2c\_start\_w**;** // Start signal from I2C\_Interface to I2C\_Core

**wire** i2c\_ready\_w**;** // Ready signal from I2C\_Core to I2C\_Interface

// Communication Protocol Device

// Generating output clock signal on SCL line with frequency of 100KHz

// while transmitting 8-bit address and data. Also indicating a ready

// signal to microprocessor before initiating the next transmission.

I2C\_Core i2c\_core **(**

**.**clk **(**clk **),**

**.**rst **(**sync\_rst **),**

**.**i2c\_addr\_i **(**i2c\_addr\_w **),**

**.**i2c\_data\_i **(**i2c\_data\_w **),**

**.**i2c\_start\_i **(**i2c\_start\_w **),**

**.**i2c\_ready\_o **(**i2c\_ready\_w **),**

**.**SCL **(**SCL **),**

**.**SDA **(**SDA **)**

**);**

// Interface

// Wrapping all wires, generating ready pulse signal, setting/resetting

// interrupt request, and holding Address and Data in registers

// to be ready before initiating an I2C transmission.

I2C\_Interface i2c\_interface**(**

**.**clk **(**clk **),**

**.**rst **(**sync\_rst **),**

**.**bt\_fire\_i **(**bt\_fire\_i **),**

**.**sw\_addr\_i **(**sw\_addr\_i **),**

**.**sw\_data\_i **(**sw\_data\_i **),**

// I2C I/O

**.**i2c\_ready\_i **(**i2c\_ready\_w **),**

**.**i2c\_address\_o**(**i2c\_addr\_w **),**

**.**i2c\_data\_o **(**i2c\_data\_w **),**

**.**i2c\_start\_o **(**i2c\_start\_w **),**

// Tramel Blaze I/O

**.**tb\_port\_id\_i **(**tb\_port\_id\_w **),**

**.**tb\_data\_i **(**tb\_out\_port\_w**),**

**.**tb\_write\_st\_i**(**tb\_wr\_st\_w **),**

**.**tb\_read\_st\_i **(**tb\_rd\_st\_w **),**

**.**tb\_intr\_ack\_i**(**tb\_intr\_ack\_w**),**

**.**tb\_intr\_r\_o **(**tb\_intr\_r\_w **),**

**.**tb\_data\_o **(**tb\_data\_w **)**

**);**

// Microprocessor

// Created by John Tramel

// Utilizing Scratch RAM to load instructions

// and execute them before initiates the transmission

// \*note: In this version, there is only writing to

// but not reading from other devices.

tramelblaze\_top tb\_top**(**

**.**CLK **(**clk **),**

**.**RESET **(**sync\_rst **),**

**.**IN\_PORT **(**tb\_data\_w **),**

**.**INTERRUPT **(**tb\_intr\_r\_w **),**

**.**OUT\_PORT **(**tb\_out\_port\_w**),**

**.**PORT\_ID **(**tb\_port\_id\_w **),**

**.**READ\_STROBE **(**tb\_rd\_st\_w **),**

**.**WRITE\_STROBE **(**tb\_wr\_st\_w **),**

**.**INTERRUPT\_ACK**(**tb\_intr\_ack\_w**)**

**);**

// Seven Segment Display

// Display Address on switches and data

// to send out over I2C as for observation

SSG\_Driver ssg\_driver**(**

**.**clk**(**clk**),**

**.**rst**(**sync\_rst**),**

**.**count**({**1'b0**,** sw\_addr\_i**,** sw\_data\_i**}),**

**.**anode**(**anode**[**3**:**0**]),**

**.**sseg**(**sseg**)**

**);**

**assign** anode**[**7**:**4**]** **=** 4'b1111**;**

// Asynchronize-In-Synchronize-Out Reset

// In used for initiating synchronous reset signal

// to the whole design.

AISO\_RST aiso\_rst**(**

**.**clk **(**clk **),**

**.**ref\_rst **(**rst **),**

**.**sync\_rst **(**sync\_rst**)**

**);**

**endmodule**

//\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ ASIO Reset \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

// Receives reset signal input from a reset button then generates

// synchronized output at rising edge to other module in the design.

// input: clk - reference clock (on-board clock)

// input: ref\_rst - reference reset (Asynchronized reset signal input)

// output: sync\_rst - synchronized reset

**module** AISO\_RST**(**

**input** **wire** clk**,** ref\_rst**,**

**output** **wire** sync\_rst

**);**

**reg** r1**,**r2**;**

**assign** sync\_rst **=** **~**r2**;** // one-clock impulsive reset signal

**always@(posedge** clk**,** **posedge** ref\_rst**)** **begin**

**if(**ref\_rst**)** **{**r1**,** r2**}** **<=** 2'b00**;**

**else** **{**r1**,** r2**}** **<=** **{**1'b1**,** r1**};**

**end**

**endmodule**

//\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ End ASIO Reset \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**reg** r1**,**r2**;**

**assign** sync\_rst **=** **~**r2**;** // one-clock impulsive reset signal

**always@(posedge** clk**,** **posedge** ref\_rst**)** **begin**

**if(**ref\_rst**)** **{**r1**,** r2**}** **<=** 2'b00**;**

**else** **{**r1**,** r2**}** **<=** **{**1'b1**,** r1**};**

**end**

**endmodule**

//\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ End ASIO Reset \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Appendix c: I2C\_Core Source Code

`timescale 1ns **/** 1ps

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

// Class: CECS490B Senior Projects //

// Project name: Garbage Collector //

// File name: I2C\_Core.v //

// //

// Created by Chanartip Soonthornwan on October 3, 2018. //

// Copyright @ 2018 Chanartip Soonthornwan. All rights reserved. //

// //

// Abstract: I2C at bit operation controlling data path and controls //

// via a state machine with eight states. //

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

// Version 2.1 (November 10, 2018)

// - Added al\_count for to check if there is an arbitrary lost from

// the master waiting from a slave's signal. If there is no response

// from a slave, then the master will return to idle state.

//

// Version 2.0 (October 21, 2018)

// - Remodeled Clock Divider to generate 400khz clock

// - Remodeled State Machine with more states

//

// Version 1.1 (October 12, 2018)

// - Added sw\_addr\_i and sw\_data\_i for interacting to a user input

// to select slave address and data to send

// - Added tb\_port\_i to allow input into INPORT of TramelBlaze

//

// Version 1.0 (October 3, 2018)

//

module I2C\_Core(

input wire clk, // On-board Clock 100MHz

input wire rst, // Reset signal

input wire [6:0] i2c\_addr\_i, // Incoming Address

input wire [7:0] i2c\_data\_i, // Incoming Data

input wire i2c\_start\_i, // Start signal

output reg i2c\_ready\_o, // Ready Status Register

output wire SCL, // I2C SCL (clock) output

inout wire SDA // I2C SDA (data) input/output

);

//\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ IO Padder \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

//

// The middle ground for SDA and SCL bus to stop by since bidirectional

// bus cannot be directly assigning the value. Therefore, there should be a

// pad or middle ground taking place to make open-drain-like circuit.

// The bus will be high impedance when no one takes control of the bus.

// In order to control the bus, an output enable seize the control of

// the bus. Once output is done, the output enable will be released, and

// the bus from this device perspective will be high-impedance.

// High-impedance is like open-drain which will allow incoming input when

// there is an input from other devices.

//

reg scl\_pad\_o; // i2c\_clock output

reg scl\_pad\_oen; // i2c\_clock output enable

wire sda\_pad\_i; // i2c\_data input

reg sda\_pad\_o; // i2c\_data output

reg sda\_pad\_oen; // i2c\_data output enable

assign SCL = (scl\_pad\_oen) ? scl\_pad\_o : 1'bz;

assign SDA = (sda\_pad\_oen) ? sda\_pad\_o : 1'bz;

assign sda\_pad\_i = (~sda\_pad\_oen)? SDA : 1'bz;

//\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Clk Divider \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

//

// Generates slow clock (400khz) by ticking every 2.5 us, and each tick

// will toggle the slow clock level from low to high and vice versa.

// Calculation: DELAY = 100Mhz / 400Khz = 250

// DELAY/2 = 125

parameter DELAY = 125; // Constant for toggleling a clock at 1.25 us

reg [ 7:0] count; // Time counter

reg slow\_clk; // slow\_clock from on\_board clk

wire tick; // Toggling the clock indicator

assign tick = (count == (DELAY -1)); // Tick at 5us

always@(posedge clk, posedge rst) begin

if(rst) begin

count <= 0;

slow\_clk <= 0;

end else

if(tick) begin // got a tick signal

count <= 0; // reset the counter

slow\_clk <= ~slow\_clk; // switch the slow clock edge

end else begin

count <= count + 8'b1; // increment the counter

slow\_clk <= slow\_clk; // slow clock edge stay the same.

end

end

//\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ State Machine \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

//

// Generates controls and outputs according to an events of each state.

// There are 20 states in total, but the main states are

// IDLE -> START -> WRITE\_ADDR -> READ\_ACK

// -> WRITE\_DATA -> READ\_ACK2 -> STOP -> IDLE

// Explain:

// IDLE - Both SCL and SDA are HIGH, and ready is HIGH indicating that the

// I2C is ready for i2c\_start\_i signal. When i2c\_start\_i is HIGH,

// ready is low as the I2C is busy.

// slow\_clk: /~~\\_\_/~~\\_\_

// SCL: ~~~~~~~~~~~~

// SDA: ~~~~~~~~~~~~

//

// START - First clock will pull SDA LOW while SCL HIGH. Second clock, both

// SCL and SDA are LOW, and set the data register by concentrating

// Address and Write bit before shifting on the next clock.

// slow\_clk: /~~\\_\_/~~\\_\_/~~\\_\_/

// SCL: ~~~~~~~~~~~~\\_\_\_\_\_\_

// SDA: ~~~~~~\\_\_\_\_\_\_\_\_\_\_\_\_

//

// WRITE - Shifting the data register from MSB to LSB. Each 4 clocks will

// decrement the register index by one, and check if it is the LSB.

// by utilizing command(CMD). If so, move to the read acknowledge.

// slow\_clk: /~~\\_\_/~~\\_\_/~~\\_\_/~~\\_\_/

// SCL: \_\_\_\_\_\_/~~~~~~~~~~~\\_\_\_\_\_/

// SDA: ==X=======================X==

//

// READ - Wait for acknowledge input from a slave device. The input should be

// ready before the rising edge of SCL and should not change while

// the SCL is HIGH.

// slow\_clk: /~~\\_\_/~~\\_\_/~~\\_\_/~~\\_\_/

// SCL: \_\_\_\_\_\_/~~~~~~~~~~~\\_\_\_\_\_/

// SDA: ==X=====\\_\_\_\_\_\_\_\_\_\_\_/======X==

//

// STOP - First clock will pull both SCL and SDA LOW, then SCL HIGH while

// SDA LOW.

// slow\_clk: /~~\\_\_/~~\\_\_/~~\\_\_/

// SCL: ~~~~~~\\_\_\_\_\_/~~~~~~

// SDA: ~~~~~~\\_\_\_\_\_\_\_\_\_\_\_/

//

localparam

S\_IDLE = 0,

S\_START\_0 = 1, S\_START\_1 = 2,

S\_WR\_ADDR\_0 = 3, S\_WR\_ADDR\_1 = 4, S\_WR\_ADDR\_2 = 5, S\_WR\_ADDR\_3 = 6,

S\_RD\_ACK1\_0 = 7, S\_RD\_ACK1\_1 = 8, S\_RD\_ACK1\_2 = 9, S\_RD\_ACK1\_3 = 10,

S\_WR\_DATA\_0 = 11, S\_WR\_DATA\_1 = 12, S\_WR\_DATA\_2 = 13, S\_WR\_DATA\_3 = 14,

S\_RD\_ACK2\_0 = 15, S\_RD\_ACK2\_1 = 16, S\_RD\_ACK2\_2 = 17, S\_RD\_ACK2\_3 = 18,

S\_STOP\_0 = 19, S\_STOP\_1 = 20,

CMD\_IDLE = 1, CMD\_START = 2, CMD\_WR\_ADDR = 3, CMD\_RD\_ACK = 4,

CMD\_WR\_DATA = 5, CMD\_STOP = 6, CMD\_AR\_LOST = 7,

WRITE\_BIT = 1'b0,

HIGH = 1'b1,

LOW = 1'b0;

// Registers

reg [7:0] data\_reg; // Holding immediate data(address or data)

reg [2:0] bit\_count; // Index for data\_reg

reg [2:0] al\_count; // Arbitrary lost count

reg [2:0] cmd; // Special command on some states

reg [4:0] state; // Present State

reg [4:0] next\_state; // Next State

// Update Present State on fast clock

always@(posedge clk, posedge rst) begin

if(rst) state <= S\_IDLE;

else state <= next\_state;

end

// State Machine

always@(posedge slow\_clk, posedge rst) begin

if(rst) begin

cmd <= CMD\_IDLE;

scl\_pad\_oen <= HIGH;

sda\_pad\_oen <= HIGH;

scl\_pad\_o <= HIGH;

sda\_pad\_o <= HIGH;

data\_reg <= 0;

bit\_count <= 0;

al\_count <= 0;

next\_state <= S\_IDLE;

i2c\_ready\_o <= HIGH;

end

else begin

case(state)

// IDLE STATE

// SCL and SDA are high until receive a start signal

S\_IDLE: begin

if(i2c\_start\_i) begin // receive start signal

cmd <= CMD\_START;

scl\_pad\_oen <= HIGH;

sda\_pad\_oen <= HIGH;

scl\_pad\_o <= HIGH;

sda\_pad\_o <= HIGH;

data\_reg <= data\_reg;

bit\_count <= bit\_count;

al\_count <= al\_count;

next\_state <= S\_START\_0;

i2c\_ready\_o <= LOW; // now i2c is busy

end

else begin // still idling

cmd <= CMD\_IDLE;

scl\_pad\_oen <= HIGH;

sda\_pad\_oen <= HIGH;

scl\_pad\_o <= HIGH;

sda\_pad\_o <= HIGH;

data\_reg <= data\_reg;

bit\_count <= bit\_count;

al\_count <= al\_count;

next\_state <= S\_IDLE;

i2c\_ready\_o <= HIGH;

end

end

// START STATE

// SDA is LOW while SCL is HIGH,

// then SCL is LOW on the next clock

S\_START\_0: begin

cmd <= CMD\_START;

scl\_pad\_oen <= HIGH;

sda\_pad\_oen <= HIGH;

scl\_pad\_o <= HIGH;

sda\_pad\_o <= LOW;

data\_reg <= data\_reg;

bit\_count <= bit\_count;

al\_count <= al\_count;

next\_state <= S\_START\_1;

end

S\_START\_1: begin

cmd <= CMD\_WR\_ADDR;

scl\_pad\_oen <= HIGH;

sda\_pad\_oen <= HIGH;

scl\_pad\_o <= LOW;

sda\_pad\_o <= LOW;

data\_reg <= {i2c\_addr\_i, WRITE\_BIT}; // Loading data

bit\_count <= 3'd7; // MSB Index

al\_count <= 0;

next\_state <= S\_WR\_ADDR\_0;

end

// WRITE STATE

// Shifting MSB to LSB,

// data in data\_reg remains the same from clock 0 to 4,

// decrement bit\_count and move to the new state at

// clock 4.

S\_WR\_ADDR\_0: begin

cmd <= cmd;

scl\_pad\_oen <= HIGH;

sda\_pad\_oen <= HIGH;

scl\_pad\_o <= LOW;

sda\_pad\_o <= data\_reg[bit\_count];

data\_reg <= data\_reg;

bit\_count <= bit\_count;

al\_count <= al\_count;

next\_state <= S\_WR\_ADDR\_1;

end

S\_WR\_ADDR\_1: begin

cmd <= cmd;

scl\_pad\_oen <= HIGH;

sda\_pad\_oen <= HIGH;

scl\_pad\_o <= HIGH;

sda\_pad\_o <= sda\_pad\_o;

data\_reg <= data\_reg;

bit\_count <= bit\_count;

al\_count <= al\_count;

next\_state <= S\_WR\_ADDR\_2;

end

S\_WR\_ADDR\_2: begin

cmd <= (bit\_count == 0)? CMD\_RD\_ACK : cmd;

scl\_pad\_oen <= HIGH;

sda\_pad\_oen <= HIGH;

scl\_pad\_o <= HIGH;

sda\_pad\_o <= sda\_pad\_o;

data\_reg <= data\_reg;

bit\_count <= bit\_count;

al\_count <= al\_count;

next\_state <= S\_WR\_ADDR\_3;

end

S\_WR\_ADDR\_3: begin

cmd <= cmd;

scl\_pad\_oen <= HIGH;

sda\_pad\_oen <= HIGH;

scl\_pad\_o <= LOW;

sda\_pad\_o <= sda\_pad\_o;

data\_reg <= data\_reg;

bit\_count <= bit\_count -1;

al\_count <= al\_count;

next\_state <= (cmd == CMD\_RD\_ACK)? S\_RD\_ACK1\_0: S\_WR\_ADDR\_0;

end

// READ STATE

// Waiting for an acknowledge from a slave device.

// Expected the ack signal to be ready before rising edge

// of SCL clock. If the signal is valid, set data\_reg with

// i2c\_data\_i, and move to the next state.

// Otherwise, come back to wait again.

S\_RD\_ACK1\_0: begin

cmd <= cmd;

scl\_pad\_oen <= HIGH;

sda\_pad\_oen <= LOW;

scl\_pad\_o <= LOW;

sda\_pad\_o <= sda\_pad\_o;

data\_reg <= data\_reg;

bit\_count <= bit\_count;

al\_count <= al\_count;

next\_state <= (al\_count == 3)? S\_IDLE : S\_RD\_ACK1\_1;

end

S\_RD\_ACK1\_1: begin

cmd <= (sda\_pad\_i == LOW)? CMD\_WR\_DATA: cmd;

scl\_pad\_oen <= HIGH;

sda\_pad\_oen <= LOW;

scl\_pad\_o <= HIGH;

sda\_pad\_o <= sda\_pad\_o;

data\_reg <= data\_reg;

bit\_count <= bit\_count;

al\_count <= al\_count;

next\_state <= S\_RD\_ACK1\_2;

end

S\_RD\_ACK1\_2: begin

cmd <= (sda\_pad\_i == LOW)? cmd: CMD\_RD\_ACK;

scl\_pad\_oen <= HIGH;

sda\_pad\_oen <= LOW;

scl\_pad\_o <= HIGH;

sda\_pad\_o <= sda\_pad\_o;

data\_reg <= data\_reg;

bit\_count <= bit\_count;

al\_count <= al\_count;

next\_state <= S\_RD\_ACK1\_3;

end

S\_RD\_ACK1\_3: begin

cmd <= cmd;

scl\_pad\_oen <= HIGH;

sda\_pad\_oen <= LOW;

scl\_pad\_o <= LOW;

sda\_pad\_o <= sda\_pad\_o;

if(cmd == CMD\_WR\_DATA) begin

data\_reg <= i2c\_data\_i;

bit\_count <= 3'd7;

al\_count <= 0;

next\_state <= S\_WR\_DATA\_0;

end

else begin

data\_reg <= data\_reg;

bit\_count <= bit\_count;

al\_count <= al\_count +1;

next\_state <= S\_RD\_ACK1\_0;

end

end

// WRITE STATE

// Similar to above state, but outputting data instead of

// the concentration of {slave address, write bit}

S\_WR\_DATA\_0: begin

cmd <= cmd;

scl\_pad\_oen <= HIGH;

sda\_pad\_oen <= HIGH;

scl\_pad\_o <= LOW;

sda\_pad\_o <= data\_reg[bit\_count];

data\_reg <= data\_reg;

bit\_count <= bit\_count;

al\_count <= al\_count;

next\_state <= S\_WR\_DATA\_1;

end

S\_WR\_DATA\_1: begin

cmd <= cmd;

scl\_pad\_oen <= HIGH;

sda\_pad\_oen <= HIGH;

scl\_pad\_o <= HIGH;

sda\_pad\_o <= sda\_pad\_o;

data\_reg <= data\_reg;

bit\_count <= bit\_count;

al\_count <= al\_count;

next\_state <= S\_WR\_DATA\_2;

end

S\_WR\_DATA\_2: begin

cmd <= (bit\_count == 0)? CMD\_RD\_ACK : cmd;

scl\_pad\_oen <= HIGH;

sda\_pad\_oen <= HIGH;

scl\_pad\_o <= HIGH;

sda\_pad\_o <= sda\_pad\_o;

data\_reg <= data\_reg;

bit\_count <= bit\_count;

al\_count <= al\_count;

next\_state <= S\_WR\_DATA\_3;

end

S\_WR\_DATA\_3: begin

cmd <= cmd;

scl\_pad\_oen <= HIGH;

sda\_pad\_oen <= HIGH;

scl\_pad\_o <= LOW;

sda\_pad\_o <= sda\_pad\_o;

data\_reg <= data\_reg;

bit\_count <= bit\_count -1;

al\_count <= al\_count;

next\_state <= (cmd == CMD\_RD\_ACK)? S\_RD\_ACK2\_0: S\_WR\_DATA\_0;

end

// READ STATE

// Similar to above state, but destination is STOP State

S\_RD\_ACK2\_0: begin

cmd <= cmd;

scl\_pad\_oen <= HIGH;

sda\_pad\_oen <= LOW;

scl\_pad\_o <= LOW;

sda\_pad\_o <= sda\_pad\_o;

data\_reg <= data\_reg;

bit\_count <= bit\_count;

al\_count <= al\_count;

next\_state <= (al\_count == 3)? S\_IDLE : S\_RD\_ACK2\_1;

end

S\_RD\_ACK2\_1: begin

cmd <= (sda\_pad\_i == LOW)? CMD\_STOP: CMD\_RD\_ACK;

scl\_pad\_oen <= HIGH;

sda\_pad\_oen <= LOW;

scl\_pad\_o <= HIGH;

sda\_pad\_o <= sda\_pad\_o;

data\_reg <= data\_reg;

bit\_count <= bit\_count;

al\_count <= al\_count;

next\_state <= S\_RD\_ACK2\_2;

end

S\_RD\_ACK2\_2: begin

cmd <= (sda\_pad\_i == LOW)? cmd: CMD\_RD\_ACK;

scl\_pad\_oen <= HIGH;

sda\_pad\_oen <= LOW;

scl\_pad\_o <= HIGH;

sda\_pad\_o <= sda\_pad\_o;

data\_reg <= data\_reg;

bit\_count <= bit\_count;

al\_count <= al\_count;

next\_state <= S\_RD\_ACK2\_3;

end

S\_RD\_ACK2\_3: begin

cmd <= cmd;

scl\_pad\_oen <= HIGH;

sda\_pad\_oen <= LOW;

scl\_pad\_o <= LOW;

sda\_pad\_o <= sda\_pad\_o;

data\_reg <= data\_reg;

bit\_count <= bit\_count;

al\_count <= al\_count +1;

next\_state <= (cmd == CMD\_STOP)? S\_STOP\_0 : S\_RD\_ACK2\_0;

end

// STOP STATE

// Both SCL and SDA are LOW, then SCL HIGH and SDA LOW

// before moving to IDLE STATE

S\_STOP\_0: begin

cmd <= CMD\_STOP;

scl\_pad\_oen <= HIGH;

sda\_pad\_oen <= HIGH;

scl\_pad\_o <= LOW;

sda\_pad\_o <= LOW;

data\_reg <= data\_reg;

bit\_count <= bit\_count;

next\_state <= S\_STOP\_1;

end

S\_STOP\_1: begin

cmd <= CMD\_IDLE;

scl\_pad\_oen <= HIGH;

sda\_pad\_oen <= HIGH;

scl\_pad\_o <= HIGH;

sda\_pad\_o <= LOW;

data\_reg <= data\_reg;

bit\_count <= bit\_count;

next\_state <= S\_IDLE;

end

endcase

end

end

endmodule

Appendix d: I2C\_Interface Source Code

`timescale 1ns **/** 1ps

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

// Class: CECS490B Senior Projects //

// Project name: Garbage Collector //

// File name: I2C\_Interface.v //

// //

// Created by Chanartip Soonthornwan on October 10, 2018. //

// Copyright @ 2018 Chanartip Soonthornwan. All rights reserved. //

// //

// Abstract: Interconnection of Tramel Blaze and I2C. //

// This module contains register for holding //

// I2C address and I2C data to be ready before //

// the I2C Core initiates the transmission. //

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

**module** I2C\_Interface **(**

**input** **wire** clk**,** // System Clock Input

**input** **wire** rst**,** // Synchronous Reset input

**input** **wire** bt\_fire\_i**,** // Button Down for firing next Transmission

**input** **wire** **[**6**:**0**]** sw\_addr\_i**,** // Switch[15:9] address input

**input** **wire** **[**7**:**0**]** sw\_data\_i**,** // Switch[7:0] data input

// From/To I2C

**input** **wire** i2c\_ready\_i**,** // Ready status from I2C Core

**output** **reg** **[**6**:**0**]** i2c\_address\_o**,** // Address output to I2C Core

**output** **reg** **[**7**:**0**]** i2c\_data\_o**,** // Data output to I2C Core

**output** **reg** i2c\_start\_o**,** // Start signal to I2C Core

// From/To Tramel Blaze

**input** **wire** **[**15**:**0**]** tb\_port\_id\_i**,** // Port ID from Tramel Blaze

**input** **wire** **[**15**:**0**]** tb\_data\_i**,** // Out-port data from Tramel Blaze

**input** **wire** tb\_write\_st\_i**,** // Write Strobe from Tramel Blaze

**input** **wire** tb\_read\_st\_i**,** // Read Strobe from Tramel Blaze

**input** **wire** tb\_intr\_ack\_i**,** // Interrupt Acknowledge from Tramel Blaze

**output** **reg** tb\_intr\_r\_o**,** // Interrupt Request to Tramel Blaze

**output** **reg** **[**15**:**0**]** tb\_data\_o // ready status to Tramel Blaze

**);**

// Parameters for reserved I2C PORTs

**localparam** I2C\_ADDR\_PORT **=** 16'h01**;**

**localparam** I2C\_DATA\_PORT **=** 16'h02**;**

**localparam** I2C\_CMD\_PORT **=** 16'h03**;**

**localparam** I2C\_STATUS\_PORT **=** 16'h04**;**

**localparam** SW\_ADDR\_PORT **=** 16'h05**;**

**localparam** SW\_DATA\_PORT **=** 16'h06**;**

// Reset signal for all flops

**wire** reset**;**

**wire** ready\_pulse\_w**;**

**assign** reset **=** rst **|** ready\_pulse\_w**;**

**wire** bt\_fire\_db**;**

**wire** bt\_fire\_pulse\_w**;**

// Positive Edge Detector

// Generate an impulsive signal from

// the edge changing from LOW to HIGH.

PED

ready\_pulse\_ped**(**

**.**clk **(**clk **),**

**.**rst **(**rst **),**

**.**d\_in **(**i2c\_ready\_i **),**

**.**pulse **(**ready\_pulse\_w**)**

**),**

button\_ready\_pulse\_ped**(**

**.**clk **(**clk **),**

**.**rst **(**rst **),**

**.**d\_in **(**bt\_fire\_db **),**

**.**pulse **(**bt\_fire\_pulse\_w**)**

**);**

// Debounce Module

// to stabilize firing input from the button down

// because the button is a mechanical input which might cause

// an unstable input in a short period of time,

// such that this module will generate a stable signal.

Debounce db\_mod**(**

**.**clk**(**clk**),**

**.**rst**(**rst**),**

**.**db\_in**(**bt\_fire\_i**),**

**.**db\_out**(**bt\_fire\_db**)**

**);**

// Tramel Blaze Data\_input

// Assign a 16-bit TB input either input data or I2C status.

// If port\_id is I2C\_STATUS\_PORT and read strobe is HIGH,

// TB input is 15'b0 with i2c\_ready signal, otherwise 16'b0.

// \*\*Note: it will be only I2C status for this project

**always@(\*)** **begin**

**if(**tb\_read\_st\_i**)** **begin**

**case(**tb\_port\_id\_i**)**

I2C\_STATUS\_PORT**:** tb\_data\_o **=** **{**15'b0**,** i2c\_ready\_i**};**

SW\_ADDR\_PORT**:** tb\_data\_o **=** **{** 9'b0**,** sw\_addr\_i**};**

SW\_DATA\_PORT**:** tb\_data\_o **=** **{** 8'b0**,** sw\_data\_i**};**

**default** **:** tb\_data\_o **=** 16'b0**;**

**endcase**

**end**

**else**

tb\_data\_o **=** 16'b0**;**

**end**

// RS-Flop for resetting the interrupt request

// for the Tramel Blaze when I2C is ready

**always@(posedge** clk**,** **posedge** rst**)** **begin**

**if(**rst**)** tb\_intr\_r\_o **<=** 1'b0**;**

**else** **if(**tb\_intr\_ack\_i**)** tb\_intr\_r\_o **<=** 1'b0**;**

**else** **if(**bt\_fire\_pulse\_w**)** tb\_intr\_r\_o **<=** 1'b1**;**

**else** tb\_intr\_r\_o **<=** tb\_intr\_r\_o**;**

**end**

// I2C Address Register

// Change output when port id is I2C\_ADDR\_PORT,

// otherwise output stays the same.

**always@(posedge** clk**,** **posedge** reset**)** **begin**

**if(**reset**)**

i2c\_address\_o **<=** 7'b0**;**

**else** **if(**tb\_write\_st\_i **&(**tb\_port\_id\_i **==** I2C\_ADDR\_PORT**))**

i2c\_address\_o **<=** tb\_data\_i**[**6**:**0**];**

**else**

i2c\_address\_o **<=** i2c\_address\_o**;**

**end**

// I2C Data Register

// Change output when port id is I2C\_DATA\_PORT,

// otherwise output stays the same.

**always@(posedge** clk**,** **posedge** reset**)** **begin**

**if(**reset**)**

i2c\_data\_o **<=** 8'b0**;**

**else** **if(**tb\_write\_st\_i **&(**tb\_port\_id\_i **==** I2C\_DATA\_PORT**))**

i2c\_data\_o **<=** tb\_data\_i**[**7**:**0**];**

**else**

i2c\_data\_o **<=** i2c\_data\_o**;**

**end**

// I2C Start signal Register

// Change output when port id is I2C\_CMD\_PORT,

// data in[0] is high, and write\_strobe is high,

// otherwise output stays the same.

**wire** i2c\_write\_w**;**

**assign** i2c\_write\_w **=** tb\_data\_i**[**0**]** **&** tb\_write\_st\_i

**&** **(**tb\_port\_id\_i **==** I2C\_CMD\_PORT**);**

**always@(posedge** clk**,** **posedge** reset**)** **begin**

**if(**reset**)** i2c\_start\_o **<=** 1'b0**;**

**else** **if(!**i2c\_ready\_i**)** i2c\_start\_o **<=** 1'b0**;**

**else** **if(**i2c\_write\_w**)** i2c\_start\_o **<=** tb\_data\_i**[**0**];**

**else** i2c\_start\_o **<=** i2c\_start\_o**;**

**end**

**endmodule**

//\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Positive Edge Detector\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

// A module to detect Positive Edge input then returns one-shot pulse

// output. If the input is HIGH at the first clock and second clock period,

// PED would detect this and output HIGH for one clock period.

**module** PED**(**clk**,** rst**,** d\_in**,** pulse**);**

**input** clk**,** rst**;** // on-board clock, and AISO reset signal

**input** d\_in**;** // input signal

**output** **wire** pulse**;** // one-shot pulse

**reg** q1**,**q2**;** // registers

**always@(posedge** clk**,** **posedge** rst**)**

**if(**rst**)** **{**q1**,** q2**}** **<=** 2'b00**;** // reset

**else** **{**q1**,** q2**}** **<=** **{**d\_in**,** q1**};** // q2 gets q1, and q1 get new signal

// output at the moment of input change

// q1 \_\_\_\_------------\_\_\_\_\_\_\_\_\_

// q2 \_\_\_\_\_\_\_\_------------\_\_\_\_\_

// pulse \_\_\_\_----\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**assign** pulse **=** q1 **&** **~**q2**;**

**endmodule**

//\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ End of PED \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Appendix e: TSI Source Code

`timescale 1ns / 1ps

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

// Class: CECS490B Senior Projects //

// Project name: Garbage Collector //

// File name: TSI.v //

// //

// Created by Chanartip Soonthornwan on November 12, 2018. //

// Copyright @ 2018 Chanartip Soonthornwan. All rights reserved. //

// //

// Abstract: Technology Specific Interface (TSI) is a module //

// includes buffers for inputs from other devices //

// to the SOPC, and the buffers for outputs from //

// the SOPC to outside devices. //

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

module TSI(

input wire clk, // on-board 100MHz Oscilloscope

input wire rst, // on-board button up reset signal

input wire bt\_fire\_i, // button down I2C firing command

input wire [6:0] sw\_addr\_i, // switches[15:9] I2C address

input wire [7:0] sw\_data\_i, // switches[7:0] I2C data

output wire clk\_o,

output wire rst\_o,

output wire bt\_fire\_o,

output wire [6:0] sw\_addr\_o,

output wire [7:0] sw\_data\_o,

output wire SCL, // I2C SCL bus

output wire [7:0] anode, // Seven-segment display's anodes

output wire [6:0] sseg, // seven segments output

input wire SCL\_i,

input wire [7:0] anode\_i,

input wire [6:0] sseg\_i

);

// Global Clock Buffer

BUFG BUFG\_inst (

.O(clk\_o),

.I(clk)

);

// Input

IBUF #(.IOSTANDARD("DEFAULT"))

rst\_inst(

.O(rst\_o),

.I(rst)

),

bt\_fire\_inst(

.O(bt\_fire\_o),

.I(bt\_fire\_i)

),

sw\_addr\_inst[6:0](

.O(sw\_addr\_o[6:0]),

.I(sw\_addr\_i[6:0])

),

sw\_data\_inst[7:0](

.O(sw\_data\_o[7:0]),

.I(sw\_data\_i[7:0])

);

OBUF #(.IOSTANDARD("DEFAULT"))

SCL\_inst(

.O(SCL),

.I(SCL\_i)

),

anode\_inst[7:0](

.O(anode[7:0]),

.I(anode\_i[7:0])

),

sseg\_inst[6:0](

.O(sseg[6:0]),

.I(sseg\_i[6:0])

);

endmodule

Appendix f: I2C\_Top\_Level Testbench

`timescale 1ns **/** 1ps

**module** I2C\_Top\_Level\_tb**;**

// Inputs

**reg** clk**;**

**reg** rst**;**

**reg** bt\_fire\_i**;** // button down input

**reg** **[**6**:**0**]** sw\_addr\_i**;** // switches[15:9] for 7-bit address input

**reg** **[**7**:**0**]** sw\_data\_i**;** // switches[7:0] for 8-bit data input

// Outputs

**wire** SCL**;**

// Bidirs

**wire** SDA**;**

**localparam** DB\_TIME **=** 6050000**\***5**;** // 30 ms debounce time

// Instantiate the Unit Under Test (UUT)

I2C\_Top\_Level master **(**

**.**clk**(**clk**),**

**.**rst**(**rst**),**

**.**SCL**(**SCL**),**

**.**SDA**(**SDA**),**

**.**bt\_fire\_i**(**bt\_fire\_i**),**

**.**sw\_addr\_i**(**sw\_addr\_i**),**

**.**sw\_data\_i**(**sw\_data\_i**)**

**);**

// Instantiate a Slave unit with address of 7'h72

I2CTest **#(** **.**slaveaddress**(**7'h72**)** **)**

slave1**(**

**.**CLCK**(**clk**),**

**.**SCL**(**SCL**),**

**.**SDA**(**SDA**)**

**);**

// Instantiate a Slave unit with address of 7'h55

I2CTest **#(** **.**slaveaddress**(**7'h55**)** **)**

slave2**(**

**.**CLCK**(**clk**),**

**.**SCL**(**SCL**),**

**.**SDA**(**SDA**)**

**);**

// Instantiate a Slave unit with address of 7'h01

I2CTest **#(** **.**slaveaddress**(**7'h01**)** **)**

slave3**(**

**.**CLCK**(**clk**),**

**.**SCL**(**SCL**),**

**.**SDA**(**SDA**)**

**);**

**always** **#**5 clk **=** **~**clk**;** // Toggle on-board clock every 5 ns

**initial** **begin**

// Initialize Inputs

clk **=** 0**;**

rst **=** 1**;**

bt\_fire\_i **=** 0**;**

sw\_addr\_i **=** 7'h01**;**

sw\_data\_i **=** 8'hAA**;**

// Reset the design to a known state

**@(negedge** clk**)** rst **=** 0**;**

**#**100 bt\_fire\_i **=** 1**;**

**#**DB\_TIME bt\_fire\_i **=** 0**;**

// Setting switches input and press the fire button, then release

sw\_addr\_i **=** 7'h72**;**

sw\_data\_i **=** 8'hAA**;**

**#**DB\_TIME bt\_fire\_i **=** 1**;**

**#**DB\_TIME bt\_fire\_i **=** 0**;**

// Setting switches input and press the fire button, then release

sw\_addr\_i **=** 7'h72**;**

sw\_data\_i **=** 8'h01**;**

**#**DB\_TIME bt\_fire\_i **=** 1**;**

**#**DB\_TIME bt\_fire\_i **=** 0**;**

// Setting switches input and press the fire button, then release

sw\_addr\_i **=** 7'h55**;**

sw\_data\_i **=** 8'h02**;**

// Setting switches input and press the fire button, then release

sw\_addr\_i **=** 7'h12**;**

sw\_data\_i **=** 8'h03**;**

**#**DB\_TIME bt\_fire\_i **=** 1**;**

**#**DB\_TIME bt\_fire\_i **=** 0**;**

// Setting switches input and press the fire button, then release

sw\_addr\_i **=** 7'h13**;**

sw\_data\_i **=** 8'h04**;**

**#**DB\_TIME bt\_fire\_i **=** 1**;**

**#**DB\_TIME bt\_fire\_i **=** 0**;**

// Note: this time, the master will wait for slave's acknowledge forever.

**#**DB\_TIME $finish**;**

**end**

**endmodule**

Appendix g: I2C\_Core Testbench

`timescale 1ns **/** 1ps

**module** I2C\_Core\_tb**;**

// Inputs

**reg** clk**;**

**reg** rst**;**

**reg** **[**6**:**0**]** i2c\_addr\_i**;**

**reg** **[**7**:**0**]** i2c\_data\_i**;**

**reg** i2c\_start\_i**;**

// Outputs

**wire** i2c\_ready\_o**;**

**wire** SCL**;**

// Bidirs

**wire** SDA**;**

**localparam**

S\_RD\_ACK1\_0 **=** 7**,** S\_RD\_ACK1\_1 **=** 8**,** S\_RD\_ACK1\_2 **=** 9**,** S\_RD\_ACK1\_3 **=** 10**,**

S\_RD\_ACK2\_0 **=** 15**,** S\_RD\_ACK2\_1 **=** 16**,** S\_RD\_ACK2\_2 **=** 17**,** S\_RD\_ACK2\_3 **=** 18**;**

// Instantiate the Unit Under Test (UUT)

I2C\_Core uut **(**

**.**clk**(**clk**),**

**.**rst**(**rst**),**

**.**i2c\_addr\_i**(**i2c\_addr\_i**),**

**.**i2c\_data\_i**(**i2c\_data\_i**),**

**.**i2c\_start\_i**(**i2c\_start\_i**),**

**.**i2c\_ready\_o**(**i2c\_ready\_o**),**

**.**SCL**(**SCL**),**

**.**SDA**(**SDA**)**

**);**

**always** **#**5 clk **=** **~**clk**;** // Toggle on-board clock every 5 ns

// Simulate acknowledge bit from a slave at Read Acknowledge State

**assign** uut**.**sda\_pad\_i **=** **(**uut**.**next\_state **==** S\_RD\_ACK1\_0**)?** 1 **:**

**(**uut**.**next\_state **==** S\_RD\_ACK1\_1**)?** 0 **:**

**(**uut**.**next\_state **==** S\_RD\_ACK1\_2**)?** 0 **:**

**(**uut**.**next\_state **==** S\_RD\_ACK1\_3**)?** 1 **:**

**(**uut**.**next\_state **==** S\_RD\_ACK2\_0**)?** 1 **:**

**(**uut**.**next\_state **==** S\_RD\_ACK2\_1**)?** 0 **:**

**(**uut**.**next\_state **==** S\_RD\_ACK2\_2**)?** 0 **:**

**(**uut**.**next\_state **==** S\_RD\_ACK2\_3**)?** 1 **:** 1**;**

**initial** **begin**

// Initialize Inputs

clk **=** 0**;**

rst **=** 1**;**

i2c\_addr\_i **=** 0**;**

i2c\_data\_i **=** 0**;**

i2c\_start\_i **=** 0**;**

// Reset signal brings the design to a known state

**@(negedge** clk**)** rst **=** 0**;**

// Setup inputs from I2C\_Interface and then wait for some time

**@(negedge** clk**)** i2c\_addr\_i **=** 7'h72**;**

**@(negedge** clk**)** i2c\_data\_i **=** 8'h45**;**

**#**10000

**@(negedge** clk**)** i2c\_start\_i **=** 1'b1**;**

**#**10000

**@(negedge** clk**)** i2c\_start\_i **=** 1'b0**;**

**#**1000000

// Setup inputs from I2C\_Interface and then wait for some time

**@(negedge** clk**)** i2c\_addr\_i **=** 7'h41**;**

**@(negedge** clk**)** i2c\_data\_i **=** 8'h92**;**

**#**10000

**@(negedge** clk**)** i2c\_start\_i **=** 1'b1**;**

**#**10000

**@(negedge** clk**)** i2c\_start\_i **=** 1'b0**;**

**#**1000000 $stop**;**

**end**

**endmodule**

Appendix h: I2C\_Interface Testbench

`timescale 1ns **/** 1ps

**module** i2c\_interface\_tb**;**

// Inputs

**reg** clk**;**

**reg** rst**;**

**reg** bt\_fire\_i**;**

**reg** **[**6**:**0**]** sw\_addr\_i**;**

**reg** **[**7**:**0**]** sw\_data\_i**;**

**reg** i2c\_ready\_i**;**

**reg** **[**15**:**0**]** tb\_port\_id\_i**;**

**reg** **[**15**:**0**]** tb\_data\_i**;**

**reg** tb\_write\_st\_i**;**

**reg** tb\_read\_st\_i**;**

**reg** tb\_intr\_ack\_i**;**

// Outputs

**wire** **[**6**:**0**]** i2c\_address\_o**;**

**wire** **[**7**:**0**]** i2c\_data\_o**;**

**wire** i2c\_start\_o**;**

**wire** tb\_intr\_r\_o**;**

**wire** **[**15**:**0**]** tb\_data\_o**;**

**localparam** DB\_TIME **=** 6050000**\***5**;** // 30 ms debounce time

// Instantiate the Unit Under Test (UUT)

I2C\_Interface uut **(**

**.**clk**(**clk**),**

**.**rst**(**rst**),**

**.**bt\_fire\_i**(**bt\_fire\_i**),**

**.**sw\_addr\_i**(**sw\_addr\_i**),**

**.**sw\_data\_i**(**sw\_data\_i**),**

**.**i2c\_ready\_i**(**i2c\_ready\_i**),**

**.**i2c\_address\_o**(**i2c\_address\_o**),**

**.**i2c\_data\_o**(**i2c\_data\_o**),**

**.**i2c\_start\_o**(**i2c\_start\_o**),**

**.**tb\_port\_id\_i**(**tb\_port\_id\_i**),**

**.**tb\_data\_i**(**tb\_data\_i**),**

**.**tb\_write\_st\_i**(**tb\_write\_st\_i**),**

**.**tb\_read\_st\_i**(**tb\_read\_st\_i**),**

**.**tb\_intr\_ack\_i**(**tb\_intr\_ack\_i**),**

**.**tb\_intr\_r\_o**(**tb\_intr\_r\_o**),**

**.**tb\_data\_o**(**tb\_data\_o**)**

**);**

**always** **#**5 clk **=** **~**clk**;** // Toggle on-board clock every 5 ns

**initial** **begin**

// Initialize Inputs

clk **=** 0**;**

rst **=** 1**;**

bt\_fire\_i **=** 0**;**

sw\_addr\_i **=** 7'h72**;**

sw\_data\_i **=** 8'hAB**;**

i2c\_ready\_i **=** 1**;**

tb\_port\_id\_i **=** 0**;**

tb\_data\_i **=** 0**;**

tb\_write\_st\_i **=** 0**;**

tb\_read\_st\_i **=** 0**;**

tb\_intr\_ack\_i **=** 0**;**

// Reset the design to a known state

**@(negedge** clk**)** rst **=** 0**;**

// Fire a start signal for I2C\_Core,

// but it needs DB\_TIME (30ms) to stabilize

// the push button signal, then bt\_db\_pulse

// will be generated for launching the I2C\_Core

**#**100 bt\_fire\_i **=** 1**;**

**@(negedge** clk**)** tb\_read\_st\_i **=** 1**;** // start read

**@(negedge** clk**)** tb\_port\_id\_i **=** 16'h0004**;** // read status

**@(negedge** clk**)** tb\_port\_id\_i **=** 16'h0005**;** // read sw\_address

**@(negedge** clk**)** tb\_port\_id\_i **=** 16'h0006**;** // read sw\_data

**@(negedge** clk**)** tb\_port\_id\_i **=** 16'h0000**;** // reset port\_id

**@(negedge** clk**)** tb\_read\_st\_i **=** 0**;** // stop read

**@(negedge** clk**)** tb\_write\_st\_i **=** 1**;** // start write

**@(negedge** clk**)**

tb\_data\_i **=** 16'h0033**;** // write '33' to i2c\_addr\_o

tb\_port\_id\_i **=** 16'h0001**;** // write address

**@(negedge** clk**)**

tb\_data\_i **=** 16'h0044**;** // write '44' to i2c\_data\_o

tb\_port\_id\_i **=** 16'h0002**;** // write data

**@(negedge** uut**.**bt\_fire\_pulse\_w**)** tb\_port\_id\_i **=** 16'h03**;** // write start

**@(negedge** clk**)** tb\_port\_id\_i **=** 16'h0000**;** // reset port\_id

**@(negedge** clk**)** tb\_write\_st\_i **=** 0**;** // stop write

// Simulating that I2C\_Core Receive the start signal

// and move to Start condition state and set the

// ready signal to low indicating that it is busy.

**@(negedge** clk**)** i2c\_ready\_i **=** 0**;**

**#**100 i2c\_ready\_i **=** 1**;**

**#**DB\_TIME bt\_fire\_i **=** 0**;**

**#**500 $finish**;**

**end**

**endmodule**

Intentionally left blank.