

COLLEGE OF ENGINEERING

Department of Computer Engineering and Computer Science

**I2C Protocol On FPGA**

By

Chanartip Soonthornwan (014353883)

On October 24, 2018

Intentionally left blank.

Table of Contents

[1 Introduction 4](#_Toc528155931)

[2 Features 4](#_Toc528155932)

[3 General I2C Operation 4](#_Toc528155933)

[4 Data Validity 5](#_Toc528155934)

[5 Design Methodology 6](#_Toc528155935)

[6 Verilog 6](#_Toc528155936)

[7 I2C for Garbage Collector 7](#_Toc528155937)

[8 I2C With a TramelBlaze 8](#_Toc528155938)

[9 Externally Developed Blocks 8](#_Toc528155939)

[9.1 TramelBlaze 8](#_Toc528155940)

[10 Internally Developed Blocks 9](#_Toc528155941)

[10.1 I2C Top Level Design 9](#_Toc528155942)

[10.2 I2C\_Core 12](#_Toc528155943)

[10.3 I2C Interface 15](#_Toc528155944)

Table of Appendices

[Appendix a: I2C Top Level Source Code 18](#_Toc528155945)

[Appendix b: I2C\_Core Source Code 21](#_Toc528155946)

[Appendix c: I2C\_Interface Source Code 28](#_Toc528155947)

[Appendix d: I2C\_Top\_Level Testbench 31](#_Toc528155948)

[Appendix e: I2C\_Core Testbench 33](#_Toc528155949)

[Appendix f: I2C\_Interface Testbench 35](#_Toc528155950)

Table of Figures

[Figure 1: Example of I2C Bus 4](#_Toc528155951)

[Figure 2: START and STOP condition 4](https://d.docs.live.net/95182daea981e71a/2018Fall_CSULB/CECS490B/Current_Project/Document/I2C_Document.docx#_Toc528155952)

[Figure 3: I2C master to slave communication 5](#_Toc528155953)

[Figure 5: Acknowledge from a slave 5](https://d.docs.live.net/95182daea981e71a/2018Fall_CSULB/CECS490B/Current_Project/Document/I2C_Document.docx#_Toc528155954)

[Figure 4: SDA is set while SCL is LOW and remain stable while SCL is HIGH. 5](https://d.docs.live.net/95182daea981e71a/2018Fall_CSULB/CECS490B/Current_Project/Document/I2C_Document.docx#_Toc528155955)

[Figure 6: I2C simple state diagram 6](https://d.docs.live.net/95182daea981e71a/2018Fall_CSULB/CECS490B/Current_Project/Document/I2C_Document.docx#_Toc528155956)

[Figure 7: I2C Timing Diagram with States 7](https://d.docs.live.net/95182daea981e71a/2018Fall_CSULB/CECS490B/Current_Project/Document/I2C_Document.docx#_Toc528155957)

[Figure 7: TramelBlaze Block Diagram 8](https://d.docs.live.net/95182daea981e71a/2018Fall_CSULB/CECS490B/Current_Project/Document/I2C_Document.docx#_Toc528155958)

[Figure 9: Top Level Detail Block Diagram 9](https://d.docs.live.net/95182daea981e71a/2018Fall_CSULB/CECS490B/Current_Project/Document/I2C_Document.docx#_Toc528155959)

[Figure 8: Top Level Block Diagram 9](https://d.docs.live.net/95182daea981e71a/2018Fall_CSULB/CECS490B/Current_Project/Document/I2C_Document.docx#_Toc528155960)

[Figure 11: Top Level Verification 10](#_Toc528155961)

[Figure 12: Top Level Complete Transmission 10](#_Toc528155962)

[Figure 12: Top Level Complete Transmission Explanation 11](https://d.docs.live.net/95182daea981e71a/2018Fall_CSULB/CECS490B/Current_Project/Document/I2C_Document.docx#_Toc528155963)

[Figure 14: Top Level Incomplete Transmission 11](#_Toc528155964)

[Figure 14: I2C Core State Diagram 12](https://d.docs.live.net/95182daea981e71a/2018Fall_CSULB/CECS490B/Current_Project/Document/I2C_Document.docx#_Toc528155965)

[Figure 15: I2C Core Block Diagram 12](https://d.docs.live.net/95182daea981e71a/2018Fall_CSULB/CECS490B/Current_Project/Document/I2C_Document.docx#_Toc528155966)

[Figure 16: I2C\_Core Verification with Simulated Acknowledgement 13](https://d.docs.live.net/95182daea981e71a/2018Fall_CSULB/CECS490B/Current_Project/Document/I2C_Document.docx#_Toc528155967)

[Figure 18: I2C\_Core Timing Diagram 14](#_Toc528155968)

[Figure 18: I2C\_Interface Block Diagram 15](https://d.docs.live.net/95182daea981e71a/2018Fall_CSULB/CECS490B/Current_Project/Document/I2C_Document.docx#_Toc528155969)

[Figure 20: I2C\_Interface Reading and Writing phrase 16](#_Toc528155970)

[Figure 20: I2C\_Interface debouncing button input and clearing registers 17](https://d.docs.live.net/95182daea981e71a/2018Fall_CSULB/CECS490B/Current_Project/Document/I2C_Document.docx#_Toc528155971)

Table of Tables

[Table 1: Bidirectional Verilog code and Diagram 6](#_Toc528155972)

[Table 2: TramelBlaze I/O 8](#_Toc528155973)

[Table 3: I2C Top Level I/O Table 10](#_Toc528155974)

[Table 4: I2C Core I/O table 13](#_Toc528155975)

[Table 5: I2C\_Interface I/O Table 16](#_Toc528155976)

**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.

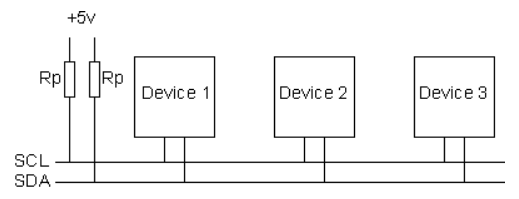


Figure 1: Example of I2C Bus

# Features

* A master and multiple slave operation
* Requires only two pins to interface I2C bus (SCL and SDA)
* Support standard rate 100 Kbps

# General I2C Operation

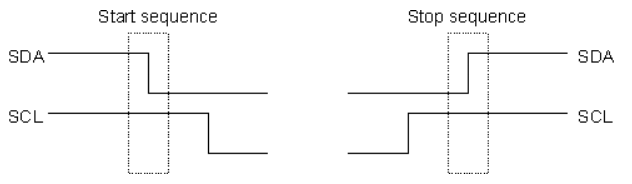
 When the I2C master demands to communicate to a slave, the master initiates transmission by pulling the SDA bus LOW while SCL is still HIGH to indicating the START condition (Figure 2). Then, shifting one bit at the time from the most significant bit (MSB) to the least significant bit (LSB) of the address 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

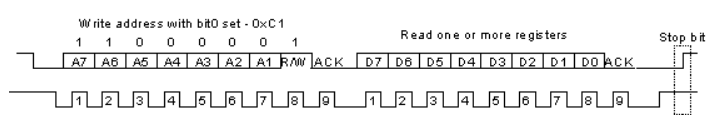
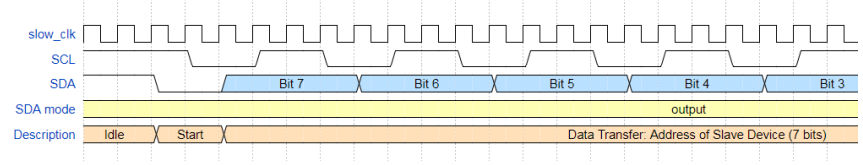


Figure 3: I2C master to slave communication

# 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 4). 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 5).

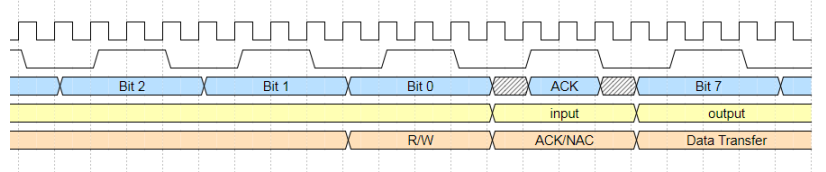


Figure 4: Acknowledge from a slave

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.

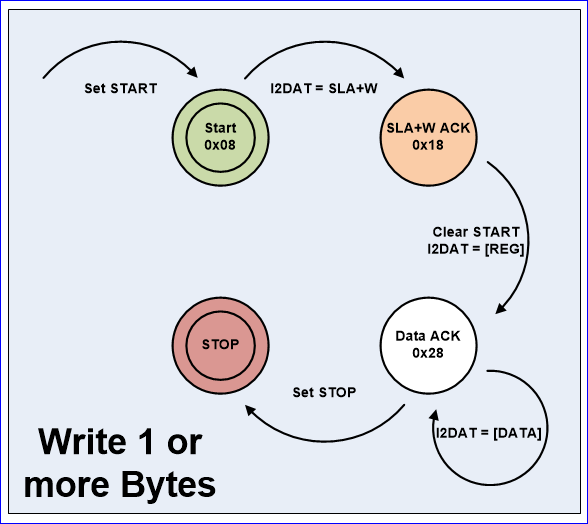


Figure 6: I2C simple state diagram

# 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).

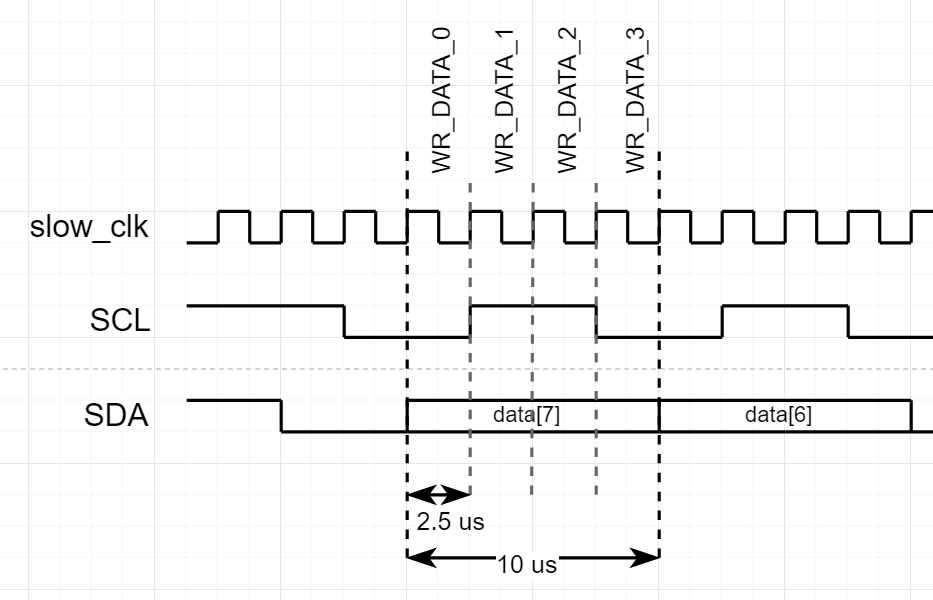


Figure 7: I2C Timing Diagram with States

# 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.

# Externally Developed Blocks

## TramelBlaze

**Description**

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.

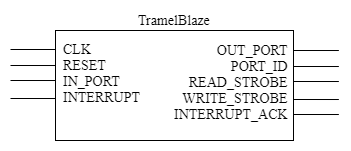


Figure 8: TramelBlaze Block Diagram

**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 2: TramelBlaze I/O

# Internally Developed Blocks

## I2C Top Level Design

**Description**

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**

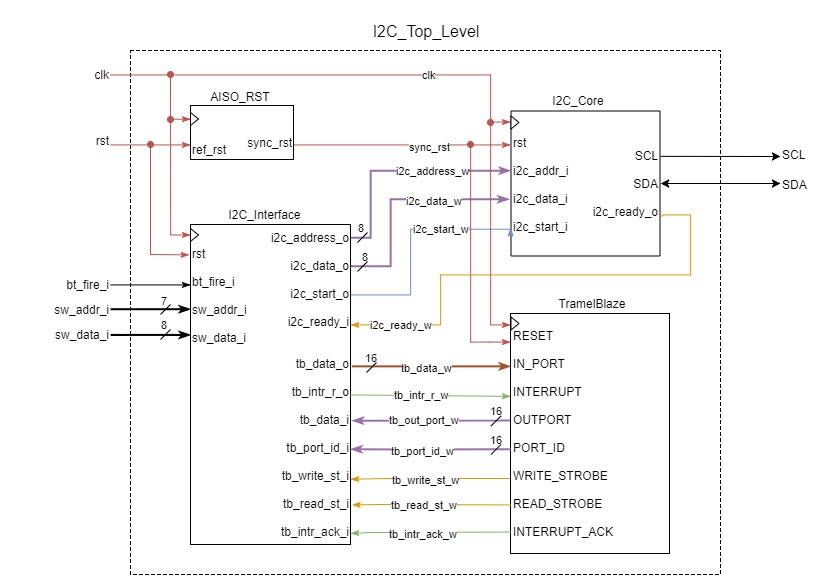


Figure 9: Top Level Detail Block Diagram

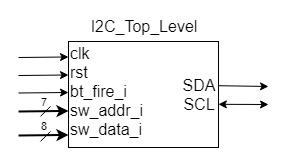


Figure 10: Top Level Block Diagram

**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 |

Table 3: I2C Top Level I/O Table

**Source Code:** Appendix a: 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.

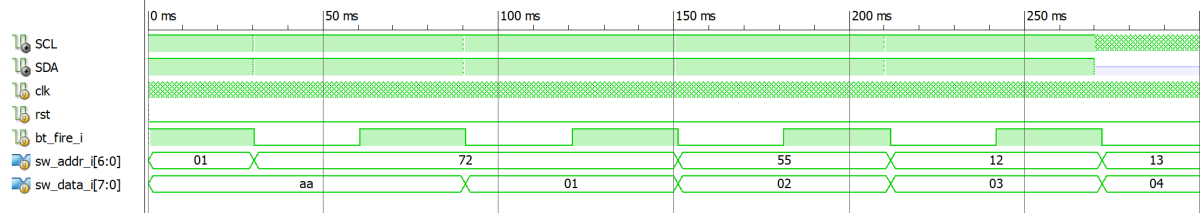


Figure 11: Top Level Verification

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.

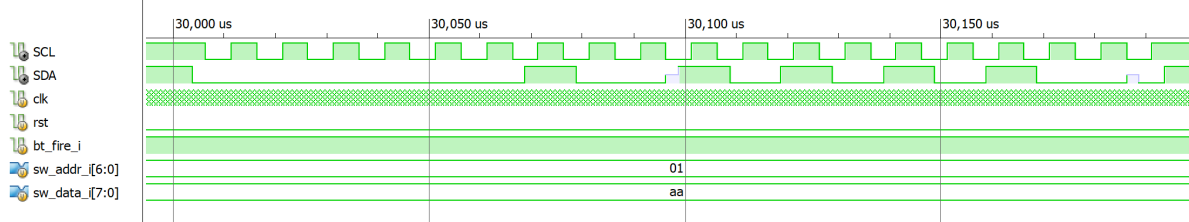


Figure 12: Top Level Complete Transmission

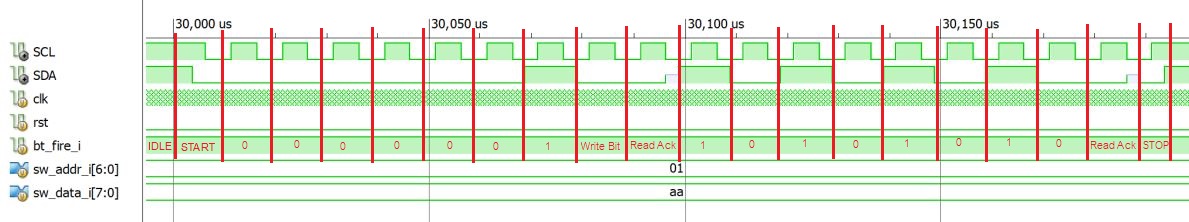
According to the state diagram (Figure 6), a complete transmission begins with a START condition which SDA line is pulled LOW following by SCL line, then transmits 7-bit data with a write bit (LOW) before waiting for an acknowledge from a slave. Since the slave is not created for specifically for this 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 12 ).

Figure 13: 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.

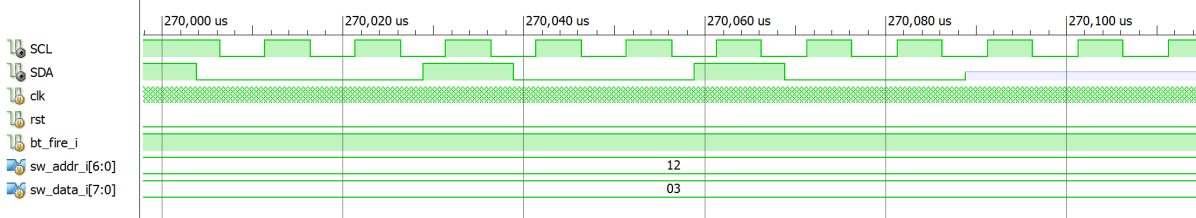


Figure 14: Top Level Incomplete Transmission

**Verification Source Code:** Appendix d: 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.

**Block Diagrams**

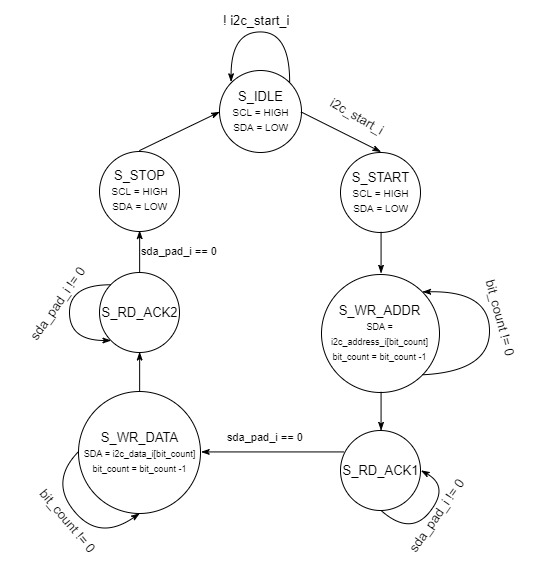


Figure 15: I2C Core State Diagram

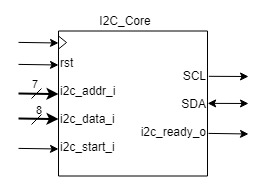


Figure 16: I2C Core Block Diagram

**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 | 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 4: I2C Core I/O table

**Source Code:** Appendix b: 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.

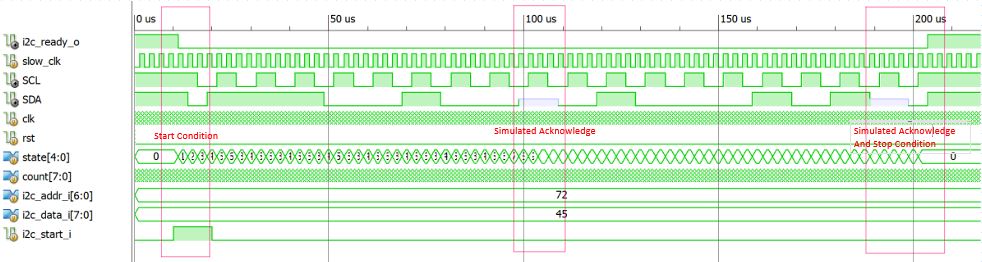


Figure 17: I2C\_Core Verification with Simulated Acknowledgement

As shown in Figure 16, 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.

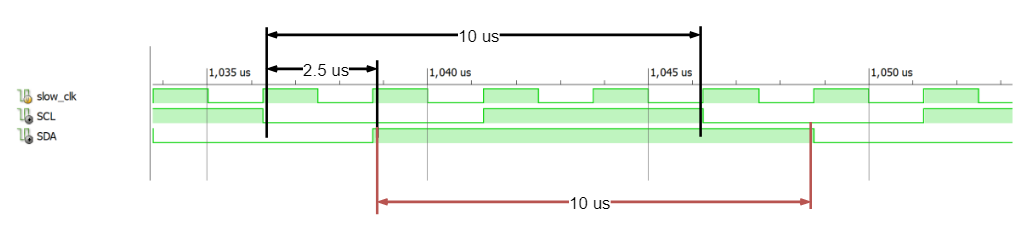


Figure 18: I2C\_Core Timing Diagram

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 e: I2C\_Core Testbench

## I2C Interface

**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.

**Block Diagrams**

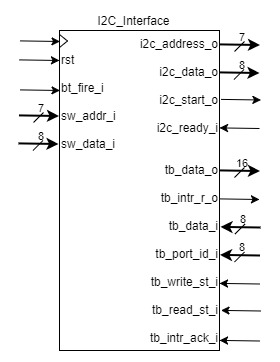


Figure 19: I2C\_Interface Block Diagram

**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] |
| 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 5: I2C\_Interface I/O Table

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

**Verification**

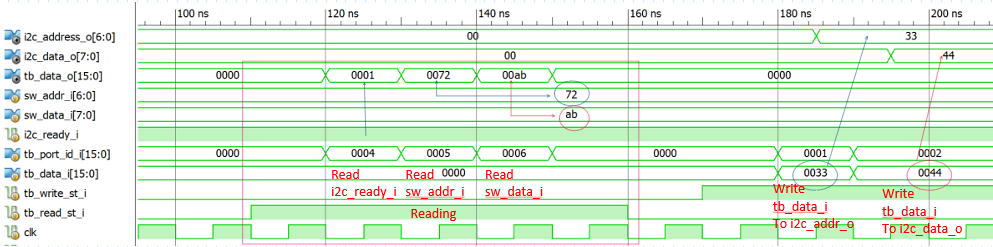
I2C\_Interface typically operates by combinational logic from inputs and outputs from TrambelBlaze(TB), I2C\_Core, and the FPGA. It is difficult to simulate the output without them, but I2C\_Interface could be breakdown to three parts; reading to TB, writing from TB, and clearing registers.

Figure 20: 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 20, 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.

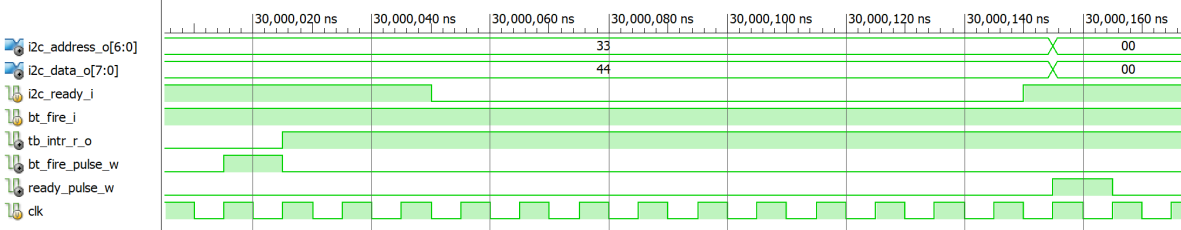


Figure 21: I2C\_Interface debouncing button input and clearing registers

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

Appendix a: 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 1.1 (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

**);**

**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**)**

**);**

// 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 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Appendix b: 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 2.0 (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**,**

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**]** 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**;**

next\_state **<=** S\_IDLE**;**

i2c\_ready\_o **<=** HIGH**;**

**end**

**else** **begin**

**case(**state**)**

// IDLE STATE

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

S\_IDLE**:** **begin**

**if(**i2c\_start\_i**)** **begin** // recieve 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**;**

next\_state **<=** S\_START\_0**;**

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

**end**

**else** **begin** // still idleling

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**;**

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**;**

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

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**;**

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**;**

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**;**

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**;**

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.

// \*\*Note: This version doesn't have Artibute Lost\*\*

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**;**

next\_state **<=** S\_RD\_ACK1\_1**;**

**end**

S\_RD\_ACK1\_1**:** **begin**

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

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**;**

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**;**

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

**end**

**else** **begin**

data\_reg **<=** data\_reg**;**

bit\_count **<=** bit\_count**;**

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

**end**

**end**

// WRITE STATE

// Similar to above state, but outputing 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**;**

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**;**

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**;**

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

**end**

S\_WR\_DATA\_3**:** **begin**

cmd **<=** cmd**;**

scl\_pad\_oen **<=** HIGH**;**

sda\_pad\_oen **<=** HIGH**;**//(cmd == CMD\_RD\_ACK)? LOW: HIGH;

scl\_pad\_o **<=** LOW**;**

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

data\_reg **<=** data\_reg**;**

bit\_count **<=** bit\_count **-**1**;**

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**;**

next\_state **<=** 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**;**

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**;**

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**;**

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 c: 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 d: 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 e: 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\_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 f: 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.