Qiming College of Huazhong University of Science & Technology

Name: ______________________ ID: _____________ Start Date: Wednesday, August 11, 2021

Name: ______________________ ID: _____________ Due Date: Thursday, August 12, 2021

Software and Hardware Co-Design with Zybo, Summer 2021 HUST
Lab #2 Phase I
I2C Master Module Implementation in Verilog on Zybo Board

This is a group lab to be performed by groups of two students. You are free to form your own groups. Each group will need to submit one report that should include names and CM numbers of both students. All submitted simulation waveforms must be annotated and initialed and dated by both students.

The objective of this part is to implement an I2C controller that can send the first byte to the receivers, which contains, Start, Address, Read/Write, Acknowledgment, and Stop.

1. Deliverables Due by Thursday, August 12, 2021

- *Submit a copy of your simulation waveforms of your controller and top-level circuits. Your screenshots should include the header of the window that shows files names as you can see from my example screenshots in this handout.*
- *Submit a copy of the I2C master waveforms to write address 7'b1001011 and READ operation to I2C bus captured on an oscilloscope that is annotated, initialed and dated by both students. Connect input signal Go to a slide switch. Show screen shots of a complete transmission from Idle to Start until Stop and another close-up screen shot of one frame.*

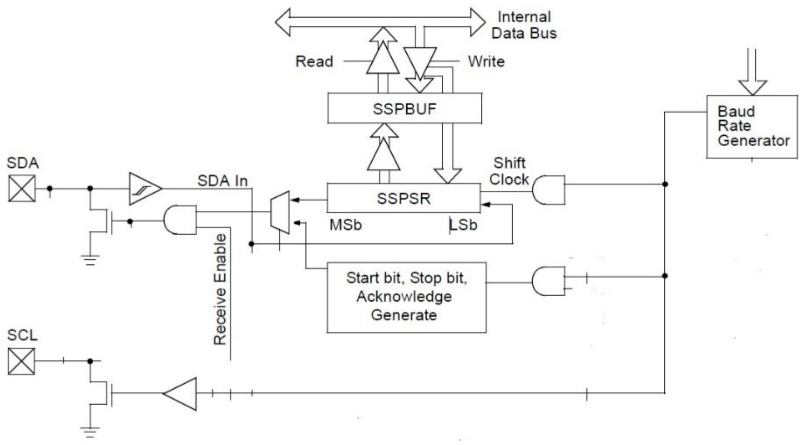
2. Available Source files

Verilog source files: ClockedNegativeOneShot.v, ClockedPositiveOneShot.v, I2C_DataUnit.v, Lab2I2Cphase1summer2021JJS.v, ShiftRegister4bits.v, SquareWaveGenerator.v, I2C_Controller_phase1_temp.v.

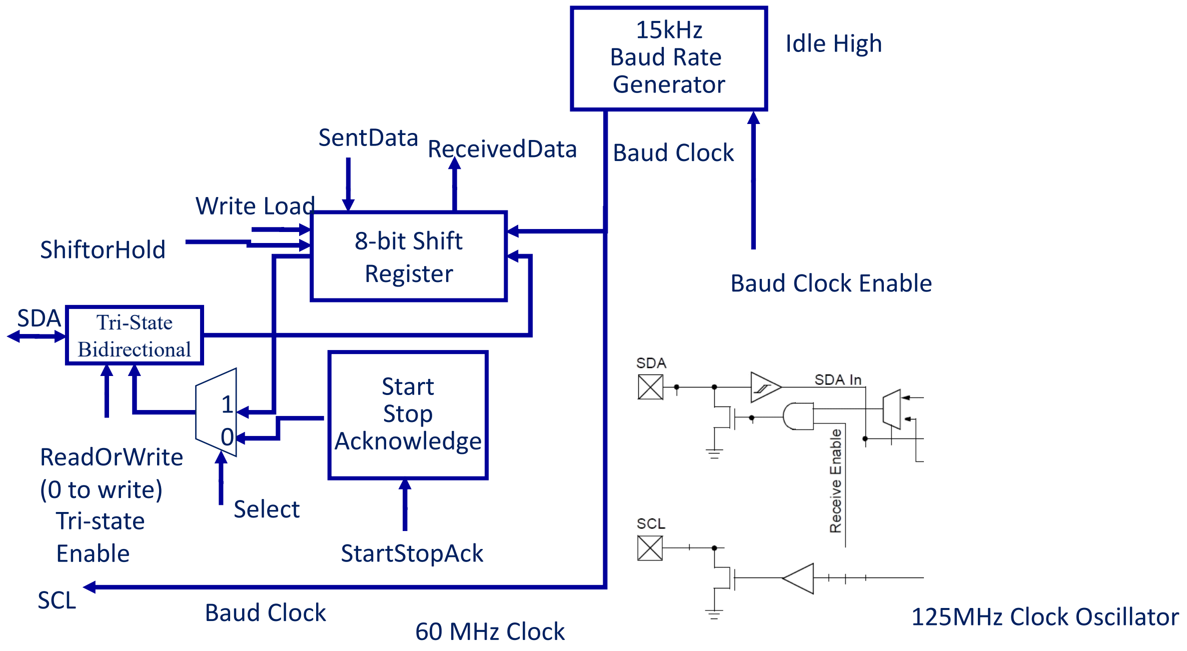
All test benches files for the master data unit, the shift register and the controller are available under lab2summer2021phase1files folder from the instructor. See appendix to this lab handout for some of these files.

3. Objectives

To implement a simplified I2C Master module in Verilog. The Master controller is similar to that of PIC16F887 microcontroller as shown on the right hand side.

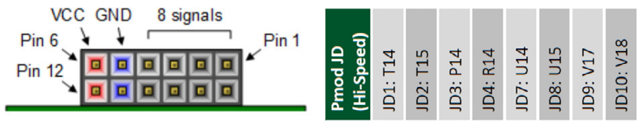


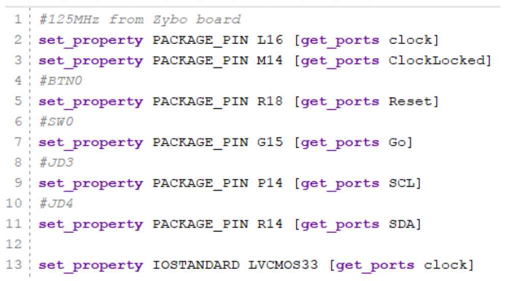
Some modules need to be simulated and their simulation test benches and expected waveforms are provided in the Appendix of this handout. The test benches files are also available from Moodle.



4. I2C Master Module Design and Implementation

The I2C Sender module is composed of data unit and control unit. The data unit is composed of three main components: baud rate clock module, shift register module and SDA signal module. The sender module has two I/O pins SDA and SCL to be connect to an I2C device through the Pmod connectors of Zybo board, for example, JD3 for SCL and JD4 for SDA. You are free to use other pins on Pmod connectors.

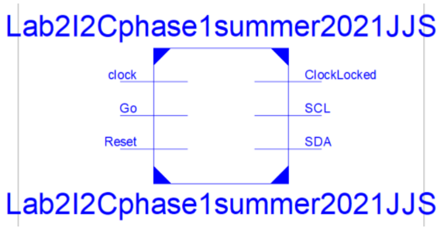


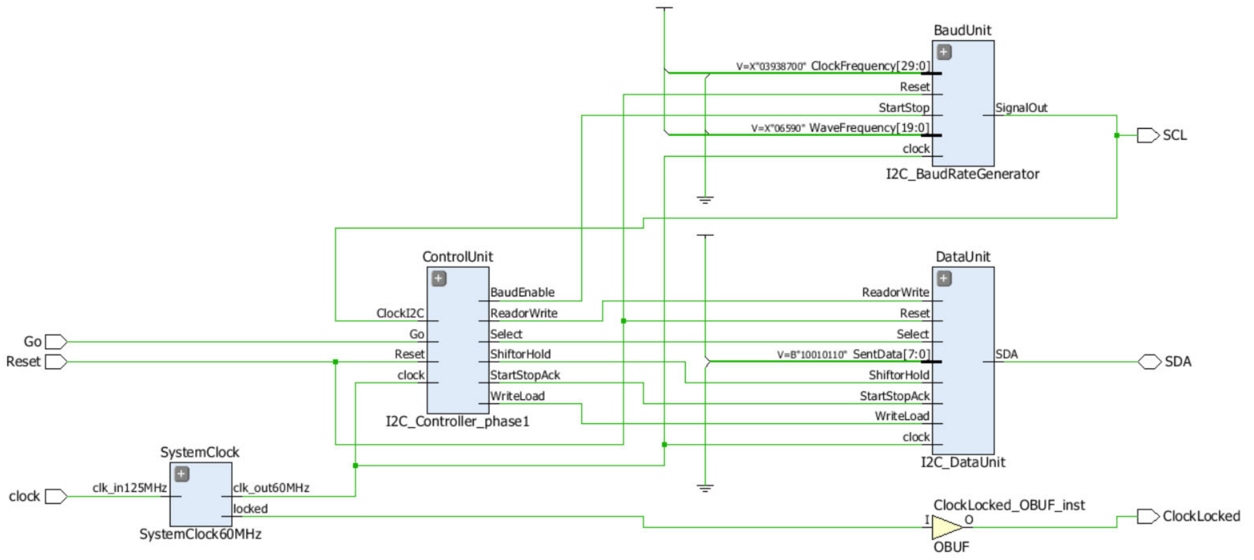

```
#125MHz from Zybo board
set_property PACKAGE_PIN L16 [get_ports clock]
set_property PACKAGE_PIN M14 [get_ports ClockLocked]
#BTN0
set_property PACKAGE_PIN R18 [get_ports Reset]
#SW0
set_property PACKAGE_PIN G15 [get_ports Go]
#JD3
set_property PACKAGE_PIN P14 [get_ports SCL]
#JD4
set_property PACKAGE_PIN R14 [get_ports SDA]

set_property IOSTANDARD LVCMOS33 [get_ports clock]
```

.

4.1 Top-level module Lab2I2Cphase1summer2021JJS.v Is Provided

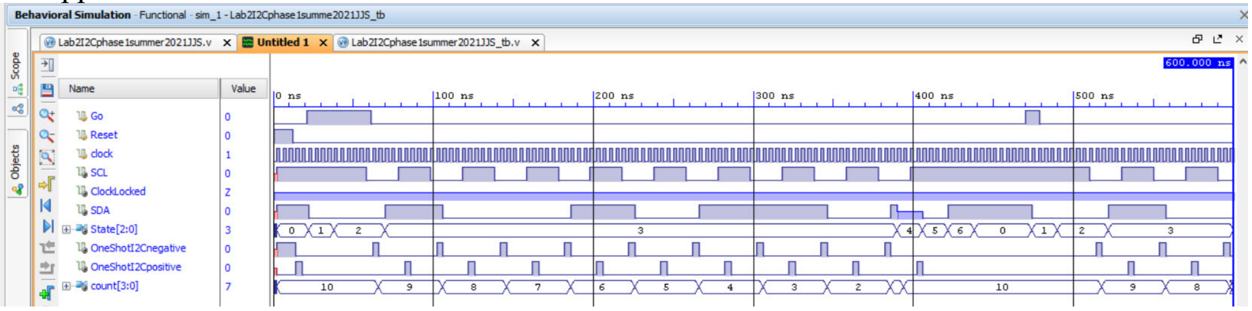
Lab2I2Cphase1summer2021JJS.v, where JJS is your name initials, is provided by the instructor. Change the name initials to your name. This module is created to test the master module. It has one button Go to start and stop sending the first byte continuously through I2C. Go is "1" to start and "0" to stop transmitting. The SDA and SCL outputs can then be observed on an oscilloscope.





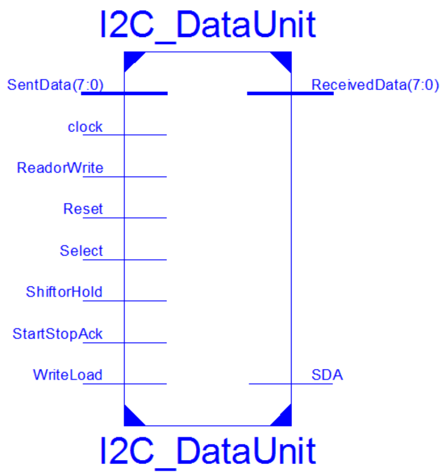
4.2 I2C Data Unit Module Is Provided

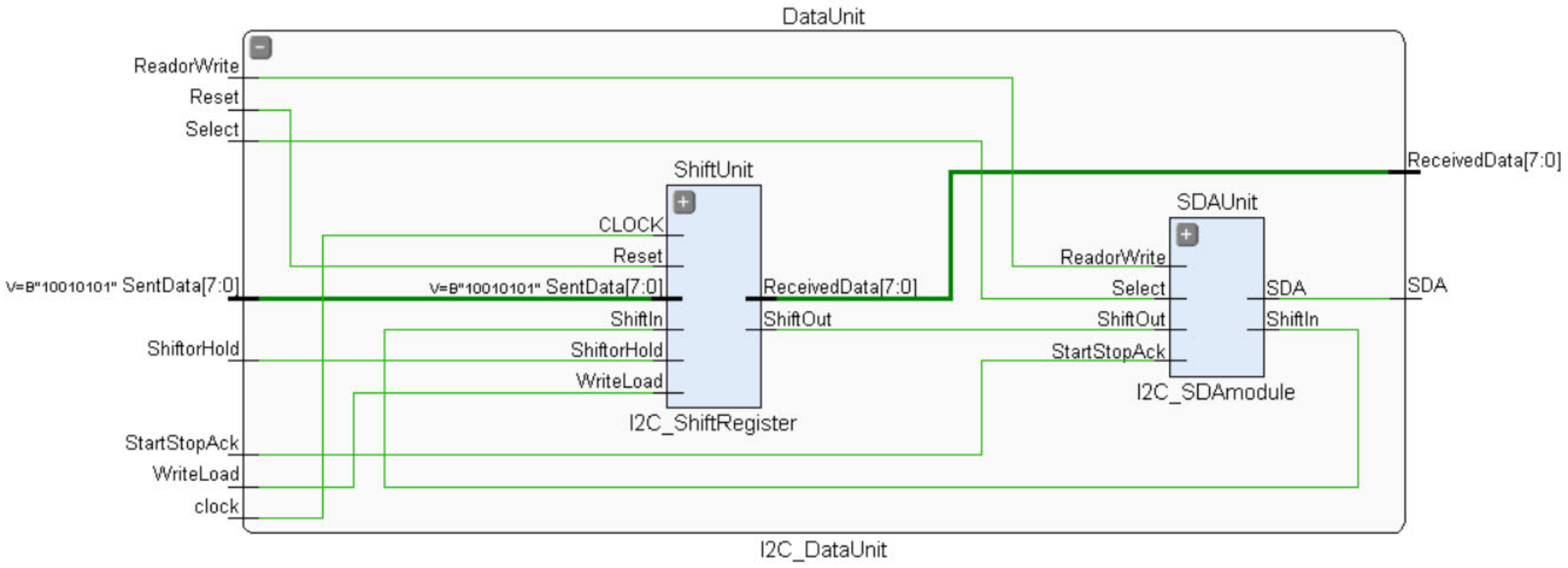
The data unit, I2C_DataUnit is also provided by the instructor. It is composed of three main components: baud rate clock module, shift register module and SDA signal module. See the appendix for more details.



module I2C_DataUnit (WriteLoad, ReadorWrite, ShiftorHold, Select, SentData,

```
ReceivedData, SDA, StartStopAck, Reset, clock);
parameter       LENGTH = 8;

input WriteLoad, ReadorWrite, ShiftorHold, Select, StartStopAck;
inout SDA;
input Reset, clock;
input [LENGTH-1:0]    SentData;
output [LENGTH-1:0]  ReceivedData;

//module I2C_ShiftRegister(WriteLoad, SentData, ReceivedData,ShiftIn,ShiftOut,ShiftorHold,Reset,CLOCK) ;
wire ShiftDataIn, ShiftDataOut;
I2C_ShiftRegister ShiftUnit(WriteLoad, SentData, ReceivedData,ShiftDataIn, ShiftDataOut,ShiftorHold,
Reset,clock) ;

//module I2C_SDAmodule(SDA, ReadorWrite, Select, StartStopAck, ShiftIn, ShiftOut) ;

I2C_SDAmodule SDAUnit(SDA, ReadorWrite, Select, StartStopAck, ShiftDataIn, ShiftDataOut) ;
endmodule
```


I2C_DataUnit
SentData(7:0)
clock
ReadorWrite
Reset
Select
ShiftorHold
StartStopAck
WriteLoad
ReceivedData(7:0)
SDA
I2C_DataUnit

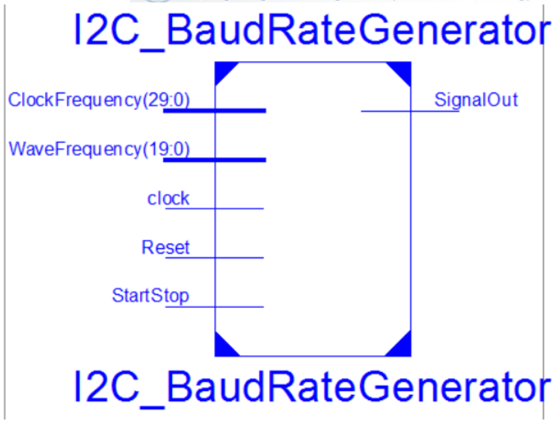

DataUnit
ReadorWrite
Reset
Select
V=B"10010101" SentData[7:0]
ShiftorHold
StartStopAck
WriteLoad
clock
ShiftUnit
CLOCK
Reset
V=B"10010101" SentData[7:0]
ShiftIn
ShiftorHold
WriteLoad
ReceivedData[7:0]
ShiftOut
I2C_ShiftRegister
SDAUnit
ReadorWrite
Select
ShiftOut
StartStopAck
SDA
ShiftIn
I2C_SDAmodule
ReceivedData[7:0]
SDA
I2C_DataUnit

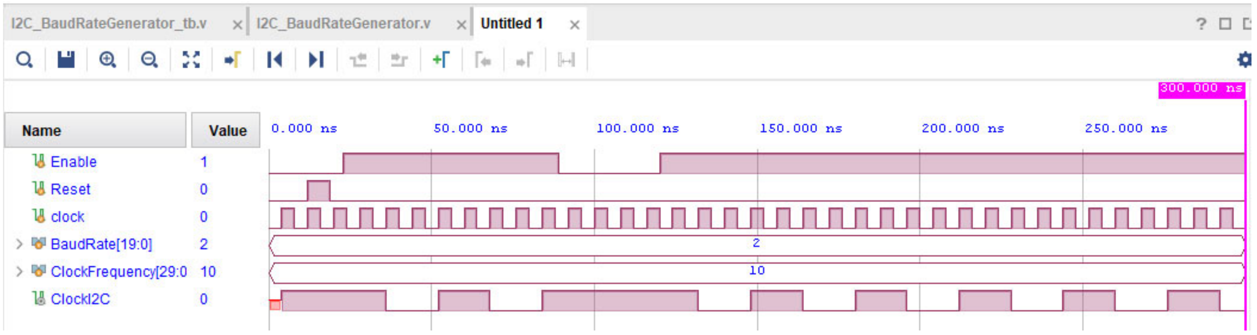
4.3 I2C Baud Rate Clock Generator Module is Provided

The I2C clock generation module can be designed from the uart baud rate generation module. However, the I2C clock is a square wave and is controlled by an enable pin.

It takes Baud Rate and System Clock Values and generates a square wave that matches the baud rate.

Enable pin is "0" to clear Baud Rate Counter and Set ClockI2C. Enable becomes active to clear ClockI2C to start the I2C clock on negative edge.



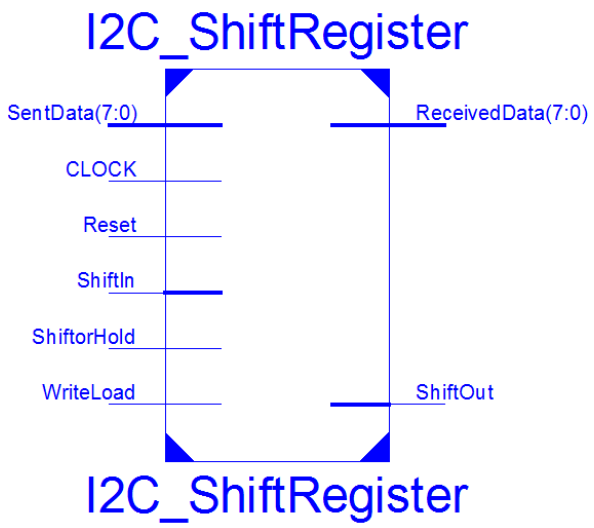


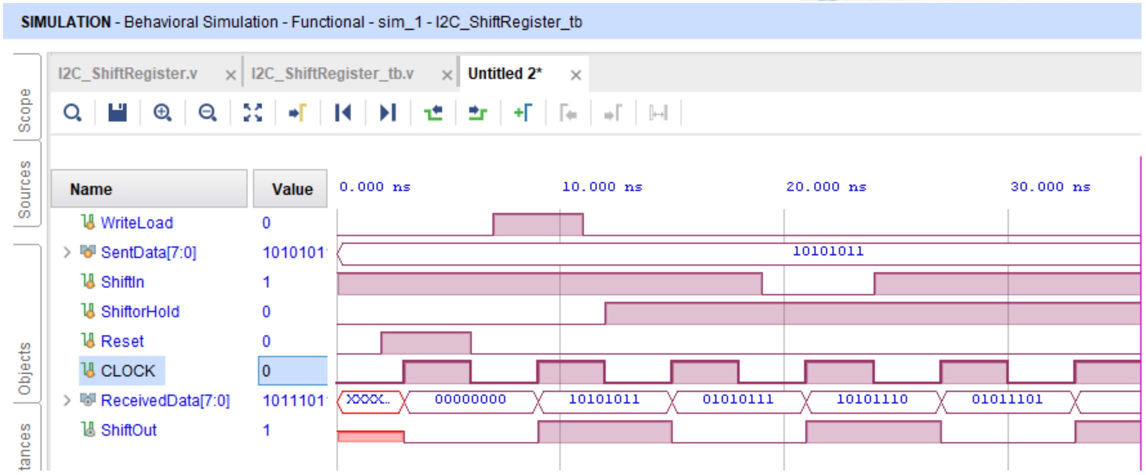
Name	Value
Enable	1
Reset	0
clock	0
BaudRate[19:0]	2
ClockFrequency[29:0]	10
ClockI2C	0

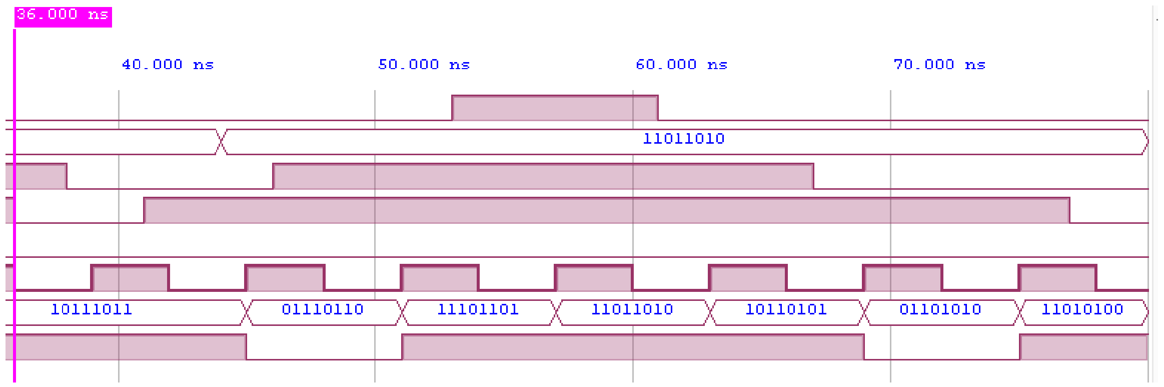
4.4 I2C Shift Register Module is Provided

Here are the specifications for this shift register.

- 8-bit SentData provides data to be sent out through I2C bus.
- 8-bit ReceivedData stores data received from reading I2C bus
- ShiftIn is a serial input of the shift register
- ShiftOut is a serial output of the shift register
- ShiftorHold controls shift or hold action
- WriteLoad=1 to parallel load SentData into shift register
- Shifting occurs when I2C clock is low as required
  by I2C standard, i.e., negative edge triggered.



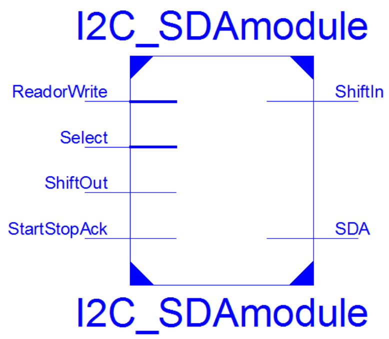


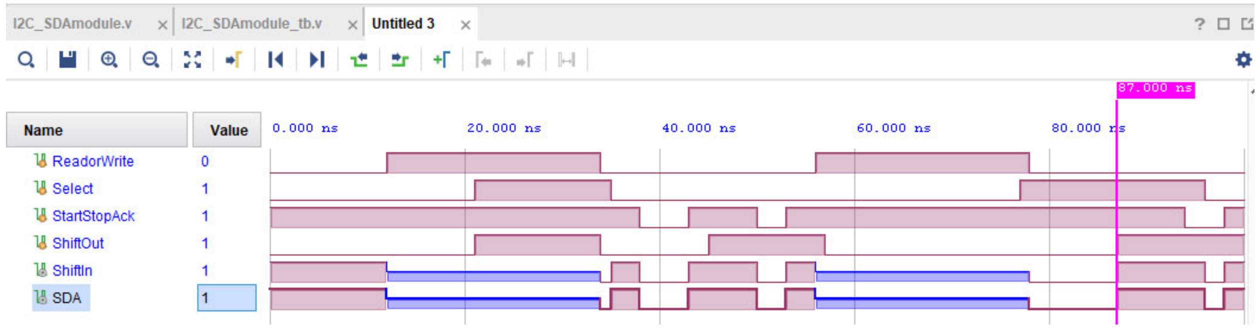


4.5 I2C SDA Signal Module to be developed

The serial data (SDA) module implements data operations such as Start, Stop, Acknowledge, Tri-state (Z) output, bidirectional input/output, etc.

- *SDA is tri-state (Z) and bidirectional*
- *Select chooses StopStartAck or ShiftOut to be sent to SDA*
- *ReadorWrite selects Read (==1) or Write (==0) operation*
- *SDA is tri-state or Hi-Z on Read operation*
- *ShiftIn is an input from SDA*
- *ShiftOut is an output to SDA*

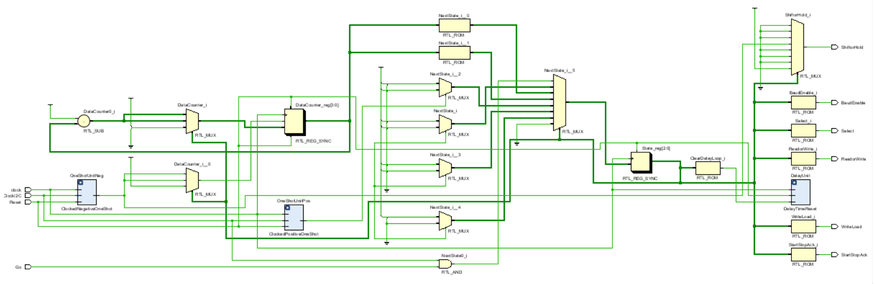


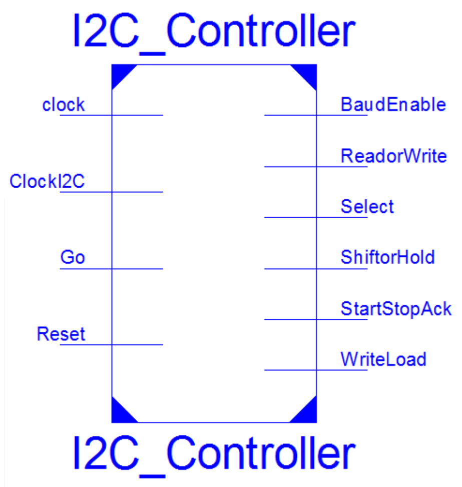


4.6 I2C Master Controller Module I2C_Controller.v to be Developed

The master controller, I2C_Controller.v, generates control signals to implement I2C Start, Stop, Acknowledge, Write and Read operations. Here is the header of the controller.

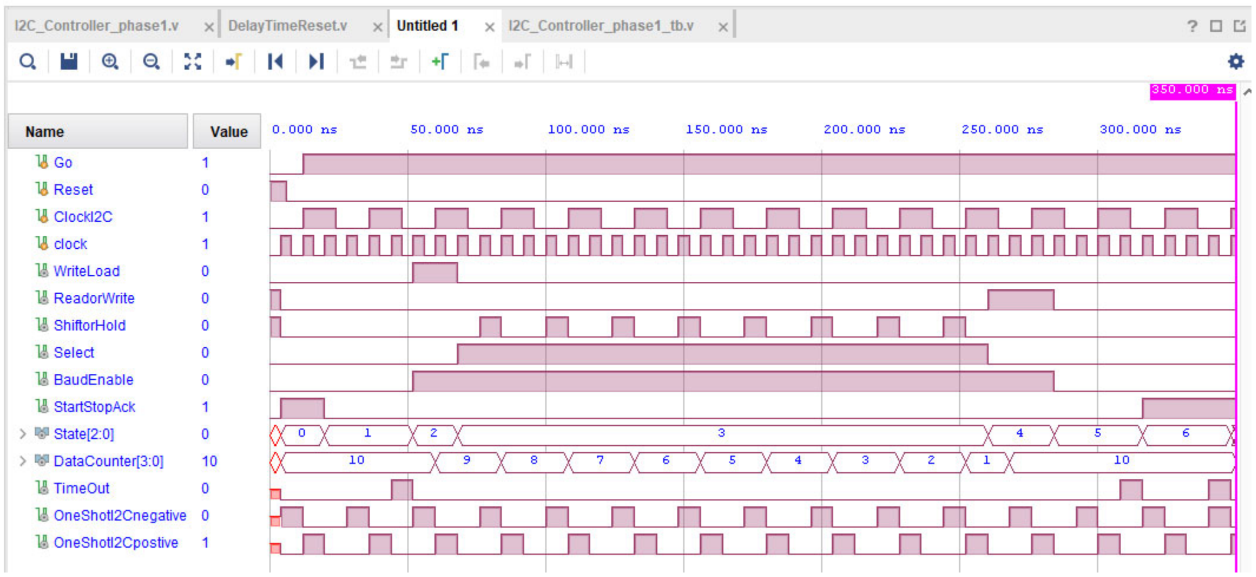
```
module I2C_Controller (Go, WriteLoad, ReadorWrite, ShiftorHold, Select, BaudEnable, StartStopAck, Reset,
ClockI2C, clock);

output reg WriteLoad, ReadorWrite, ShiftorHold, Select, BaudEnable, StartStopAck;
input Go, Reset, clock, ClockI2C;
parameter InitialState=3'd0, StartState=3'd1, LoadState=3'd2, WriteState=3'd3, AcknowledgeState=3'd4;
parameter TransitState=3'd5, StopState=3'd6;

reg [3:0] DataCounter;
reg [2:0] State, NextState;

endmodule
```





4.7 Simulation and an ASM chart for your controller

Here is an example simulation of the controller.



Name	Value
Go	1
Reset	0
ClockI2C	1
clock	1
WriteLoad	0
ReadorWrite	0
ShiftorHold	0
Select	0
BaudEnable	0
StartStopAck	1
State[2:0]	0
DataCounter[3:0]	10
TimeOut	0
OneShotI2Cnegative	0
OneShotI2Cpostive	1

Here is a recommended ASM chart of the controller. You are welcome to use it or revise it as you like.

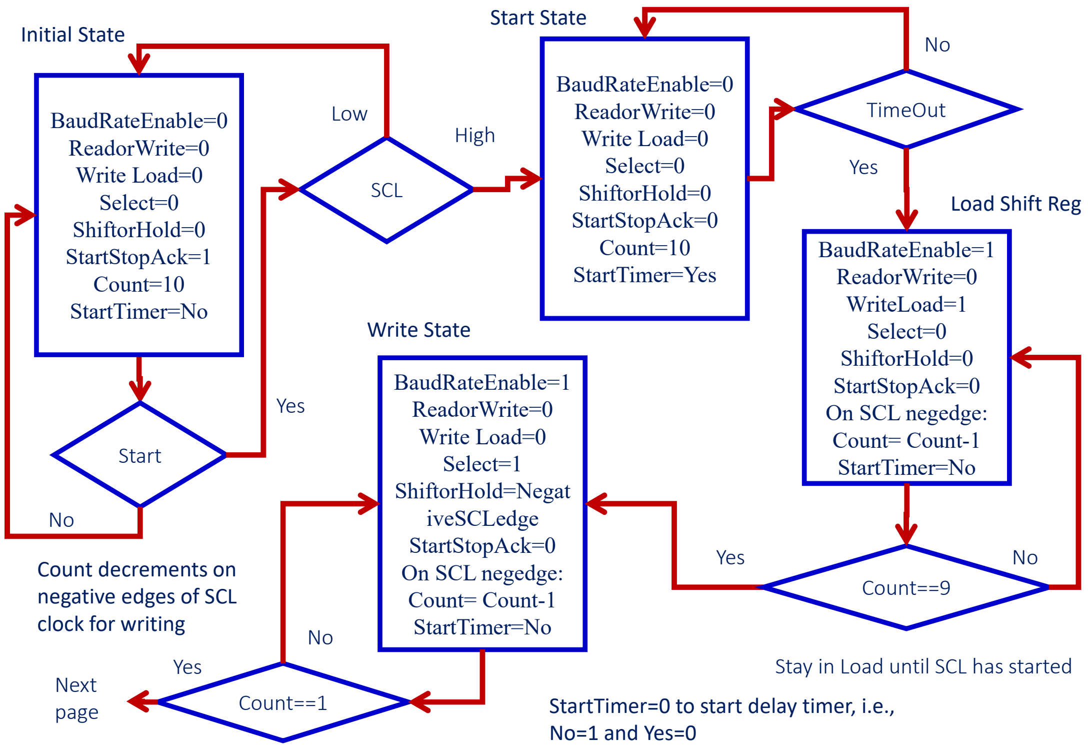
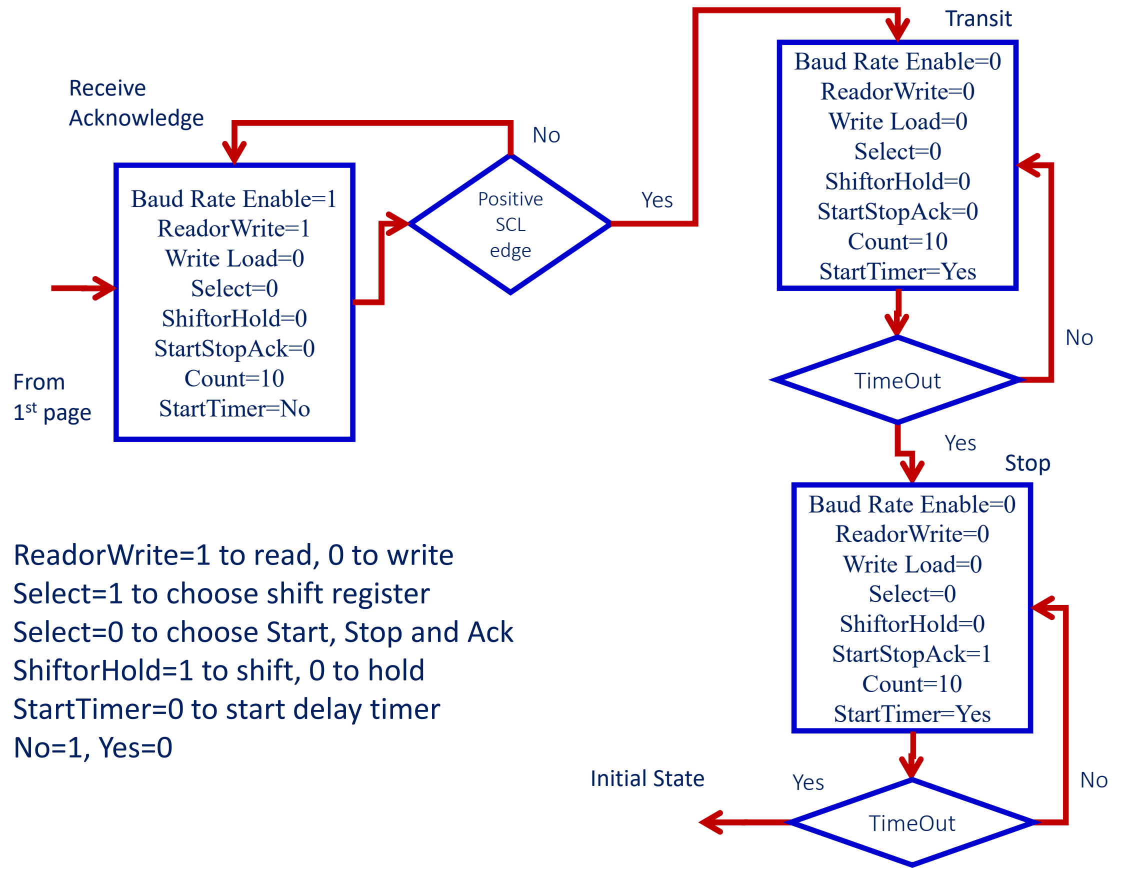


Figure 1. An recommended ASM chart of the controller (page1 of 2)

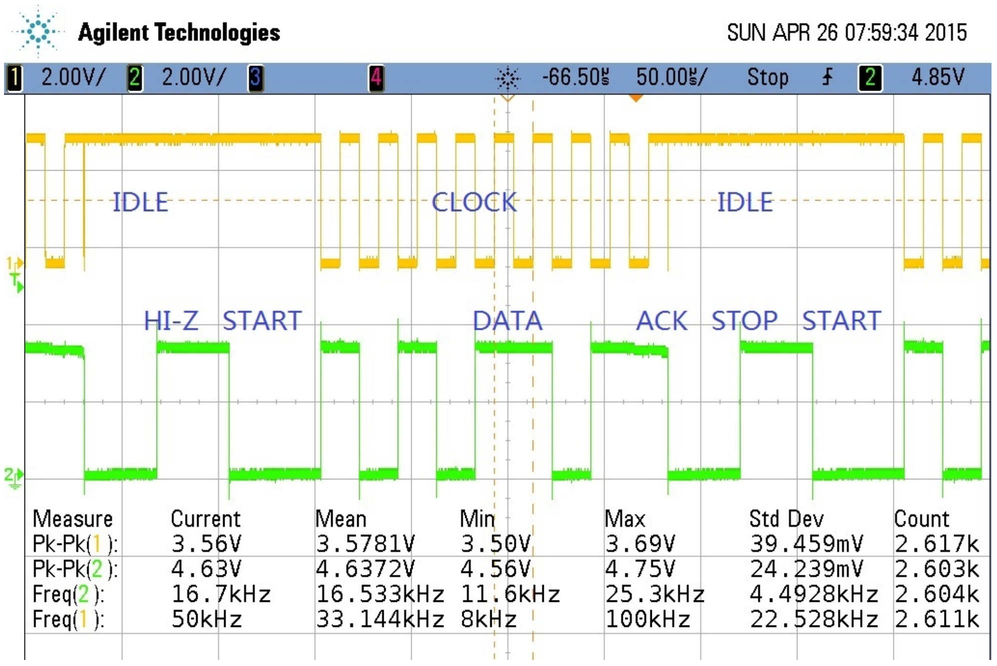


ReadorWrite=1 to read, 0 to write
Select=1 to choose shift register
Select=0 to choose Start, Stop and Ack
ShiftorHold=1 to shift, 0 to hold
StartTimer=0 to start delay timer
No=1, Yes=0

Figure 1. An recommended ASM chart of the controller (page 2 of 2).

4.8 Obtain waveforms of your I2C master module on an oscilloscope

lab2phase1I2C_controller_jjs.v sends one byte continuously through I2C pin with Start, Data, Stop, etc. An oscilloscope can then observe the I2C transfer waveforms on SCL clock and SDA data lines when SCL and SDA signals are wired to two pins, JB3 (V20) is SCL and JB4 (W20) is SDA on the Zybo board. You are free to choose two pins. Here are two sample waveforms for SDA and SCL that are similar to what you will need to generate and submit. Notice your waveforms should show that the I2C clock frequency is 100kHz and the address is 7'b1001010 and READ.



```
Measure     Current   Mean       Min      Max      Std Dev    Count
Pk-Pk(1):   3.56V     3.5781V    3.50V    3.69V    39.459mV   2.617k
Pk-Pk(2):   4.63V     4.6372V    4.56V    4.75V    24.239mV   2.603k
Freq(2):    16.7kHz   16.533kHz  11.6kHz  25.3kHz  4.4928kHz  2.604k
Freq(1):    50kHz     33.144kHz  8kHz     100kHz   22.528kHz  2.611k
```

Figure 2. 50kHz I2C clock and data waveforms to send address 7'b1010110 and READ

Figure 3. One frame of I2C transmission of first byte, Address=7[b1010110 and READ.

5. Pmod Connectors on the Zybo Board

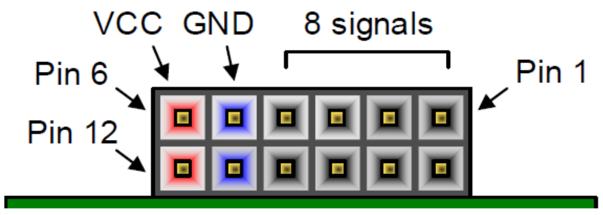
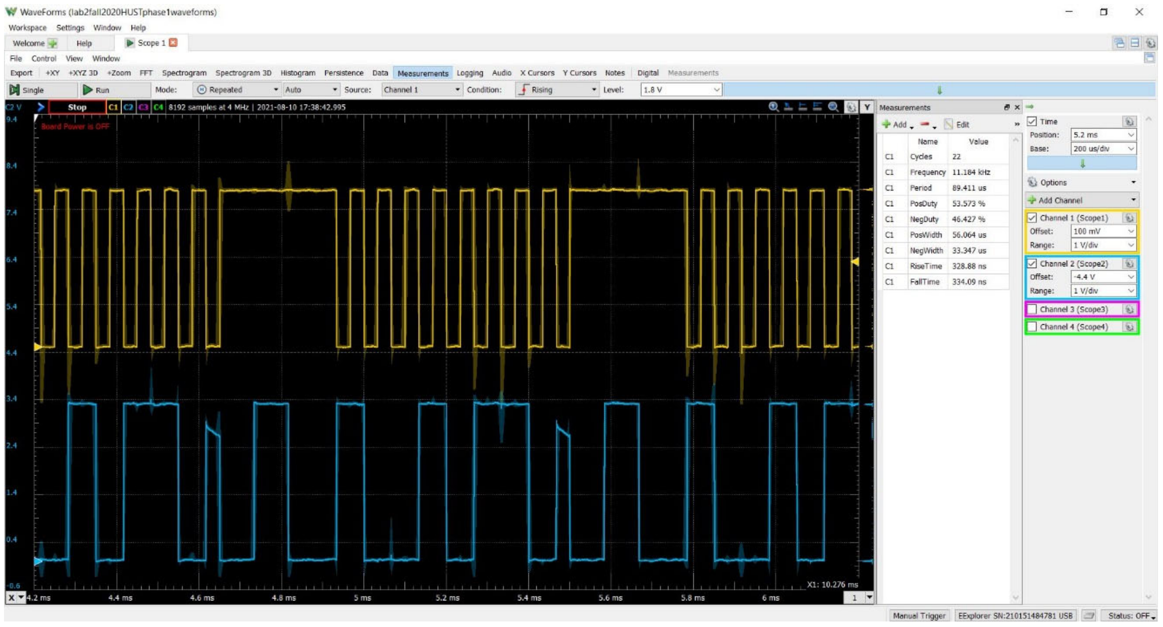


Figure 16. Pmod diagram.

Pmod JA (XADC)	Pmod JB (Hi-Speed)	Pmod JC (Hi-Speed)	Pmod JD (Hi-Speed)	Pmod JE (Std.)	Pmod JF (MIO)
JA1: N15	JB1: T20	JC1: V15	JD1: T14	JE1: V12	JF1: MIO-13
JA2: L14	JB2: U20	JC2: W15	JD2: T15	JE2: W16	JF2: MIO-10
JA3: K16	JB3: V20	JC3: T11	JD3: P14	JE3: J15	JF3: MIO-11
JA4: K14	JB4: W20	JC4: T10	JD4: R14	JE4: H15	JF4: MIO-12
JA7: N16	JB7: Y18	JC7: W14	JD7: U14	JE7: V13	JF7: MIO-0
JA8: L15	JB8: Y19	JC8: Y14	JD8: U15	JE8: U17	JF8: MIO-9
JA9: J16	JB9: W18	JC9: T12	JD9: V17	JE9: T17	JF9: MIO-14
JA10: J14	JB10: W19	JC10: U12	JD10: V18	JE10: Y17	JF10: MIO-15


WaveForms (lab2fall2020HUSTphase1waveforms)
Workspace Settings Window Help
Welcome Help Scope 1
File Control View Window
Export +XY +XYZ 3D +Zoom FFT Spectrogram Spectrogram 3D Histogram Persistence Data Measurements Logging Audio X Cursors Y Cursors Notes Digital Measurements
Single Run
Stop C1 C2 C3 C4
Board Power is OFF
Add Edit
Options
Add Channel
Manual Trigger

Mode: Repeated | Auto | Source: Channel 1 | Condition: Rising | Level: 1.8 V

8192 samples at 4 MHz | 2021-08-10 17:38:42.995

Measurements

	Name	Value
C1	Cycles	22
C1	Frequency	11.184 kHz
C1	Period	89.411 us
C1	PosDuty	53.573 %
C1	NegDuty	46.427 %
C1	PosWidth	56.064 us
C1	NegWidth	33.347 us
C1	RiseTime	328.88 ns
C1	FallTime	334.09 ns

Time: Position: 5.2 ms, Base: 200 us/div

Channel 1 (Scope1): Offset: 100 mV, Range: 1 V/div

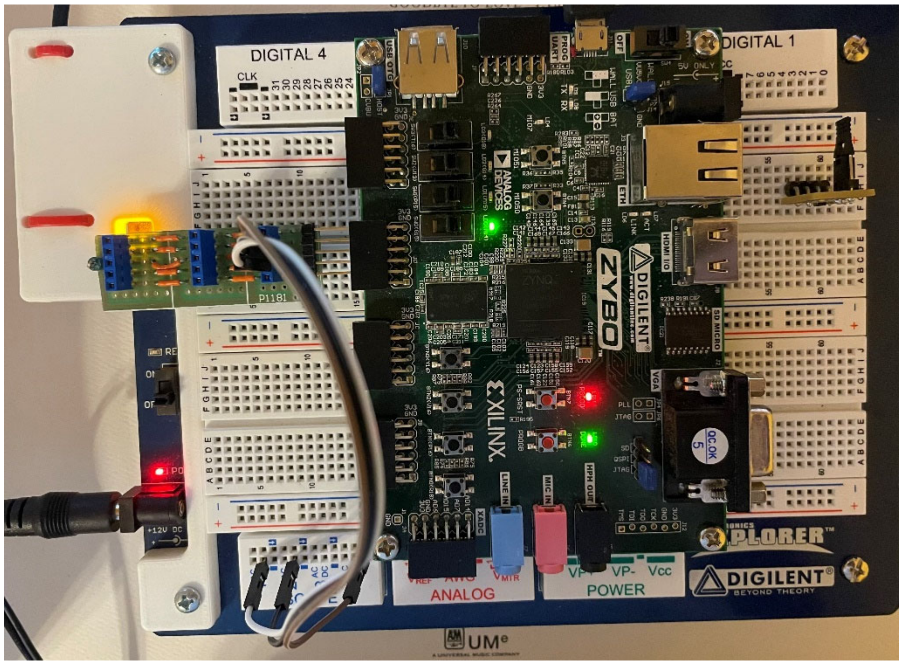
Channel 2 (Scope2): Offset: -4.4 V, Range: 1 V/div

Channel 3 (Scope3)

Channel 4 (Scope4)

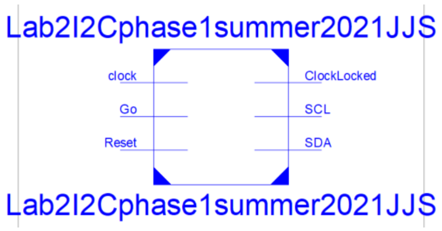
X1: 10.276 ms

EExplorer SN:210151484781 USB | Status: OFF



6. Appendix Lab 2 Phase I: I2C Data Unit and Controller Summer 2021

6.1 Lab6phase1 top-level module, Lab2I2Cphase1summer2021JJS.v

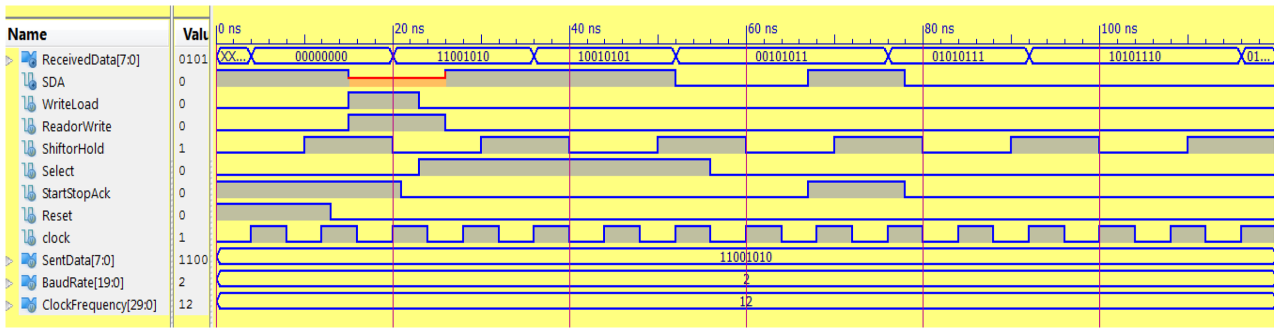


```
//File Name: Lab2I2Cphase1summer2021JJS.v
//Author: Jianjian Song
//Date: August 10, 2021
//HUST Summer 2021
//Phase 1 of Lab #2 I2C driver and TMP101 temperature sensor
//send first byte to I2C bus with slave address
//Output:
//Go is "1" to start communication

module Lab2I2Cphase1summer2021JJS (input Reset, clock, Go,
output SCL, ClockLocked, inout SDA);
//send this byte as address
parameter FirstByte=8'b10010110;
//I2C speed frequency and system clock frequency
//parameter BaudRate=20'd26000, ClockFrequency=30'd60000000;
wire clk60MHz;

//These are simulation parameters.
//Comment the following two lines before making bit stream file
parameter BaudRate=2, ClockFrequency=20;
assign clk60MHz=clock;

//disable SystemClock60MHz to simulate this circuit
//SystemClock60MHz SystemClock(.clk_in125MHz(clock),.clk_out60MHz(clk60MHz),.locked(ClockLocked));

wire WriteLoad, ReadorWrite, ShiftorHold, Select, BaudEnable, StartStopAck;

//module I2C_BaudRateGenerator (input Reset, clock, StartStop,
//input [19:0] WaveFrequency,  //up to 1,000,000
//input [29:0] ClockFrequency,
//output reg  SignalOut);

I2C_BaudRateGenerator  BaudUnit(Reset, clk60MHz, BaudEnable, BaudRate, ClockFrequency,  SCL);

//module I2C_Controller_phase1(
//input Reset, clock, Go, ClockI2C,
//output reg WriteLoad, ReadorWrite, ShiftorHold, Select, BaudEnable, StartStopAck);


I2C_Controller_phase1  ControlUnit(Reset, clk60MHz, Go, SCL,
WriteLoad, ReadorWrite, ShiftorHold, Select, BaudEnable, StartStopAck);

//module I2C_DataUnit #(parameter        LENGTH = 8) (
//input Reset, clock, WriteLoad, ReadorWrite, ShiftorHold, Select, StartStopAck,
//input           [LENGTH-1:0]    SentData,
//output [LENGTH-1:0]             ReceivedData, inout SDA);

wire [7:0] ReceivedData;
I2C_DataUnit DataUnit(Reset, clk60MHz, WriteLoad, ReadorWrite, ShiftorHold, Select, StartStopAck, FirstByte,
ReceivedData, SDA);

endmodule
```

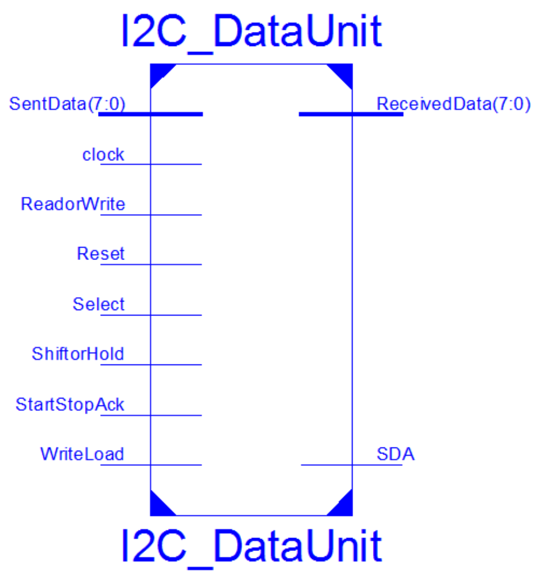
6.2 Data Unit Top Level, I2C_DataUnit.v

```
module I2C_DataUnit (WriteLoad, ReadorWrite, ShiftorHold, Select, SentData,
ReceivedData, SDA, StartStopAck, Reset, clock);
parameter       LENGTH = 8;

input WriteLoad, ReadorWrite, ShiftorHold, Select, StartStopAck;
inout SDA;
input Reset, clock;
input [LENGTH-1:0]    SentData;
output [LENGTH-1:0] ReceivedData;

//module I2C_ShiftRegister(WriteLoad, SentData, ReceivedData,ShiftIn,ShiftOut,ShiftorHold,Reset,CLOCK) ;
wire ShiftDataIn, ShiftDataOut;
I2C_ShiftRegister ShiftUnit(WriteLoad, SentData, ReceivedData,ShiftDataIn, ShiftDataOut,ShiftorHold, Reset,clock) ;

//module I2C_SDAmodule(SDA, ReadorWrite, Select, StartStopAck, ShiftIn, ShiftOut) ;

I2C_SDAmodule SDAUnit(SDA, ReadorWrite, Select, StartStopAck, ShiftDataIn, ShiftDataOut) ;

endmodule
```

6.3 I2C Data Unit Simulation

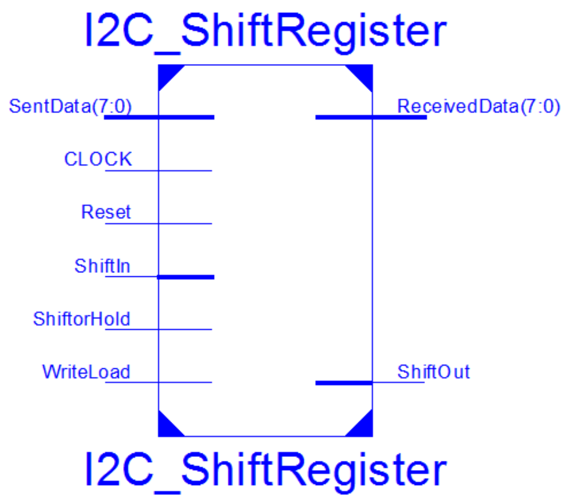


```
`timescale 1ns / 1ps

module I2C_DataUnit_tb;
reg WriteLoad, ReadorWrite, ShiftorHold, Select, StartStopAck, Reset, clock;
reg [7:0] SentData;
        reg [19:0] BaudRate;
        reg [29:0] ClockFrequency;
        wire [7:0] ReceivedData;
        wire SDA;
//module I2C_DataUnit (WriteLoad, ReadorWrite, ShiftorHold, Select,
SentData,
//ReceivedData, SDA, StartStopAck, Reset, clock);
 I2C_DataUnit uut (WriteLoad, ReadorWrite, ShiftorHold, Select, SentData,
ReceivedData, SDA, StartStopAck, Reset, clock);

initial begin WriteLoad = 0;   ReadorWrite = 0; ShiftorHold = 0;  Select = 0;
SentData = 0; StartStopAck = 0; Reset = 0;
clock = 0; BaudRate = 0; ClockFrequency = 0; end
        always #4 clock=~clock;
                always #10 ShiftorHold=~ShiftorHold;
        initial fork
        #0 Reset = 1;  #13 Reset = 0;    #0 BaudRate = 2;   #0 ClockFrequency =
12;
        #0 WriteLoad = 0;   #15 WriteLoad = 1;  #23 WriteLoad = 0;
        #56 WriteLoad = 1;   #56 WriteLoad = 0;
        #0 SentData = 8'b11001010;
        #0 Select = 0;      #23 Select = 1;   #56 Select = 0; #90 Select = 0;
        #0 StartStopAck = 1;   #21 StartStopAck = 0;  #67 StartStopAck = 1; #78 StartStopAck = 0;
        #0 ReadorWrite = 0;  #15 ReadorWrite = 1;   #26 ReadorWrite = 0; #90 ReadorWrite = 0;
        #120 $stop;
        join
endmodule
```

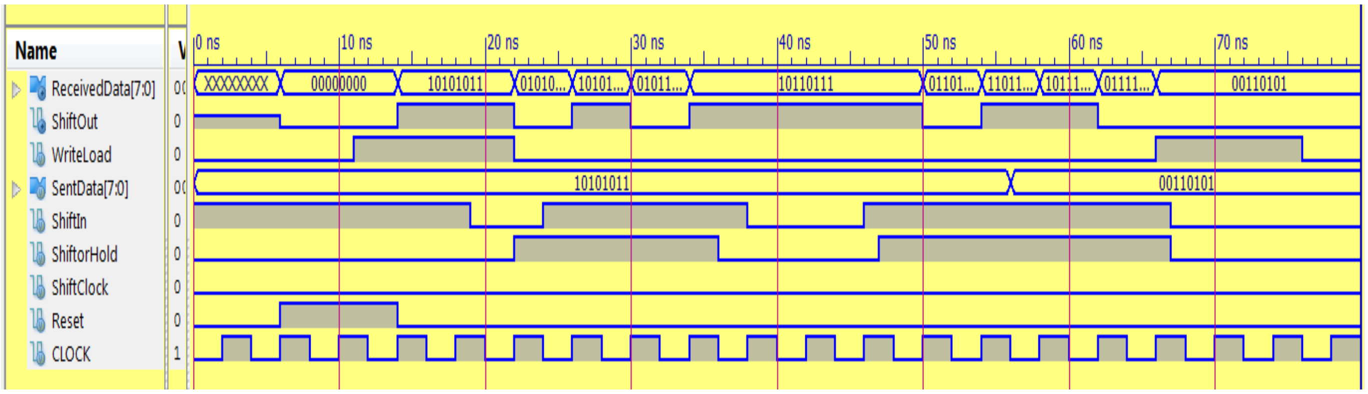


6.4 I2C Shift Register

```
`timescale 1ns / 1ps

module ShiftRegisterI2C_tb;

        reg WriteLoad;
        reg [7:0] SentData;
        reg ShiftIn, ShiftorHold, ShiftClock, Reset, CLOCK;

        wire [7:0] ReceivedData;
        wire ShiftOut;
//      wire ShiftClockOneShot=uut.ShiftClockOneShot;

        I2C_ShiftRegister uut (WriteLoad, SentData,
ReceivedData,ShiftIn,ShiftOut,ShiftorHold,Reset,CLOCK);

        initial begin  WriteLoad = 0;  SentData = 0;  ShiftIn = 0;
ShiftorHold = 0;   ShiftClock = 0;  Reset = 0;  CLOCK = 0; end
        always #2 CLOCK=~CLOCK;
//      always #4 ShiftClock=~ShiftClock;
```

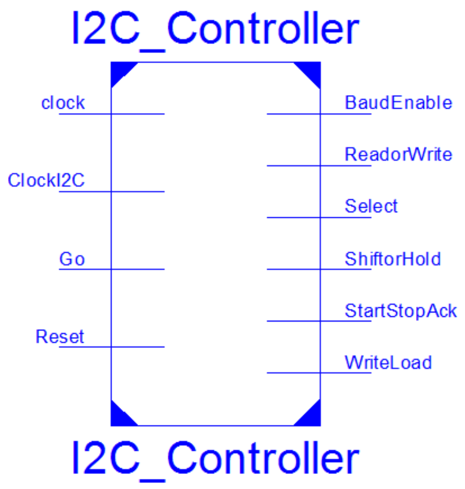


```
        initial fork
        #0 Reset = 0;  #6 Reset = 1;  #14 Reset = 0;
        #0 WriteLoad = 0;  #11 WriteLoad = 1; #22 WriteLoad = 0; #66 WriteLoad = 1;  #76 WriteLoad = 0;
        #0 SentData = 8'b10101011;          #56 SentData = 8'b00110101;
        #0 ShiftIn = 1;   #13 ShiftIn = 1;   #19 ShiftIn = 0;   #24 ShiftIn = 1;   #38 ShiftIn = 0;    #46 ShiftIn = 1;
        #67 ShiftIn = 0; #98 ShiftIn = 1;
        #0 ShiftorHold = 0;   #22 ShiftorHold = 1;    #36 ShiftorHold = 0;
        #47 ShiftorHold = 1; #67 ShiftorHold = 0;
        #80 $stop;
        join
endmodule
```



6.5 Controller Design

```
module I2C_Controller (Go, WriteLoad, ReadorWrite, ShiftorHold, Select,
BaudEnable, StartStopAck, Reset, ClockI2C, clock);

output reg WriteLoad, ReadorWrite, ShiftorHold, Select, BaudEnable,
StartStopAck;
input Go, Reset, clock, ClockI2C;
parameter InitialState=3'd0, StartState=3'd1, LoadState=3'd2, WriteState=3'd3,
AcknowledgeState=3'd4;
parameter TransitState=3'd5, StopState=3'd6;

reg [3:0] DataCounter;
reg [2:0] State, NextState;
```



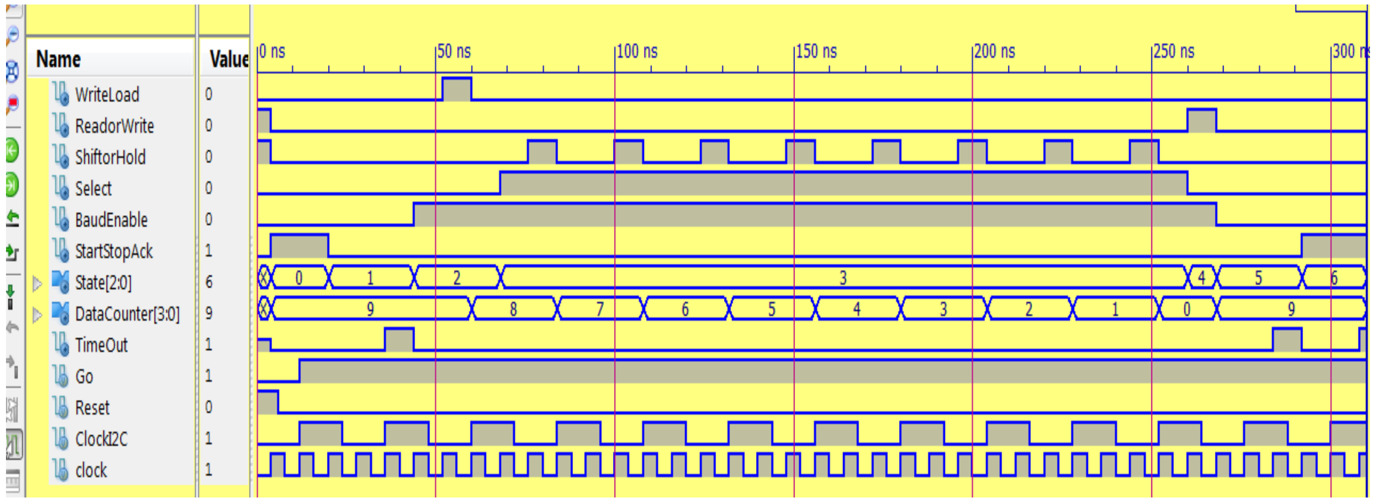
The controller ASM chart is given in Figure 1 and Figure 2 on pages 8 and 9. The controller waits for a level signal Go to become logic “1” to start the transmission. It will then wait for the baud rate clock SCL to become logic “1” to start the frame.

The controller uses a count variable, Count, to record the number of data bits. Count is implemented with an always block that decreases Count on the positive edge of SCL as shown below.

```
always@(posedge clock)
        if(Reset==1) begin DataCounter<=4'd9; end
        else
                case (State)
                LoadState: if(OneShotI2C==0) DataCounter<=DataCounter-1'b1; else DataCounter<=DataCounter;
                WriteState: if(OneShotI2C==0) DataCounter<=DataCounter-1'b1; else DataCounter<=DataCounter;
                default: DataCounter<=4'd9;   endcase
```

```
module I2C_Controller (Go, WriteLoad, ReadorWrite, ShiftorHold, Select, BaudEnable, StartStopAck, Reset, ClockI2C,
clock);

output reg WriteLoad, ReadorWrite, ShiftorHold, Select, BaudEnable, StartStopAck;
input Go, Reset, clock, ClockI2C;

endmodule
```

Here is an example simulation of the controller signals.



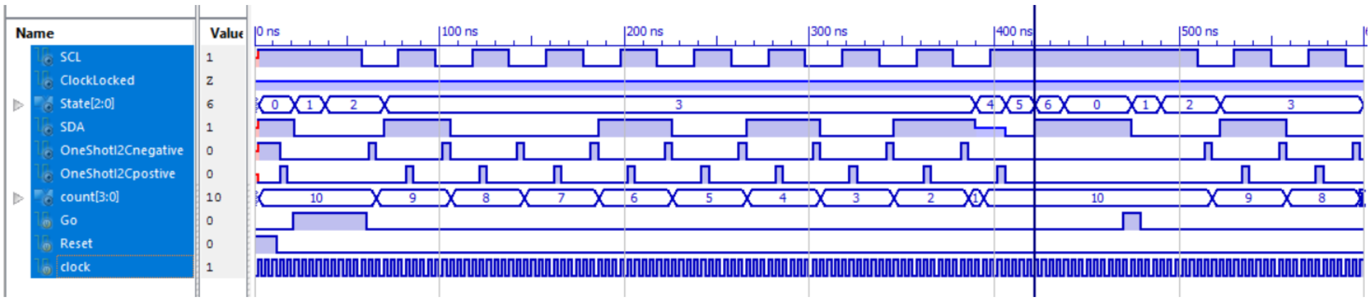
6.6 Test Bench for the controller

```
`timescale 1ns / 1ps

module I2C_Controller_tb;

        reg Go,Reset,ClockI2C,clock;

        wire WriteLoad, ReadorWrite, ShiftorHold, Select, BaudEnable, StartStopAck;
        wire [2:0] State=uut.State;
        wire [3:0] DataCounter=uut.DataCounter;
        wire TimeOut=uut.TimeOut;

        I2C_Controller uut (Go, WriteLoad, ReadorWrite, ShiftorHold, Select,
        BaudEnable, StartStopAck, Reset, ClockI2C, clock);

        initial begin  Go = 0;  Reset = 0;  ClockI2C = 0;  clock = 0; end
        always #4 clock=~clock;
        always #12 ClockI2C=~ClockI2C;
        initial fork
        #0 Go = 0;  #12 Go = 1;
        #0 Reset = 1;  #6 Reset = 0;
        #310 $stop;
        join

endmodule
```

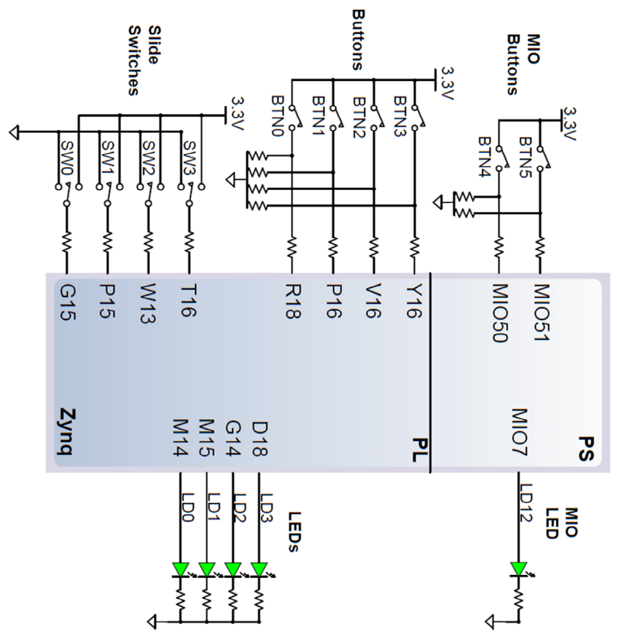
6.7 Overall Simulation

Set clock to be 20 and baud rate to be 2 for simulation.

```
`timescale 1ns / 1ps
//HUST summer 2021
//These are simulation parameters.
//in lab2I2Cphase1summer2021JJS.v
//parameter BaudRate=2, ClockFrequency=20;
//assign clk60MHz=clock;
//SystemClock60MHz SystemClock(clock,clk60MHz,ClockLocked);
//Change the delay loop to 3 counts in DelayUnit of Control Unit

module Lab2I2Cphase1summe2021JJS_tb;
        reg Go, Reset, clock;
        wire SCL, ClockLocked;
        // Bidirs
        wire SDA;
        wire [2:0] State=DUT.ControlUnit.State;
        wire OneShotI2Cnegative=DUT.ControlUnit.OneShotI2Cnegative;
        wire OneShotI2Cpositive=DUT.ControlUnit.OneShotI2Cpositive;
        wire [3:0] count=DUT.ControlUnit.DataCounter;

//module Lab7I2Cphase1summer2021JJS (
//input Reset, clock, Go,
//output SCL, ClockLocked, inout SDA);

        Lab2I2Cphase1summer2021JJS DUT(Reset, clock, Go, SCL, ClockLocked, SDA);

        initial begin Go = 0;  Reset = 0;  clock = 0; end
        always #2 clock=~clock;
        initial fork
        #0 Reset=1;  #12 Reset=0;
        #0 Go=0;  #21 Go=1;     #61 Go=0;   #470 Go=1;  #479 Go=0;
        #600 $stop;
        join

endmodule
```



7. Zybo I/O Pin Assignment



Pmod JA (XADC)	Pmod JB (Hi-Speed)	Pmod JC (Hi-Speed)	Pmod JD (Hi-Speed)	Pmod JE (Std.)	Pmod JF (MIO)
JA1: N15	JB1: T20	JC1: V15	JD1: T14	JE1: V12	JF1: MIO-13
JA2: L14	JB2: U20	JC2: W15	JD2: T15	JE2: W16	JF2: MIO-10
JA3: K16	JB3: V20	JC3: T11	JD3: P14	JE3: J15	JF3: MIO-11
JA4: K14	JB4: W20	JC4: T10	JD4: R14	JE4: H15	JF4: MIO-12
JA7: N16	JB7: Y18	JC7: W14	JD7: U14	JE7: V13	JF7: MIO-0
JA8: L15	JB8: Y19	JC8: Y14	JD8: U15	JE8: U17	JF8: MIO-9
JA9: J16	JB9: W18	JC9: T12	JD9: V17	JE9: T17	JF9: MIO-14
JA10: J14	JB10: W19	JC10: U12	JD10: V18	JE10: Y17	JF10: MIO-15

Table 9. Pmod pinout.

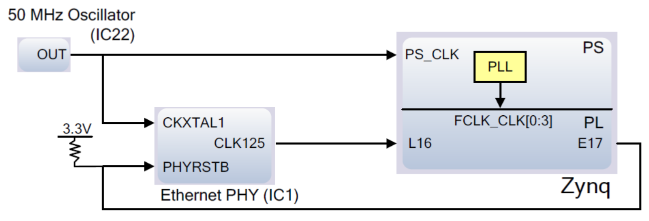


Figure 13. ZYBO clocking.