Maseeh College of Engineering and Computer Science

PORTLAND STATE UNIVERSITY

Final Project

Design and Verification of a I2C based Memory Subsystem

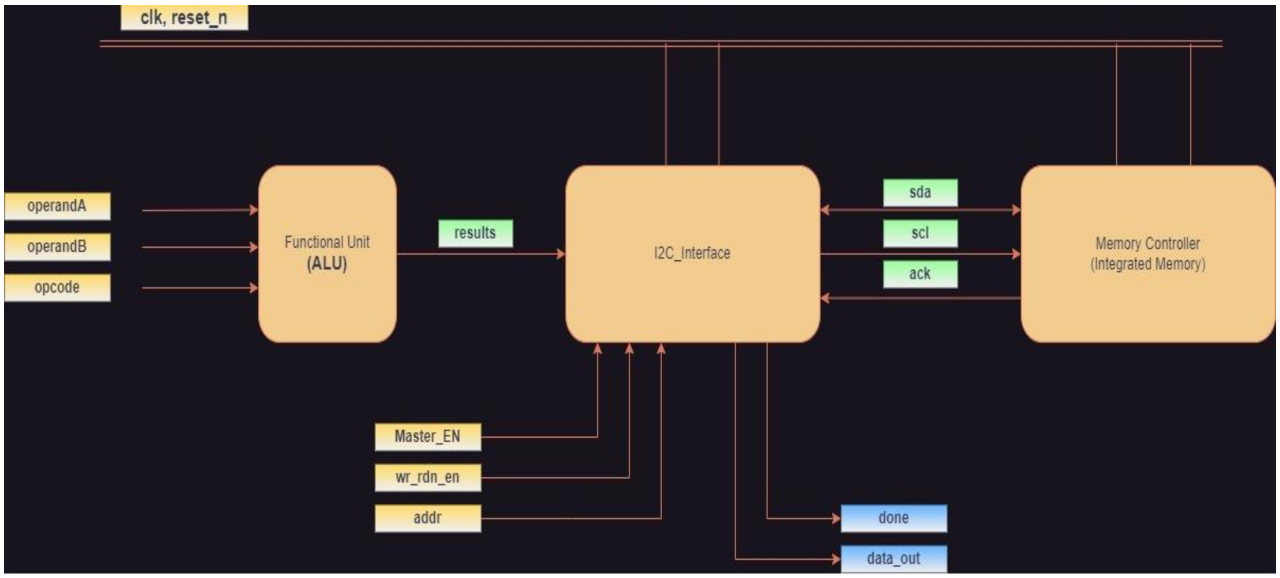
Submitted to: Cory Davidson

ECE-571, Group-7

Submitted by:
Fardeen Wasey: - wasey@pdx.edu
Mohamed Gnedi: - gnedi@pdx.edu
Adeel Ahmed: - adeel@pdx.edu
Daniyal Ahmad: - daniyal@pdx.edu
Priyanka Pavana Bekal: - pbekal@pdx.edu
Mahika Godbole: - mgodbole@pdx.edu

Objective:

Design an I2C based memory subsystem that's composed of four blocks: A Functional Unit, an I2C Interface, a Memory Interface, and a Static Memory.



Goals

1. Gaining an understanding of the fundamental behavior of the I2C protocol, including addressing, data transfer, and protocol constraints.
2. Developing the project using System Verilog to gain experience in designing complex digital systems, implementing state machines, managing signals, and utilizing System Verilog constructs for design structuring.
3. Designing and integrating a memory subsystem, which includes addressing and accessing memory, implementing memory controllers, and ensuring proper data flow between different blocks.
4. Developing testbench development and functional verification skills by creating testbenches to validate the behavior of sub-systems and the entire design.
5. Improving debugging and troubleshooting skills by identifying and resolving issues in the design and testbenches throughout the development phases, such as dealing with unexpected behavior or simulation issues.

Solution

The design and verification of the I2C-based memory subsystem involves developing four key blocks: Functional Unit, I2C Interface, Memory Interface, and Static Memory. The Functional Unit can be implemented using complex components like an ALU or shifter, while the I2C Interface utilizes an FSM-based design to handle I2C protocol behavior. The Memory Interface connects the I2C Interface to the Static Memory, implementing read and write operations with address decoding. Each block requires a corresponding testbench to validate its behavior and functionality. The blocks are interfaced by connecting the output of the Functional Unit to the input of the I2C Interface, the output of the I2C Interface to the input of the Memory Interface, and the appropriate signals from the Memory Interface to the Static Memory.

Block 1: Functional Unit

---

1. Shifter (Option 1):

- *Inputs:*
  - **Data Input**: The input data to be shifted.
  - **Control Inputs**: Signals specifying the type and direction of shifting (e.g., leftshift, rightshift).
- *Output:*
  - **Results**: The shifted output data.

---

2. Arithmetic Logic Unit (ALU) (Option 2):

- *Inputs:*
  - **Data Input A**: The first input data for arithmetic or logical operations.
  - **Data Input B**: The second input data for arithmetic or logical operations.
  - **Control Inputs**: Signals specifying the operation to be performed (e.g., addition, subtraction, AND, OR).
- *Output:*
  - **Results**: The computed output based on the selected operation.

Block 2: I2C Interface

Methodology: For the I2C Interface, a Finite State Machine (FSM) approach is commonly used. Design the FSM to handle different states of the I2C protocol, including start and stop conditions, data transmission, and acknowledgment. Utilize combinational and sequential logic to implement the required functionality and data transformation.

Ports

- *Inputs:*
  - **Results** (from the Functional Unit block): The computed result from the Functional Unit block that needs to be stored in memory.
  - **addr**: The address specifying the location in memory where the Result should be stored.
  - **Reset**: A signal indicating a reset condition, typically used to initialize the I2C interface.
  - **Clk**: The clock signal that synchronizes the data transfer on the I2C bus.
- *Outputs:*
  - **SDA** (Serial Data Line): The output line for transmitting and receiving data over the I2C bus.
  - **SCL** (Serial Clock Line): The output line that provides the clock signal for synchronizing the data transfer on the I2C bus.

Block 3: Memory Interface

Methodology: The Memory Controller can be implemented using a Finite State Machine (FSM) approach, similar to the I2C Interface block. Design the FSM to handle the conversion of data received through the SDA line back into valid data and address formats. Utilize combinational and sequential logic to implement the necessary data conversion and memory access operations.

Ports

- *Inputs:*
  - **Reset**: A signal indicating a reset condition, typically used to initialize the I2C interface.
  - **Clk**: The clock signal that synchronizes the data transfer on the I2C bus.
  - **SDA** (Serial Data Line): The output line for transmitting and receiving data over the I2C bus.
  - **SCL** (Serial Clock Line): The output line that provides the clock signal for synchronizing the data transfer on the I2C bus.
- *Output:*
  - **data out**: The output line that carries the converted data from the Memory Controller. It represents the valid data and address ready for storage in the memory or further processing.

Block 4: Static Memory

Methodology: Model the Static Memory using structural modeling techniques. Instantiate memory cells and design the necessary control logic for read and write operations. Consider address decoding and data buffering requirements.

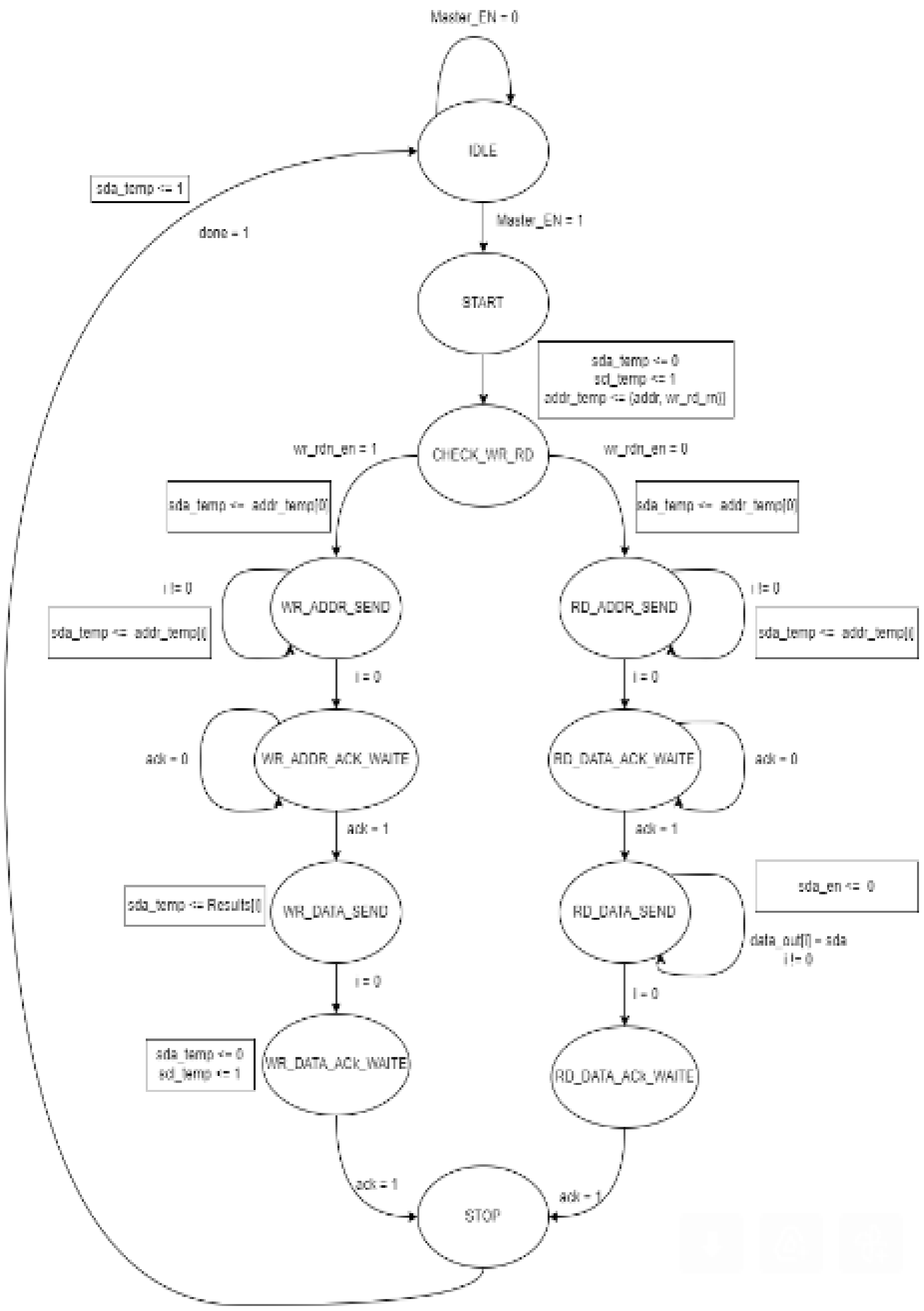
Ports

- *Inputs:*
  - **data in**: The input line that carries the converted data from the Memory Controller. It represents the valid data and address ready for storage in the memory or further processing.

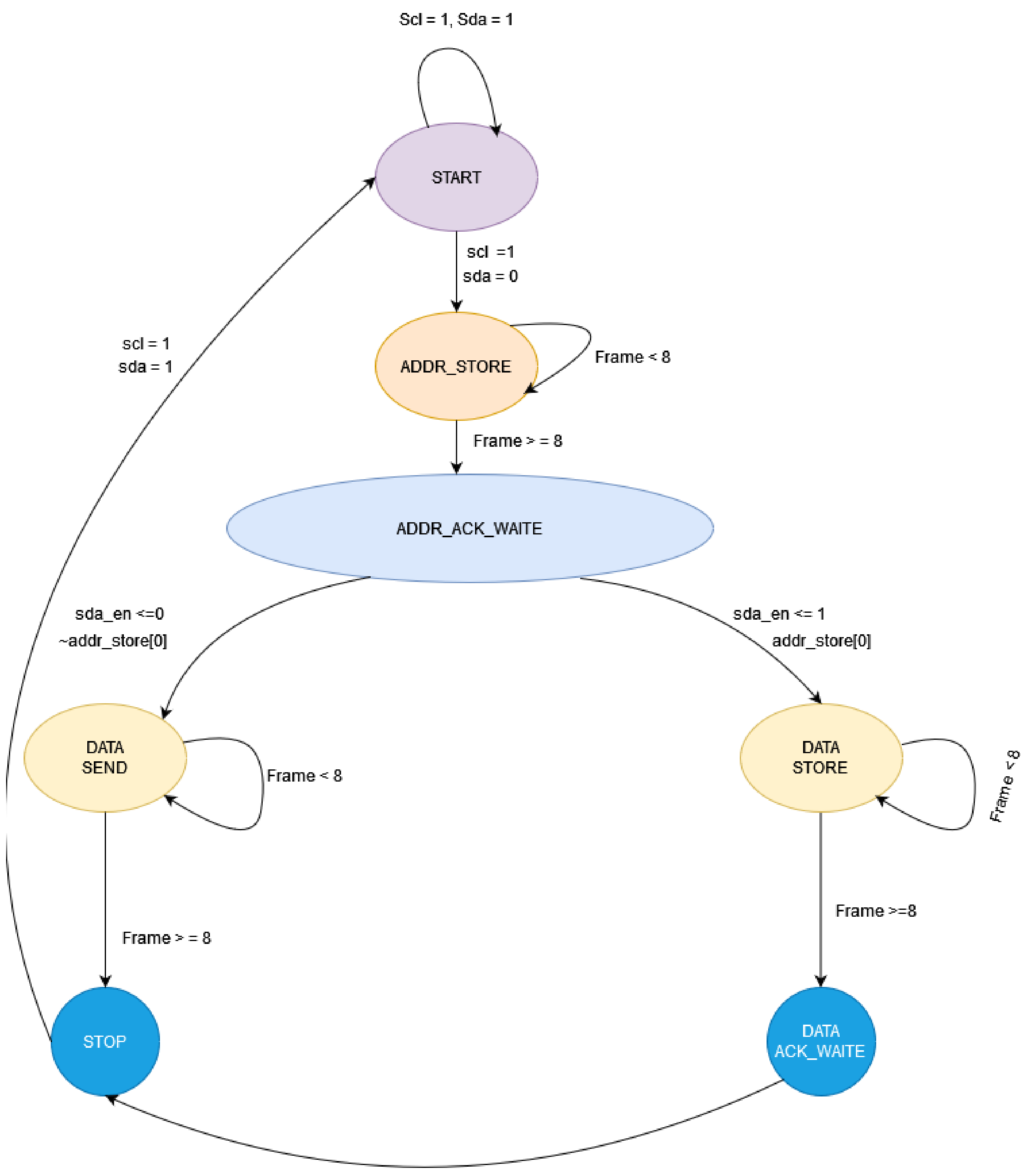
Integrating Sub systems:

- ***Functional Unit with I2C Interface***: Connect the output of the Functional Unit (Result) to the input of the I2C Interface block. Ensure compatibility between the data format and size from the Functional Unit and the input requirements of the I2C Interface.
- ***I2C Interface with Memory Controller***: Connect the output signals (SDA and SCL) of the I2C Interface to the corresponding inputs of the Memory Controller. Consider timing requirements and synchronization between the I2C bus and the Memory Controller signals.
- ***Memory Controller with Static Memory***: Connect the appropriate signals (e.g., data_out) from the Memory Controller to the corresponding inputs of the Static Memory block. Ensure proper timing and address decoding to enable read and write operations.

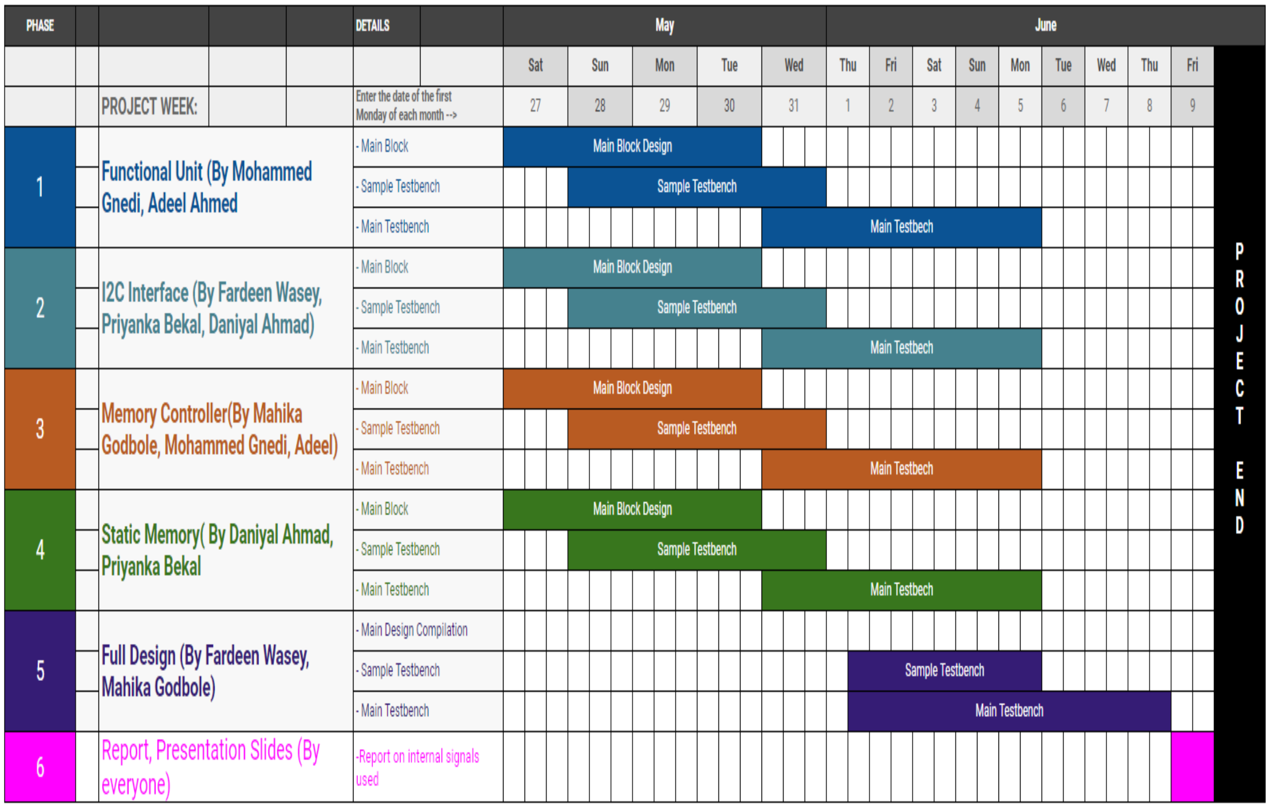
Finite State Machine of I2C:



Finite State Machine of Memory Controller:



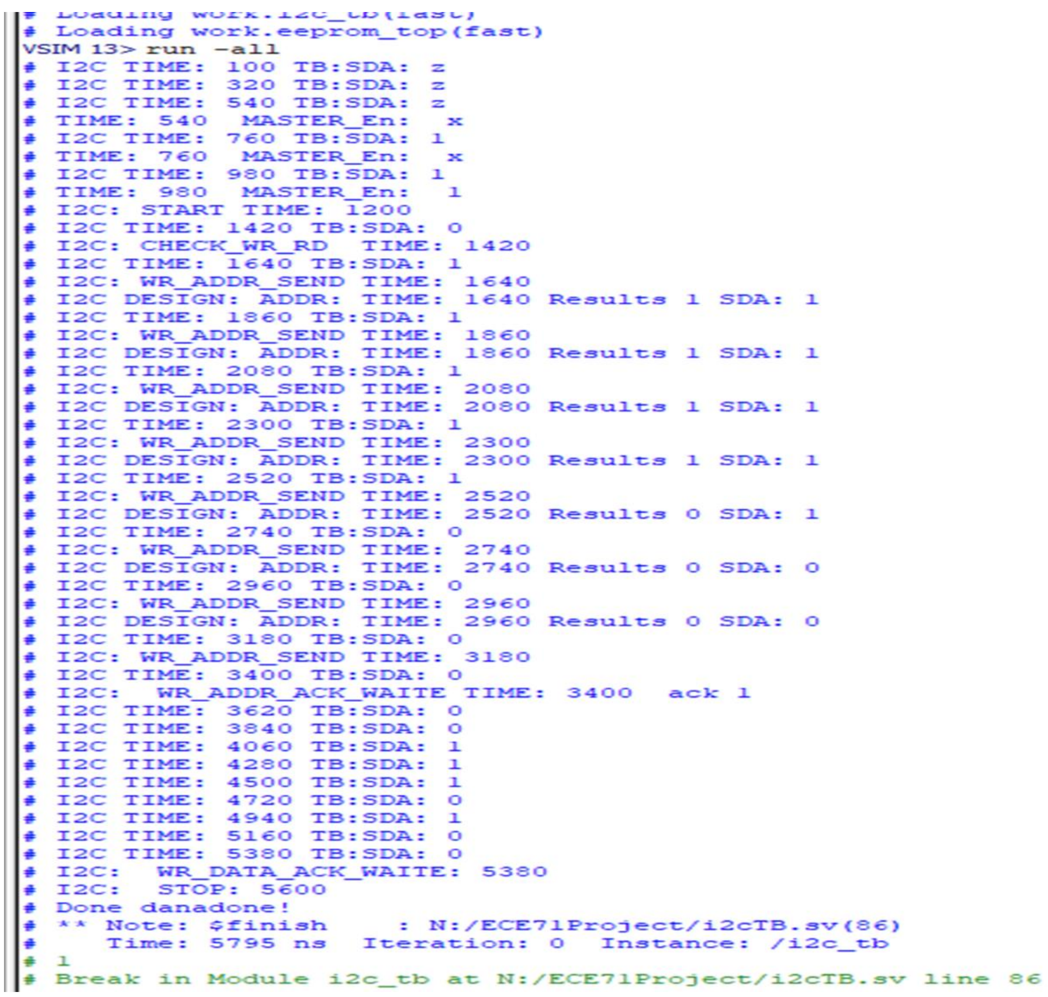
Roles and Responsibilities:



PHASE		DETAILS	May					June									
			Sat	Sun	Mon	Tue	Wed	Thu	Fri	Sat	Sun	Mon	Tue	Wed	Thu	Fri	
	PROJECT WEEK:	Enter the date of the first Monday of each month -->	27	28	29	30	31	1	2	3	4	5	6	7	8	9	
1	Functional Unit (By Mohammed Gnedi, Adeel Ahmed	- Main Block	Main Block Design	■	■	■											PROJECT END
		- Sample Testbench		Sample Testbench	■	■	■										
		- Main Testbench					Main Testbech	■	■	■	■	■					
2	I2C Interface (By Fardeen Wasey, Priyanka Bekal, Daniyal Ahmad)	- Main Block	Main Block Design	■	■	■											
		- Sample Testbench		Sample Testbench	■	■	■										
		- Main Testbench					Main Testbech	■	■	■	■	■					
3	Memory Controller(By Mahika Godbole, Mohammed Gnedi, Adeel)	- Main Block	Main Block Design	■	■	■											
		- Sample Testbench		Sample Testbench	■	■	■										
		- Main Testbench					Main Testbech	■	■	■	■	■					
4	Static Memory(By Daniyal Ahmad, Priyanka Bekal	- Main Block	Main Block Design	■	■	■											
		- Sample Testbench		Sample Testbench	■	■	■										
		- Main Testbench					Main Testbech	■	■	■	■	■					
5	Full Design (By Fardeen Wasey, Mahika Godbole)	- Main Design Compilation															
		- Sample Testbench						Sample Testbench	■	■	■	■					
		- Main Testbench						Main Testbench	■	■	■	■	■	■	■		
6	Report, Presentation Slides (By everyone)	-Report on internal signals used														■	

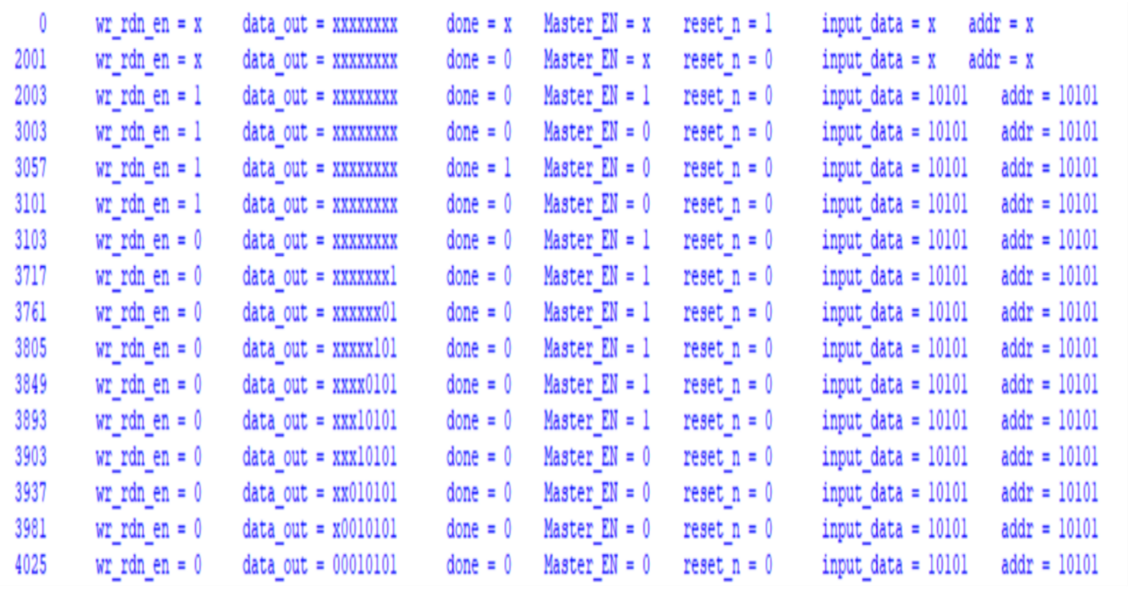
Result:

Results from I2C Module to show serial communication:



```
# Loading work.i2c_tb(fast)
# Loading work.eeprom_top(fast)
VSIM 13> run -all
# I2C TIME: 100 TB:SDA: z
# I2C TIME: 320 TB:SDA: z
# I2C TIME: 540 TB:SDA: z
# TIME: 540  MASTER_En:  x
# I2C TIME: 760 TB:SDA: 1
# TIME: 760  MASTER_En:  x
# I2C TIME: 980 TB:SDA: 1
# TIME: 980  MASTER_En:  1
# I2C: START TIME: 1200
# I2C TIME: 1420 TB:SDA: 0
# I2C: CHECK_WR_RD  TIME: 1420
# I2C TIME: 1640 TB:SDA: 1
# I2C: WR_ADDR_SEND TIME: 1640
# I2C DESIGN: ADDR: TIME: 1640 Results 1 SDA: 1
# I2C TIME: 1860 TB:SDA: 1
# I2C: WR_ADDR_SEND TIME: 1860
# I2C DESIGN: ADDR: TIME: 1860 Results 1 SDA: 1
# I2C TIME: 2080 TB:SDA: 1
# I2C: WR_ADDR_SEND TIME: 2080
# I2C DESIGN: ADDR: TIME: 2080 Results 1 SDA: 1
# I2C TIME: 2300 TB:SDA: 1
# I2C: WR_ADDR_SEND TIME: 2300
# I2C DESIGN: ADDR: TIME: 2300 Results 1 SDA: 1
# I2C TIME: 2520 TB:SDA: 1
# I2C: WR_ADDR_SEND TIME: 2520
# I2C DESIGN: ADDR: TIME: 2520 Results 0 SDA: 1
# I2C TIME: 2740 TB:SDA: 0
# I2C: WR_ADDR_SEND TIME: 2740
# I2C DESIGN: ADDR: TIME: 2740 Results 0 SDA: 0
# I2C TIME: 2960 TB:SDA: 0
# I2C: WR_ADDR_SEND TIME: 2960
# I2C DESIGN: ADDR: TIME: 2960 Results 0 SDA: 0
# I2C TIME: 3180 TB:SDA: 0
# I2C: WR_ADDR_SEND TIME: 3180
# I2C TIME: 3400 TB:SDA: 0
# I2C:  WR_ADDR_ACK_WAITE TIME: 3400  ack 1
# I2C TIME: 3620 TB:SDA: 0
# I2C TIME: 3840 TB:SDA: 0
# I2C TIME: 4060 TB:SDA: 1
# I2C TIME: 4280 TB:SDA: 1
# I2C TIME: 4500 TB:SDA: 1
# I2C TIME: 4720 TB:SDA: 0
# I2C TIME: 4940 TB:SDA: 1
# I2C TIME: 5160 TB:SDA: 0
# I2C TIME: 5380 TB:SDA: 0
# I2C:  WR_DATA_ACK_WAITE: 5380
# I2C:  STOP: 5600
# Done danadone!
# ** Note: $finish    : N:/ECE71Project/i2cTB.sv(86)
#    Time: 5795 ns  Iteration: 0  Instance: /i2c_tb
# 1
# Break in Module i2c_tb at N:/ECE71Project/i2cTB.sv line 86
```

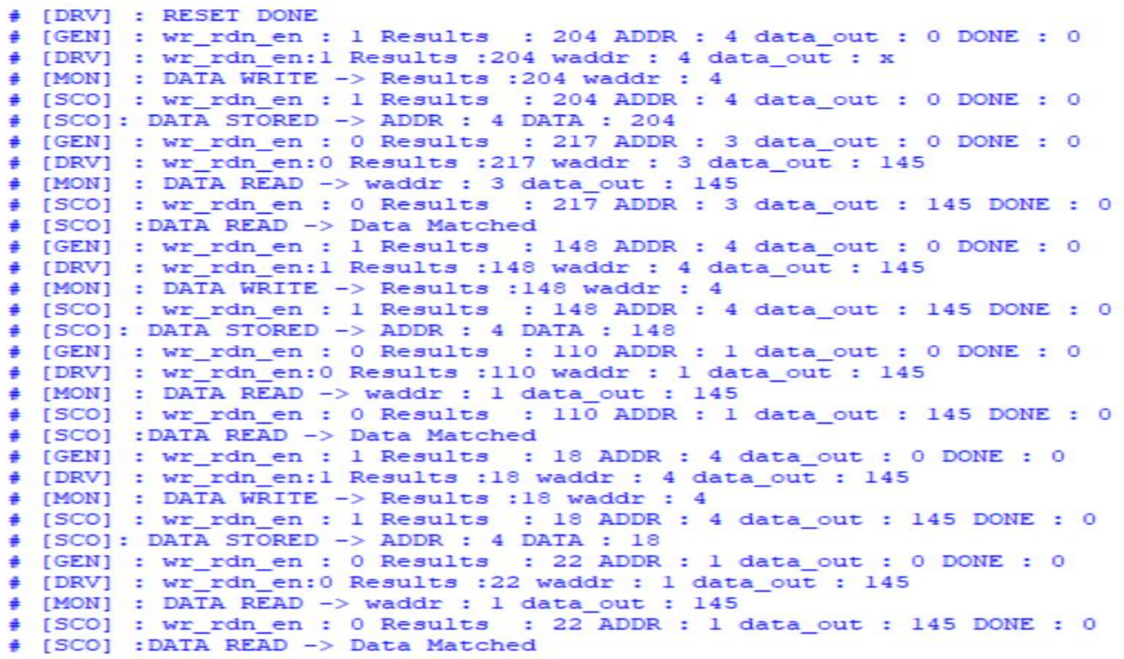
Results from integrating Memory Controller and I2C:



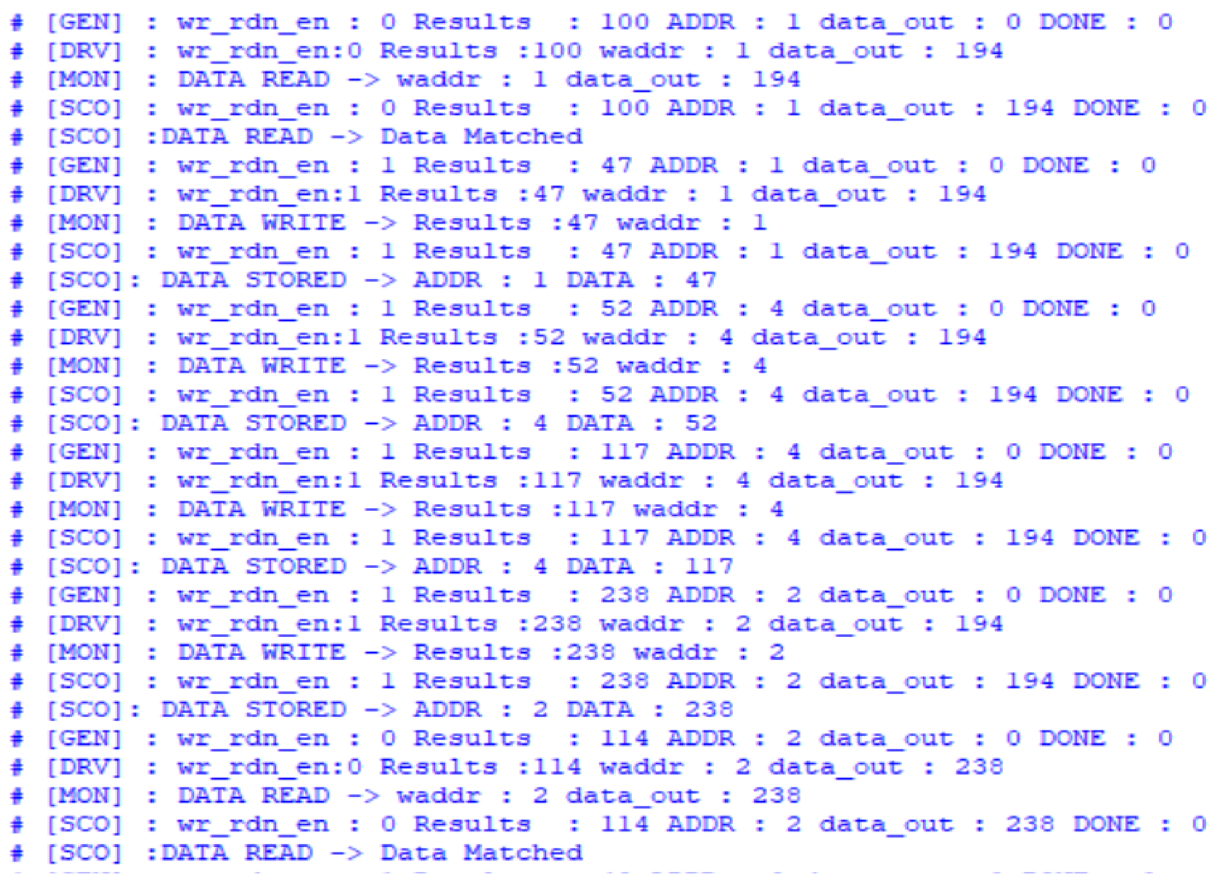
```
   0   wr_rdn_en = x   data_out = xxxxxxxx   done = x   Master_EN = x   reset_n = 1   input_data = x       addr = x
2001   wr_rdn_en = x   data_out = xxxxxxxx   done = 0   Master_EN = x   reset_n = 0   input_data = x       addr = x
2003   wr_rdn_en = 1   data_out = xxxxxxxx   done = 0   Master_EN = 1   reset_n = 0   input_data = 10101   addr = 10101
3003   wr_rdn_en = 1   data_out = xxxxxxxx   done = 0   Master_EN = 0   reset_n = 0   input_data = 10101   addr = 10101
3057   wr_rdn_en = 1   data_out = xxxxxxxx   done = 1   Master_EN = 0   reset_n = 0   input_data = 10101   addr = 10101
3101   wr_rdn_en = 1   data_out = xxxxxxxx   done = 0   Master_EN = 0   reset_n = 0   input_data = 10101   addr = 10101
3103   wr_rdn_en = 0   data_out = xxxxxxxx   done = 0   Master_EN = 1   reset_n = 0   input_data = 10101   addr = 10101
3717   wr_rdn_en = 0   data_out = xxxxxxx1   done = 0   Master_EN = 1   reset_n = 0   input_data = 10101   addr = 10101
3761   wr_rdn_en = 0   data_out = xxxxxx01   done = 0   Master_EN = 1   reset_n = 0   input_data = 10101   addr = 10101
3805   wr_rdn_en = 0   data_out = xxxxx101   done = 0   Master_EN = 1   reset_n = 0   input_data = 10101   addr = 10101
3849   wr_rdn_en = 0   data_out = xxxx0101   done = 0   Master_EN = 1   reset_n = 0   input_data = 10101   addr = 10101
3893   wr_rdn_en = 0   data_out = xxx10101   done = 0   Master_EN = 1   reset_n = 0   input_data = 10101   addr = 10101
3903   wr_rdn_en = 0   data_out = xxx10101   done = 0   Master_EN = 0   reset_n = 0   input_data = 10101   addr = 10101
3937   wr_rdn_en = 0   data_out = xx010101   done = 0   Master_EN = 0   reset_n = 0   input_data = 10101   addr = 10101
3981   wr_rdn_en = 0   data_out = x0010101   done = 0   Master_EN = 0   reset_n = 0   input_data = 10101   addr = 10101
4025   wr_rdn_en = 0   data_out = 00010101   done = 0   Master_EN = 0   reset_n = 0   input_data = 10101   addr = 10101
```

Results from Top Level Testbench:

Writing into a random address location and reading from the same address location



```
# [DRV] : RESET DONE
# [GEN] : wr_rdn_en : 1 Results  : 204 ADDR : 4 data_out : 0 DONE : 0
# [DRV] : wr_rdn_en:1 Results :204 waddr : 4 data_out : x
# [MON] : DATA WRITE -> Results :204 waddr : 4
# [SCO] : wr_rdn_en : 1 Results  : 204 ADDR : 4 data_out : 0 DONE : 0
# [SCO]: DATA STORED -> ADDR : 4 DATA : 204
# [GEN] : wr_rdn_en : 0 Results  : 217 ADDR : 3 data_out : 0 DONE : 0
# [DRV] : wr_rdn_en:0 Results :217 waddr : 3 data_out : 145
# [MON] : DATA READ -> waddr : 3 data_out : 145
# [SCO] : wr_rdn_en : 0 Results  : 217 ADDR : 3 data_out : 145 DONE : 0
# [SCO] :DATA READ -> Data Matched
# [GEN] : wr_rdn_en : 1 Results  : 148 ADDR : 4 data_out : 0 DONE : 0
# [DRV] : wr_rdn_en:1 Results :148 waddr : 4 data_out : 145
# [MON] : DATA WRITE -> Results :148 waddr : 4
# [SCO] : wr_rdn_en : 1 Results  : 148 ADDR : 4 data_out : 145 DONE : 0
# [SCO]: DATA STORED -> ADDR : 4 DATA : 148
# [GEN] : wr_rdn_en : 0 Results  : 110 ADDR : 1 data_out : 0 DONE : 0
# [DRV] : wr_rdn_en:0 Results :110 waddr : 1 data_out : 145
# [MON] : DATA READ -> waddr : 1 data_out : 145
# [SCO] : wr_rdn_en : 0 Results  : 110 ADDR : 1 data_out : 145 DONE : 0
# [SCO] :DATA READ -> Data Matched
# [GEN] : wr_rdn_en : 1 Results  : 18 ADDR : 4 data_out : 0 DONE : 0
# [DRV] : wr_rdn_en:1 Results :18 waddr : 4 data_out : 145
# [MON] : DATA WRITE -> Results :18 waddr : 4
# [SCO] : wr_rdn_en : 1 Results  : 18 ADDR : 4 data_out : 145 DONE : 0
# [SCO]: DATA STORED -> ADDR : 4 DATA : 18
# [GEN] : wr_rdn_en : 0 Results  : 22 ADDR : 1 data_out : 0 DONE : 0
# [DRV] : wr_rdn_en:0 Results :22 waddr : 1 data_out : 145
# [MON] : DATA READ -> waddr : 1 data_out : 145
# [SCO] : wr_rdn_en : 0 Results  : 22 ADDR : 1 data_out : 145 DONE : 0
# [SCO] :DATA READ -> Data Matched
```



```
# [GEN] : wr_rdn_en : 0 Results  : 100 ADDR : 1 data_out : 0 DONE : 0
# [DRV] : wr_rdn_en:0 Results :100 waddr : 1 data_out : 194
# [MON] : DATA READ -> waddr : 1 data_out : 194
# [SCO] : wr_rdn_en : 0 Results  : 100 ADDR : 1 data_out : 194 DONE : 0
# [SCO] :DATA READ -> Data Matched
# [GEN] : wr_rdn_en : 1 Results  : 47 ADDR : 1 data_out : 0 DONE : 0
# [DRV] : wr_rdn_en:1 Results :47 waddr : 1 data_out : 194
# [MON] : DATA WRITE -> Results :47 waddr : 1
# [SCO] : wr_rdn_en : 1 Results  : 47 ADDR : 1 data_out : 194 DONE : 0
# [SCO]: DATA STORED -> ADDR : 1 DATA : 47
# [GEN] : wr_rdn_en : 1 Results  : 52 ADDR : 4 data_out : 0 DONE : 0
# [DRV] : wr_rdn_en:1 Results :52 waddr : 4 data_out : 194
# [MON] : DATA WRITE -> Results :52 waddr : 4
# [SCO] : wr_rdn_en : 1 Results  : 52 ADDR : 4 data_out : 194 DONE : 0
# [SCO]: DATA STORED -> ADDR : 4 DATA : 52
# [GEN] : wr_rdn_en : 1 Results  : 117 ADDR : 4 data_out : 0 DONE : 0
# [DRV] : wr_rdn_en:1 Results :117 waddr : 4 data_out : 194
# [MON] : DATA WRITE -> Results :117 waddr : 4
# [SCO] : wr_rdn_en : 1 Results  : 117 ADDR : 4 data_out : 194 DONE : 0
# [SCO]: DATA STORED -> ADDR : 4 DATA : 117
# [GEN] : wr_rdn_en : 1 Results  : 238 ADDR : 2 data_out : 0 DONE : 0
# [DRV] : wr_rdn_en:1 Results :238 waddr : 2 data_out : 194
# [MON] : DATA WRITE -> Results :238 waddr : 2
# [SCO] : wr_rdn_en : 1 Results  : 238 ADDR : 2 data_out : 194 DONE : 0
# [SCO]: DATA STORED -> ADDR : 2 DATA : 238
# [GEN] : wr_rdn_en : 0 Results  : 114 ADDR : 2 data_out : 0 DONE : 0
# [DRV] : wr_rdn_en:0 Results :114 waddr : 2 data_out : 238
# [MON] : DATA READ -> waddr : 2 data_out : 238
# [SCO] : wr_rdn_en : 0 Results  : 114 ADDR : 2 data_out : 238 DONE : 0
# [SCO] :DATA READ -> Data Matched
```

Conclusion:

I2c is an easy and cheap communication protocol, it can be multi-master or multi-slave. In I2c we get the acknowledgment (ACK) and not acknowledgment (NACK) bits after each transmitted byte. Some disadvantage also attaches with I2C, it is a half-duplex communication and slow as compared to SPI (serial peripheral communication).