Fall- 2017

CMPE-200

Project Assignement-1

V.Vineela 012445186

M.S.Karthika 011494288

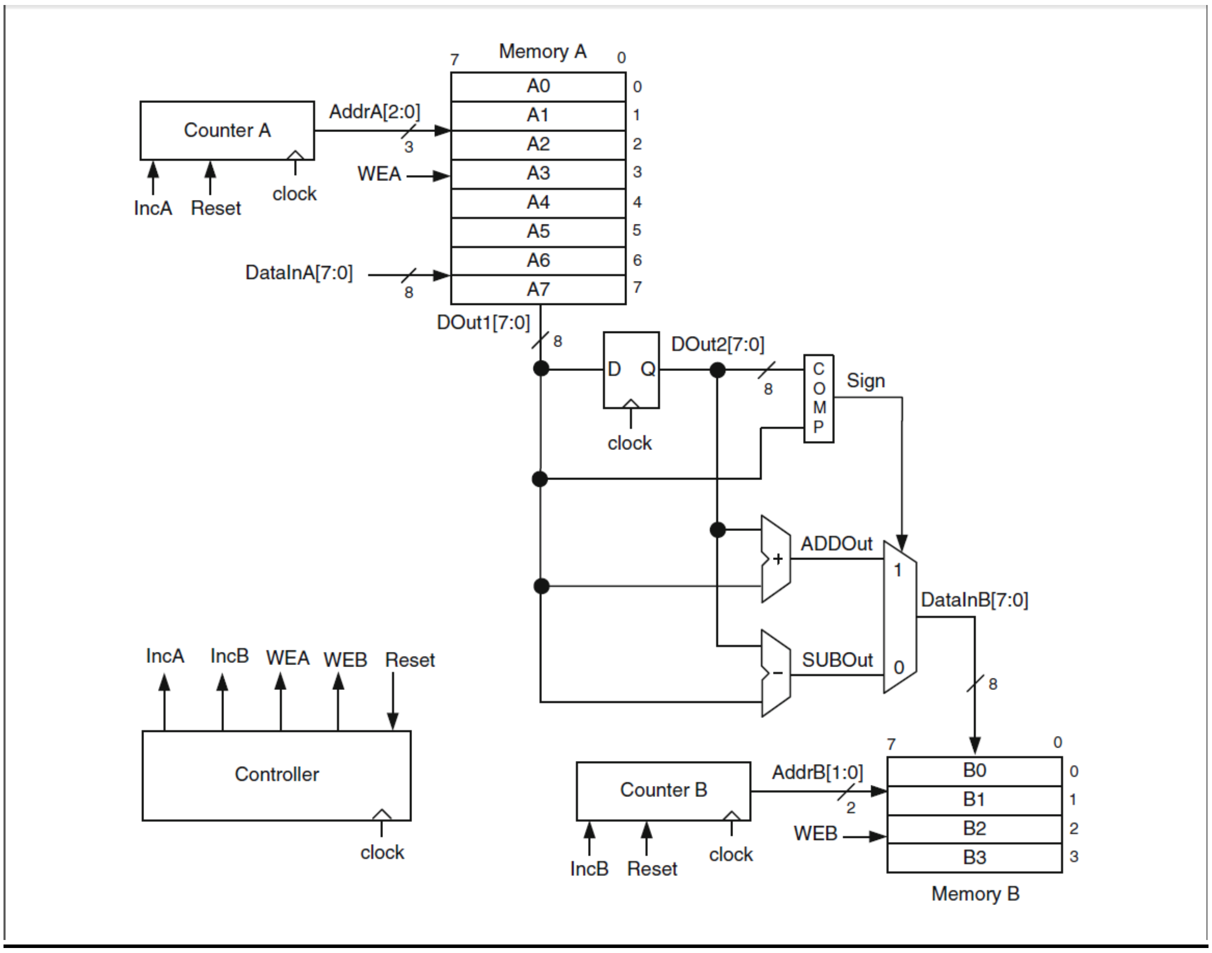
ABSTRACT

We have implemented a sequential Logic with a controller in this project. In the sequential logic we executed various concepts such as flip-flop, register, counter and memory. The hardware description language used for this project is Verilog and software used for designing and simulation is Modelsim. The project implementation involves decomposing the design circuit in to small, individual modules, which are designed and tested separately. After the successful testing of individual modules, they are integrated in a top module according to the functionality and was simulated in Modelsim to observe the waveforms. The design process primarily involved reading and writing data from/to Memory A (as shown in block diagram) and performing arithmetic operations such as adding and subtracting on the input data and writing the output to Memory B.

Introduction

Every design starts with gathering small or large logic blocks to meet the functional specifications of the design and to construct a data-path for proper data-flow. The sequential design in this project combines the data-path and controller design concepts. It also introduces the use of important sequential logic blocks such as flip-flop, register, counter and memory in the same design. Once the data-path is set among these components, then the precise data movements from one logic block to the next are described using timing diagrams. Any architectural change in the data-path should follow a corresponding change in the timing diagram or vice versa. When the data-path design and the timing diagram fully associate with each other, and each describes identical data movements, the next step in the design process is to build the controller that governs the flow of data. To define the states of the controller, the clock periods that generate different sets of outputs are separated from each other and named individually as distinct states. Similarly, the clock periods revealing identical outputs are grouped together under the same state name. The controller can be implemented by Moore-type state machine or by a counter-decoder-type design. We implemented it using counter-decoder design. The counter-decoder style design consists of a five-bit counter and four decoders to generate WEA, IncA, WEB and IncB control signals.

Block Diagram:



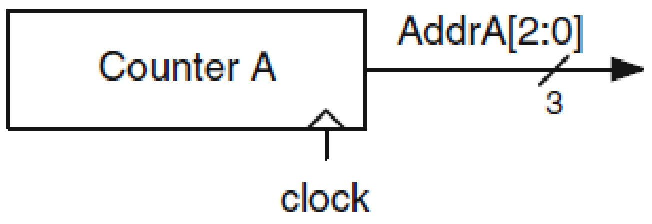
Individual Components:

1. Counter A:

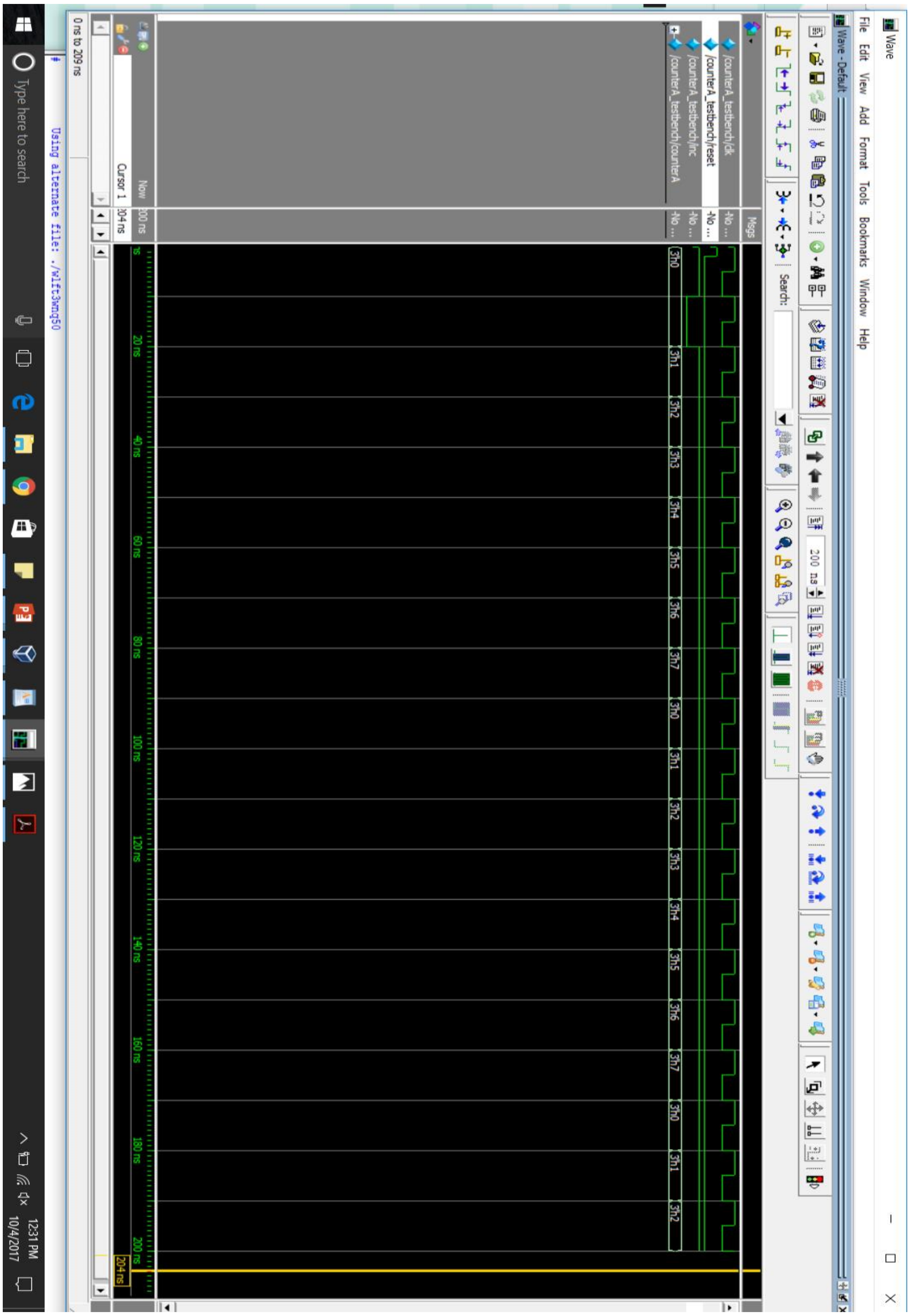
- **Functionality**: We have created a 3-bit up counter to generate addresses for memory A. Implementation involves storing the initial value (000) in register and incrementing it whenever incA is 1. Since the register is 3 bits, the maximum value is 7 and it sets to zero. This process goes on whenever the positive edge of clock comes. incA is the enable signal for the counter. We get the incA from the controller circuit.

Clock	**IncA**	**Input**	**Output**
posedge	1	000	001
posedge	1	001	010
posedge	1	010	011
posedge	1	011	100
posedge	1	100	101
posedge	1	101	110
posedge	1	110	111
Posedge	1	111	000
Posedge	0	111	111
Negedge	1	111	111

- **Block Diagram:**



- **Timing Diagram**:



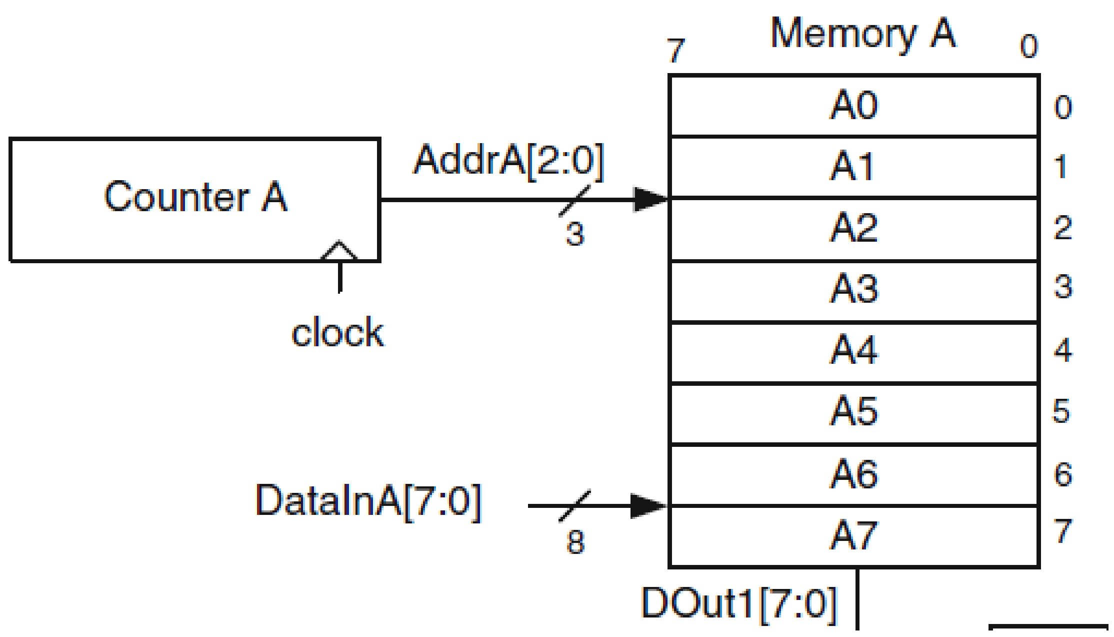
```
Using alternate file: ./wlft3wnq50
```

Wave
File Edit View Add Format Tools Bookmarks Window Help
Wave - Default
Msgs
/counterA_testbench/clk -No ...
/counterA_testbench/reset -No ...
/counterA_testbench/inc -No ...
/counterA_testbench/counterA -No ...
3'h0 3'h1 3'h2 3'h3 3'h4 3'h5 3'h6 3'h7 3'h0 3'h1 3'h2 3'h3 3'h4 3'h5 3'h6 3'h7 3'h0 3'h1 3'h2
Now 200 ns
Cursor 1 204 ns
ns 20 ns 40 ns 60 ns 80 ns 100 ns 120 ns 140 ns 160 ns 180 ns 200 ns
204 ns
0 ns to 209 ns
Search:
200 ns
Type here to search
12:31 PM
10/4/2017

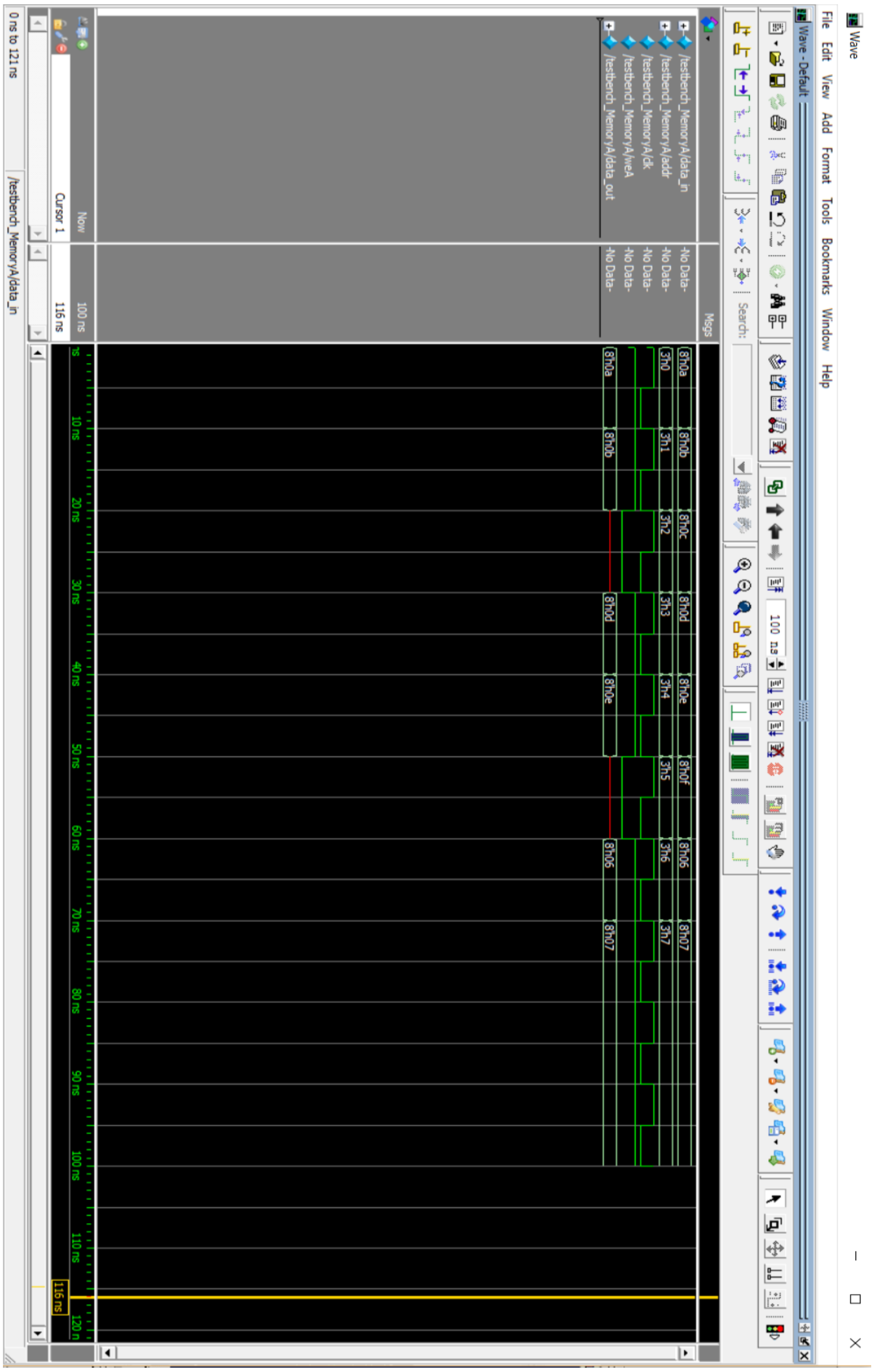
2. Memory A:

- **Functionality**: In this module we created a memory mem A of 7 rows and 7 columns. This module takes the input from the user and at every positive edge of clock cycle it is written in the memory created. Memory is written only when WEA=1.

- **Block Diagram**:



- **Timing Diagram**:



```
/testbench_MemoryA/data_in    -No Data-   8'h0a  8'h0b  8'h0c  8'h0d  8'h0e  8'h0f  8'h06  8'h07
/testbench_MemoryA/addr       -No Data-   3'h0   3'h1   3'h2   3'h3   3'h4   3'h5   3'h6   3'h7
/testbench_MemoryA/clk        -No Data-
/testbench_MemoryA/weA        -No Data-
/testbench_MemoryA/data_out   -No Data-   8'h0a  8'h0b         8'h0d  8'h0e         8'h06  8'h07

Now        100 ns
Cursor 1   116 ns

ns  10 ns  20 ns  30 ns  40 ns  50 ns  60 ns  70 ns  80 ns  90 ns  100 ns  110 ns  120 n
116 ns

0 ns to 121 ns    /testbench_MemoryA/data_in
```

3. ALU:

- **Functionality**: An ALU takes the inputs from the memory. Using the D Flip flop the first output is stored for one clock cycle and passed to Dout2. If Dout2>Dout1 then sign bit is given the value 0 and subtraction is performed. Otherwise sign bit is 1 and addition is performed.

D Flip Flop:

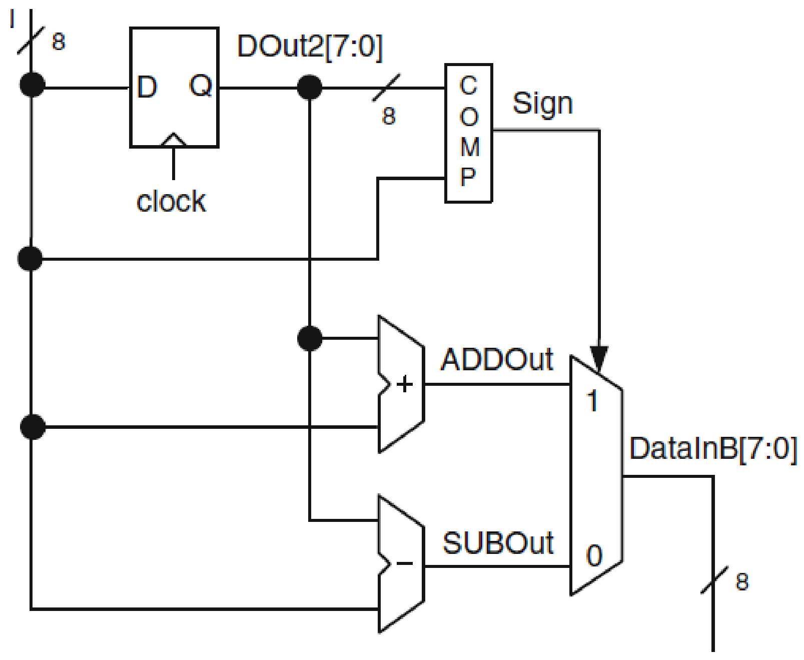
Clock	Input	Output
Posedge	0	0
Posedge	1	1
Negedge	X	Previous state

x -> any value(0 or1) , output will be in the previous state

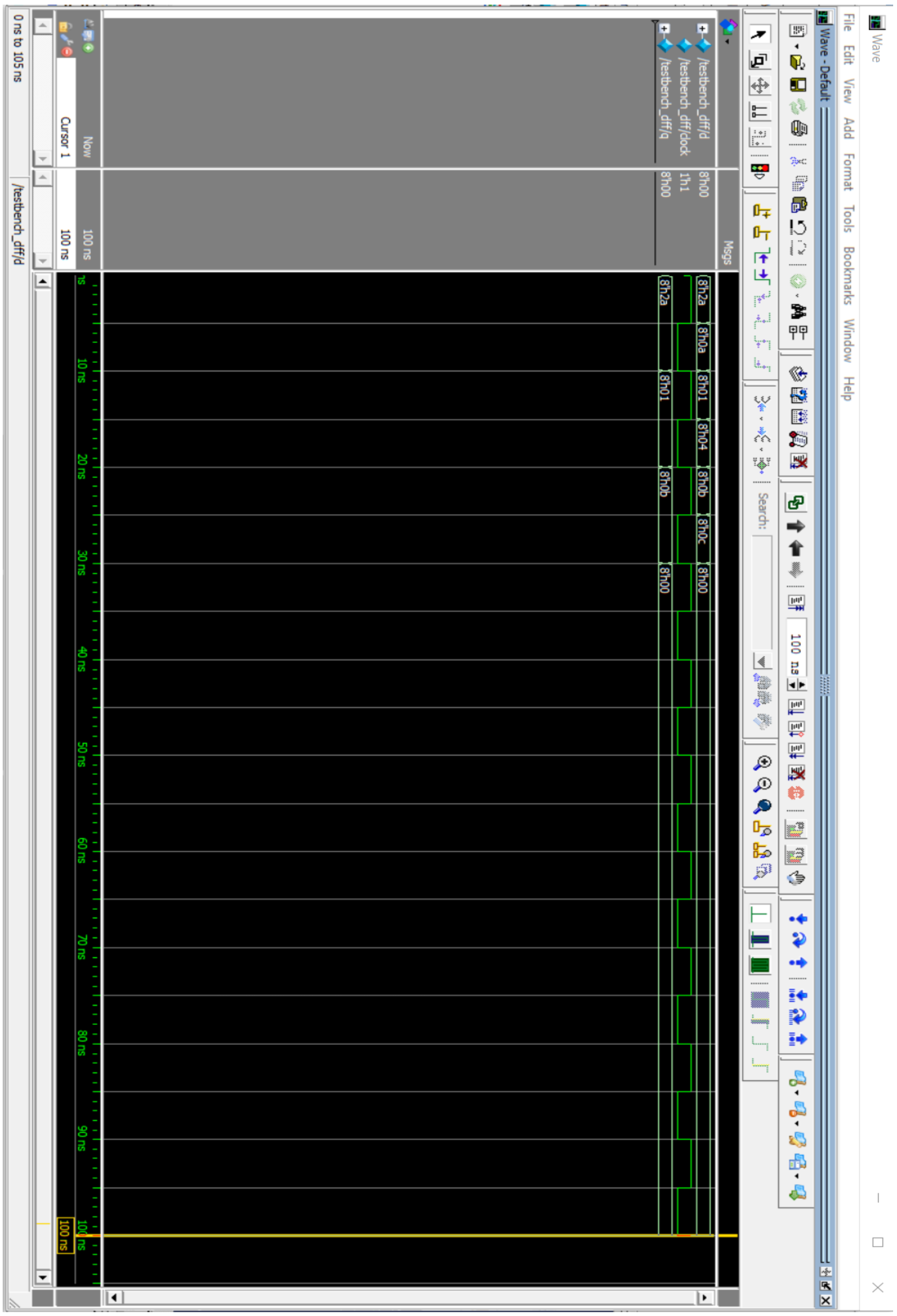
ALU Table:

Dout1	Dout2	Sign	Output(add/subtract)
1001	1011	0	0010
1101	1001	1	10000
0110	1101	0	0111
1110	0100	1	1100

- **Block Diagram**:



- **Timing Diagram**: D Flip Flop



Wave - Default

	Msgs
/testbench_dff/d	8'h00
/testbench_dff/clock	1'h1
/testbench_dff/q	8'h00

/testbench_dff/d: 8'h2a, 8'h0a, 8'h01, 8'h04, 8'h0b, 8'h0c, 8'h00

/testbench_dff/q: 8'h2a, 8'h01, 8'h0b, 8'h00

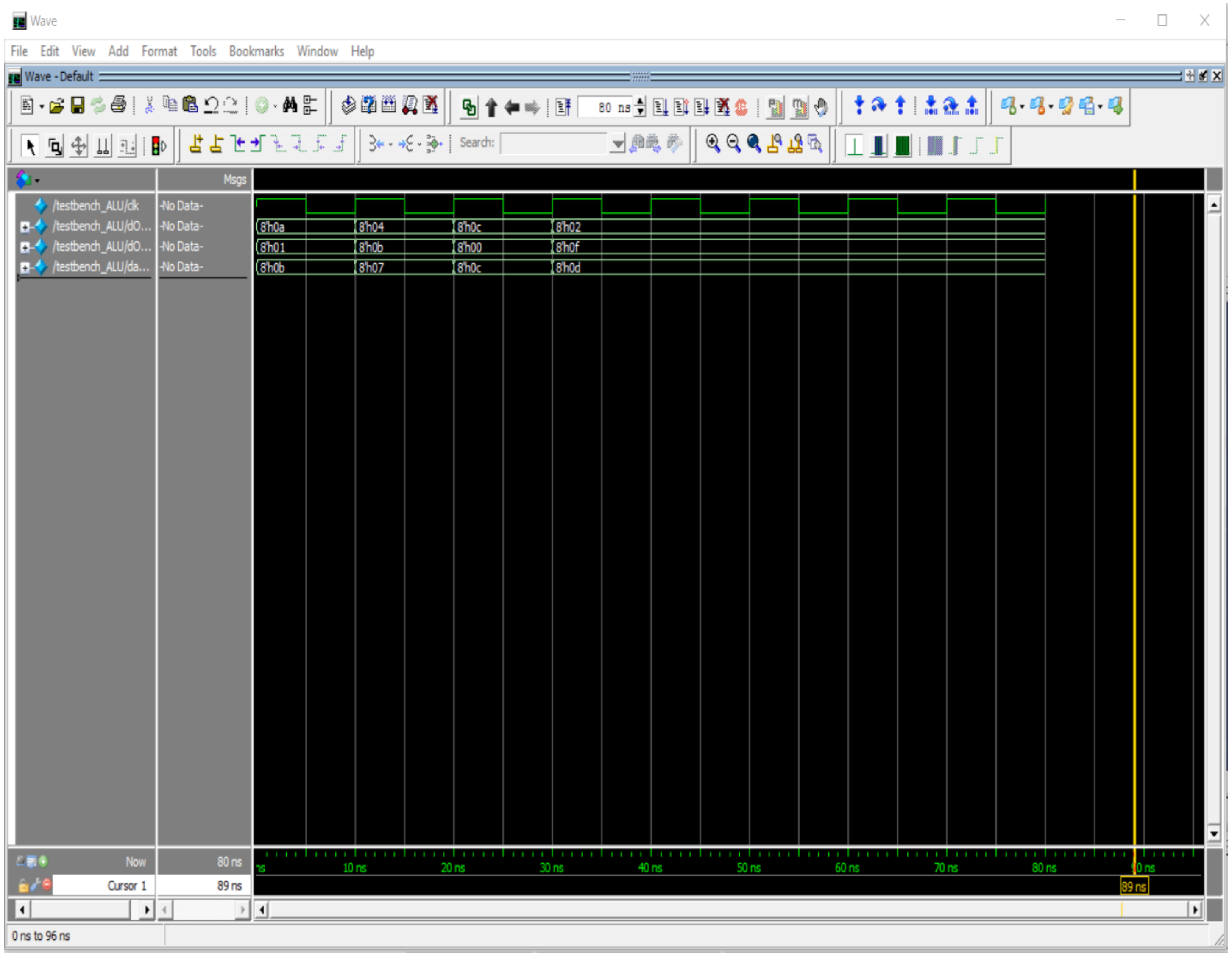
Now 100 ns

Cursor 1 100 ns

0 ns to 105 ns

/testbench_dff/d

ALU:



Wave
File Edit View Add Format Tools Bookmarks Window Help
Wave - Default
80 ns
Search:
Msgs

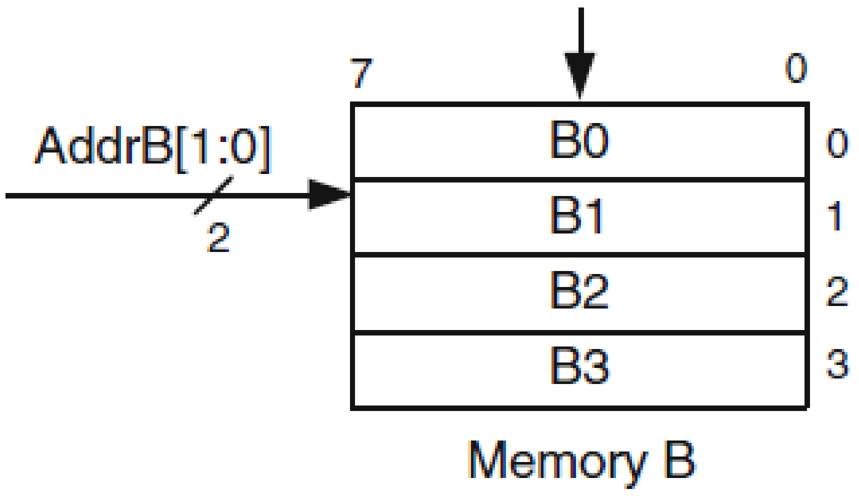
```
/testbench_ALU/clk      -No Data-
/testbench_ALU/dO...    -No Data-   8'h0a  8'h04  8'h0c  8'h02
/testbench_ALU/dO...    -No Data-   8'h01  8'h0b  8'h00  8'h0f
/testbench_ALU/da...    -No Data-   8'h0b  8'h07  8'h0c  8'h0d
```

Now 80 ns
Cursor 1 89 ns
ns 10 ns 20 ns 30 ns 40 ns 50 ns 60 ns 70 ns 80 ns 90 ns
89 ns
0 ns to 96 ns

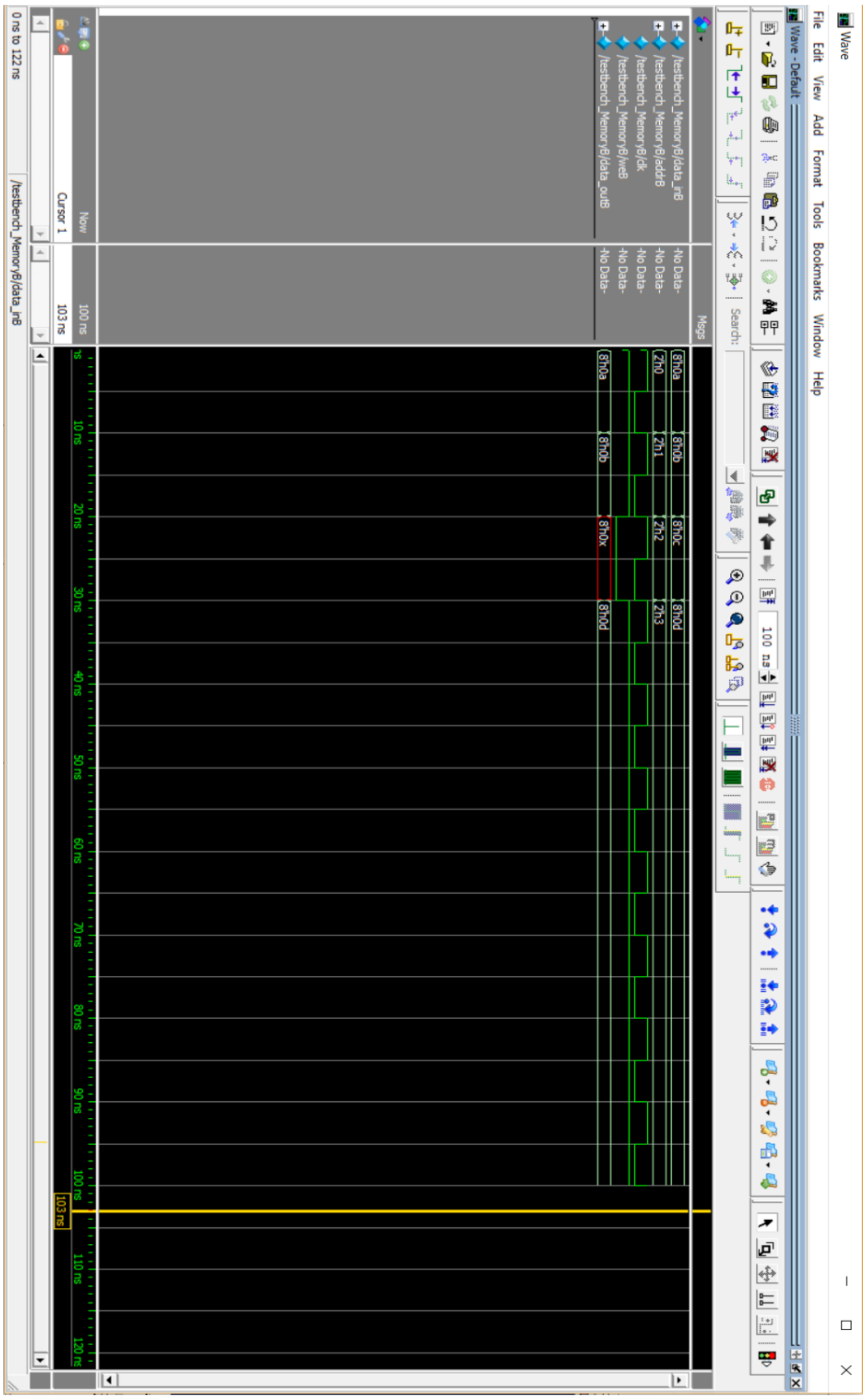
4. Memory B:

- **Functionality**: In this module we created a memory mem B of 4 rows and 7 columns. This module takes the input from the ALU and at every positive edge of clock cycle it is written in the memory we created. Memory is written only when WEB=1.

- **Block Diagram**:



- **Timing Diagram**:



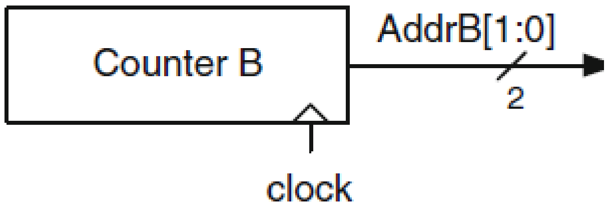
```
Msgs
/testbench_Memory8/data_in8   -No Data-   8'h0a   8'h0b   8'h0c   8'h0d
/testbench_Memory8/addr8      -No Data-   2'h0    2'h1    2'h2    2'h3
/testbench_Memory8/clk        -No Data-
/testbench_Memory8/we8        -No Data-
/testbench_Memory8/data_out8  -No Data-   8'h0a   8'h0b   8'h0x   8'h0d

Now        100 ns
Cursor 1   103 ns
0 ns to 122 ns
```

5. Counter B:

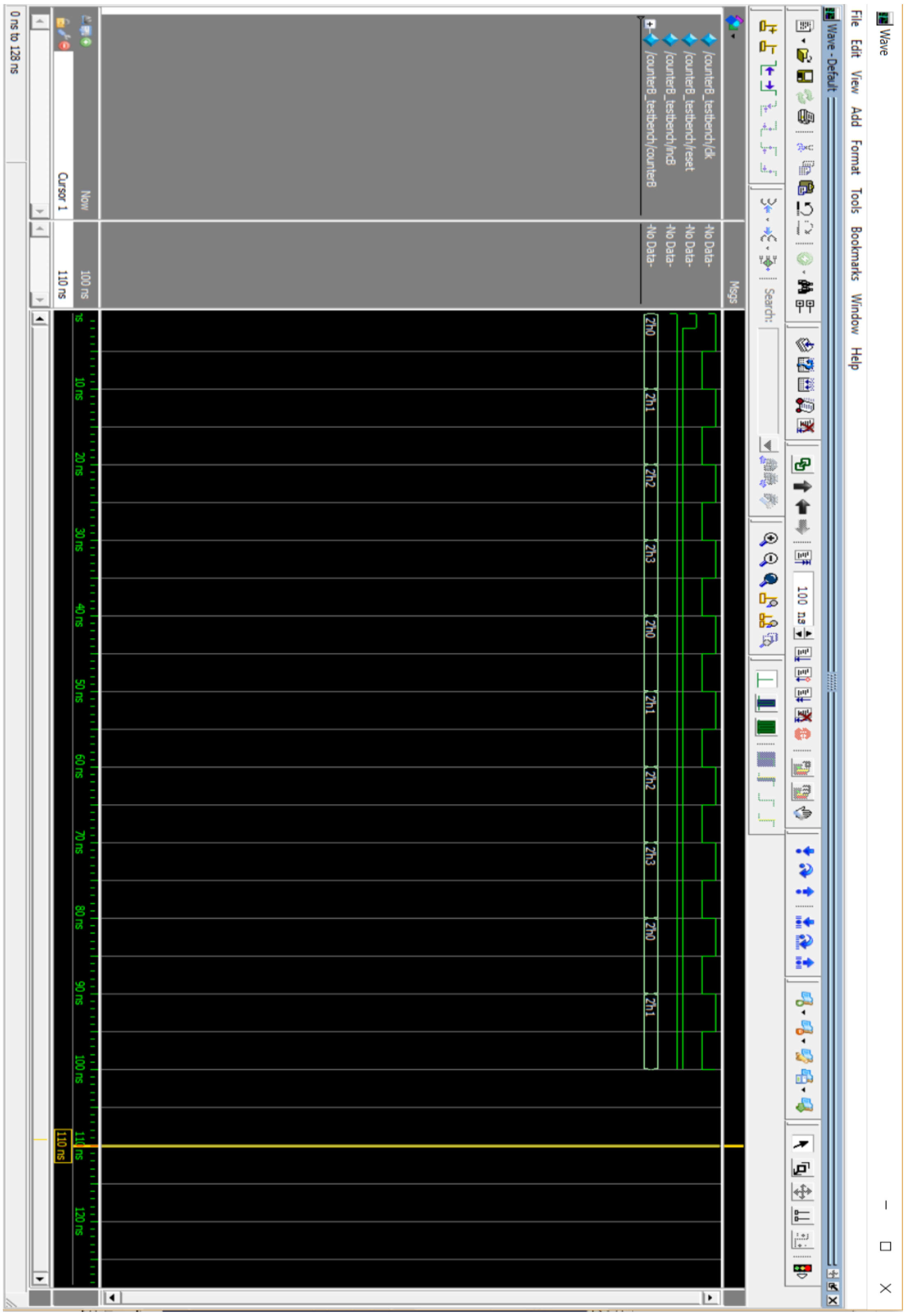
- **Functionality**: We have created a 2-bit up counter to generate addresses for memory B. Implementation involves storing the initial value (000) in register and incrementing it whenever incB is high. Since the register is 2 bits, the maximum value is 4 and it sets to zero. This process goes on when both positive edge of clock and incB is high.

Clock	IncB	Input	Output
Posedge	1	00	01
Posedge	1	01	10
Posedge	1	10	11
Posedge	1	11	00
Posedge	0	11	11
negedge	1	11	11

- **Block Diagram**:



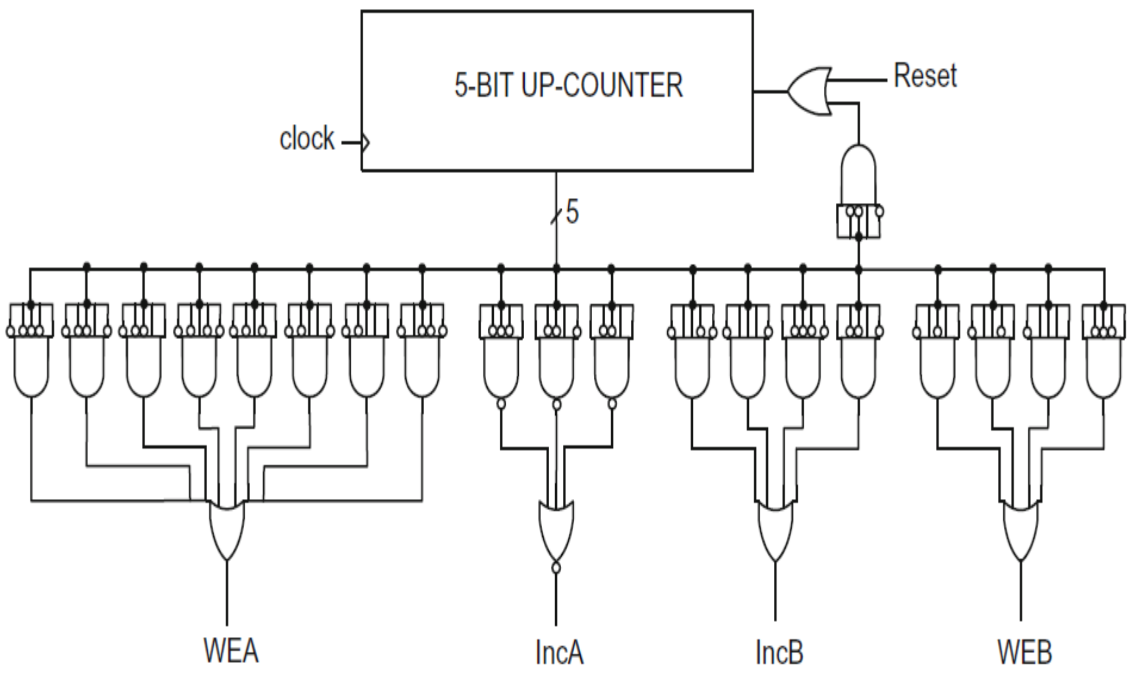
- **Timing Diagram**:


Wave
File Edit View Add Format Tools Bookmarks Window Help
Wave - Default
Msgs
/counter8_testbench/clk -No Data-
/counter8_testbench/reset -No Data-
/counter8_testbench/incB -No Data-
/counter8_testbench/counter8 -No Data-
2'h0 2'h1 2'h2 2'h3 2'h0 2'h1 2'h2 2'h3 2'h0 2'h1
Now 100 ns
Cursor 1 110 ns
10 ns 20 ns 30 ns 40 ns 50 ns 60 ns 70 ns 80 ns 90 ns 100 ns 110 ns 120 ns
0 ns to 128 ns

6. Controller:

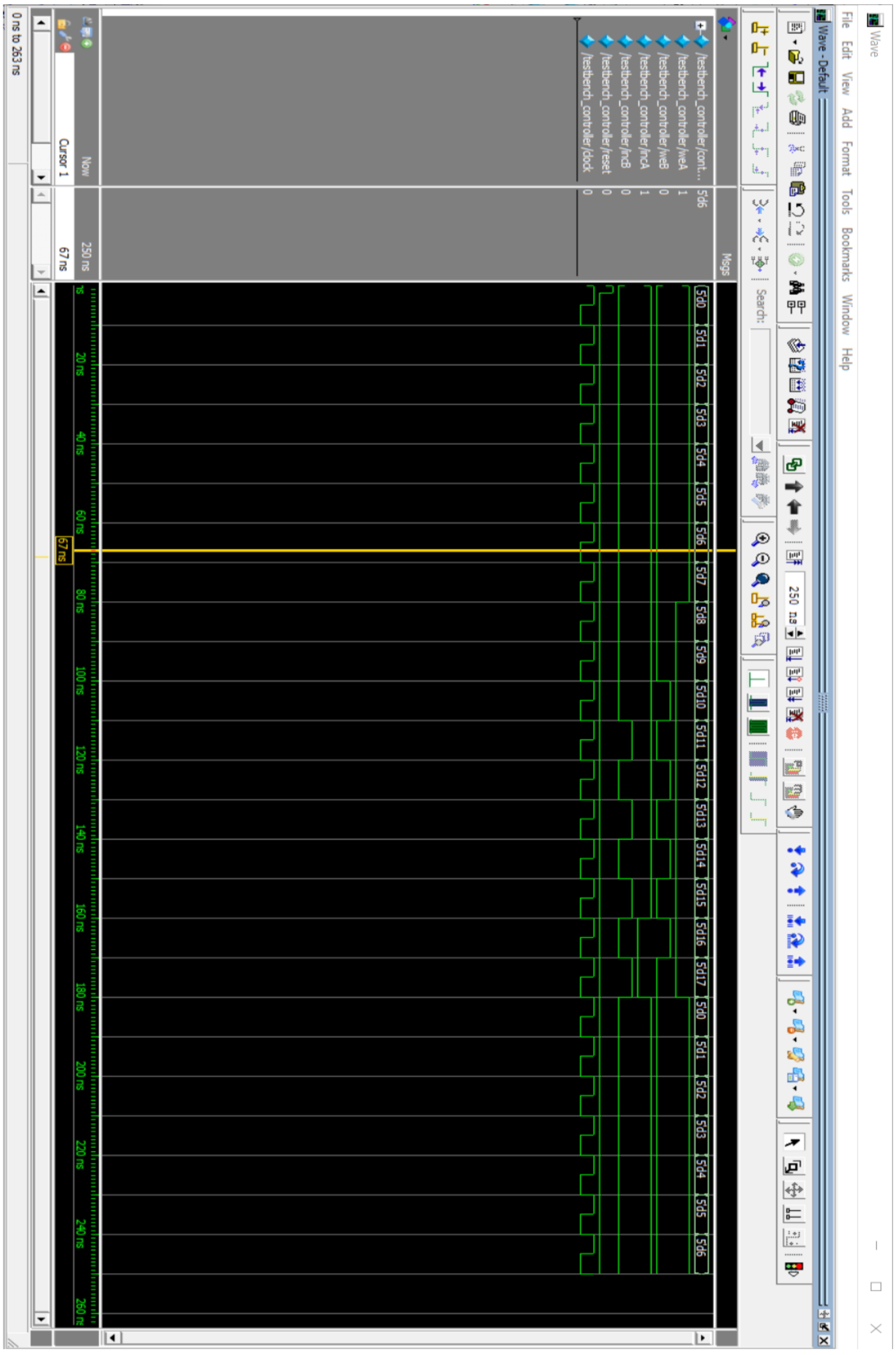
- **Functionality**: Controller is implemented using counter decoder design to generate WEA, IncA, WEB, IncB control signals.

- **Block Diagram**:



Clock Cycle	Reset	WEA	IncA	IncB	WEB
0	0	1	1	0	0
1	0	1	1	0	0
2	0	1	1	0	0
3	0	1	1	0	0
4	0	1	1	0	0
5	0	1	1	0	0
6	0	1	1	0	0
7	0	1	1	0	0
8	0	0	1	0	0
9	0	0	1	0	0
10	0	0	1	0	1
11	0	0	1	1	0
12	0	0	1	0	1
13	0	0	1	1	0
14	0	0	1	0	1
15	0	0	1	1	0
16	0	0	0	0	1
17	0	0	0	1	0
18	0	0	0	0	0
x	1	1	0	0	0

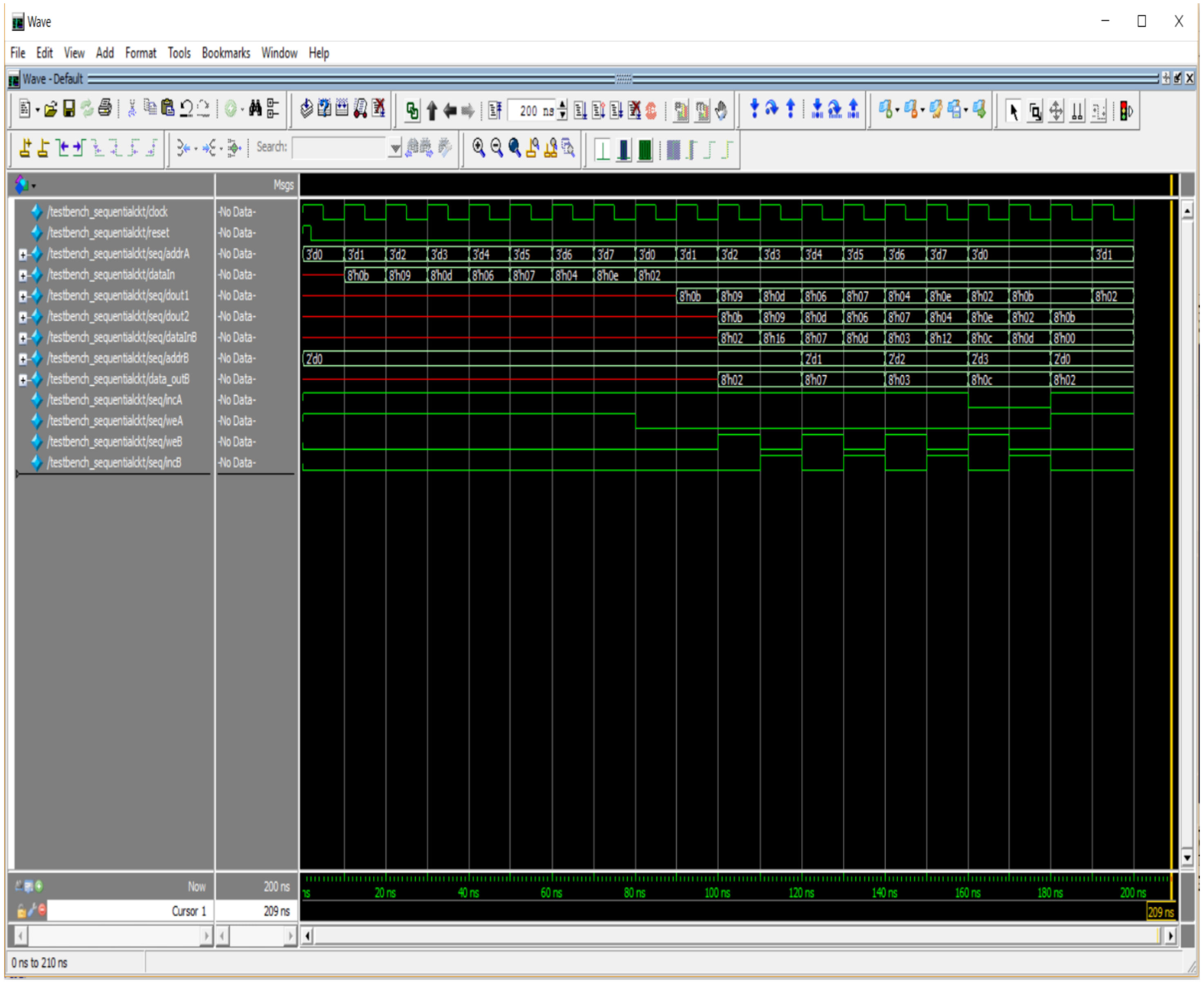
- **Timing Diagram**:



Top Module: We integrated all the modules after individually testing them. We used wires to connect the modules. The top module functionality can be described as below.

Initially, counter A generates the addresses, 0 to 7, for memory A and writes the data packets, A0 to A7, through DataInA[7:0] port. This is shown in the timing diagram from clock cycles 1 through 8. When this task is complete, counter A resets and reads the first data packet A0 from AddrA[2:0] = 0 in clock cycle 9. In the next clock cycle, A0 becomes available at DOut1, and the counter A increments by one. In cycle 11, AddrA [2:0] becomes 2, the data packet A1 is read from DOut1[7:0], and the data packet A0 transfers to DOut2[7:0]. In this cycle, the contents of the data packets A0 and A1 are compared with each other by subtracting A1 (at DOut1) from A0 (at DOut2). If the contents of A0 are less than A1, then the sign bit, Sign, of (A0 – A1) becomes negative. Sign = 1 selects (A0 + A1) at ADDOut[7:0] and routes this value to DataInB[7:0]. However, if the contents of A0 are greater than A1, (A0 – A1) becomes positive. Sign = 0 selects (A0 – A1) and routes this value from SUBOut[7:0] to DataInB[7:0]. The result at DataInB[7:0] is written at AddrB[1:0] = 0 of memory B at the positive edge of clock cycle 12. In the same cycle, A1 is transferred to DOut2[7:0], and A2 becomes available at DOut1[7:0]. A comparison between A1 and A2 takes place, and either (A1 + A2) or (A1 – A2) is prompted to be written to memory B depending on the value of the Sign node. However, this is an unwarranted step in the data transfer process because the design requirement states that the comparison has to be done only once between two data packets from memory A. Since A1 is used in an earlier comparison with A0, A1 cannot be used in a subsequent comparison with A2, and neither (A1 + A2) nor (A1 – A2) should be written to memory B. The remaining clock cycles from 13 through 18 compare the values of A2 with A3, A4 with A5, and A6 with A7, and write the added or subtracted results into memory B. After clock cycle 19, all operations on this data-path suspend, the counters are reset and all writes to the memory core are disabled.

Timing Diagram:



```
/testbench_sequentialckt/clock          -No Data-
/testbench_sequentialckt/reset          -No Data-
/testbench_sequentialckt/seq/addrA      -No Data-   3'd0 3'd1 3'd2 3'd3 3'd4 3'd5 3'd6 3'd7 3'd0 3'd1 3'd2 3'd3 3'd4 3'd5 3'd6 3'd7 3'd0 3'd1
/testbench_sequentialckt/dataIn         -No Data-   8'h0b 8'h09 8'h0d 8'h06 8'h07 8'h04 8'h0e 8'h02
/testbench_sequentialckt/seq/dout1      -No Data-   8'h0b 8'h09 8'h0d 8'h06 8'h07 8'h04 8'h0e 8'h02 8'h0b 8'h02
/testbench_sequentialckt/seq/dout2      -No Data-   8'h0b 8'h09 8'h0d 8'h06 8'h07 8'h04 8'h0e 8'h02 8'h0b
/testbench_sequentialckt/seq/dataInB    -No Data-   8'h02 8'h16 8'h07 8'h0d 8'h03 8'h12 8'h0c 8'h0d 8'h00
/testbench_sequentialckt/seq/addrB      -No Data-   2'd0 2'd1 2'd2 2'd3 2'd0
/testbench_sequentialckt/data_outB      -No Data-   8'h02 8'h07 8'h03 8'h0c 8'h02
/testbench_sequentialckt/seq/incA       -No Data-
/testbench_sequentialckt/seq/weA        -No Data-
/testbench_sequentialckt/seq/weB        -No Data-
/testbench_sequentialckt/seq/incB       -No Data-

Now       200 ns    ns 20 ns 40 ns 60 ns 80 ns 100 ns 120 ns 140 ns 160 ns 180 ns 200 ns
Cursor 1  209 ns    209 ns
0 ns to 210 ns
```

Difficulties Encountered:

Below listed are the problems we encountered while doing our project.

1) While writing the input data to memory A during the first positive cycle, all elements in data set except the first one was being written in to the memory. As a result, one data was missing in both read and write operations.

   Solution: We included a delay of 10ns while inputting the first set of data. After that the read and write operations were performed correctly.
2) We found it difficult to synchronize the output of D flipflop (dout2) and dout1 as dout2 = dout1 .

   Solution: We created a d flip flop module separately by giving dout1 as input from alu and taking its output after one cycle. So both dout1 and dout2 are different values and output can be obtained.
3) Writing alternated values in to memory B

   Solution: We were storing values only when data input to memory B was changing. We gave a condition as posedge DataInB.

Conclusion: We implemented the given sequential design using Verilog. We showed various timing diagrams for each module. We integrated all the individual modules using wires to form a top module. The controller is provided with the clock, reset which generate a 5 bit counter. This counter decides the values for WeA, WeB, IncA, IncB. These values decide the various operations to be done in another modules. We provide Data which is written into Memory A with the help of counter A. Now the arithmetic operations are performed on this data and new data is read into Memory B with the help of Counter B. This is hoe the implementation takes place in this design.