

## Programs: Computer Engineering

Course Number	<b>COE608</b>
Course Title	<b>Computer Organization and Architecture</b>
Semester/Year	<b>Summer 2020</b>
Instructor	<b>Patrick Siddavaatam</b>

<b>Lab Report No.</b>	<b>4a</b>
-----------------------	-----------

Lab Title	<b>Data Memory Module</b>
-----------	---------------------------

Section No.	<b>011</b>
Group No.	
Submission Date	<b>28 July 2020</b>
Due Date	<b>28 July 2020</b>

Name	Student ID	Signature*
Duanwei Zhang	500824903	DZ
Xinyu Hadrian Hu	500194233	XHH

\*By signing above you attest that you have contributed to this submission and confirm that all work you have contributed to this submission is your own work. Any suspicion of copying or plagiarism in this work will result in an investigation of Academic Misconduct and may result in a "0" on the work, an "F" in the course, or possibly more severe penalties, as well as a Disciplinary Notice on your academic record under the Student Code of Academic Conduct, which can be found online at:

[www.ryerson.ca/senate/current/pol60.pdf](http://www.ryerson.ca/senate/current/pol60.pdf).

Name: Xinyu Hadrian Hu, and Duan Wei Zhang

Student Number: 500194233, and 500824903

TA: Jasminder Singh

Date: June 28, 2020

COE 608: Computer Architecture and Design

---

## **COE 608: Lab 4, Part 1 Report**

### **Objective**

The purpose of this lab is to create a memory module for the 32-bit ALU, and later the instruction set architecture (ISA) for the million instructions per second (MIPS) ISA.

### **Design and Implementation**

Provided below is the truth table for the memory module. The address is an 8-bit data, and the data-in and data-outs are 32-bit data. See Figure 1: Memory Module Truth Table.

Memory Module (sample from waveform)					
clk	en	wen (unsigned)	data_in (unsigned)	data_out(unsigned)	
0	1	0	0	0	
1	0	1	0	0	
0	1	2	2	0	
1	1	3	2	0	
0	1	4	4	0	
1	0	5	4	0	
0	1	6	6	0	
1	1	7	6	0	
0	0	8	8	0	
1	0	9	8	0	
0	1	10	10	0	
1	1	11	10	0	
0	1	12	12	4	
1	1	13	12	4	

Figure 1: Memory Module Truth Table

## Observations and Results

Below are the observations from functional and timing waveforms: See Figure 2: Memory Module Functional Waveform and Figure 3: Memory Module Timing Waveform.

## Memory Module: Functional and Timing

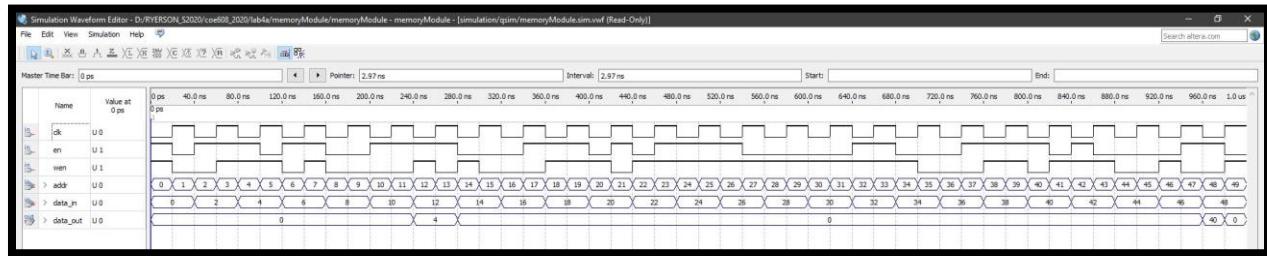


Figure 2: Memory Module Functional Waveform

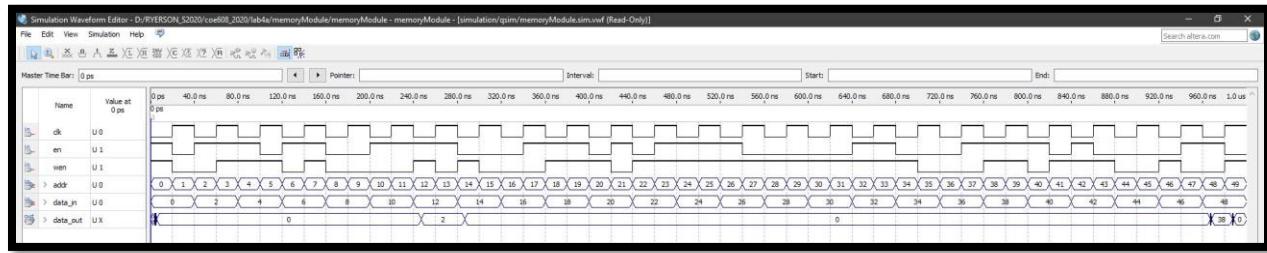


Figure 3: Memory Module Timing Waveform

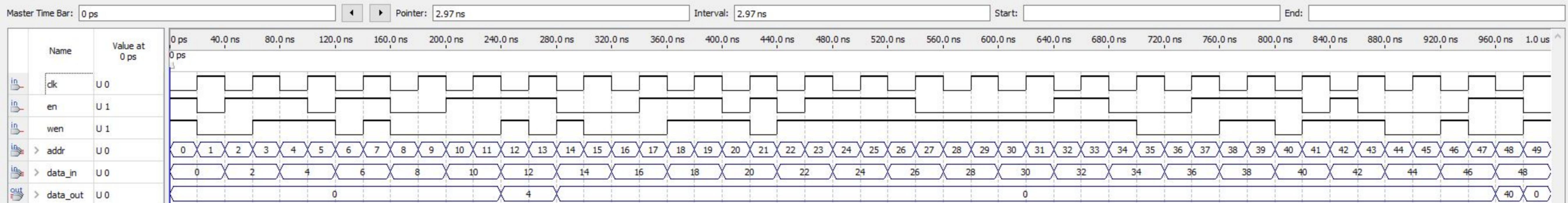
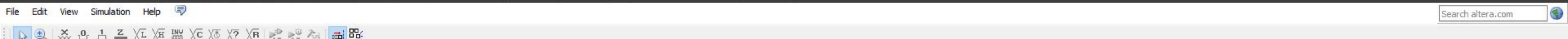
## Discussions and Conclusions

The memory module works as intended. The worst case delays for the memory module is 5 ns, as can be observed from the timing simulation waveform.

## Appendix: VHDL Codes and Screenshots of Waveforms

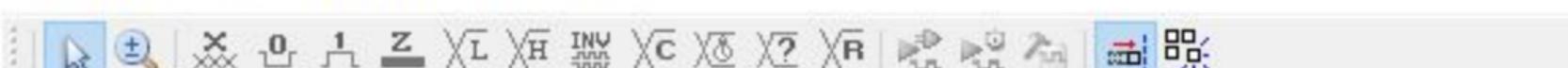
I have provided the VHDL and screenshots of the waveforms for easier reading.

```
1 library ieee;
2 use ieee.std_logic_1164.all;
3 use ieee.numeric_std.all;
4
5 entity memoryModule is
6     port (
7         clk: in std_logic;
8         addr: in unsigned (7 downto 0);
9         data_in : in std_logic_vector (31 downto 0);
10        wen, en: in std_logic;
11        data_out: out std_logic_vector (31 downto 0));
12 end memoryModule;
13
14 architecture description of memoryModule is
15     type memory2D is array (7 downto 0) of std_logic_vector (31 downto 0);
16     signal memoryModuleArray : memory2D;
17     begin
18         process(clk,en,wen)
19             begin
20                 if falling_edge (clk) then
21                     if (en = '1' and wen = '0') then
22                         data_out <= memoryModuleArray (to_integer(unsigned(addr)));
23                     elsif (en = '1' and wen = '1') then
24                         memoryModuleArray (to_integer(unsigned(addr))) <= data_in;
25                         data_out <= (others => '0');
26                     end if;
27                 end if;
28                 if en = '0' then
29                     data_out <= (others => '0');
30                 end if;
31             end process;
32     end description;
33
34
```



File Edit View Simulation Help

Search altera.com



The timing diagram displays the following signals over a period of 1.0 us:

- clk**: A square wave signal starting at U 0 at 0 ps, transitioning to U 1 at approximately 40 ns, and back to U 0 at approximately 160 ns.
- en**: A signal that is U 0 from 0 ps to approximately 160 ns, then jumps to U 1 and remains high until 1.0 us.
- wen**: A signal that is U 0 from 0 ps to approximately 160 ns, then jumps to U 1 and remains high until 1.0 us.
- > addr**: An address bus signal ranging from 0 to 49. It starts at 0, increments by 1 every 20 ns, and wraps around after 49.
- > data\_in**: An 8-bit data input bus. The values are: 0, 2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 22, 24, 26, 28, 30, 32, 34, 36, 38, 40, 42, 44, 46, 47, 48, 49.
- > data\_out**: An 8-bit data output bus. The values are: 0, 2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 22, 24, 26, 28, 30, 32, 34, 36, 38, 40, 42, 44, 46, 48, 0.

## Programs: Computer Engineering

Course Number	<b>COE608</b>
Course Title	<b>Computer Organization and Architecture</b>
Semester/Year	<b>Summer 2020</b>
Instructor	<b>Patrick Siddavaatam</b>

<b>Lab Report No.</b>	<b>4b</b>
-----------------------	-----------

Lab Title	<b>CPU Data Path Design</b>
-----------	-----------------------------

Section No.	<b>011</b>
Group No.	
Submission Date	<b>28 July 2020</b>
Due Date	<b>28 July 2020</b>

Name	Student ID	Signature*
Duanwei Zhang	500824903	DZ
Xinyu Hadrian Hu	500194233	XHH

*\*By signing above you attest that you have contributed to this submission and confirm that all work you have contributed to this submission is your own work. Any suspicion of copying or plagiarism in this work will result in an investigation of Academic Misconduct and may result in a "0" on the work, an "F" in the course, or possibly more severe penalties, as well as a Disciplinary Notice on your academic record under the Student Code of Academic Conduct, which can be found online at:*

[www.ryerson.ca/senate/current/pol60.pdf](http://www.ryerson.ca/senate/current/pol60.pdf).

Name: Xinyu Hadrian Hu, and Duan Wei Zhang

Student Number: 500194233, and 500824903

TA: Jasminder Singh

Date: June 28, 2020

COE 608: Computer Architecture and Design

---

## **COE 608: Lab 4, Part 2 Report**

### **Table of Figures**

Figure 1: UZE Truth Table .....	3
Figure 2: LZE Truth Table.....	3
Figure 3: Reducer Truth Table.....	3
Figure 4: Data-path signals truth table.....	4
Figure 5: LZE Functional Waveform .....	5
Figure 6: LZE Timing Waveform.....	6
Figure 7: UZE Functional Waveform.....	6
Figure 8: UZE Timing Waveform .....	7
Figure 9: Reducer Functional Waveform .....	7
Figure 10: Reducer Timing Waveform.....	7
Figure 11: Data-path Random Input Waveform .....	8
Figure 12: Data-path Output Functional Waveform.....	8
Figure 13: Data-path Output Timing Waveform .....	8
Figure 14: LDAI Operation Waveform .....	10
Figure 15: LDBI Operation Waveform.....	11
Figure 16: STA Operation Waveform .....	12
Figure 17: STB Operation Waveform.....	13
Figure 18: LDA Operation Waveform.....	14
Figure 19: LDB Operation Waveform.....	15
Figure 20: LUI Operation Waveform .....	16
Figure 21: JMP Operation Waveform.....	17

Figure 22: ADD Operation Waveform .....	18
Figure 23: ADDI Operation Waveform .....	19
Figure 24: SUB Operation Waveform .....	20
Figure 25: INCA Operation Waveform .....	21
Figure 26: ROL Operation Waveform .....	22
Figure 27: CLRA Operation Waveform .....	23
Figure 28: ANDI Operation Waveform .....	24
Figure 29: ORI Operation Waveform .....	25
Figure 30: DECA Operation Waveform .....	26
Figure 31: ROR Operation Waveform.....	27

## **Objective**

The purpose of this lab is to put together everything we have done so far, with the addition of the upper zero extender, lower zero extender, and reducer with the program counter, arithmetic logical unit, multiplexers, and operational inverters into a functioning data-path for the MIPS instructional set architecture.

## **Design and Implementation**

I first started with the reducers, LZE and UZE implementations. I did the data-path last. The following truth-tables provide examples of some of the purposes of the LZE, UZE, and reducer. The LZE extends zeroes by padding the bits from 15 down to 0. The UZE does the inverse of the LZE operation by padding zeroes from 31 down to 16. The reducer helps to reduce a 32-bit data into 8-bit data. The data-path is programmed in VHDL using the structural behavior, by utilizing port mapping and signals to denote the wires and buses between the many devices used throughout the course.

The truth tables for each of the new devices are illustrated in the following pages.

See the following figures below: Figure 1: UZE Truth Table, Figure 2: LZE Truth Table, Figure 3: Reducer Truth Table and Figure 4: Data-path signals truth table.

Upper Zero Extender	
In (hex)	Out (hex)
0 0 0 0 0 0 1	0 0 0 0 0 0 0
0 0 0 0 0 0 A	0 0 0 0 0 0 0
0 0 0 F 9 E D F	0 0 0 F 0 0 0 0
A B C 8 5 F G 1	A B C 8 0 0 0 0

Figure 1: UZE Truth Table

Next, is the Lower Zero Extender Truth Table:

Lower Zero Extender	
In (hex)	Out (hex)
0 0 0 0 0 0 1	0 0 0 0 0 0 1
0 0 0 0 0 0 A	0 0 0 0 0 0 0 A
0 0 0 F 9 E D F	0 0 0 0 9 E D F
A B C 8 5 F G 1	0 0 0 0 5 F G 1

Figure 2: LZE Truth Table

Next, is the Reducer Truth Table:

Reducer	
In (hex)	Out (hex)
0 0 0 0 0 0 1	0 1
0 0 0 0 0 0 A	0 A
0 0 0 F 9 E D F	9 E D F
A B C 8 5 F G 1	5 F G 1

Figure 3: Reducer Truth Table

Finally, the Data-path Truth Table:

Name: Xinyu Hadrian Hu		Student #: 500194233		Section: 01									
INST		CLR_IR LD_IR	LD_PC INC_PC	CLR_A LD_A	CLR_B LD_B	CLR_C LD_C	CLR_Z LD_Z	ALU_OP	EN_WEN	A/B_MUX	REG_MUX	Data_MUX	IM_MUX1 IM_MUX2
LDA	00	00	01	00	00	00	00	XXX	10	0/X	X	1	X
LDB	00	00	00	01	00	00	00	XXX	10	X/0	X	1	X
STA	00	00	00	00	00	00	00	XXX	11	X	0	X	X
JMP	00	00	00	00	00	00	00	XXX	11	X	1	X	X
STAB	00	10	00	00	00	00	00	XXX	X	X	X	X	X
LDAI	00	00	01	00	00	00	00	XXX	X	1/X	X	X	X
LDBI	00	00	00	01	00	00	00	XXX	X	X/1	X	X	X
LUI	00	00	01	10	00	00	00	1	X	0/X	X	10	1/X
ANDI	00	00	01	00	01	01	01	0	X	0/X	X	10	0/01
DECA	00	00	01	00	01	01	01	110	X	0/X	X	10	0/10
AND	00	00	01	00	01	01	01	10	X	0/X	X	10	0/00
ADDI	00	00	01	00	01	01	01	110	X	0/X	X	10	0/00
SUB	00	00	01	00	01	01	01	10	X	0/X	X	10	0/10
INCA	00	00	01	00	01	01	01	0	X	0/X	X	10	0/00
ORI	00	00	01	00	01	01	01	10	X	0/X	X	10	0/01
ROL	00	00	01	00	01	01	01	1	X	0/X	X	10	0/01
ROR	00	00	01	00	01	01	01	100	X	0/X	X	10	0/X
CLRA	00	00	01	00	01	01	01	101	X	0/X	X	10	0/X
CLRC	00	00	10	00	00	00	00	XXX	X	X	X	X	X
CLRZ	00	00	00	10	00	00	00	XXX	X	X	X	X	X
PC <= PC + 4	00	10	00	00	00	00	00	XXX	X	X	X	X	X
IR <= M[INT]	01	00	00	00	00	00	00	XXX	X	X	X	0	X
PC IR [15..0]	00	10	00	00	00	00	00	XXX	X	X	X	X	X

Figure 4: Data-path signals truth table

## Observations and Results

Below are the functional and timing waveforms derived for each of the new components listed for this lab.

The following figures are the observations of this project:

Figure 5: LZE Functional Waveform, Figure 6: LZE Timing Waveform, Figure 7: UZE Functional Waveform, Figure 8: UZE Timing Waveform, Figure 9: Reducer Functional Waveform, Figure 10: Reducer Timing Waveform, Figure 11: Data-path Random Input Waveform, Figure 12: Data-path Output Functional Waveform, and Figure 13: Data-path Output Timing Waveform are the figures of interest.

## LZE: Functional and Timing

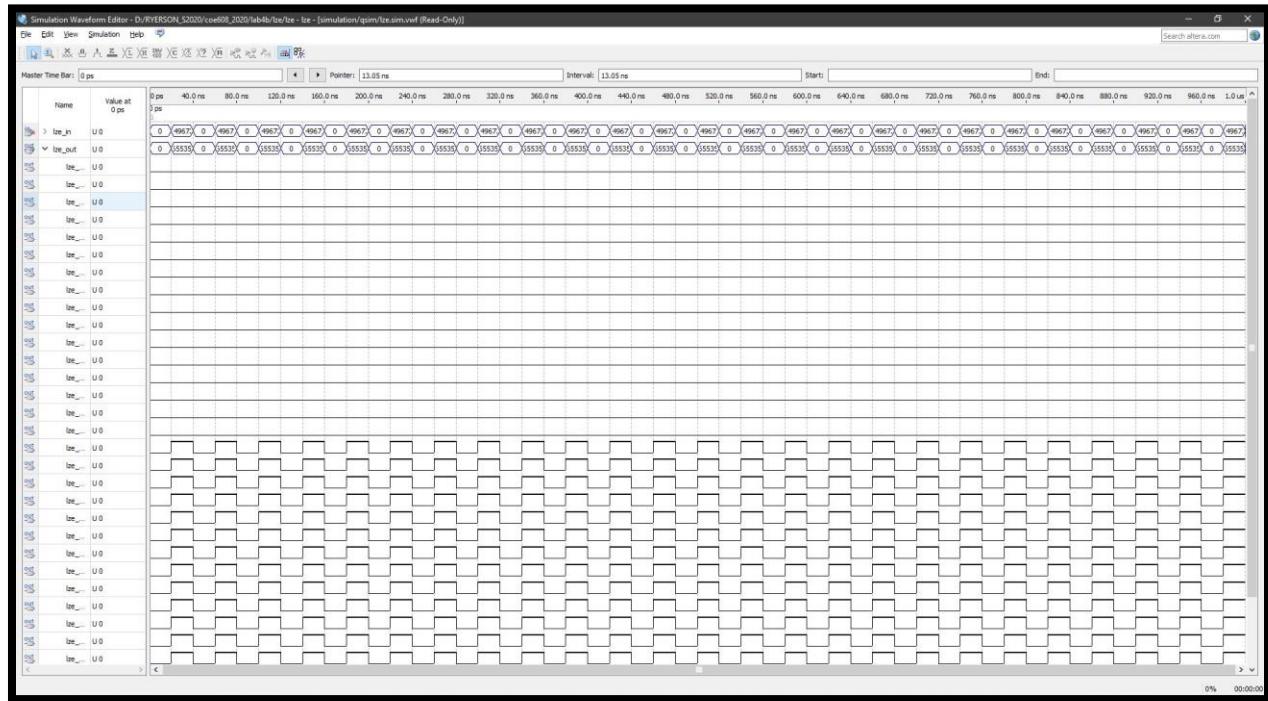


Figure 5: LZE Functional Waveform

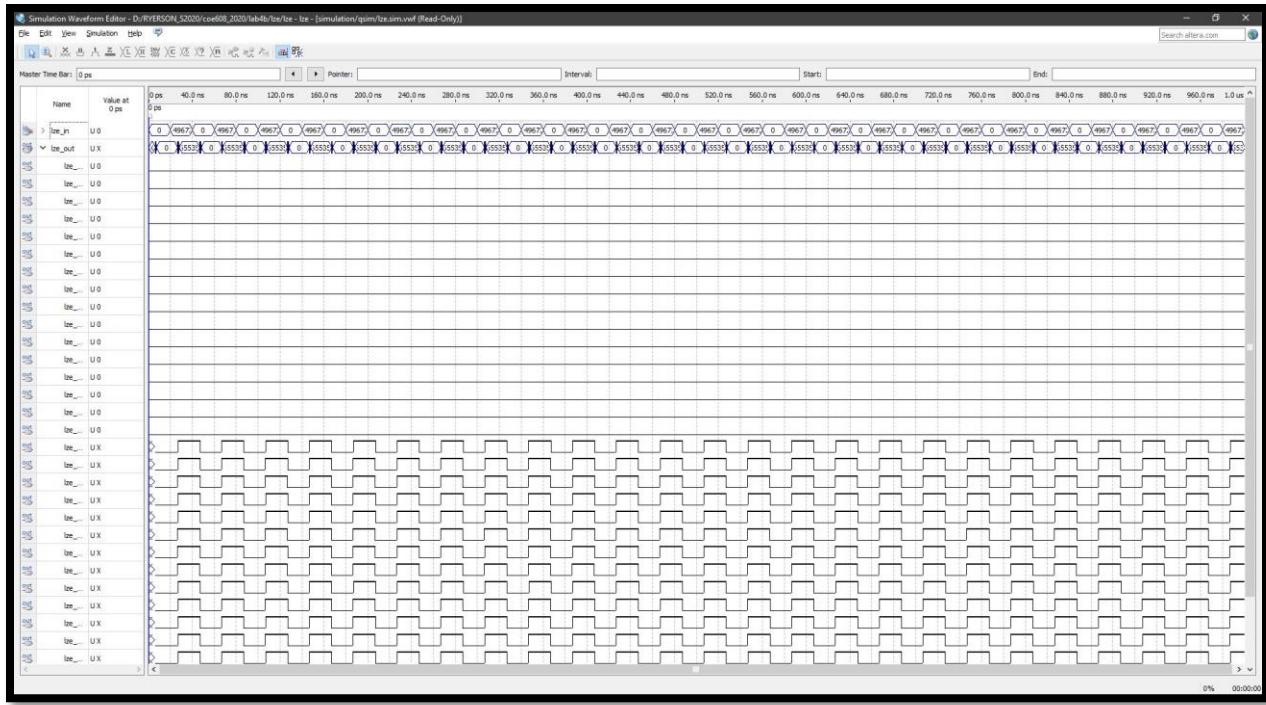


Figure 6: LZE Timing Waveform

## UZE: Functional and Timing

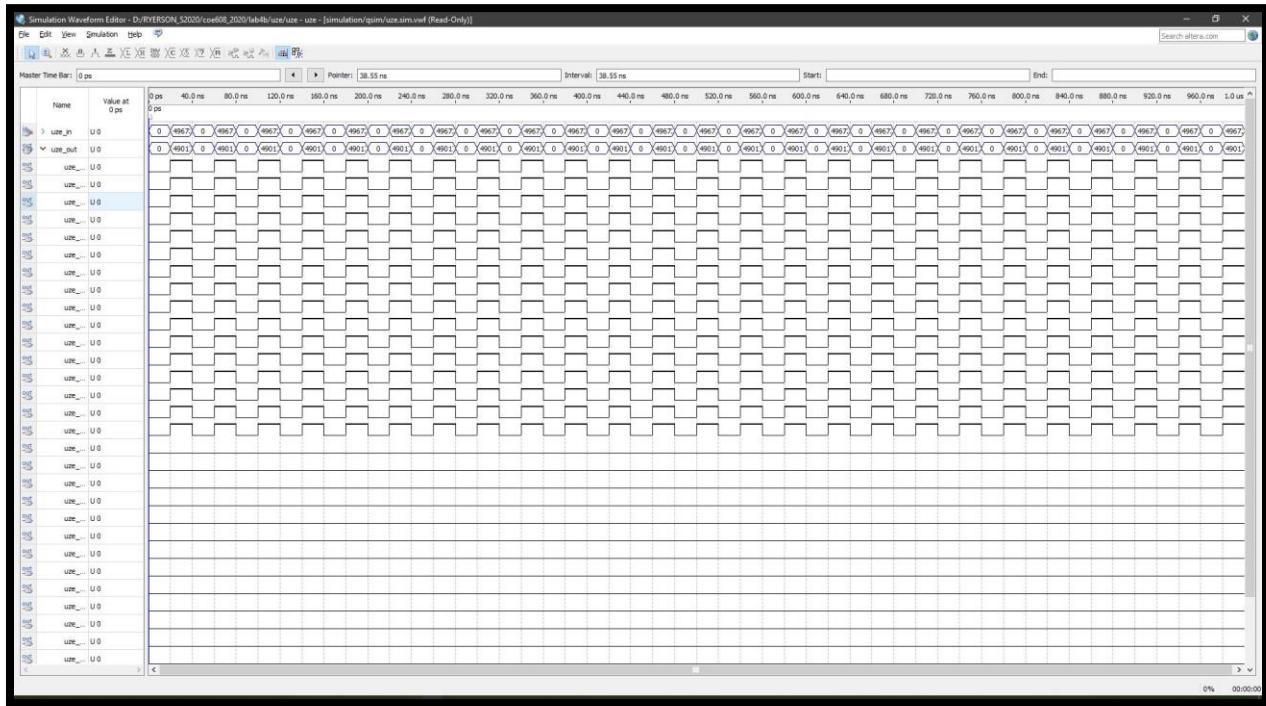


Figure 7: UZE Functional Waveform

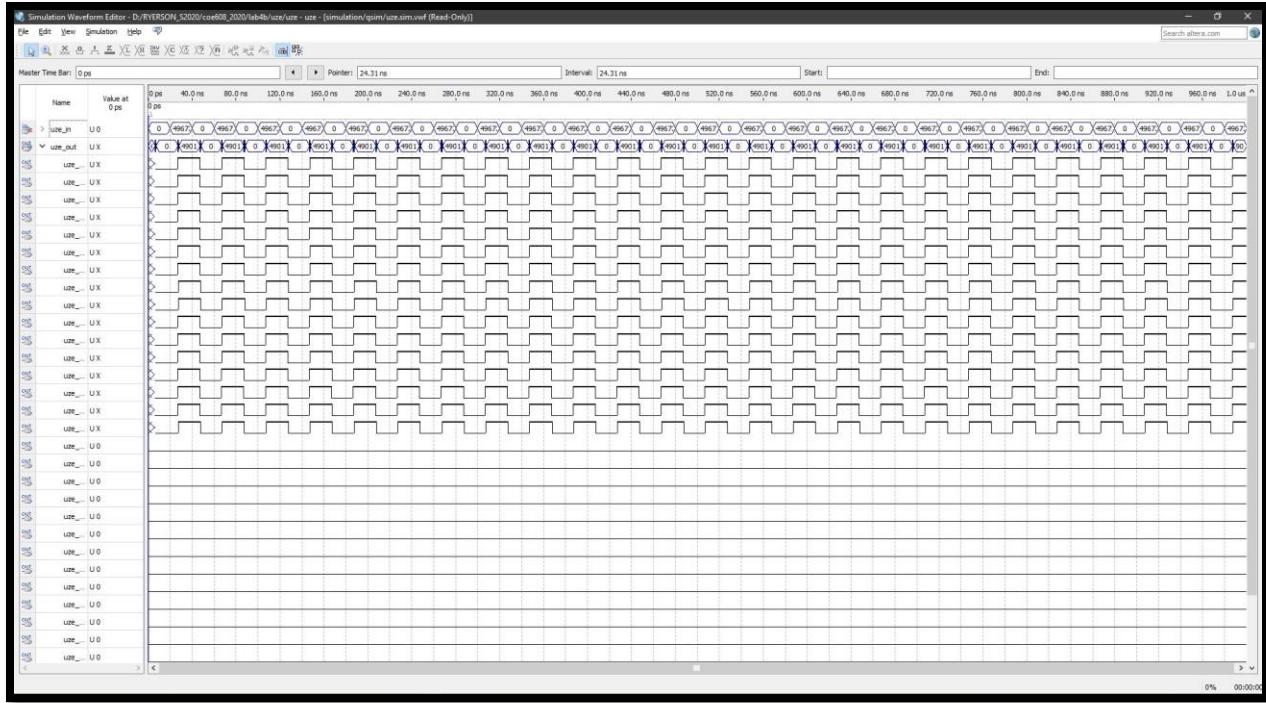


Figure 8: UZE Timing Waveform

## Reducer: Functional and Timing

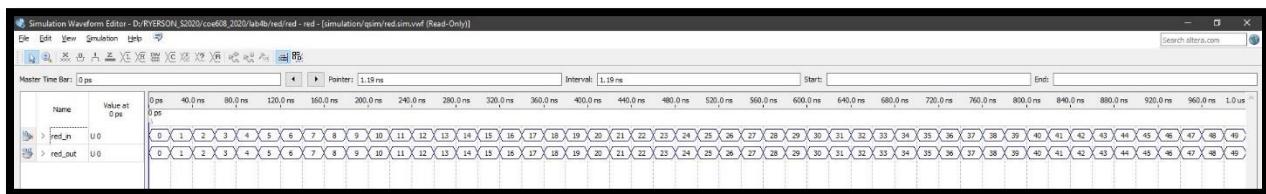


Figure 9: Reducer Functional Waveform



Figure 10: Reducer Timing Waveform

## Data-path: Functional and Timing

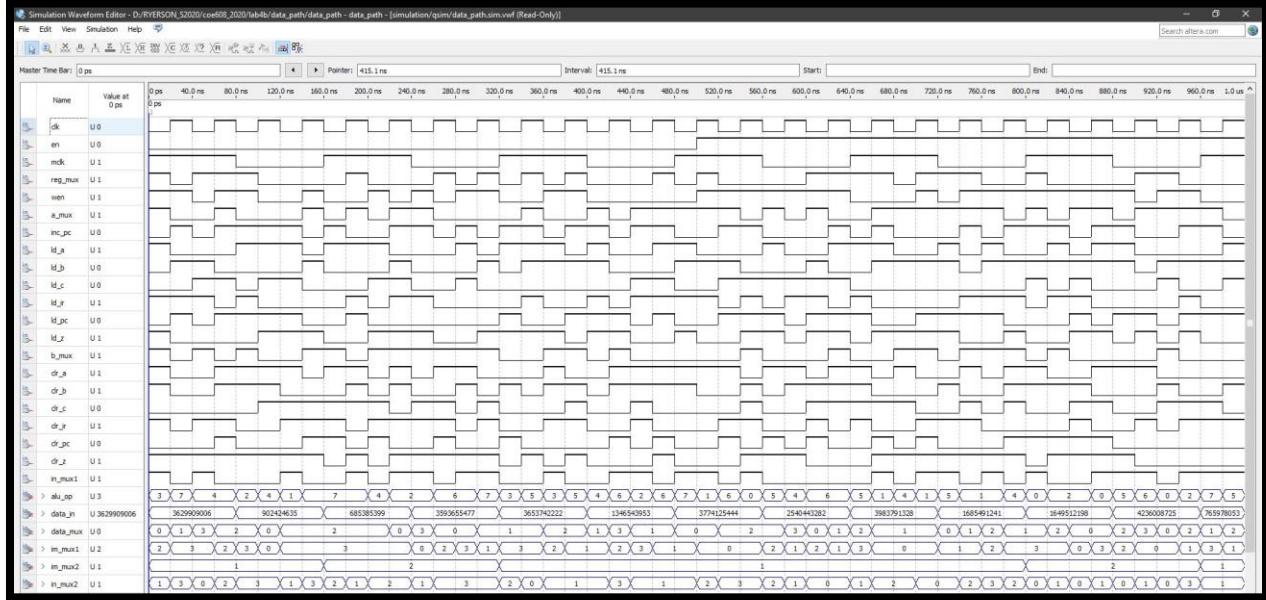


Figure 11: Data-path Random Input Waveform

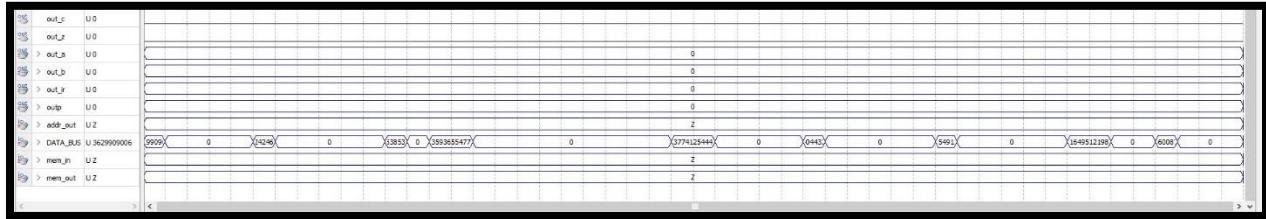


Figure 12: Data-path Output Functional Waveform

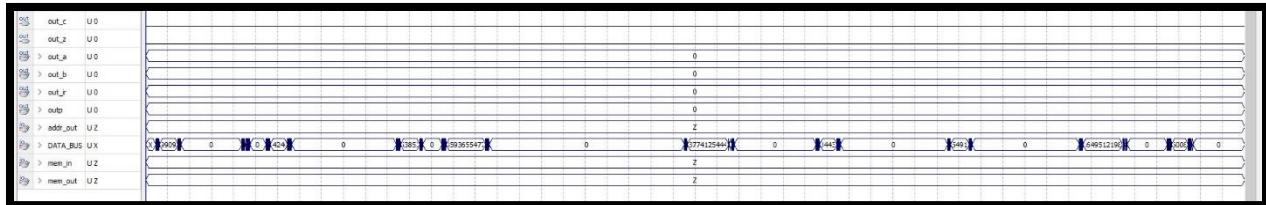


Figure 13: Data-path Output Timing Waveform

For the purposes of understand the operations of MIPS data-path, we have also included the results of specific operations. The random operations waveforms above are not very helpful in demonstrating whether a specific operation works or not.

The following waveform screenshots are courtesy of Duanwei Zhang:

The following figures illustrate the most important operations for the MIPS data-path:

*Figure 14: LDAI Operation Waveform, Figure 15: LDBI Operation Waveform, Figure 16: STA Operation Waveform, Figure 17: STB Operation Waveform, Figure 18: LDA Operation Waveform, Figure 19: LDB Operation Waveform, Figure 20: LUI Operation Waveform, Figure 21: JMP Operation Waveform, Figure 22: ADD Operation Waveform, Figure 23: ADDI Operation Waveform, Figure 24: SUB Operation Waveform, Figure 25: INCA Operation Waveform, Figure 26: ROL Operation Waveform, Figure 27: CLRA Operation Waveform, Figure 28: ANDI Operation Waveform, Figure 29: ORI Operation Waveform, Figure 30: DECA Operation Waveform, and Figure 31: ROR Operation Waveform.*

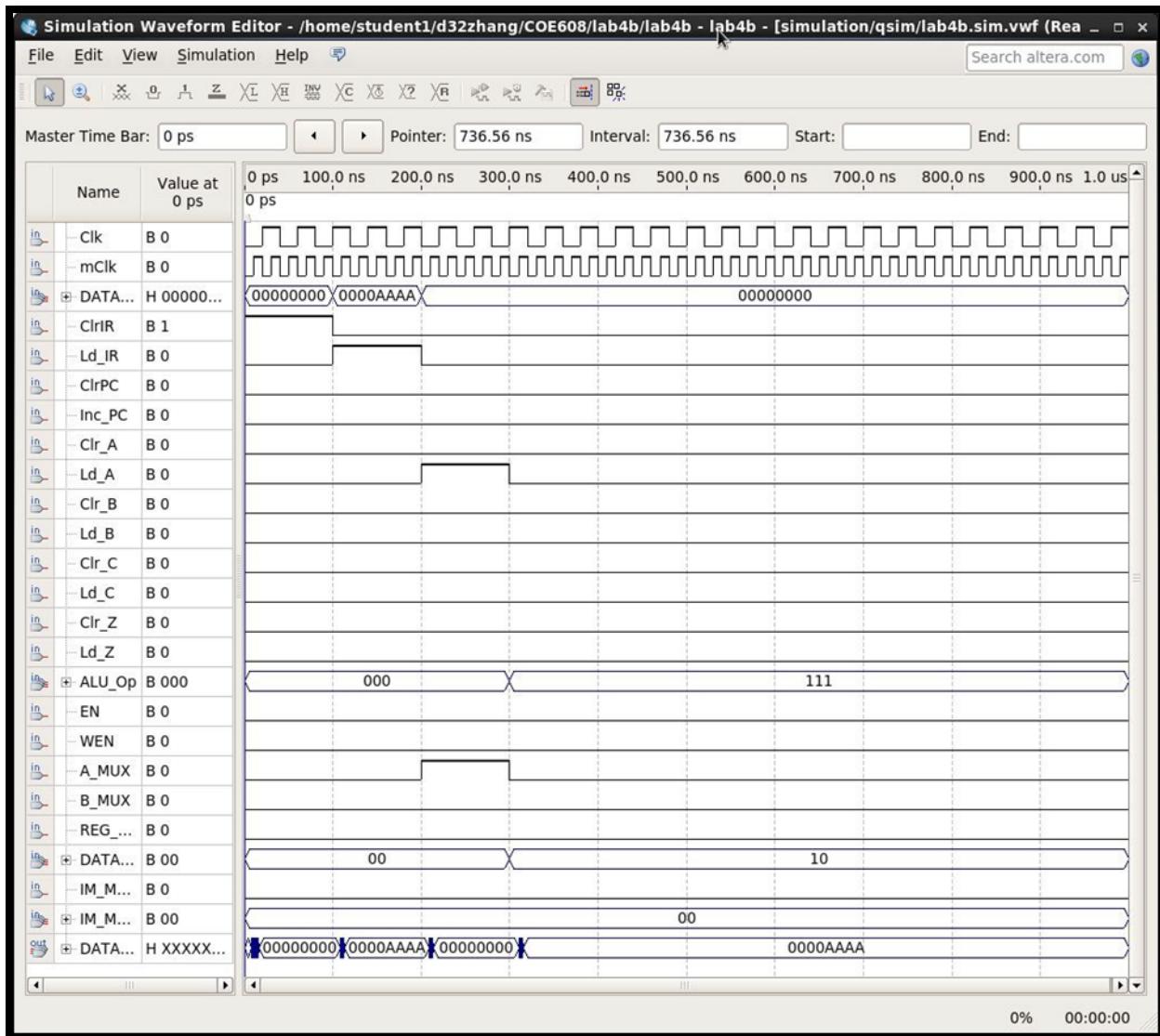


Figure 14: LDAI Operation Waveform

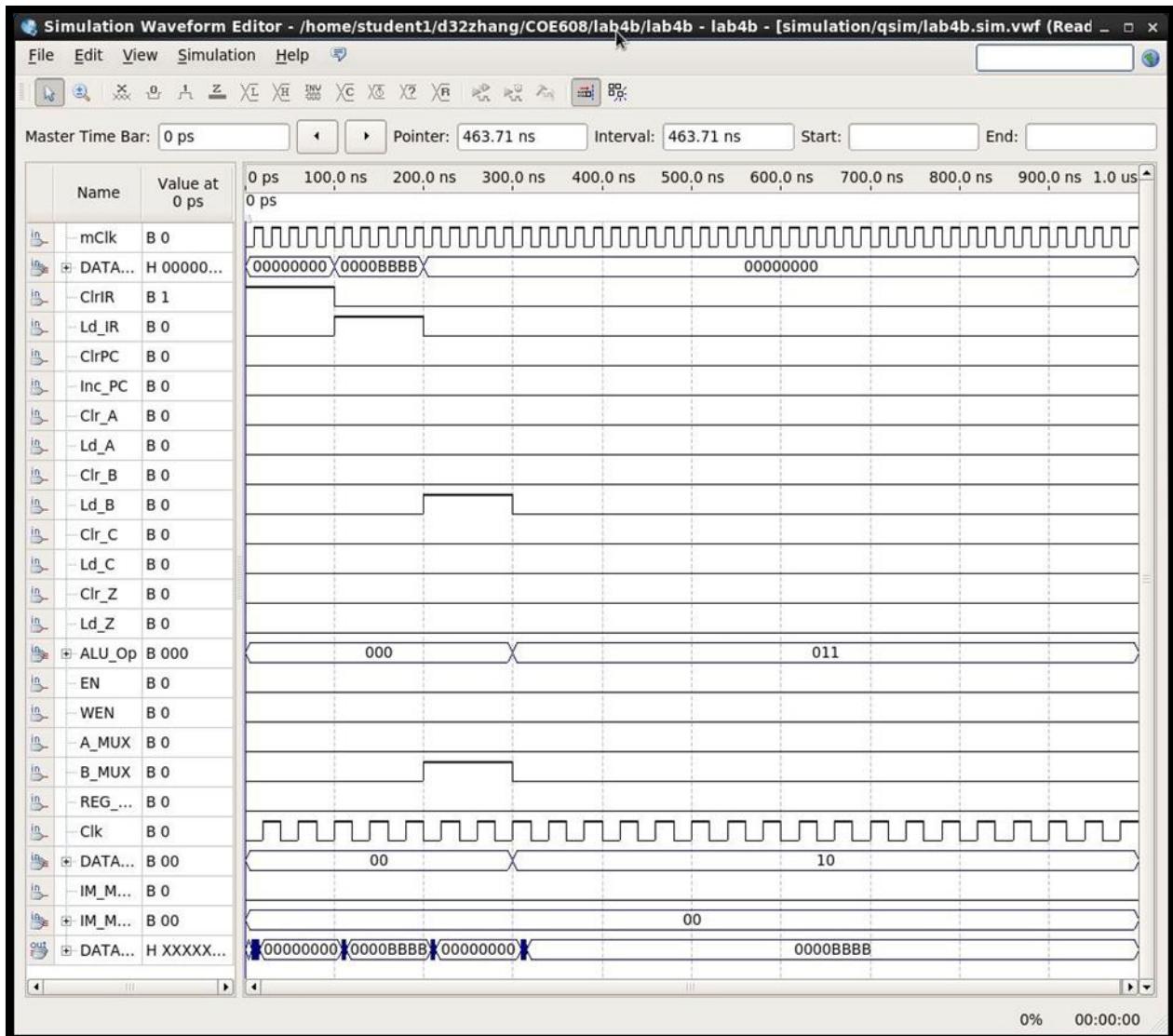


Figure 15: LDBI Operation Waveform

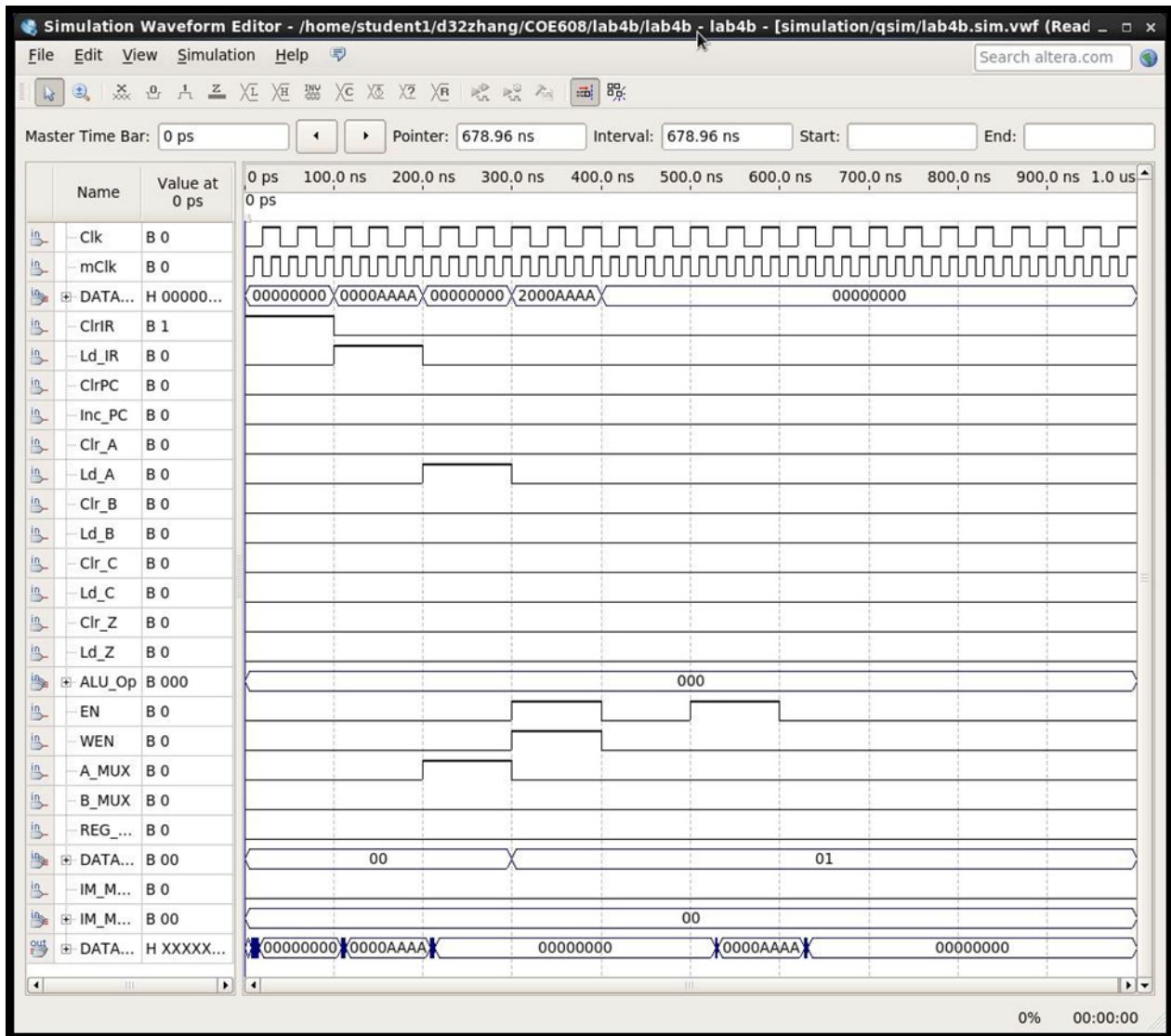


Figure 16: STA Operation Waveform

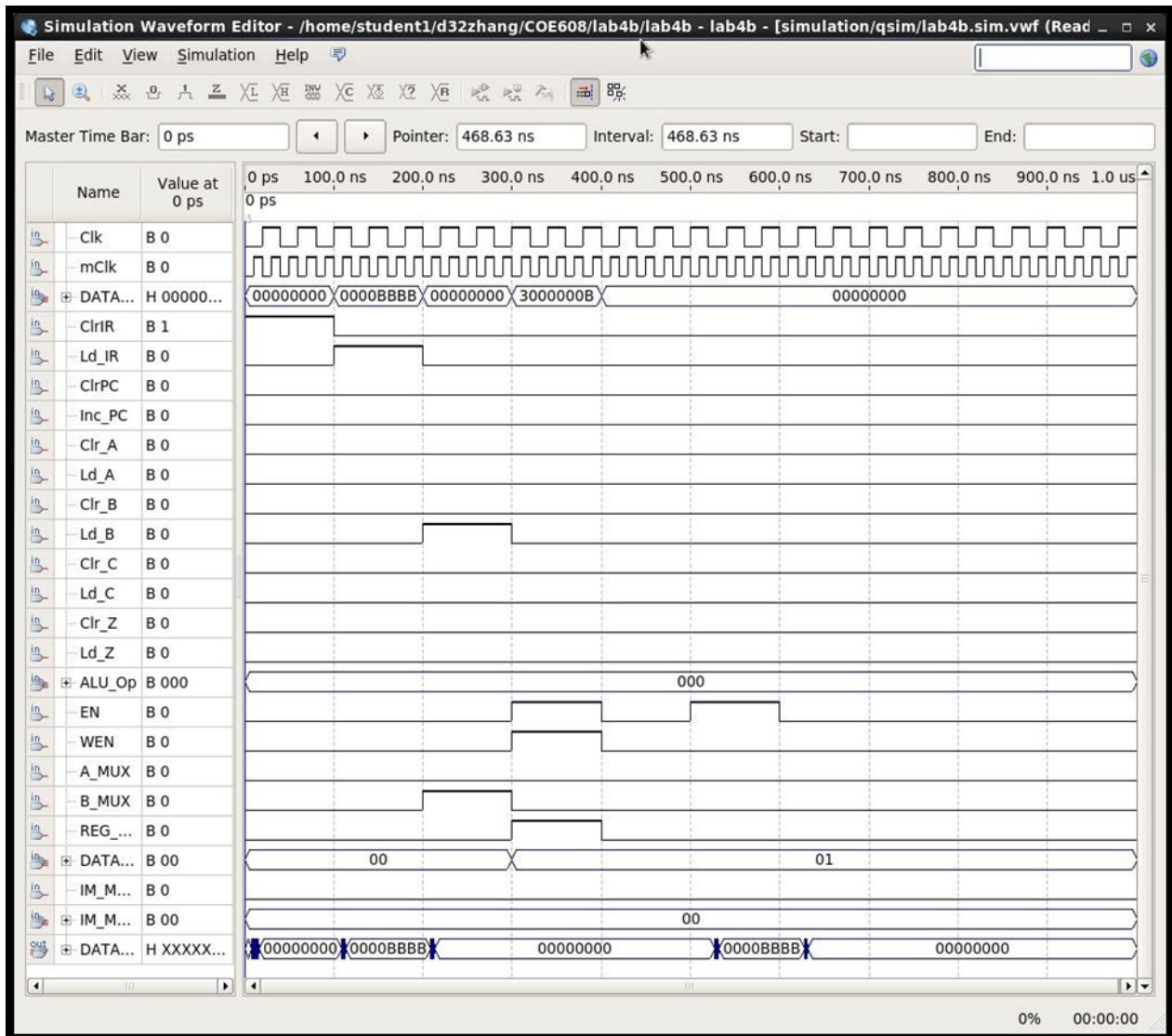


Figure 17: STB Operation Waveform

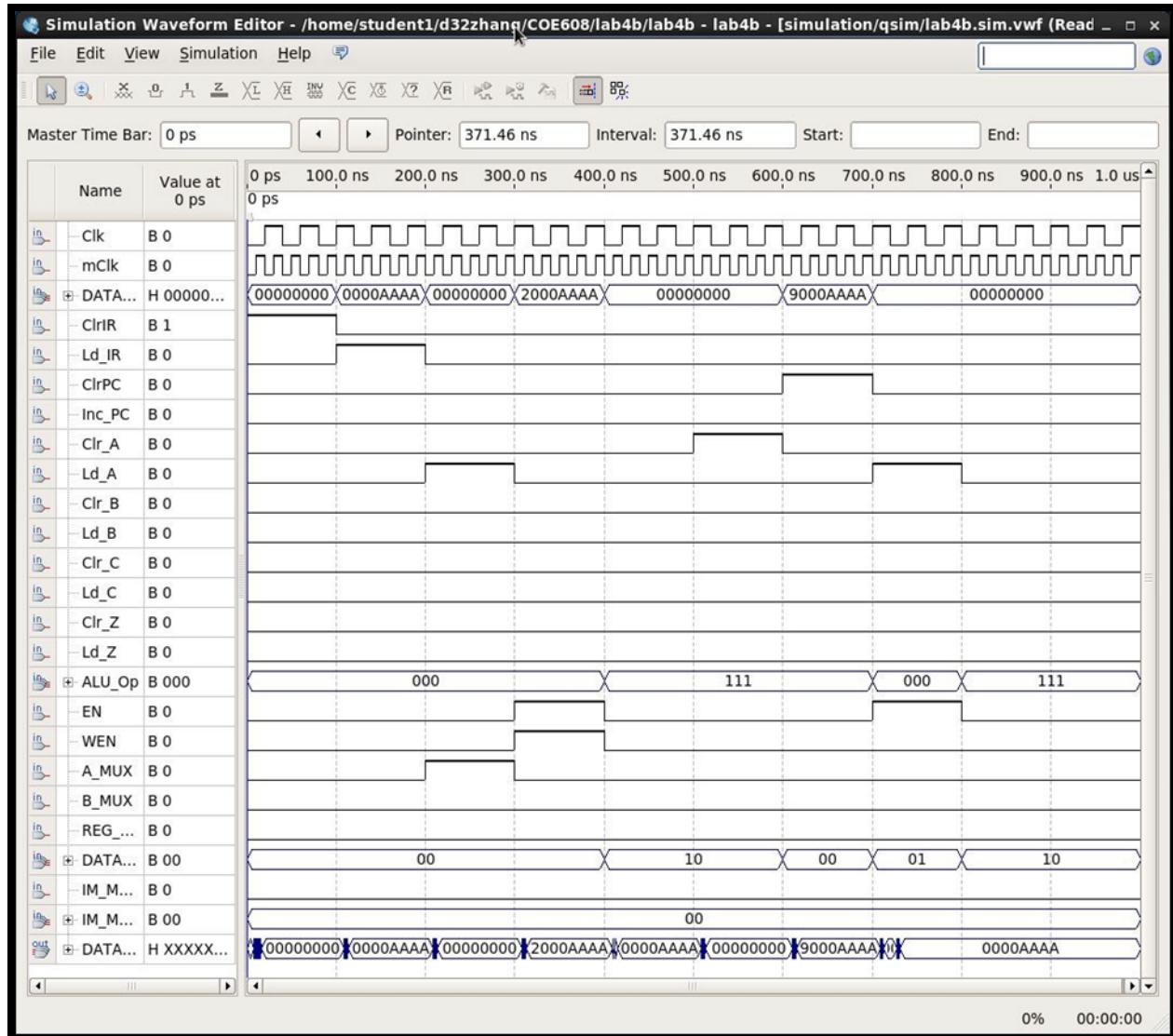


Figure 18: LDA Operation Waveform

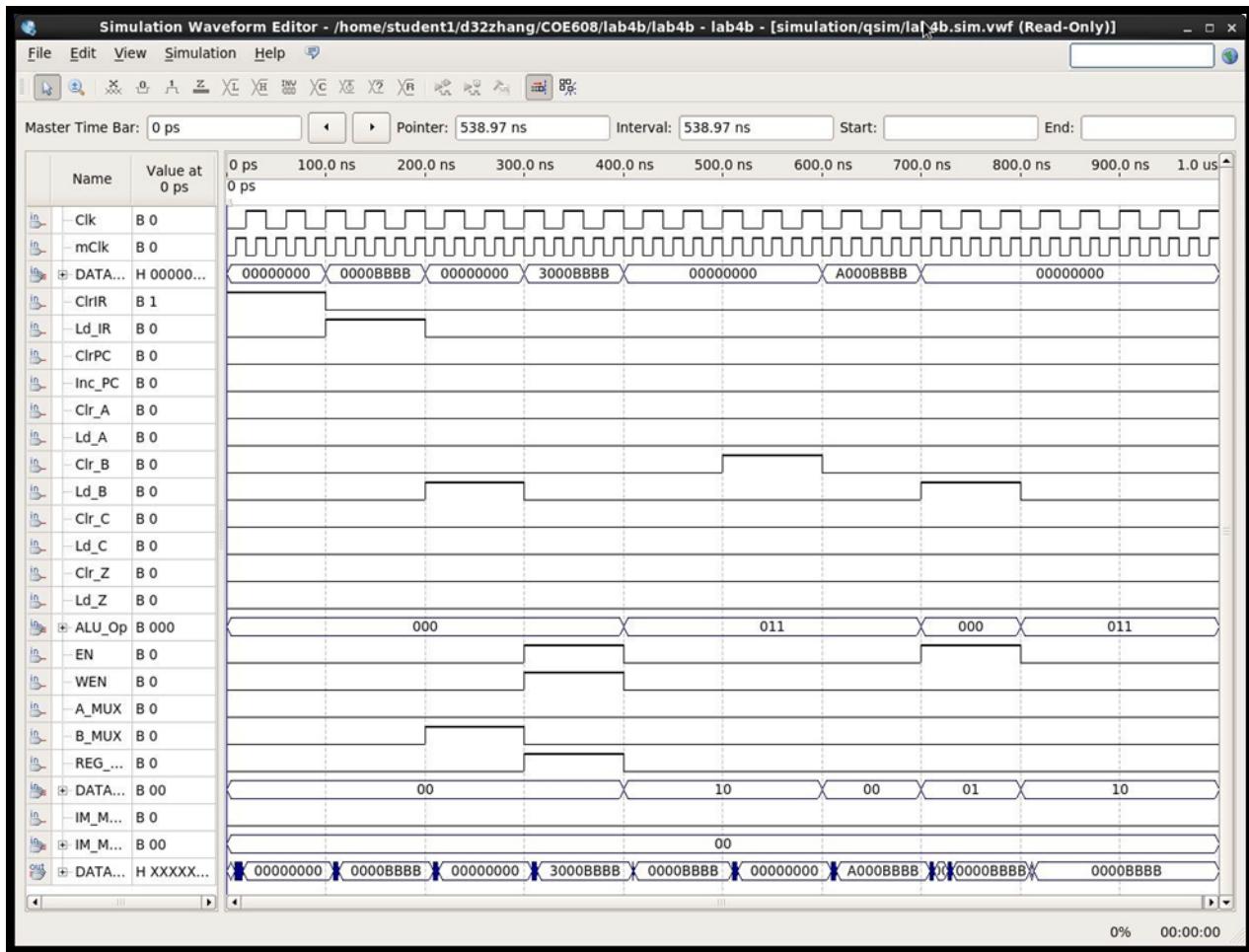


Figure 19: LDB Operation Waveform

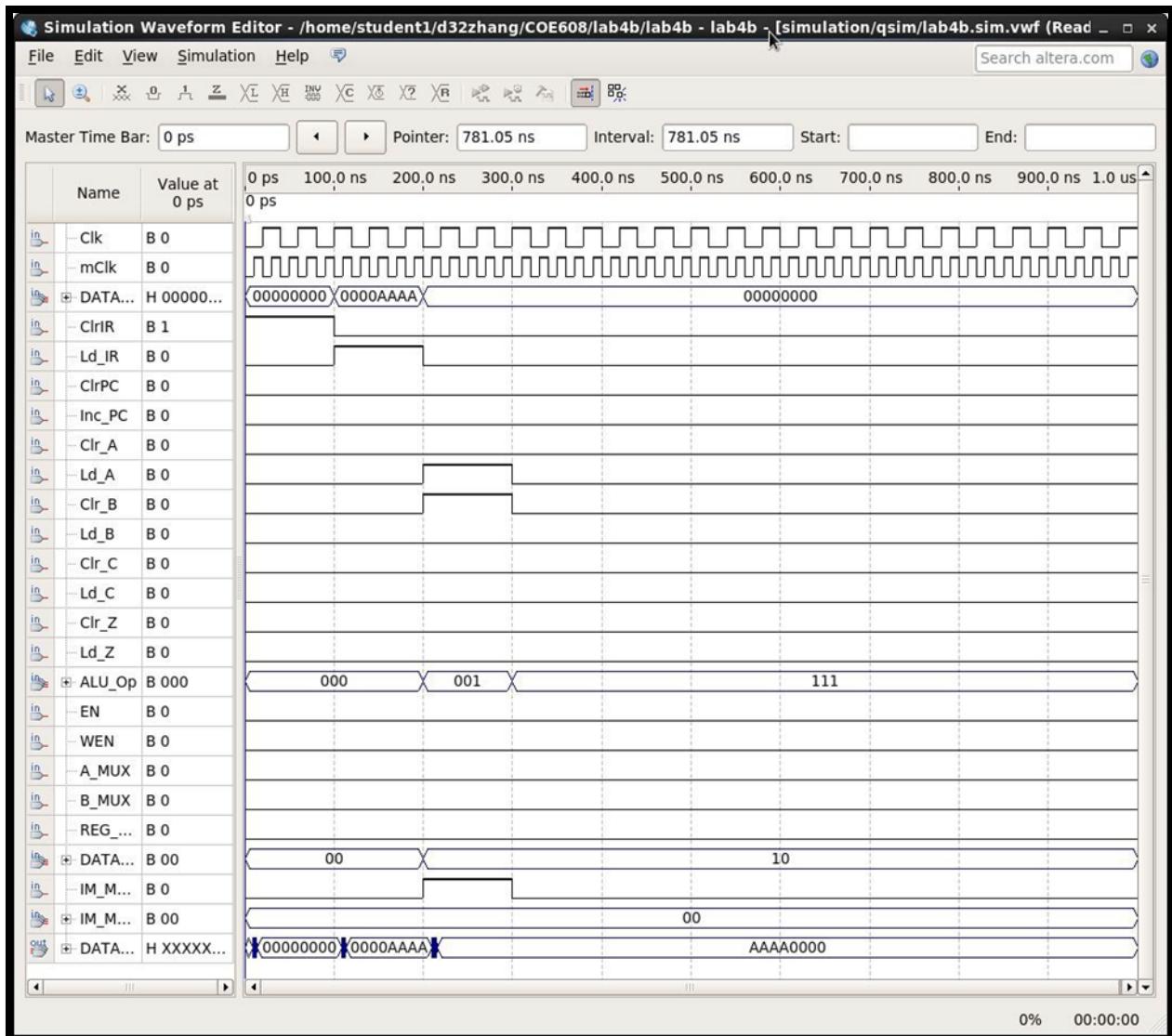


Figure 20: LUI Operation Waveform

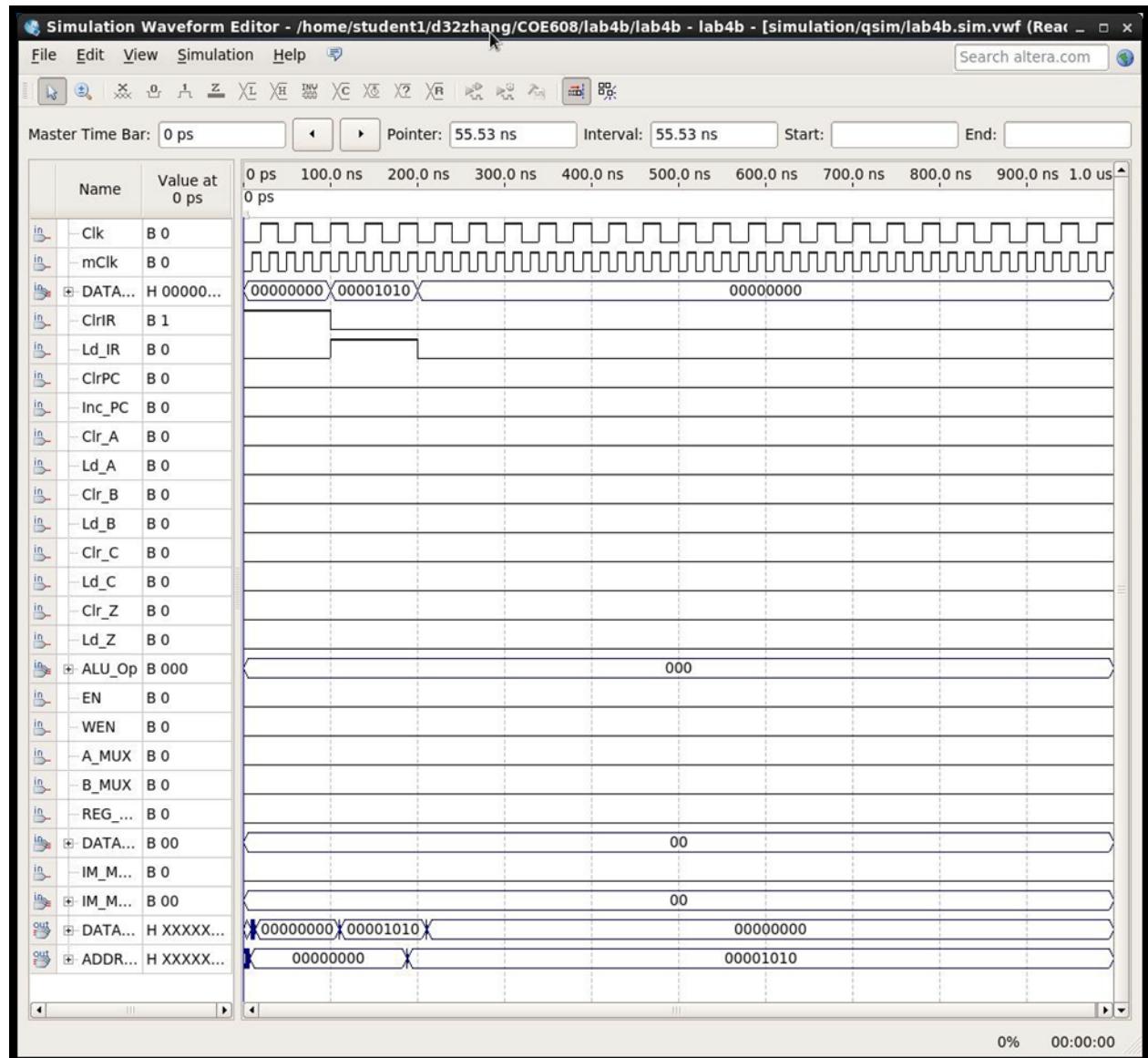


Figure 21: JMP Operation Waveform

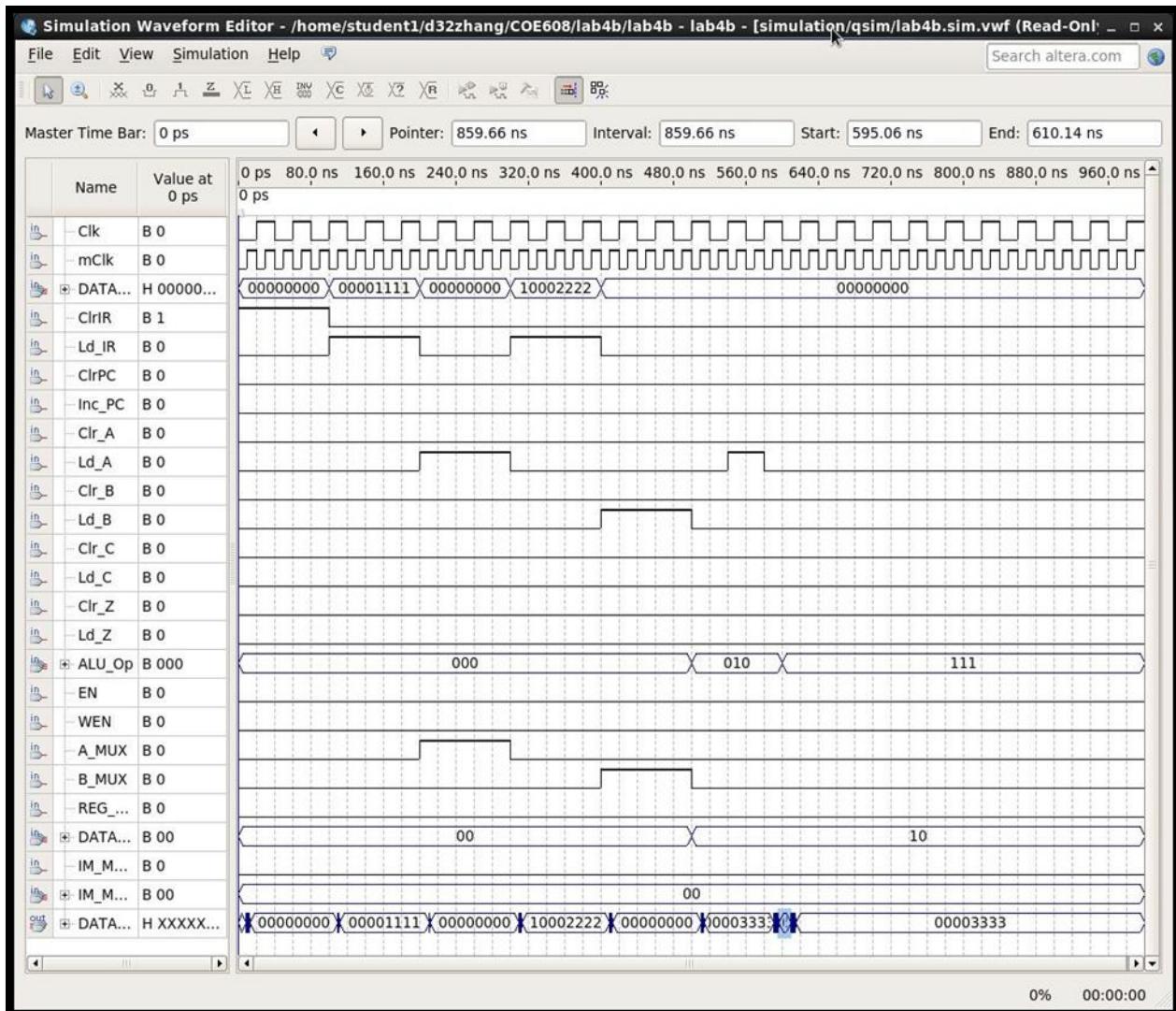


Figure 22: ADD Operation Waveform

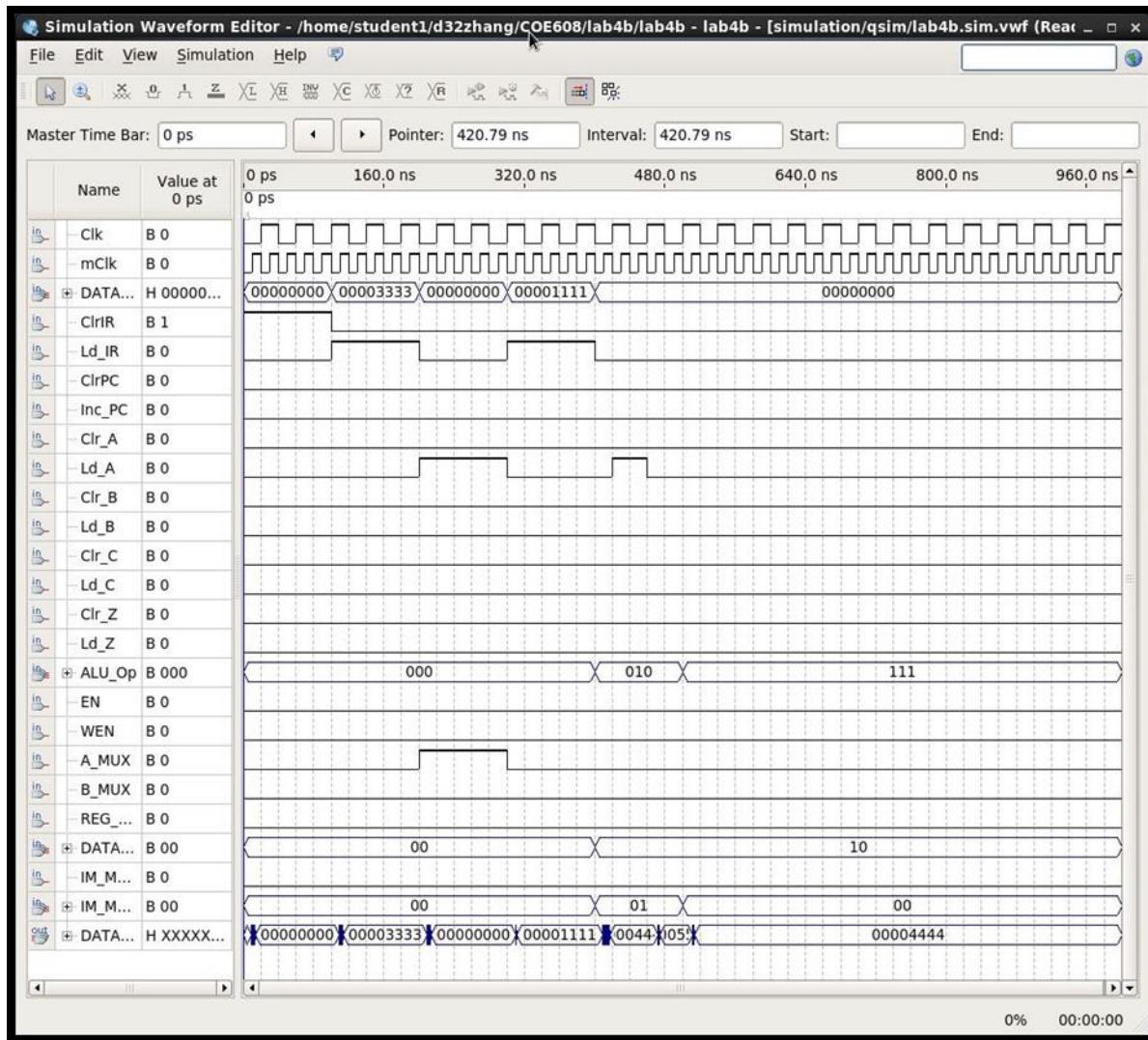


Figure 23: ADDI Operation Waveform

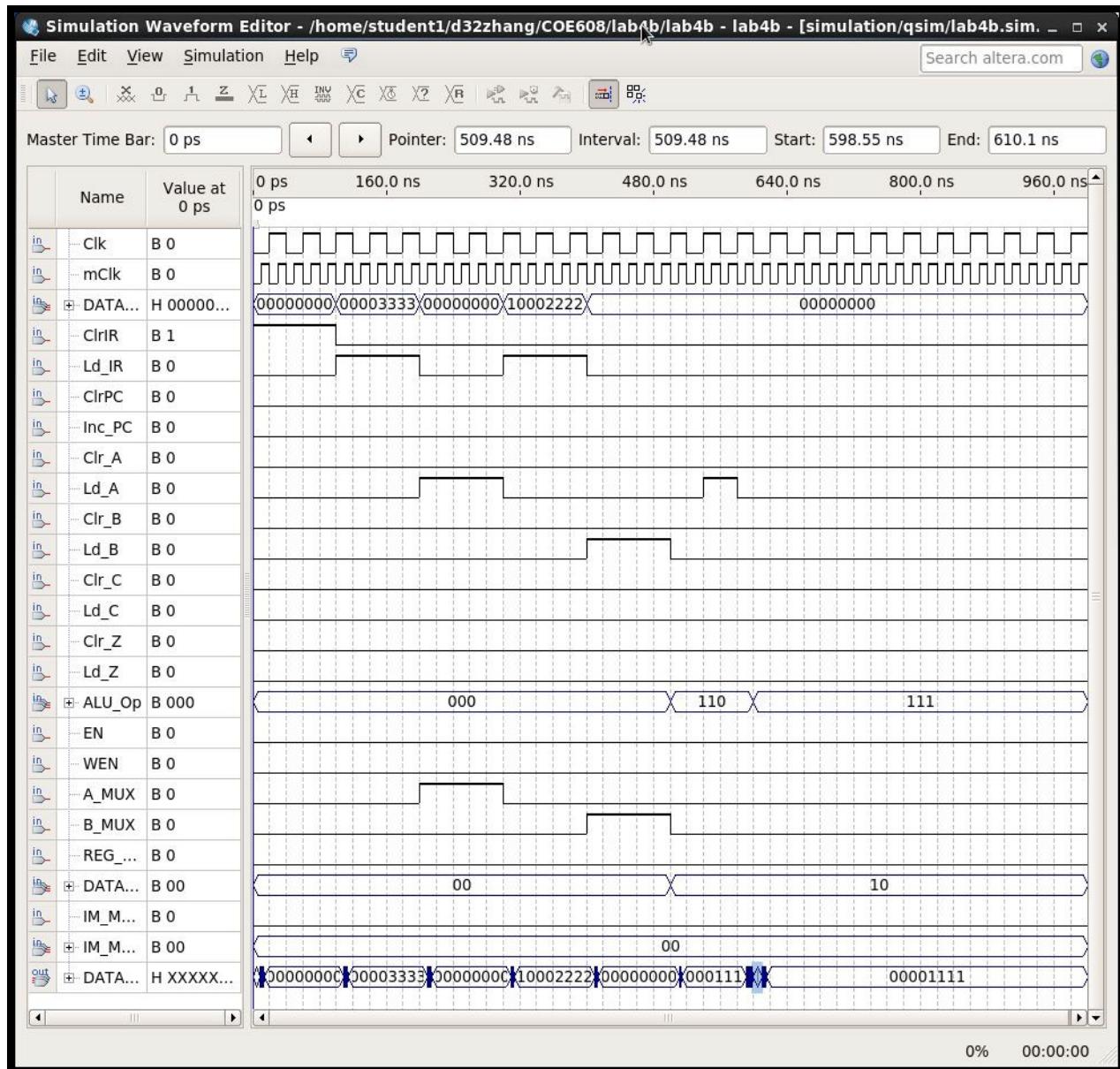


Figure 24: SUB Operation Waveform

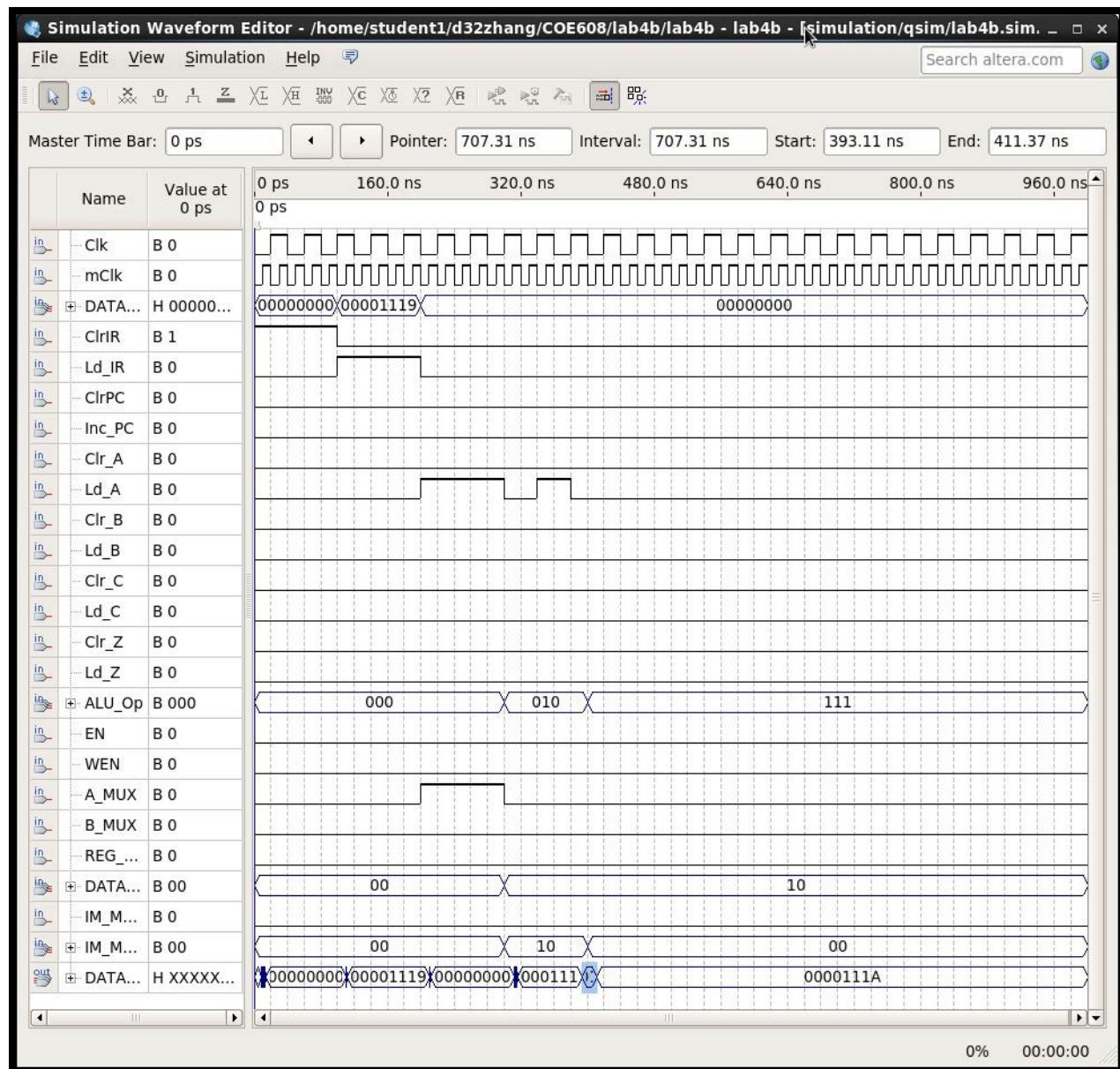


Figure 25: INCA Operation Waveform

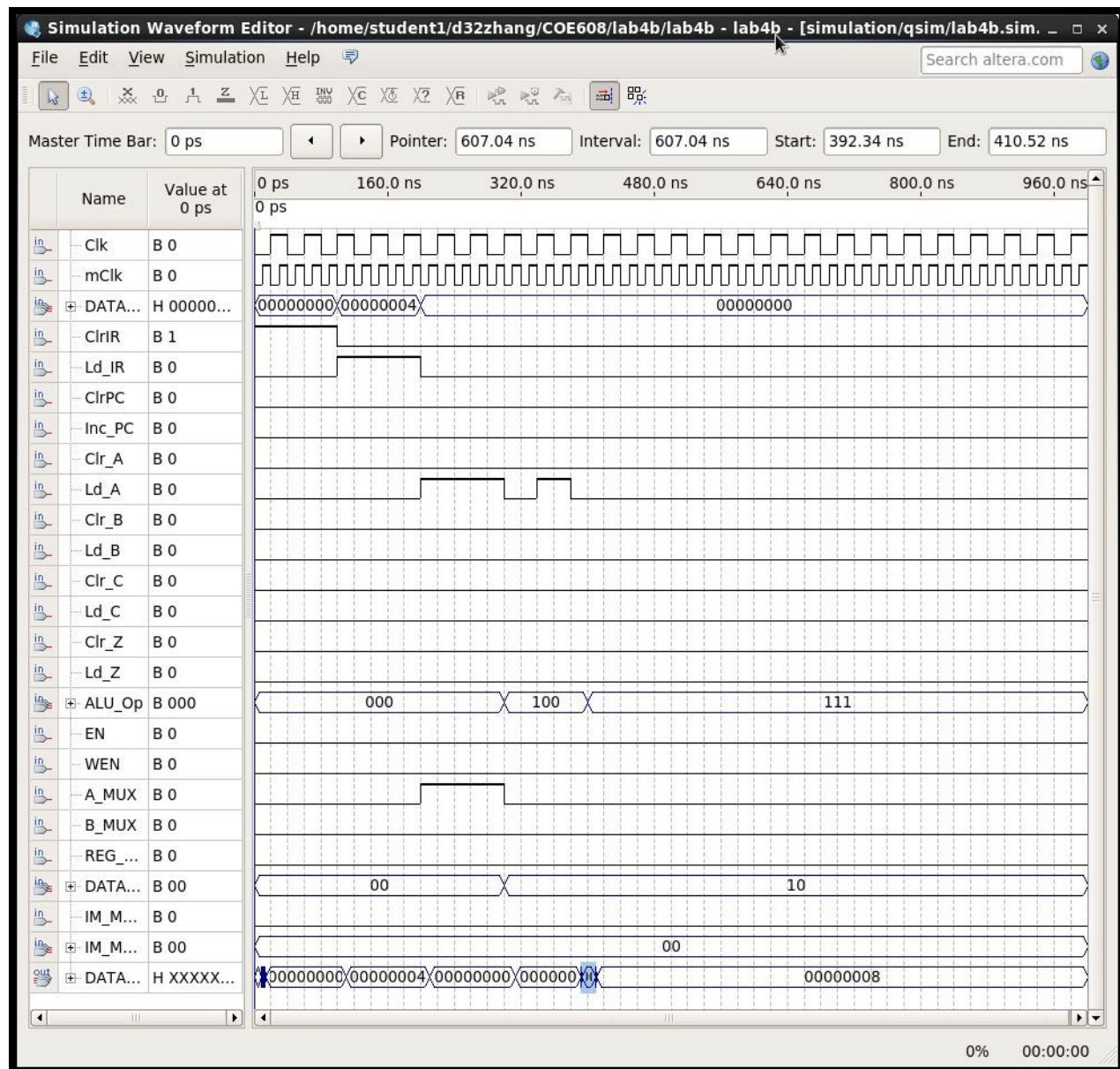


Figure 26: ROL Operation Waveform

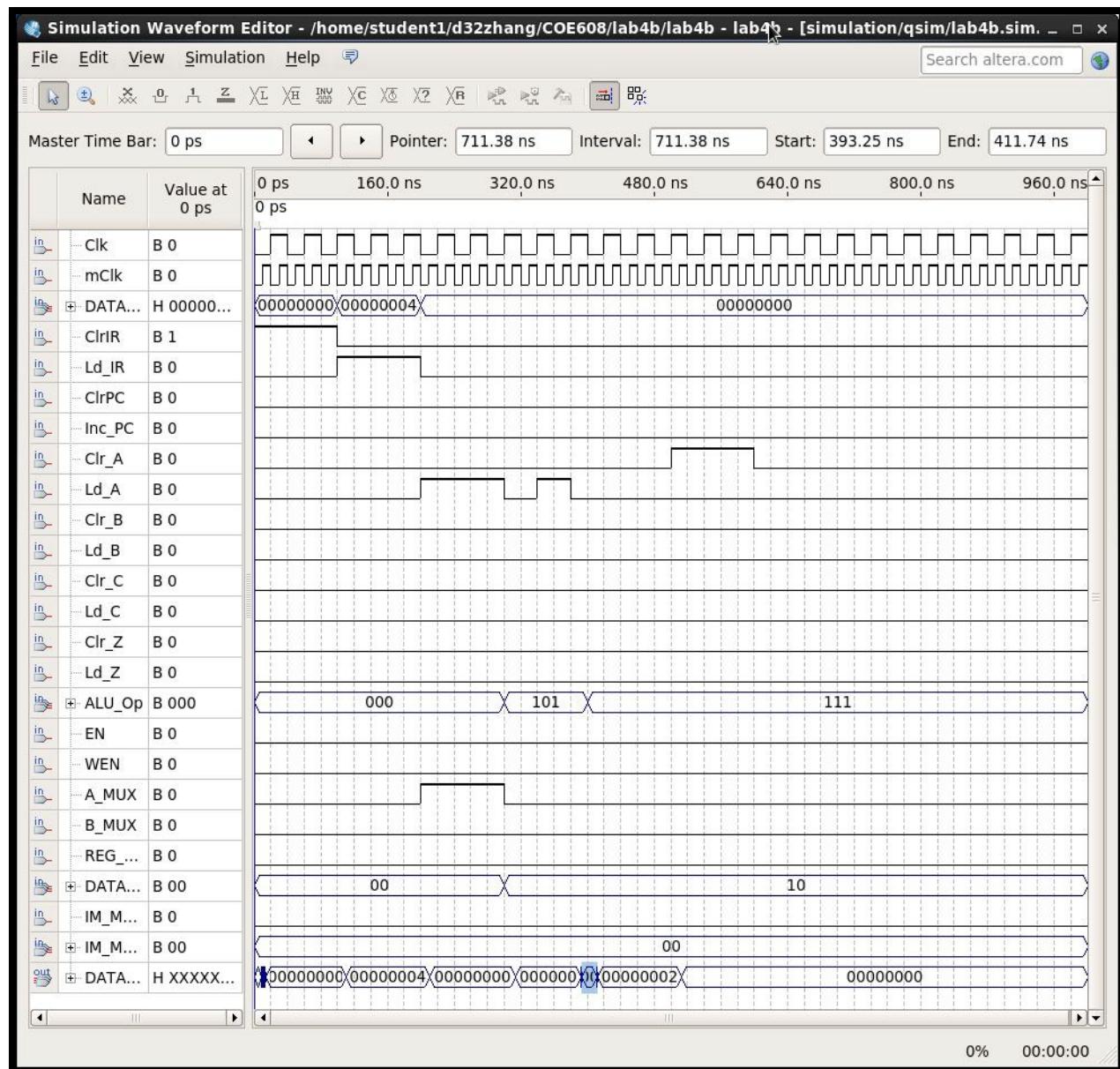


Figure 27: CLRA Operation Waveform

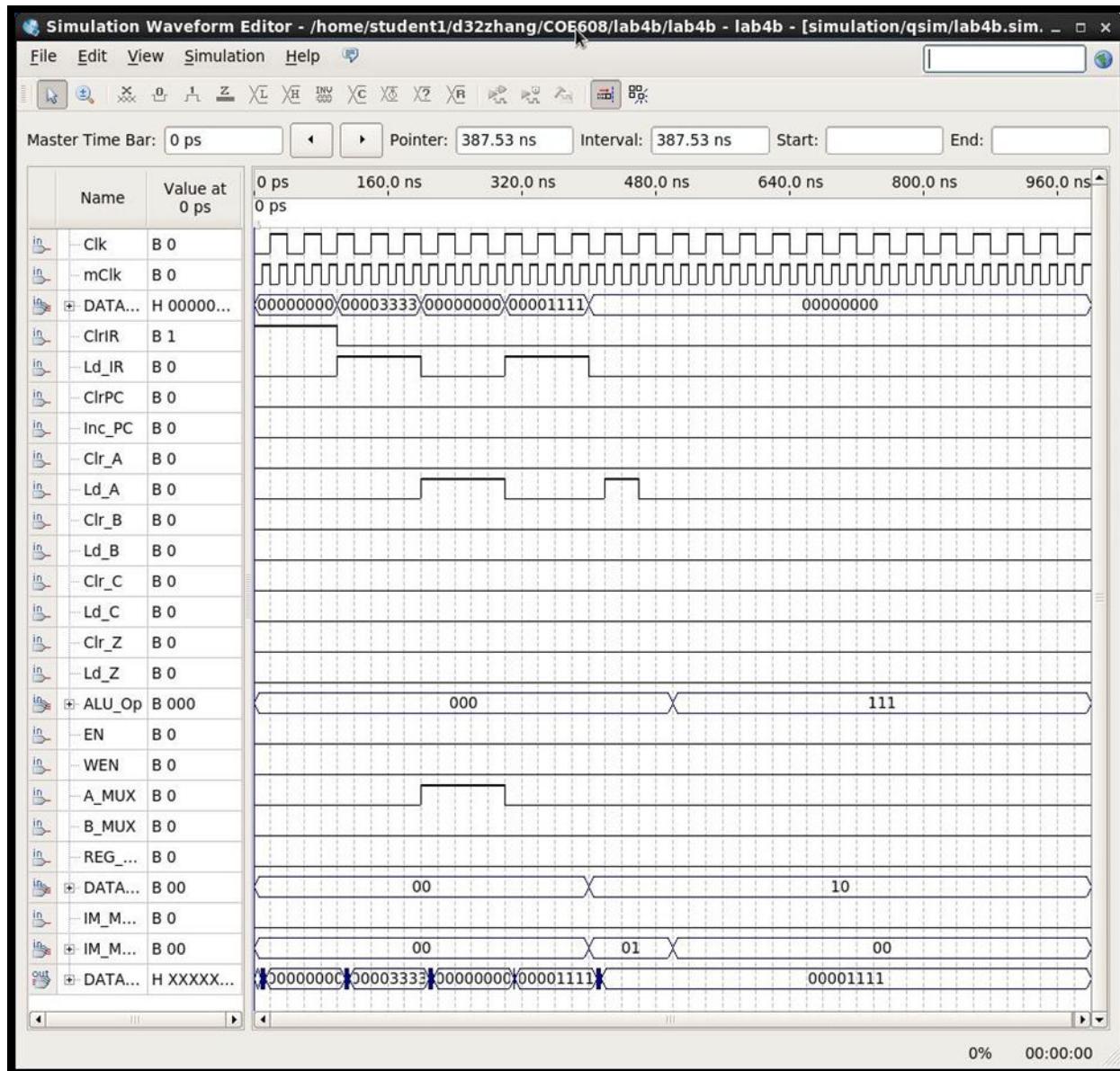


Figure 28: ANDI Operation Waveform

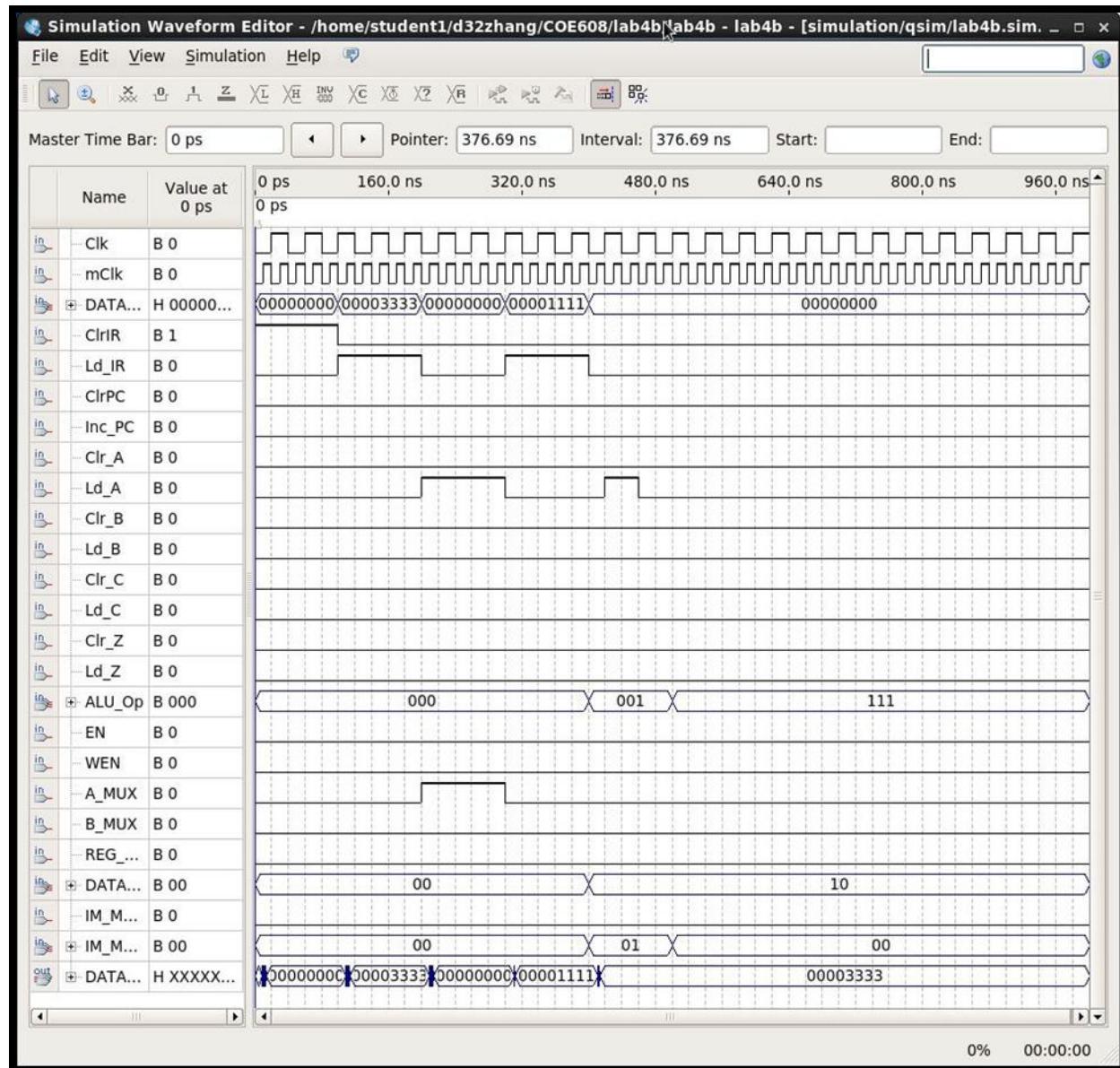


Figure 29: ORI Operation Waveform

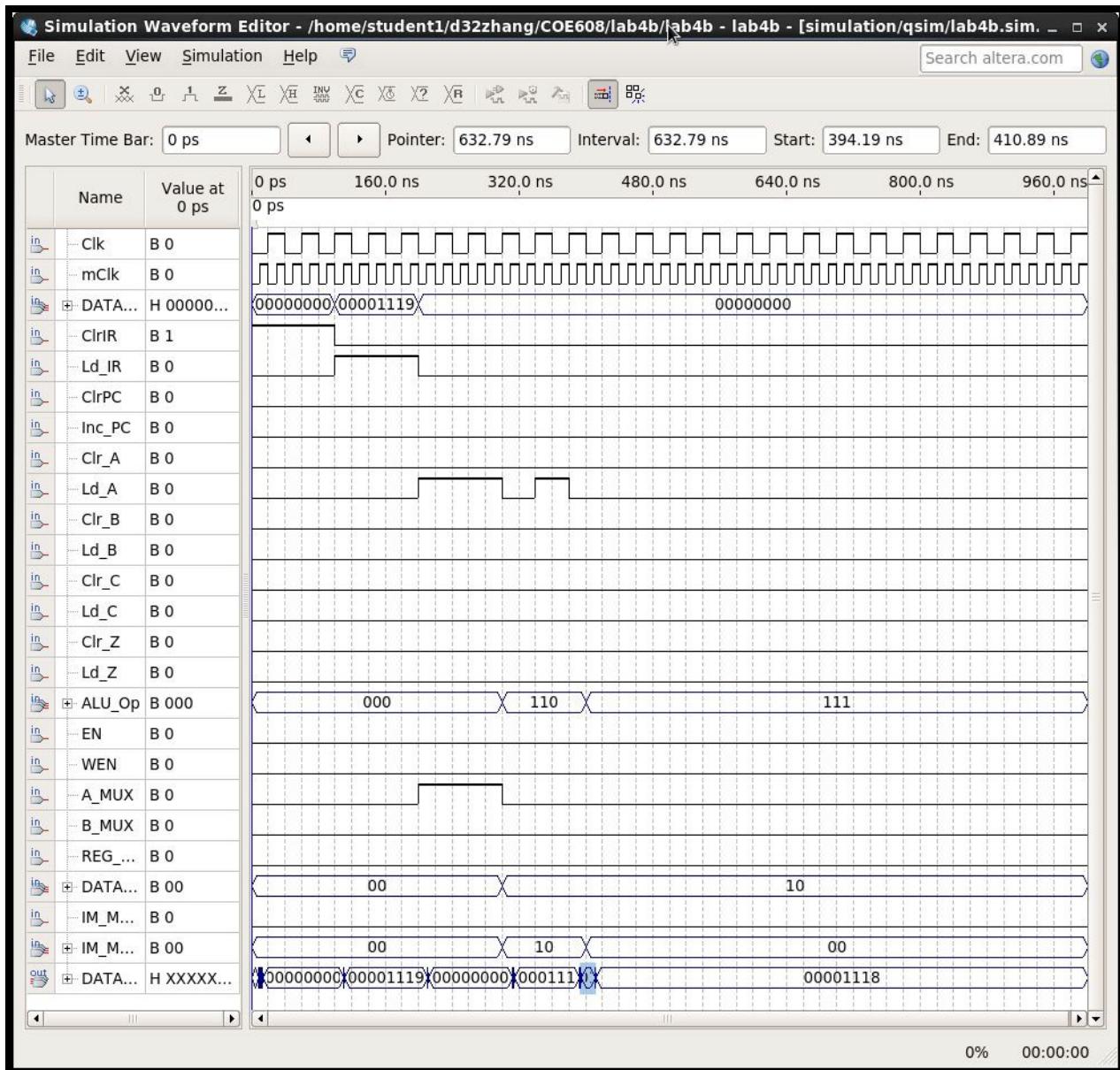


Figure 30: DECA Operation Waveform

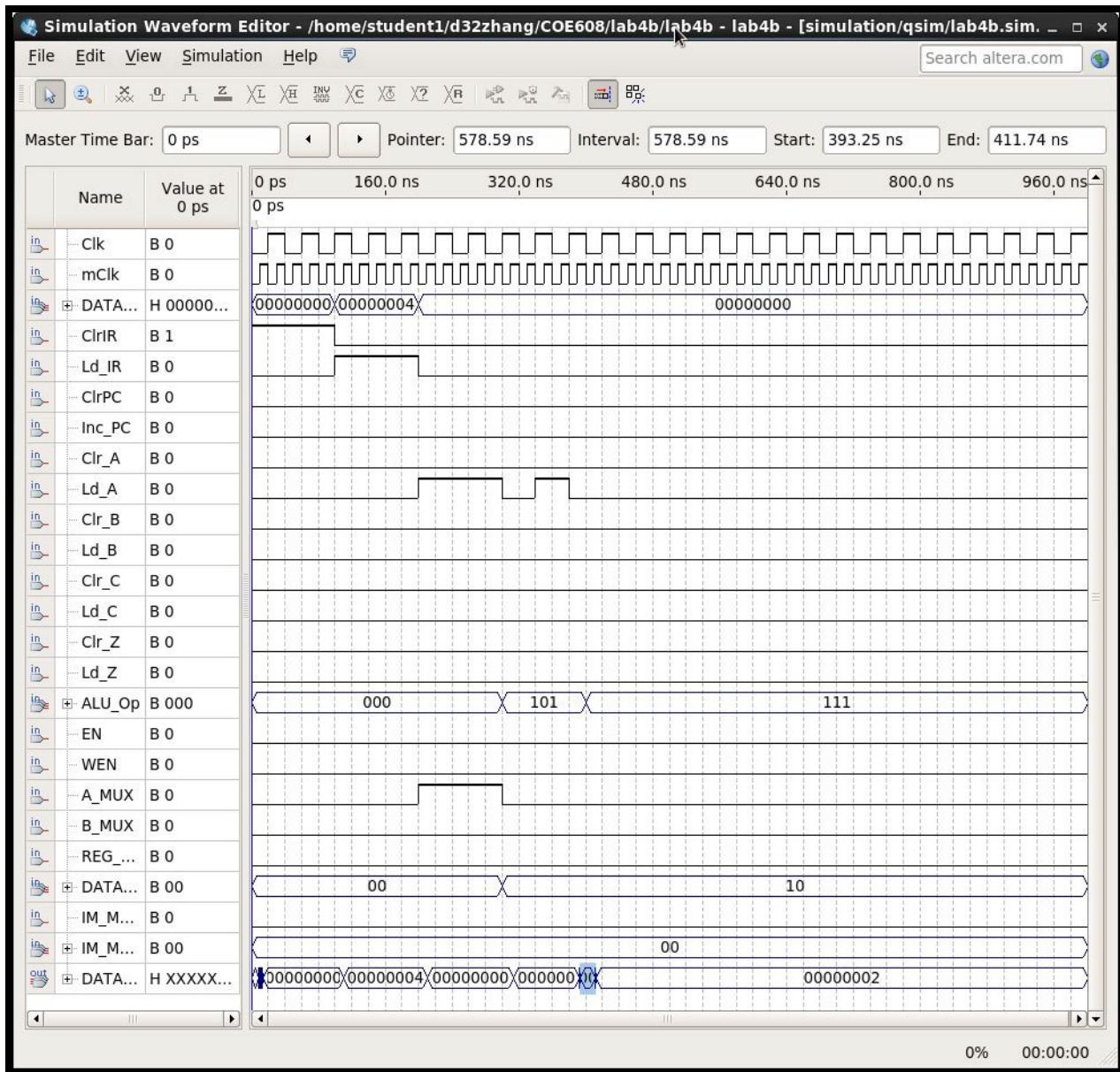


Figure 31: ROR Operation Waveform

## **Discussions and Conclusions**

The data-path implements INCA, ADDI, LDBI and LDA operations by using a finite state machine to select these advanced operations.

The main determinant of the data-path speed is the longest operation performed. In this case, the worst-case timing depends on the operation that is most time-consuming. From Figure 13, it appears the worst-case delay is 10 ns.

The most reliable limit for the data-path clock is 10 ns.

## **Appendix: VHDL Codes and Screenshots of Waveforms**

I have provided the VHDLD and screenshots of the waveforms for easier reading. I also included Duan Wei Zhang's draft report to include the actual data that was captured in each operation waveform listed from Figure 14 to Figure 31.

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3  use ieee.std_logic_arith.all;
4  use ieee.std_logic_unsigned.all;
5
6  entity data_path is
7      port(
8          --clock and memory clock
9          clk, mclk : in std_logic;
10         --memory signals
11         wen, en: in std_logic;
12         --register control signals
13         clr_a, ld_a: in std_logic;
14         clr_b, ld_b: in std_logic;
15         clr_c, ld_c: in std_logic;
16         clr_z, ld_z: in std_logic;
17         clr_pc, ld_pc: in std_logic;
18         clr_ir, ld_ir: in std_logic;
19         --register outputs
20         out_a, out_b: out std_logic_vector(31 downto 0);
21         out_c, out_z: out std_logic;
22         outp, out_ir: out std_logic_vector(31 downto 0);
23         --pc special input
24         inc_pc: in std_logic;
25         --address and data bus signal debug
26         addr_out : inout std_logic_vector(31 downto 0);
27         data_in: in std_logic_vector(31 downto 0);
28         DATA_BUS, mem_out, mem_in: inout std_logic_vector(31 downto 0);
29         in_mux1: in std_logic;
30         in_mux2: in std_logic_vector(1 downto 0);
31         --alu op
32         alu_op: in std_logic_vector(2 downto 0);
33         --mux controllers
34         data_mux : in std_logic_vector(1 downto 0);
35         reg_mux: in std_logic;
36         a_mux, b_mux: in std_logic;
37         im_mux1, im_mux2: in std_logic_vector(1 downto 0));
38     end data_path;
39
40     architecture description of data_path is
41     --components
42
43     component alu
44         port(
45             a, b: in std_logic_vector(31 downto 0);
46             op: in std_logic_vector(2 downto 0);
47             result: inout std_logic_vector(31 downto 0);
48             cout: out std_logic;
49             zero: out std_logic);
50     end component alu;
51
52     component register1
53         port(
54             d,ld,clr,clk: in std_logic;
55             Q: out std_logic);
56     end component register1;
57
58     component register32
59         port(
60             d: in std_logic_vector(31 downto 0);
61             ld,clr,clk: in std_logic;
62             Q: out std_logic_vector(31 downto 0));
```

```

63      end component register32;
64
65  component pc
66    port(
67      clr, clk, ld, inc: in std_logic;
68      d: in std_logic_vector(31 downto 0);
69      q: inout std_logic_vector(31 downto 0));
70  end component pc;
71
72  component memoryModule
73    port (
74      clk: in std_logic;
75      addr: in unsigned (7 downto 0);
76      data_in : in std_logic_vector(31 downto 0);
77      wen, en: in std_logic;
78      data_out: out std_logic_vector(31 downto 0));
79  end component memoryModule;
80
81  component lze
82    port(
83      lze_in: in std_logic_vector(31 downto 0);
84      lze_out: out std_logic_vector(31 downto 0));
85  end component lze;
86
87  component uze
88    port(
89      uze_in : in std_logic_vector(31 downto 0);
90      uze_out: out std_logic_vector(31 downto 0));
91  end component uze;
92
93  component red
94    port(
95      red_in : in std_logic_vector(31 downto 0);
96      red_out : out unsigned(7 downto 0));
97  end component red;
98
99  component mux2to1
100   port (
101     s : in std_logic;
102     w0, w1 : in std_logic_vector(31 downto 0);
103     f: out std_logic_vector(31 downto 0));
104  end component mux2to1;
105
106 component mux4to1
107   port (
108     s : in std_logic_vector(1 downto 0);
109     x0, x1, x2, x3 : in std_logic_vector(31 downto 0);
110     f : out std_logic_vector(31 downto 0));
111  end component mux4to1;
112
113  --signals
114
115  signal IR_out,data_bus1,lze_pc_out,lze_Amux_out,lze_Bmux_out,Amux_out, Bmux_out,
registerA_out, registerB_out, registerMux_out, memory_out : std_logic_vector(31 downto 0);
116  signal uze_imMUX1_out, imMUX1_out, lze_imMUX2_out, imMUX2_out, alu_out : std_logic_vector
(31 downto 0);
117  signal red_DataMem_out: unsigned (7 downto 0);
118  signal zero_flag, carry_flag : std_logic;
119  signal temp : std_logic_vector(30 downto 0) := (others => '0');
120  begin
121    ir_register : register32 port map (data_bus, ld_ir, clr_ir, clk, IR_out);
122    a_register: register32 port map (Amux_out, ld_a, clr_a, clk, registerA_out);

```

```
123      b_register: register32 port map(Bmux_out, ld_b, clr_b, clk, registerB_out);
124      alu_component: alu port map(imMuX1_out, imMUX2_out, alu_op, alu_out, carry_flag,
125      zero_flag);
126      memory_component: memoryModule port map (mclk, RED_DataMem_out, registerMux_out,
127      wen, en, memory_out);
128      DATA_BUS <= data_bus1;
129      lze_pc : lze port map (IR_out, lze_pc_out);
130      lze_A_mux : lze port map (IR_out, lze_Amux_out);
131      A_mux0 : mux2to1 port map (a_mux, data_bus, lze_Amux_out, Amux_out);
132      lze_B_mux : lze port map (IR_out, lze_Bmux_out);
133      B_mux0: mux2to1 port map (b_mux, data_bus, lze_Bmux_out, Bmux_out);
134      reg_mux0: mux2to1 port map (reg_mux, registerA_out, registerB_out, registerMux_out
135 );
136      RED_DATA_MEM: red port map (IR_out, red_DataMem_out);
137      UZE_IM_MUX1: uze port map (IR_out, UZE_imMUX1_out);
138      IM_MUX1a : mux2to1 port map (in_mux1,registerA_out, UZE_imMUX1_out, imMUX1_out);
139      LZE_IM_MUX2 : lze port map (IR_out, LZE_imMUX2_out);
140      IM_MUX2a : mux4to1 port map (in_mux2, registerB_out, LZE_imMUX2_out, (temp & '1'),
141      (others => '0'), imMUX2_out);
142      DATA_MUX0: mux4to1 port map (data_mux, data_in, memory_out, alu_out, (others => '0'
143 ), data_bus1);
144      end description;
145
146
```

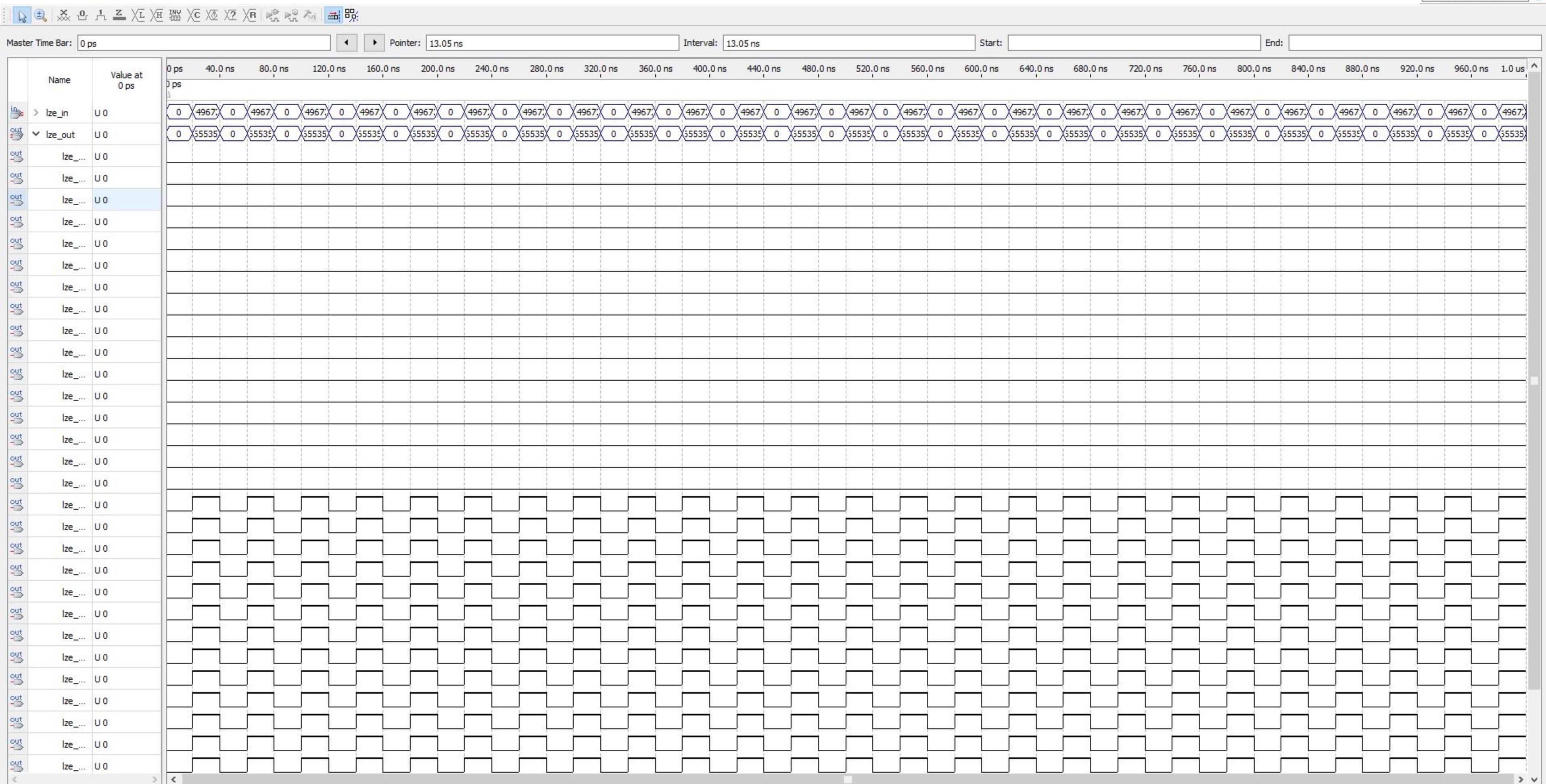
out_c	U 0	
out_z	U 0	
> out_a	U 0	0
> out_b	U 0	0
> out_ir	U 0	0
> outp	U 0	0
> addr_out	U Z	Z
> DATA_BUS	U 3629909006	9909 0 24246 0 53853 0 3593655477 0 3774125444 0 0443 5491 1649512198 0 6008 0
> mem_in	U Z	Z
> mem_out	U Z	Z

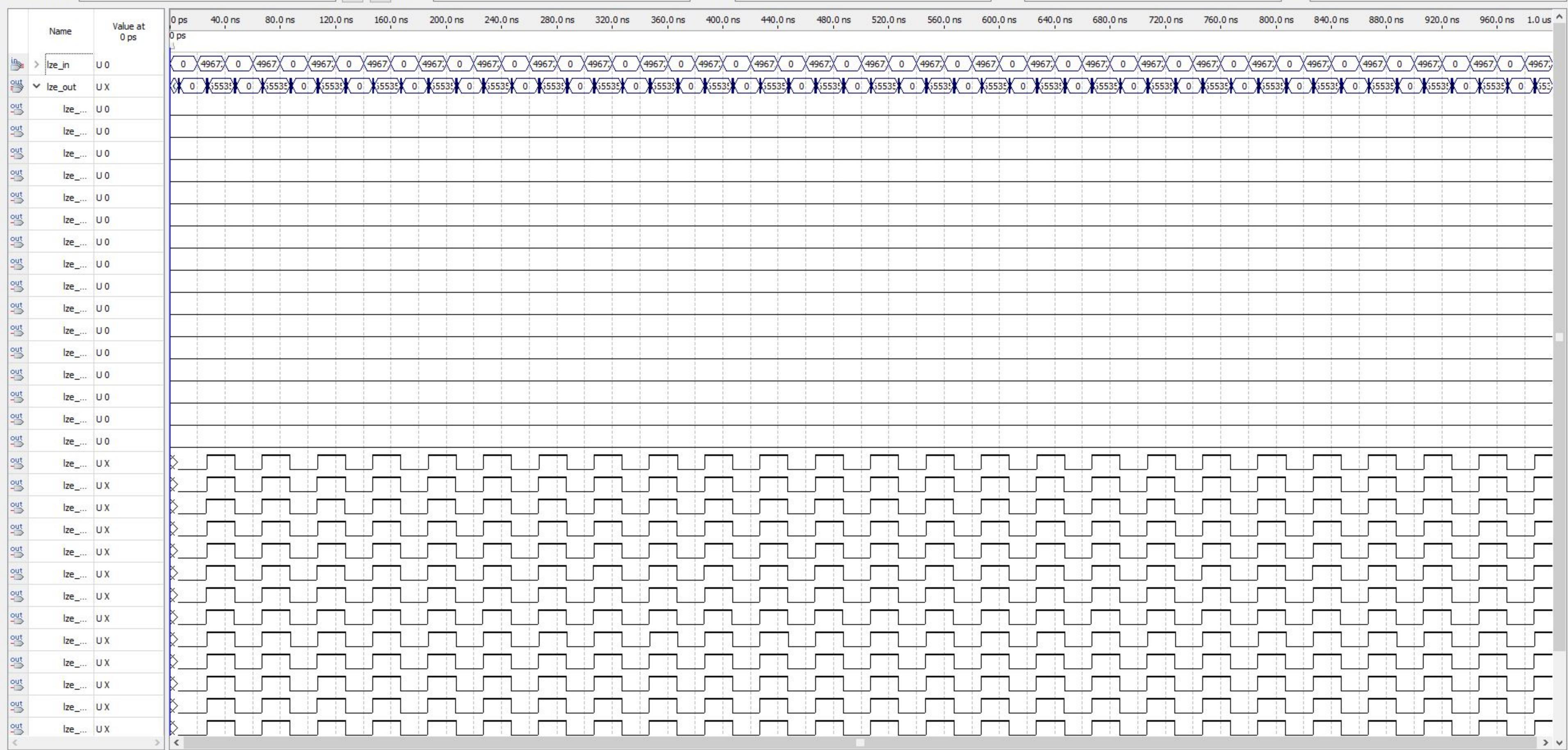
out	out_c	U 0
out	out_z	U 0
out	> out_a	U 0
out	> out_b	U 0
out	> out_ir	U 0
out	> outp	U 0
io	> addr_out	U Z
io	> DATA_BUS	UX
io	> mem_in	U Z
io	> mem_out	U Z

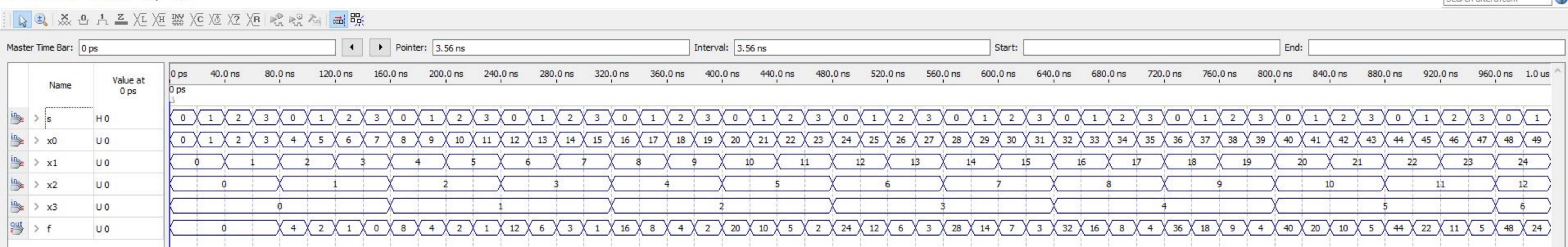
The timing diagram displays the state of each signal over time. The horizontal axis represents time, and the vertical axis lists the signals. The DATA\_BUS signal is highlighted with a blue background and shows specific data values at each sampling point.

Time Step	out_c	out_z	out_a	out_b	out_ir	outp	addr_out	DATA_BUS	mem_in	mem_out
1	0	0	0	0	0	0	Z	X	0	0
2	0	0	0	0	0	0	Z	3909	0	0
3	0	0	0	0	0	0	Z	0	0	0
4	0	0	0	0	0	0	Z	424	0	0
5	0	0	0	0	0	0	Z	0	0	0
6	0	0	0	0	0	0	Z	385	0	0
7	0	0	0	0	0	0	Z	159365547	0	0
8	0	0	0	0	0	0	Z	0	0	0
9	0	0	0	0	0	0	Z	377412544	0	0
10	0	0	0	0	0	0	Z	0	0	0
11	0	0	0	0	0	0	Z	443	0	0
12	0	0	0	0	0	0	Z	0	0	0
13	0	0	0	0	0	0	Z	5491	0	0
14	0	0	0	0	0	0	Z	0	0	0
15	0	0	0	0	0	0	Z	649512198	0	0
16	0	0	0	0	0	0	Z	0	0	0
17	0	0	0	0	0	0	Z	6008	0	0
18	0	0	0	0	0	0	Z	0	0	0

```
1 library ieee;
2 use ieee.std_logic_1164.all;
3 use ieee.std_logic_unsigned.all;
4
5 entity lze is
6     port(
7         lze_in: in std_logic_vector(31 downto 0);
8         lze_out: out std_logic_vector(31 downto 0));
9 end lze;
10
11 architecture behavior of lze is
12     signal zeros : std_logic_vector(31 downto 16) := (others => '0');
13     begin
14         lze_out <= zeros & lze_in(15 downto 0);
15     end behavior;
16
17
```







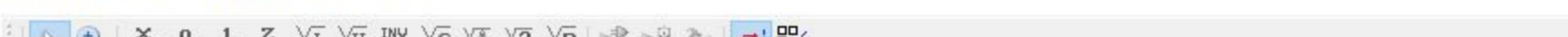


```
1 library ieee;
2 use ieee.std_logic_1164.all;
3 use ieee.std_logic_unsigned.all;
4
5 entity mux4to1 is
6     port
7         (s : in std_logic_vector(1 downto 0);
8          x0, x1, x2, x3 : in std_logic_vector(31 downto 0);
9          f : out std_logic_vector(31 downto 0));
10 end mux4to1;
11
12 architecture behavior of mux4to1 is
13 begin
14     with s select
15         f <= x0 when "00",
16                     x1 when "01",
17                     x2 when "10",
18                     x3 when "11";
19 end behavior;
20
21
22
```

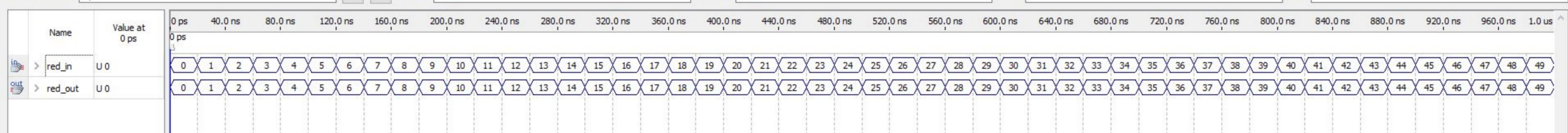
```
1 library ieee;
2 use ieee.std_logic_1164.all;
3 use ieee.numeric_std.all;
4
5 entity red is
6     port(
7         red_in : in std_logic_vector(31 downto 0);
8         red_out : out unsigned(7 downto 0));
9 end red;
10
11 architecture behavior of red is
12 begin
13     red_out <= unsigned(red_in(7 downto 0));
14 end behavior;
15
16
```

File Edit View Simulation Help

Search altera.com



Master Time Bar: 0 ps Pointer: 1.19 ns Interval: 1.19 ns Start: End:

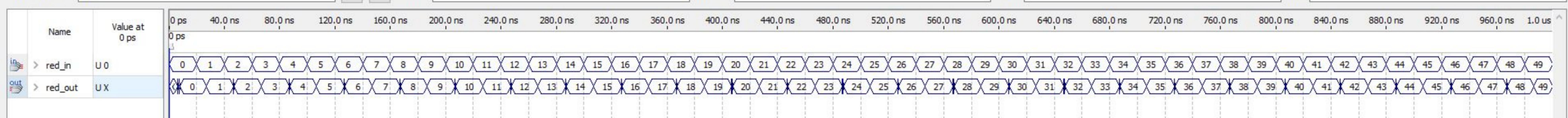


File Edit View Simulation Help

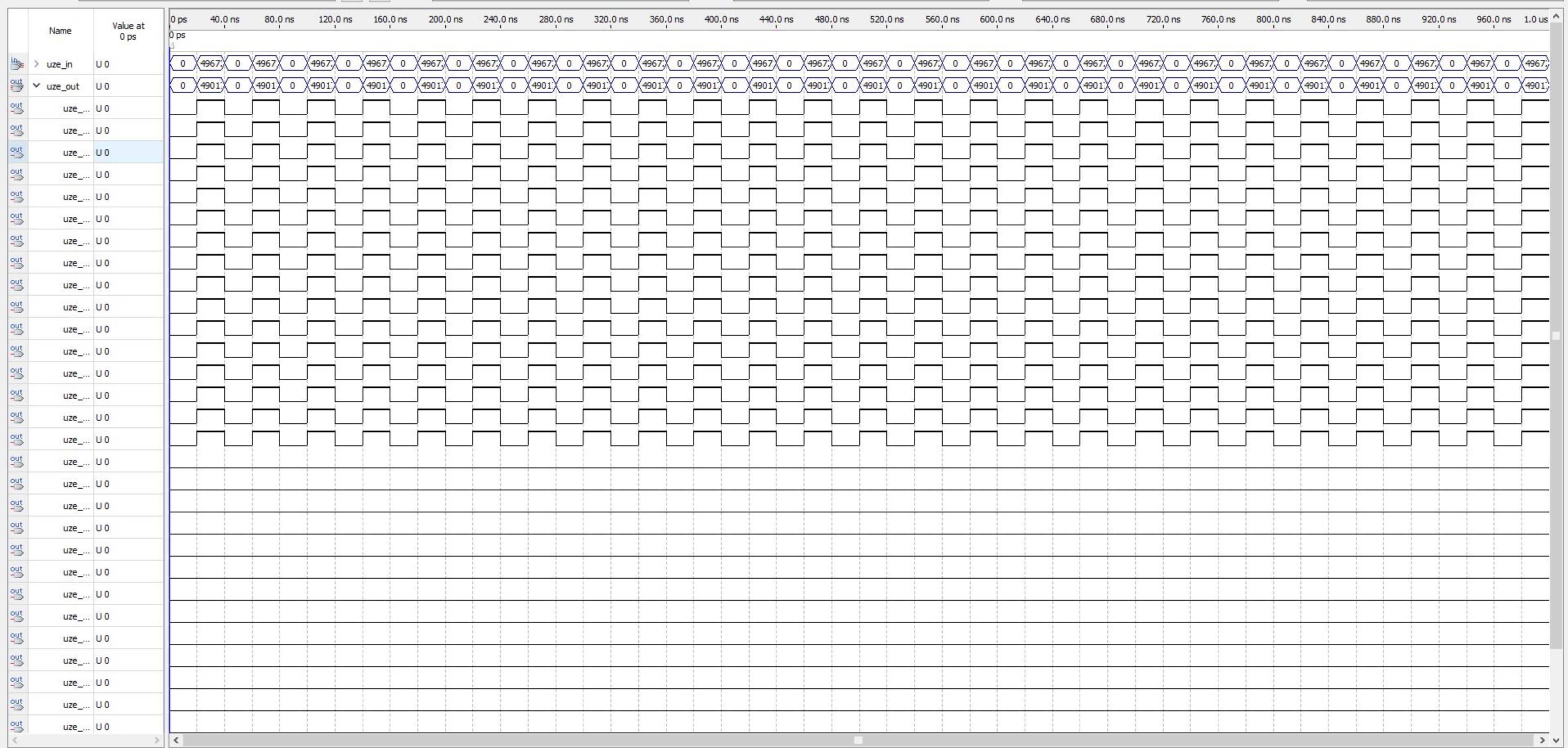
Search altera.com

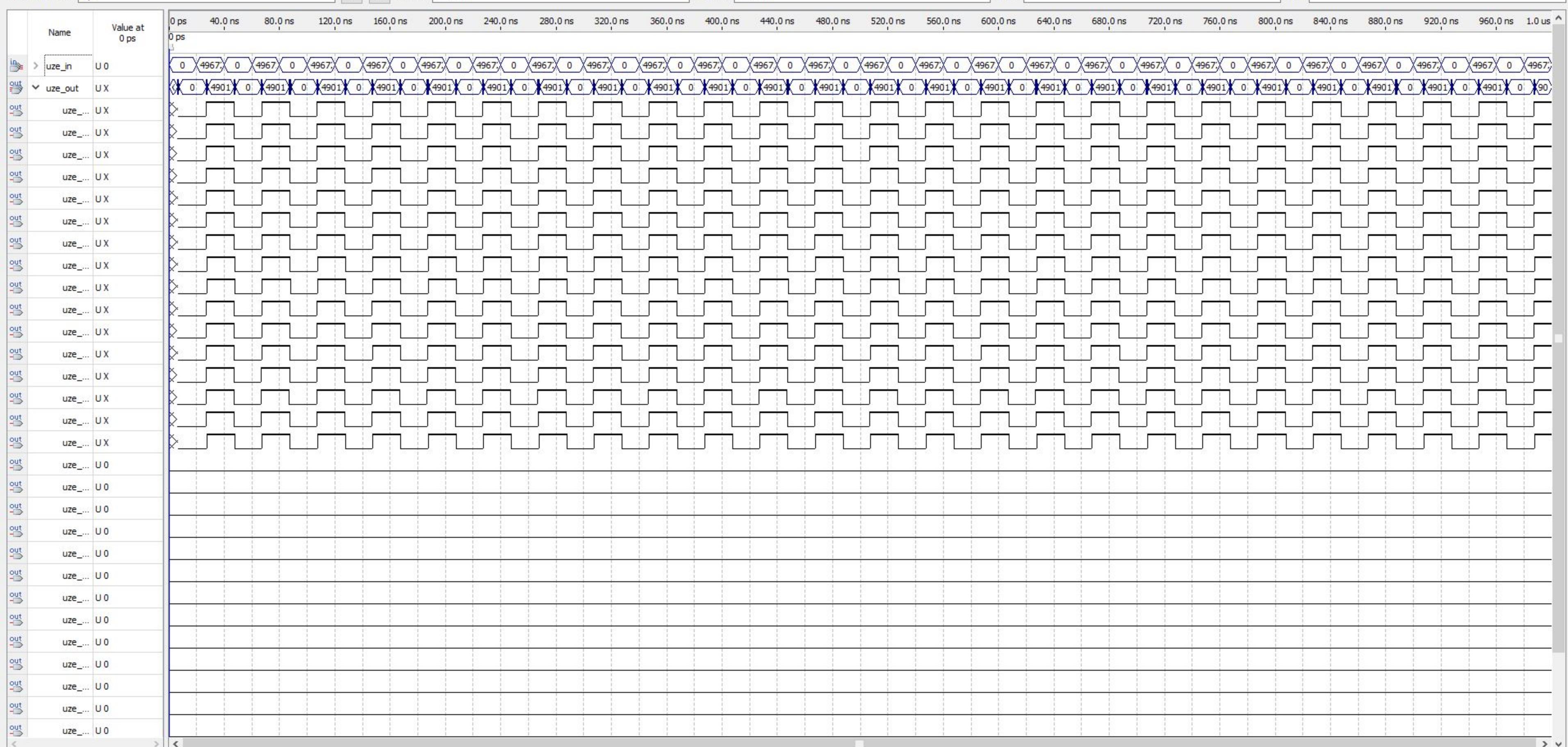


Master Time Bar: 0 ps   Pointer:  Interval:  Start:  End:



```
1 library ieee;
2 use ieee.std_logic_1164.all;
3 use ieee.std_logic_unsigned.all;
4
5 entity uze is
6     port(
7         uze_in : in std_logic_vector(31 downto 0);
8         uze_out: out std_logic_vector(31 downto 0));
9 end uze;
10
11 architecture behavior of uze is
12     signal zeros : std_logic_vector(15 downto 0) := (others => '0');
13     begin
14         uze_out <= uze_in(31 downto 16) & zeros;
15     end behavior;
16
17
```





Programs: Computer Engineering

Course Number	<b>COE608</b>
Course Title	<b>Computer Organization and Architecture</b>
Semester/Year	<b>Winter 2020</b>
Instructor	<b>Nagi Mekhiel</b>

<b>Lab Report No.</b>	<b>4b</b>
-----------------------	-----------

Lab Title	<b>Data-Path Design</b>
-----------	-------------------------

Section No.	<b>04</b>
Group No.	
Submission Date	<b>03/10/2020</b>
Due Date	<b>03/10/2020</b>

Name	Student ID	Signature*
Duanwei Zhang	500824903	

\*By signing above you attest that you have contributed to this submission and confirm that all work you have contributed to this submission is your own work. Any suspicion of copying or plagiarism in this work will result in an investigation of Academic Misconduct and may result in a “0” on the work, an “F” in the course, or possibly more severe penalties, as well as a Disciplinary Notice on your academic record under the Student Code of Academic Conduct, which can be found online at:

[www.ryerson.ca/senate/current/pol60.pdf](http://www.ryerson.ca/senate/current/pol60.pdf).

## 1. Register32.vhd

The screenshot shows the Quartus II 64-Bit software interface. The main window displays the VHDL code for `register32.vhd`. The code defines an entity `register32` with a port containing inputs `d`, `ld`, `clr`, and `clk`, and an output `Q`. It includes an architecture `Behavior` with a process that updates `Q` based on `clr` and `ld` controls. The Project Navigator on the left lists other files in the project, and the Messages window at the bottom shows compilation progress and warnings.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity register32 is
port(
    d      : in std_logic_vector(31 downto 0);
    ld     : in std_logic;
    clr   : in std_logic;
    clk   : in std_logic;
    Q      : out std_logic_vector(31 downto 0)
);
end register32;

architecture Behavior of register32 is
begin
    process (ld, clr, clk)
    begin
        if clr = '1' then
            Q <= (others => '0');
        elsif ((clk'event and clk = '1') and (ld = '1')) then
            Q <= d;
        end if;
    end process;
end Behavior;
```

## 2. Pc.vhd

The screenshot shows the Quartus II 64-Bit interface. The Project Navigator on the left lists several files including register32.vhd, pc.vhd, mux2to1.vhd, add.vhd, adder4.vhd, adder16.vhd, adder32.vhd, alu.vhd, fulladd.vhd, data\_mem.vhd, LZE.vhd, UZE.vhd, RED.vhd, mux4to1.vhd, Data\_Path.vhd, LDAI.vwf, LDBI.vwf, STA.vwf, STB.vwf, and LDA.vwf. The 'Files' tab is selected. The main window displays the VHDL code for pc.vhd. The code defines an entity pc with a port containing clr, clk, ld, inc, d, and q. It includes an architecture Behaviour with components add, mux2to1, and register32. The add component has inputs A and B and output f. The mux2to1 component has inputs s, w0, and w1, and output f. The register32 component has inputs d, ld, clk, and clr, and outputs q\_out. The code also includes signal declarations for add\_out, mux\_out, q\_out, and ld. The 'Tasks' panel on the right shows the progress of a compilation process: Analysis & Synthesis (57%), Fitter (Place & Route) (89%), Assembler (89%), TimeQuest Timing (0%), EDA Netlist Write (0%), and Program Device (0%). The status bar at the bottom indicates Ln 1 Col 1 VHDL File, 57% completion, and 00:00:37.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity pc is
port(
    clr : in std_logic;
    clk : in std_logic;
    --ld : in std_logic;
    inc : in std_logic;
    d : in std_logic_vector(31 downto 0);
    q : out std_logic_vector(31 downto 0)
);
end pc;

architecture Behaviour of pc is
component add
port (
    A : in std_logic_vector(31 downto 0);
    B : out std_logic_vector(31 downto 0)
);
end component;

component mux2to1
port (
    s : in std_logic;
    w0, w1: in std_logic_vector(31 downto 0);
    f : out std_logic_vector(31 downto 0)
);
end component;

component register32
port (
    d : in std_logic_vector(31 downto 0);
    ld : in std_logic;
    clk : in std_logic;
    Q : out std_logic_vector(31 downto 0)
);
end component;

begin
add0: add port map(q_out, add_out);
mux0: mux2to1 port map(inc, d, add_out, mux_out);
reg0: register32 port map (mux_out, ld, clk, q_out);
q <= q_out;
end Behaviour;
```

### 3. Mux2to1.vhd

The screenshot shows the Quartus II 64-Bit interface with the project 'lab4b' open. The 'Project Navigator' panel on the left lists files: register32.vhd, pc.vhd, mux2to1.vhd, add.vhd, adder4.vhd, and adder16.vhd. The 'mux2to1.vhd' file is selected. The main editor window displays the VHDL code for the mux2to1 entity:

```
library ieee;
use ieee.std_logic_1164.all;

entity mux2to1 is
port (s : in std_logic;
      w0,w1 : in std_logic_vector(31 downto 0);
      f : out std_logic_vector(31 downto 0));
end mux2to1;

architecture Behavior of mux2to1 is
begin
  with s select
    f <= w0 when '0',
    w1 when others;
end Behavior;
```

The code defines an entity 'mux2to1' with a single port 's' of type std\_logic and two inputs 'w0' and 'w1' of type std\_logic\_vector(31 downto 0). The output 'f' is also a std\_logic\_vector(31 downto 0). The architecture 'Behavior' uses a 'with select' statement to map 's' to either 'w0' or 'w1'. The 'with' keyword is highlighted in red.

### 4. Add.vhd

The screenshot shows the Quartus II 64-Bit interface with the project 'lab4b' open. The 'Project Navigator' panel on the left lists files: register32.vhd, pc.vhd, mux2to1.vhd, and add.vhd. The 'add.vhd' file is selected. The main editor window displays the VHDL code for the add entity:

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity add is
port (A : in std_logic_vector(31 downto 0);
      B : out std_logic_vector(31 downto 0));
end add;

architecture Behavior of add is
begin
  B <= A + 4;
end Behavior;
```

The code defines an entity 'add' with a single port 'A' of type std\_logic\_vector(31 downto 0) and one output 'B' of the same type. The architecture 'Behavior' contains a single line of code: 'B <= A + 4;'. The '+ 4' part is highlighted in red.

## 5. Adder4.vhd

The screenshot shows the Quartus II 64-Bit interface with the project 'lab4b' open. The left pane displays the 'Project Navigator' with files like register32.vhd, pc.vhd, mux2to1.vhd, add.vhd, adder4.vhd, adder16.vhd, adder32.vhd, alu.vhd, and fulladd.vhd. The right pane shows the VHDL code for 'adder4.vhd'. The code defines an entity 'adder4' with a port containing Cin, X, Y, S, and Cout. It uses a 'fulladd' component with four stages of addition. The architecture 'Behavior' maps the port to the component and specifies four stages of addition using 'fulladd' port maps.

```
library ieee;
use ieee.std_logic_1164.all;

entity adder4 is
port(
    Cin : in std_logic;
    X,Y : in std_logic_vector(3 downto 0);
    S : out std_logic_vector(3 downto 0);
    Cout : out std_logic
);
end adder4;

architecture Behavior of adder4 is
component fulladd
port(
    Cin, x, y : in std_logic;
    s, Cout : out std_logic
);
end component;

signal C : std_logic_vector (1 to 3);
begin
    stage0: fulladd port map (Cin, X(0), Y(0), S(0), C(1));
    stage1: fulladd port map (C(1), X(1), Y(1), S(1), C(2));
    stage2: fulladd port map (C(2), X(2), Y(2), S(2), C(3));
    stage3: fulladd port map (C(3), X(3), Y(3), S(3), Cout);
end Behavior;
```

## 6. Adder16.vhd

The screenshot shows the Quartus II 64-Bit interface with the project 'lab4b' open. The left pane displays the 'Project Navigator' with files like register32.vhd, pc.vhd, mux2to1.vhd, add.vhd, adder4.vhd, adder16.vhd, adder32.vhd, alu.vhd, fulladd.vhd, data mem.vhd, and EDA Netlist Wr. The right pane shows the VHDL code for 'adder16.vhd'. The code defines an entity 'adder16' with a port containing Cin, X, Y, S, and Cout. It uses an 'adder4' component with four stages of addition. The architecture 'Behavior' maps the port to the component and specifies four stages of addition using 'adder4' port maps.

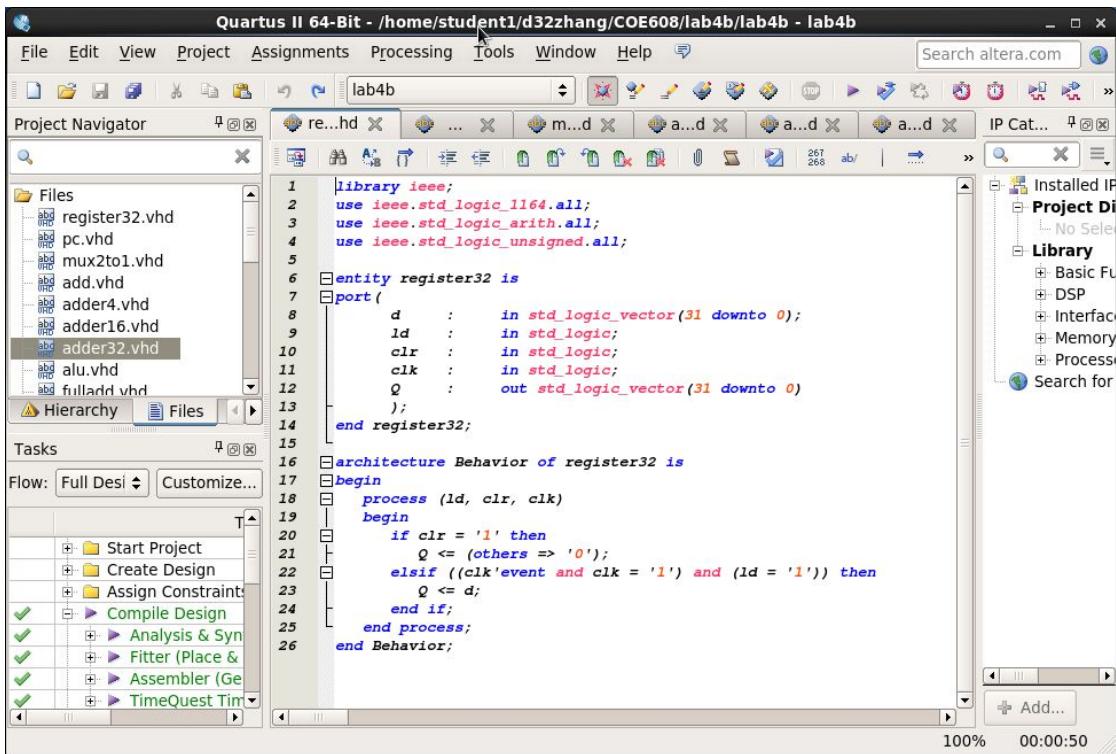
```
library ieee;
use ieee.std_logic_1164.all;

entity adder16 is
port(
    Cin : in std_logic;
    X,Y : in std_logic_vector(15 downto 0);
    S : out std_logic_vector(15 downto 0);
    Cout : out std_logic
);
end adder16;

architecture Behavior of adder16 is
component adder4
port(
    Cin : in std_logic;
    X,Y : in std_logic_vector(3 downto 0);
    S : out std_logic_vector(3 downto 0);
    Cout : out std_logic
);
end component;

signal C : std_logic_vector (1 to 3);
begin
    stage0: adder4 port map (Cin, X(3 downto 0), Y(3 downto 0), S(3 downto 0), C(1));
    stage1: adder4 port map (C(1), X(7 downto 4), Y(7 downto 4), S(7 downto 4), C(2));
    stage2: adder4 port map (C(2), X(11 downto 8), Y(11 downto 8), S(11 downto 8), C(3));
    stage3: adder4 port map (C(3), X(15 downto 12), Y(15 downto 12), S(15 downto 12), Cout);
end Behavior;
```

## 7. Adder32.vhd



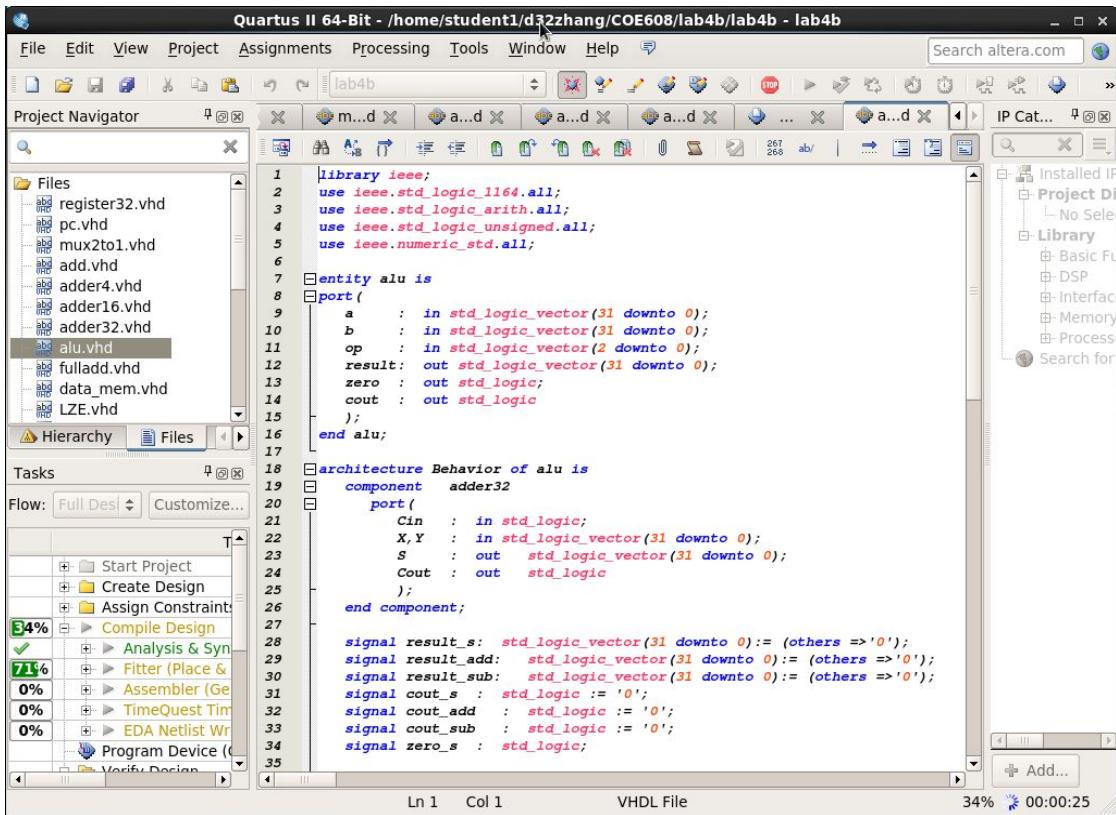
The screenshot shows the Quartus II 64-Bit interface with the project "lab4b" open. The Project Navigator on the left lists several files including register32.vhd, pc.vhd, mux2to1.vhd, add.vhd, adder4.vhd, adder16.vhd, adder32.vhd, alu.vhd, and fulladd.vhd. The "Files" tab is selected. The main window displays the VHDL code for "adder32.vhd". The code defines an entity "register32" with a port containing "d", "ld", "clr", "clk", and "Q". It includes a process that updates "Q" based on "ld" and "clr" inputs, and another process that updates "Q" based on "clk" and "d". The architecture ends with a "Behavior" section. The right panel shows the "Library" tree under "Project Dir". The bottom status bar shows "100% 00:00:50".

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity register32 is
port(
    d : in std_logic_vector(31 downto 0);
    ld : in std_logic;
    clr : in std_logic;
    clk : in std_logic;
    Q : out std_logic_vector(31 downto 0)
);
end register32;

architecture Behavior of register32 is
begin
    process (ld, clr, clk)
    begin
        if clr = '1' then
            Q <= (others => '0');
        elsif ((clk'event and clk = '1') and (ld = '1')) then
            Q <= d;
        end if;
    end process;
end Behavior;
```

## 8. Alu.vhd



The screenshot shows the Quartus II 64-Bit interface with the project "lab4b" open. The Project Navigator on the left lists files including register32.vhd, pc.vhd, mux2to1.vhd, add.vhd, adder4.vhd, adder16.vhd, adder32.vhd, alu.vhd, fulladd.vhd, data\_mem.vhd, and LZE.vhd. The "Files" tab is selected. The main window displays the VHDL code for "alu.vhd". The code defines an entity "alu" with a port containing "a", "b", "op", "result", "zero", and "cout". It includes a process that performs arithmetic operations based on "op" and returns results in "result" and "cout". The architecture ends with a "Behavior" section that uses the "adder32" component. The right panel shows the "Library" tree under "Project Dir". The bottom status bar shows "34% 00:00:25".

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;
use ieee.numeric_std.all;

entity alu is
port(
    a : in std_logic_vector(31 downto 0);
    b : in std_logic_vector(31 downto 0);
    op : in std_logic_vector(2 downto 0);
    result : out std_logic_vector(31 downto 0);
    zero : out std_logic;
    cout : out std_logic
);
end alu;

architecture Behavior of alu is
component adder32
port(
    Cin : in std_logic;
    X,Y : in std_logic_vector(31 downto 0);
    S : out std_logic_vector(31 downto 0);
    Cout : out std_logic
);
end component;

signal result_s: std_logic_vector(31 downto 0):= (others =>'0');
signal result_add: std_logic_vector(31 downto 0):= (others =>'0');
signal result_sub: std_logic_vector(31 downto 0):= (others =>'0');
signal cout_s : std_logic := '0';
signal cout_add : std_logic := '0';
signal cout_sub : std_logic := '0';
signal zero_s : std_logic;
```

Quartus II 64-Bit - /home/student1/d32zhang/COE608/lab4b/lab4b - lab4b

File Edit View Project Assignments Processing Tools Window Help

Project Navigator

Files

- register32.vhd
- pc.vhd
- mux2to1.vhd
- add.vhd
- adder4.vhd
- adder16.vhd
- adder32.vhd
- alu.vhd
- fulladd.vhd
- data\_mem.vhd
- LZE.vhd
- UZE.vhd
- RED.vhd
- mux4to1.vhd
- Data\_Path.vhd
- LDAI.wvf
- LDBI.wvf

Hierarchy Files

Tasks

Flow: Full Desi ▾ Customize...

Start Project

Create Design

Assign Constraints

Compile Design

Analysis & Syn

Fitter (Place & Route)

Assembler (Generate)

TimeQuest Timing

EDA Netlist Writer

Program Device ()

Verify Design

Simulate Design

On-chip Debug

PowerPlay Power

SSN Analyzer

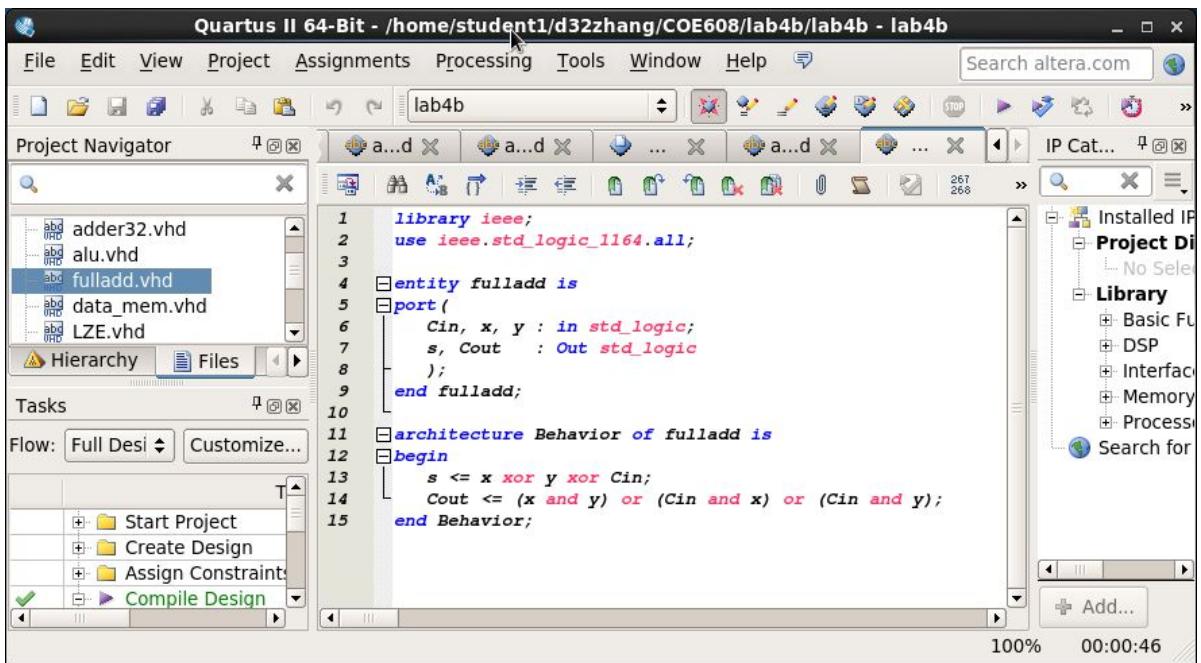
Engineering Change

35 begin  
36 add0 : adder32 port map (op(2), a, b, result\_add, cout\_add);  
37 sub0 : adder32 port map (op(2), a, not b, result\_sub, cout\_sub);  
38  
39 process (a, b, op)  
40 begin  
41 case (op) is  
42 when "000" =>  
43 result\_s <= a and b;  
44 cout\_s <= '0';  
45 when "001" =>  
46 result\_s <= a or b;  
47 cout\_s <= '0';  
48 when "010" =>  
49 result\_s <= result\_add;  
50 cout\_s <= cout\_add;  
51 when "011" =>  
52 result\_s <= b;  
53 cout\_s <= '0';  
54 when "110" =>  
55 result\_s <= result\_sub;  
56 cout\_s <= cout\_sub;  
57 when "100" =>  
58 result\_s <= a(30 downto 0) & '0';  
59 cout\_s <= a(31);  
60 when "101" =>  
61 result\_s <= '0' & a(31 downto 1);  
62 cout\_s <= '0';  
63  
64 when others =>  
65 result\_s <= a;  
66 cout\_s <= '0';  
67 end case;  
68  
69 case (result\_s) is  
70 when (others >= '0') =>  
71 zero\_s <= '1';  
72 when others =>  
73 zero\_s <= '0';  
74 end case;  
75 end process;  
76  
77 result <= result\_s;  
78 cout <= cout\_s;  
79 zero <= zero\_s;  
80  
81 end Behavior;

100% 00:00:53

+ Add...

## 9. Fulladd.vhd



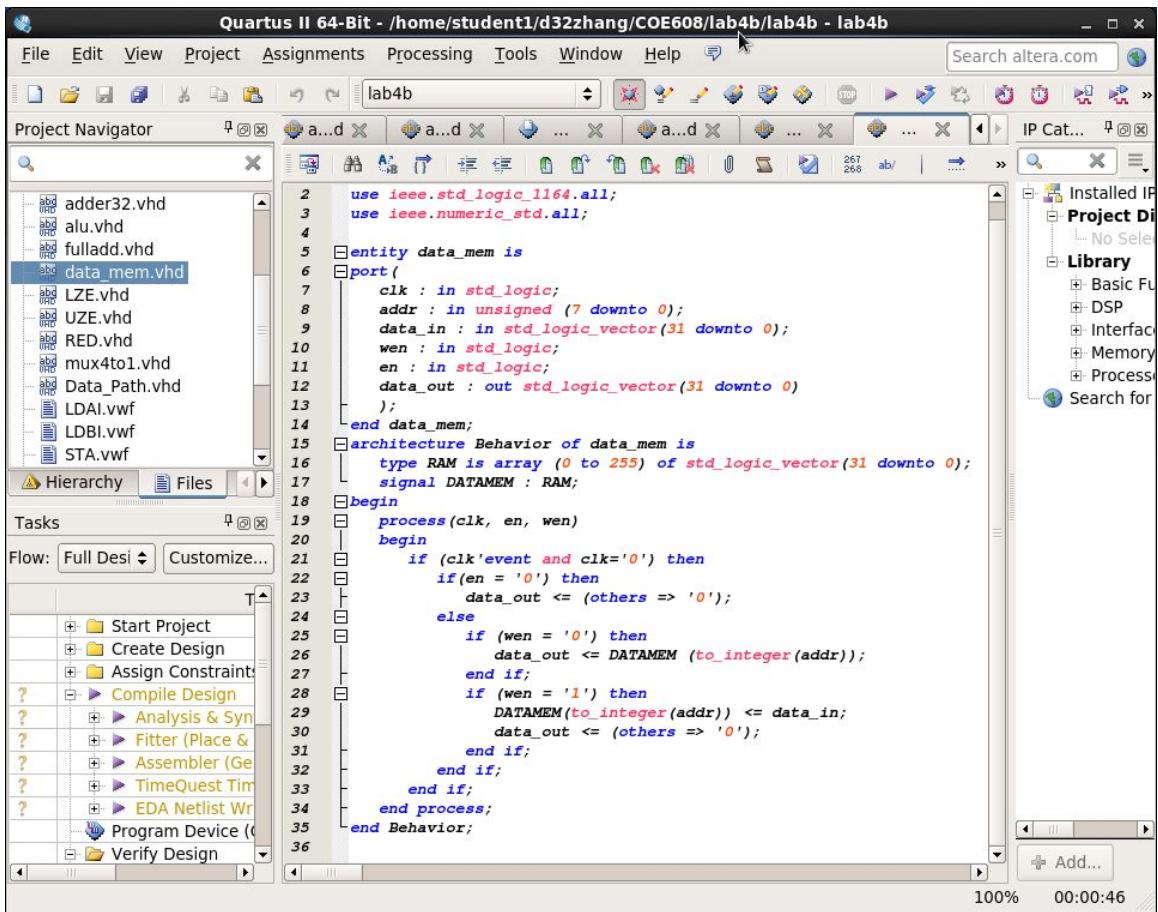
The screenshot shows the Quartus II 64-Bit interface. The Project Navigator on the left lists files: adder32.vhd, alu.vhd, fulladd.vhd (selected), data\_mem.vhd, and LZE.vhd. The Source Editor on the right displays the VHDL code for the fulladd entity:

```
library ieee;
use ieee.std_logic_1164.all;

entity fulladd is
port(
    Cin, x, y : in std_logic;
    s, Cout : out std_logic
);
end fulladd;

architecture Behavior of fulladd is
begin
    s <= x xor y xor Cin;
    Cout <= (x and y) or (Cin and x) or (Cin and y);
end Behavior;
```

## 10. Data\_mem.vhd



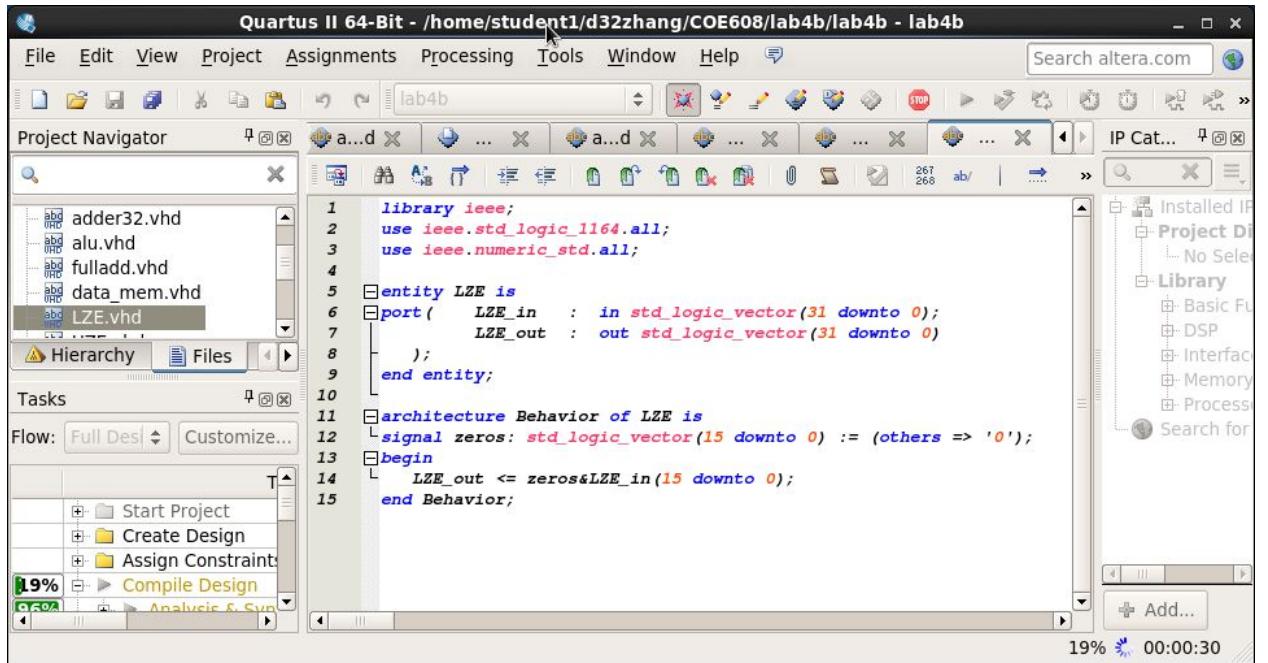
The screenshot shows the Quartus II 64-Bit interface. The Project Navigator on the left lists files: adder32.vhd, alu.vhd, fulladd.vhd, and data\_mem.vhd (selected). The Source Editor on the right displays the VHDL code for the data\_mem entity:

```
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity data_mem is
port(
    clk : in std_logic;
    addr : in unsigned (7 downto 0);
    data_in : in std_logic_vector(31 downto 0);
    wen : in std_logic;
    en : in std_logic;
    data_out : out std_logic_vector(31 downto 0)
);
end data_mem;

architecture Behavior of data_mem is
type RAM is array (0 to 255) of std_logic_vector(31 downto 0);
signal DATAMEM : RAM;
begin
process(clk, en, wen)
begin
    if (clk'event and clk='0') then
        if(en = '0') then
            data_out <= (others => '0');
        else
            if (wen = '0') then
                data_out <= DATAMEM (to_integer(addr));
            end if;
            if (wen = '1') then
                DATAMEM(to_integer(addr)) <= data_in;
                data_out <= (others => '0');
            end if;
        end if;
    end process;
end Behavior;
```

## 11. LZE.vhd



The screenshot shows the Quartus II 64-Bit interface with the project "lab4b" open. The Project Navigator on the left lists several VHDL files: adder32.vhd, alu.vhd, fulladd.vhd, data\_mem.vhd, and LZE.vhd. The LZE.vhd file is currently selected. The main editor window displays the VHDL code for the LZE entity:

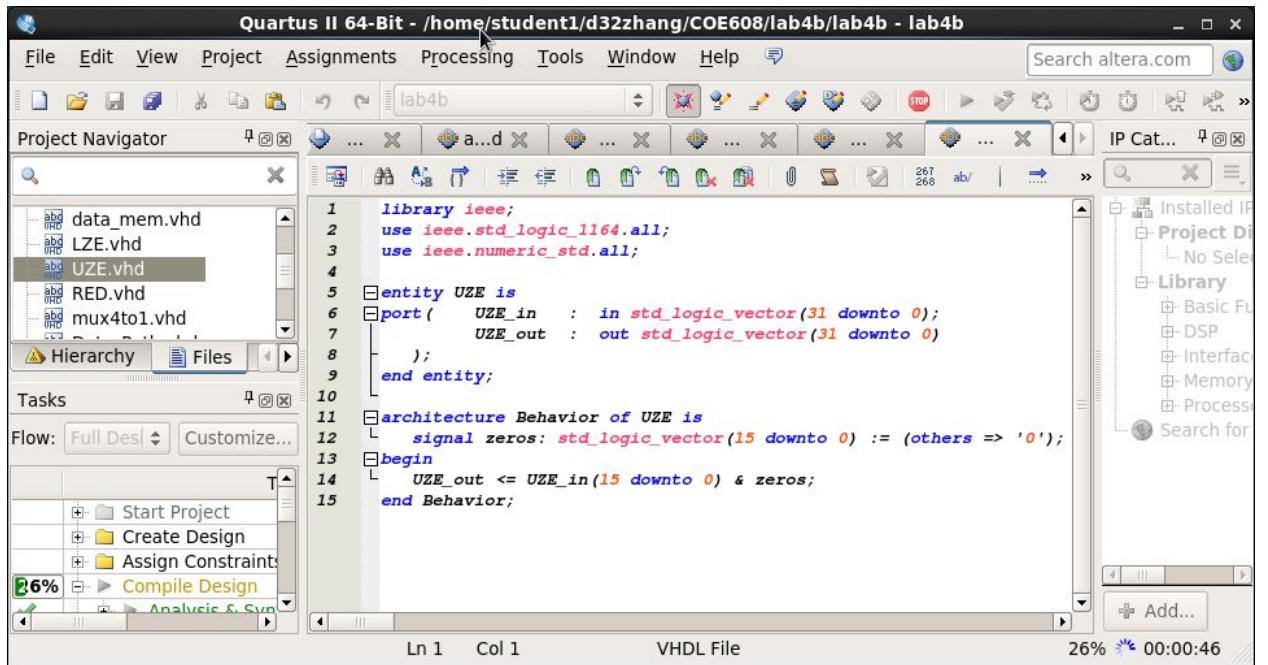
```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity LZE is
port(
    LZE_in : in std_logic_vector(31 downto 0);
    LZE_out : out std_logic_vector(31 downto 0)
);
end entity;

architecture Behavior of LZE is
signal zeros: std_logic_vector(15 downto 0) := (others => '0');
begin
    LZE_out <= zeros&LZE_in(15 downto 0);
end Behavior;
```

The status bar at the bottom indicates 19% completion and 00:00:30 time.

## 12. UZE.vhd



The screenshot shows the Quartus II 64-Bit interface with the project "lab4b" open. The Project Navigator on the left lists several VHDL files: data\_mem.vhd, LZE.vhd, and UZE.vhd. The UZE.vhd file is currently selected. The main editor window displays the VHDL code for the UZE entity:

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity UZE is
port(
    UZE_in : in std_logic_vector(31 downto 0);
    UZE_out : out std_logic_vector(31 downto 0)
);
end entity;

architecture Behavior of UZE is
signal zeros: std_logic_vector(15 downto 0) := (others => '0');
begin
    UZE_out <= UZE_in(15 downto 0) & zeros;
end Behavior;
```

The status bar at the bottom indicates 26% completion and 00:00:46 time.

### 13. RED.vhd

The screenshot shows the Quartus II 64-Bit interface with the project 'lab4b' open. The 'Project Navigator' panel on the left lists several VHDL files: data\_mem.vhd, LZE.vhd, UZE.vhd, RED.vhd, and mux4to1.vhd. The 'mux4to1.vhd' file is currently selected. The main editor window displays the following VHDL code for the RED entity:

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity RED is
port (RED_in : in std_logic_vector(31 downto 0);
      RED_out : out unsigned(7 downto 0));
end entity;

architecture Behavior of RED is
begin
  RED_out <= unsigned(RED_in(7 downto 0));
end Behavior;
```

The status bar at the bottom right indicates the code length is 28% and the time is 00:00:59.

### 14. Mux4to1.vhd

The screenshot shows the Quartus II 64-Bit interface with the project 'lab4b' open. The 'Project Navigator' panel on the left lists several VHDL files: data\_mem.vhd, LZE.vhd, UZE.vhd, RED.vhd, mux4to1.vhd, Data\_Path.vhd, and LDAl.vwf. The 'mux4to1.vhd' file is currently selected. The main editor window displays the following VHDL code for the mux4to1 entity:

```
library ieee;
use ieee.std_logic_1164.all;

entity mux4to1 is
port (s : in std_logic_vector(1 downto 0);
      X1,X2,X3,X4 : in std_logic_vector(31 downto 0);
      f : out std_logic_vector(31 downto 0));
end mux4to1;

architecture Behavior of mux4to1 is
begin
  with s select
    f <= x1 when "00",
    x2 when "01",
    x3 when "10",
    x4 when "11";
end Behavior;
```

The status bar at the bottom right indicates the code length is 34% and the time is 00:01:23.

## 15. Data\_Path.vhd

The screenshot shows the Quartus II 64-Bit interface. The Project Navigator on the left lists several files: fulladd.vhd, data\_mem.vhd, LZE.vhd, UZE.vhd, RED.vhd, mux4to1.vhd, and Data\_Path.vhd. The Data\_Path.vhd file is currently selected. The Source Editor on the right displays the VHDL code for the Data\_Path entity. The code includes library declarations for IEEE std\_logic\_1164.all, std\_logic\_arith.all, and std\_logic\_unsigned.all. It defines an entity data\_path with various input and output ports, including Clk, mClk, WEN, EN, Clr\_A, Ld\_A, Clr\_B, Ld\_B, Clr\_C, Ld\_C, Clr\_Z, Ld\_Z, ClrPC, Ld\_PC, ClrIR, Ld\_IR, Out\_A, Out\_B, Out\_C, Out\_Z, Out\_PC, Out\_IR, Inc\_PC, ADDR\_OUT, DATA\_IN, DATA\_BUS, MEM\_OUT, MEM\_IN, MEM\_ADDR, DATA\_MUX, REG\_MUX, A\_MUX, B\_MUX, IM\_MUX1, IM\_MUX2, and ALU\_Op. The code concludes with an end entity declaration. On the left, the Tasks panel shows compilation progress: 35% for Compile Design, 75% for Fitter (Place & Route), 0% for Assembler, 0% for TimeQuest Timing, and 0% for EDA Netlist Writing. The bottom status bar indicates 35% completion and 00:01:58 time.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity data_path is
    port(
        Clk, mClk : in std_logic;
        WEN, EN : in std_logic;
        Clr_A, Ld_A : in std_logic;
        Clr_B, Ld_B : in std_logic;
        Clr_C, Ld_C : in std_logic;
        Clr_Z, Ld_Z : in std_logic;
        ClrPC, Ld_PC : in std_logic;
        ClrIR, Ld_IR : in std_logic;
        Out_A : out std_logic_vector(31 downto 0);
        Out_B : out std_logic_vector(31 downto 0);
        Out_C : out std_logic;
        Out_Z : out std_logic;
        Out_PC : out std_logic_vector(31 downto 0);
        Out_IR : out std_logic_vector(31 downto 0);
        Inc_PC : in std_logic;
        ADDR_OUT : out std_logic_vector(31 downto 0);
        DATA_IN : in std_logic_vector(31 downto 0);
        DATA_BUS,
        MEM_OUT,
        MEM_IN : out std_logic_vector(31 downto 0);
        MEM_ADDR : out std_logic_vector(7 downto 0);
        DATA_MUX : in std_logic_vector(1 downto 0);
        REG_MUX : in std_logic;
        A_MUX,
        B_MUX : in std_logic;
        IM_MUX1 : in std_logic;
        IM_MUX2 : in std_logic_vector(1 downto 0);
        ALU_Op : in std_logic_vector(2 downto 0)
    );
end entity;
```

Quartus II 64-Bit - /home/student1/d32zhang/COE608/lab4b/lab4b - lab4b

File Edit View Project Assignments Processing Tools Window Help

Project Navigator

fulladd.vhd  
data\_mem.vhd  
LZE.vhd  
UZE.vhd  
RED.vhd  
mux4to1.vhd  
**Data\_Path.vhd**  
LDAI.vwf  
LDBI.vwf  
STA.vwf  
STB.vwf  
LDA.vwf  
LDB.vwf  
LUI.vwf  
JMP.vwf  
ADD.vwf

Hierarchy Files

Tasks

Flow: Full Des | Customize...

35% □ ▶ Compile Design  
75% ✓ □ ▶ Analysis & Syn  
0% □ ▶ Filter (Place &  
0% □ ▶ Assembler (Ge  
0% □ ▶ TimeQuest Tim  
0% □ ▶ EDA Netlist Wr  
Program Device (O  
Verify Design  
Simulate Design  
On-chip Debug  
PowerPlay Pow  
SSN Analyzer  
Engineering Ch  
Chip Planner

```
44 L
45 architecture Behavior of Data_path is
46   component data_mem is
47     port(
48       clk      : in std_logic;
49       addr    : in unsigned(7 downto 0);
50       data_in : in std_logic_vector(31 downto 0);
51       wen     : in std_logic;
52       en      : in std_logic;
53       data_out: out std_logic_vector(31 downto 0)
54     );
55   end component;
56
57   component register32 is
58     port(
59       d      : in std_logic_vector(31 downto 0);
60       ld     : in std_logic;
61       clr   : in std_logic;
62       clk   : in std_logic;
63       q      : out std_logic_vector(31 downto 0)
64     );
65   end component;
66
67   component pc is
68     port(
69       clr   : in std_logic;
70       clk   : in std_logic;
71       inc   : in std_logic;
72       d     : in std_logic_vector(31 downto 0);
73       q     : out std_logic_vector(31 downto 0)
74     );
75   end component;
76
77   component LZE is
78     port(  LZE_in   : in std_logic_vector(31 downto 0);
79            LZE_out  : out std_logic_vector(31 downto 0)
80          );
81   end component;
82
83   component UZE is
84     port(  UZE_in   : in std_logic_vector(31 downto 0);
85            UZE_out  : out std_logic_vector(31 downto 0)
86          );
87   end component;
```

Add...

35% 00:02:21

Quartus II 64-Bit - /home/student/t1/d32zhang/COE608/lab4b/lab4b - lab4b

File Edit View Project Assignments Processing Tools Window Help

Project Navigator

Search altera.com

IP Catalog

Installed IP

Project Details

No Selection

Library

- Basic Functions
- DSP
- Interconnect
- Memory
- Processor

Search for

adder32.vhd  
alu.vhd  
fulladd.vhd  
data\_mem.vhd  
LZE.vhd  
UZE.vhd  
RED.vhd  
mux4to1.vhd  
Data\_Path.vhd  
LDAI.vwf  
LDBI.vwf  
STA.vwf  
STB.vwf  
LDA.vwf  
LDB.vwf  
LUI.vwf  
JMP.vwf  
ADD.vwf  
ADDI.vwf

Hierarchy Files

Tasks

Flow: Full Design Customize...

Compile Design

- 6% ▶ Analysis & Synthesis
- 80% ▶ Fitter (Place & Route)
- 0% ▶ Assembler (Generate)
- 0% ▶ TimeQuest Timing Analyzer
- 0% ▶ EDA Netlist Writer

Program Device (Fitter)

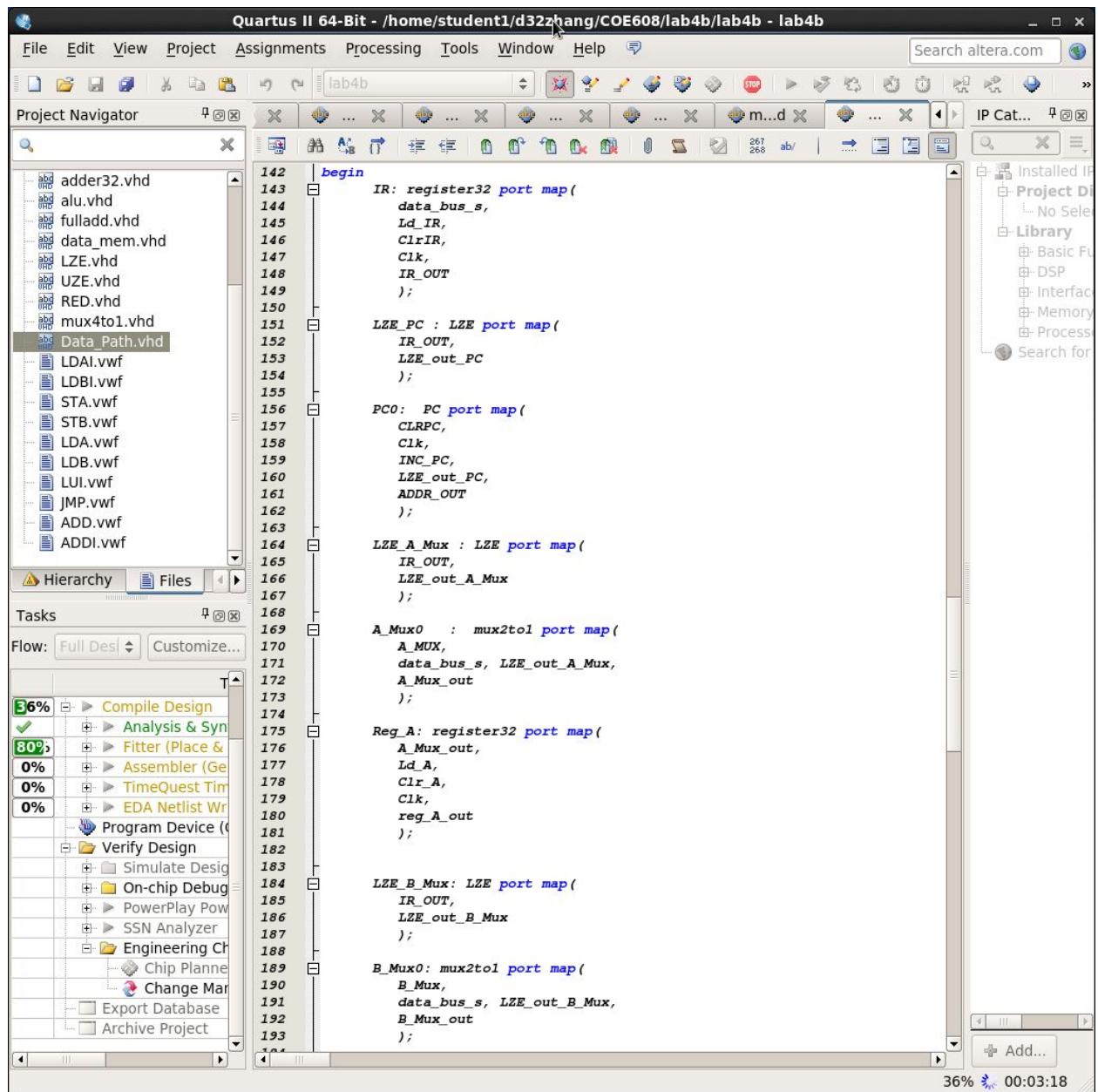
- Verify Design
- Simulate Design
- On-chip Debug
- PowerPlay Power Estimator
- SSN Analyzer
- Engineering Change Management
- Change Manager
- Export Database
- Archive Project

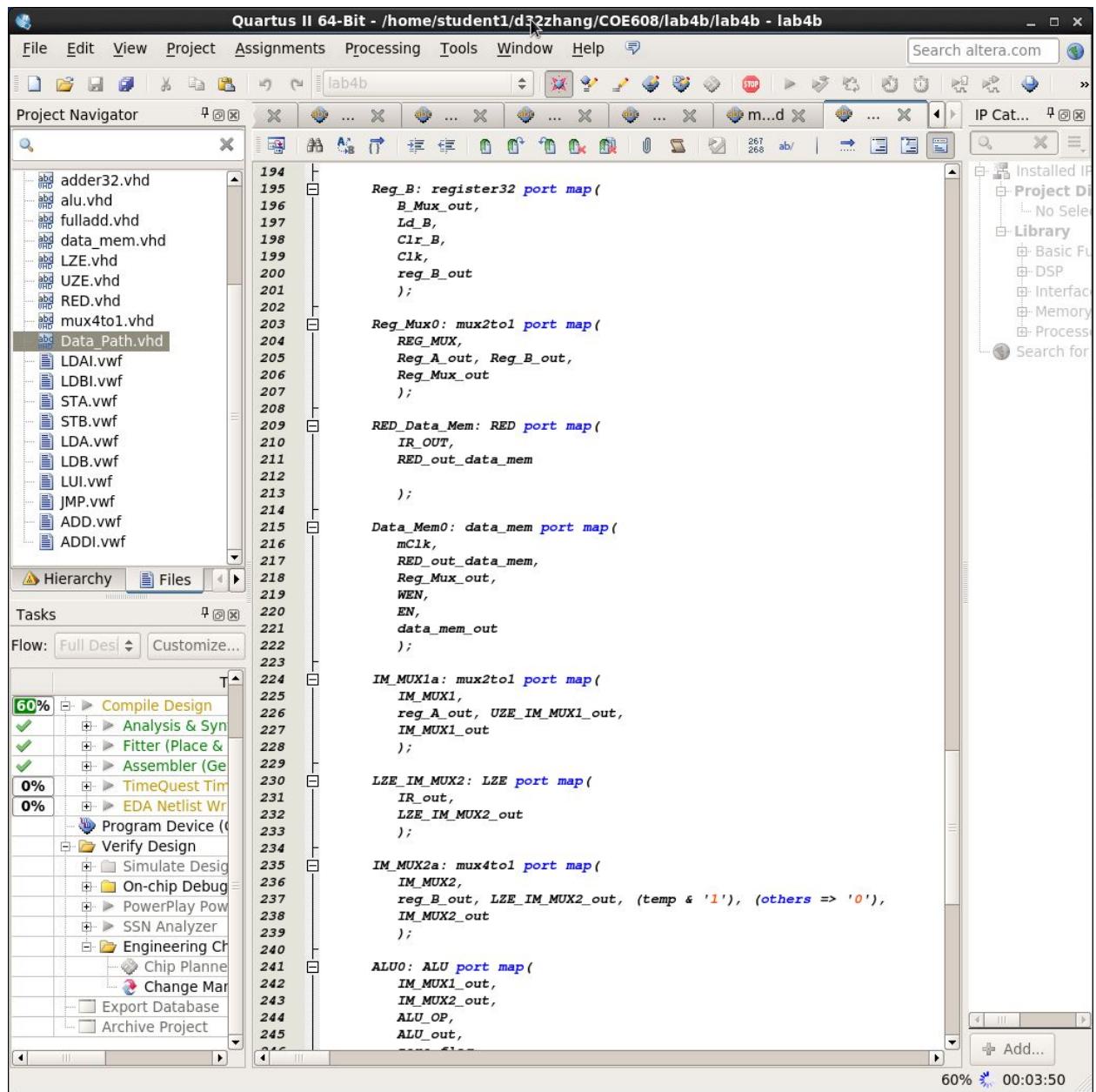
```

88
89 component RED is
90   port(   RED_in : in std_logic_vector(31 downto 0);
91           RED_out : out unsigned(7 downto 0)
92         );
93 end component;
94
95 component mux2to1 is
96   port (s : in std_logic;
97         w0,w1 : in std_logic_vector(31 downto 0);
98         f : out std_logic_vector(31 downto 0)
99       );
100 end component;
101
102 component mux4to1 is
103   port (s : in std_logic_vector(1 downto 0);
104         X1,X2,X3,X4 : in std_logic_vector(31 downto 0);
105         f : out std_logic_vector(31 downto 0)
106       );
107 end component;
108
109 component alu is
110   port(
111     a : in std_logic_vector(31 downto 0);
112     b : in std_logic_vector(31 downto 0);
113     op : in std_logic_vector(2 downto 0);
114     result: out std_logic_vector(31 downto 0);
115     zero : out std_logic;
116     cout : out std_logic
117   );
118 end component;
119
120 signal IR_OUT      : std_logic_vector(31 downto 0);
121 signal data_bus_s  : std_logic_vector(31 downto 0);
122 signal LZE_out_PC  : std_logic_vector(31 downto 0);
123 signal LZE_out_A_Mux : std_logic_vector(31 downto 0);
124 signal LZE_out_B_Mux : std_logic_vector(31 downto 0);
125 signal RED_out_Data_Mem : unsigned (7 downto 0);
126 signal A_Mux_out   : std_logic_vector(31 downto 0);
127 signal B_Mux_out   : std_logic_vector(31 downto 0);
128 signal reg_A_out   : std_logic_vector(31 downto 0);
129 signal reg_B_out   : std_logic_vector(31 downto 0);
130 signal reg_Mux_out : std_logic_vector(31 downto 0);
131 signal data_mem_out : std_logic_vector(31 downto 0);
132 signal UZE_IM_MUX1_out : std_logic_vector(31 downto 0);
133 signal IM_MUX1_out  : std_logic_vector(31 downto 0);
134 signal LZE_IM_MUX2_out : std_logic_vector(31 downto 0);
135 signal IM_MUX2_out  : std_logic_vector(31 downto 0);
136 signal ALU_out      : std_logic_vector(31 downto 0);
137 signal zero_flag    : std_logic;
138 signal carry_flag   : std_logic;
139 signal temp         : std_logic_vector(30 downto 0) := (others => '0');
140

```

36% 00:03:03





Quartus II 64-Bit - /home/student1/d32zhang/COE608/lab4b/lab4b - lab4b

File Edit View Project Assignments Processing Tools Window Help

Project Navigator

adder32.vhd  
alu.vhd  
fulladd.vhd  
data\_mem.vhd  
LZE.vhd  
UZE.vhd  
RED.vhd  
mux4to1.vhd  
Data\_Path.vhd  
LDAI.vwf  
LDI.vwf  
STA.vwf  
STB.vwf  
LDA.vwf  
LDB.vwf

Hierarchy Files

Tasks

Flow: Full Desi Customize...

85% ▶ Compile Design  
✓ ▶ Analysis & Syn  
✓ ▶ Fitter (Place &  
✓ ▶ Assembler (Ge  
✓ ▶ TimeQuest Tim  
25% ▶ EDA Netlist Wr  
Program Device (O  
Verify Design  
Simulate Design  
On-chip Debug  
PowerPlay Pow  
SSN Analyzer  
Engineering Ch

```
221      data_mem_out
222      );
223
224      IM_MUX1a: mux2to1 port map(
225          IM_MUX1,
226          reg_A_out, UZE_IM_MUX1_out,
227          IM_MUX1_out
228      );
229
230      LZE_IM_MUX2: LZE port map(
231          IR_out,
232          LZE_IM_MUX2_out
233      );
234
235      IM_MUX2a: mux4to1 port map(
236          IM_MUX2,
237          reg_B_out, LZE_IM_MUX2_out, (temp & '1'), (others => '0'),
238          IM_MUX2_out
239      );
240
241      ALU0: ALU port map(
242          IM_MUX1_out,
243          IM_MUX2_out,
244          ALU_OP,
245          ALU_out,
246          zero_flag,
247          carry_flag
248      );
249
250      DATA_MUX0: mux4to1 port map(
251          DATA_MUX,
252          DATA_IN, data_mem_out, ALU_out, (others => '0'), data_bus_s
253      );
254
255      DATA_BUS <= data_bus_s;
256
257
258
259
260
261
262
```

85% 00:04:09

## How does this data-path implement the INCA, ADDI, LDBI and LDA operations?

For INCA operation, we first load LD\_A,LD\_C,LD\_Z, assign A MUX to 0, Data MUX to 10, IM\_MUX1 TO 0, IM\_MUX2 to 10, and then go to the operation code “010” in alu, alu calculates the assigned value to produce the result.

For ADDI operation, we first load LD\_A,LD\_C,LD\_Z, assign A MUX to 0, Data MUX to 10, IM\_MUX1 TO 0, IM\_MUX2 to 01, and then go to the operation code “010” in alu, alu calculates the assigned value to produce the result.

For LDBI operation, we first load LD\_B, assign B MUX to 1.

For ADDI operation, we first load LD\_A, assign EN to 1, assign WEN to 0, assign A MUX to 0, and Data MUX to 01.

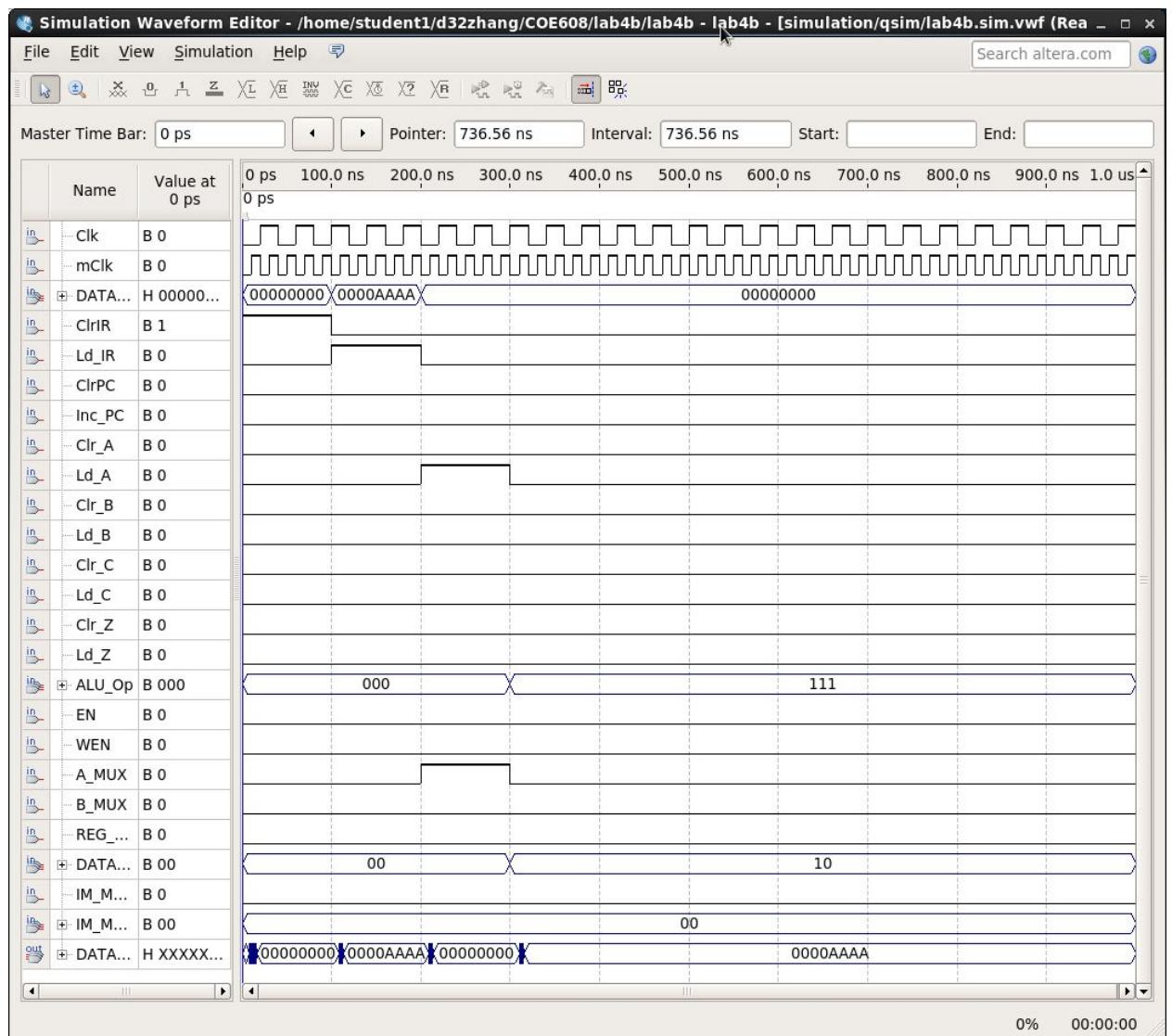
## The data-path has a maximum reliable operating speed (Clk). What determines this speed? (i.e. how would you estimate the data-path circuit clock)?

Delays determine the maximum reliable operating speed, every time a different input was passed in, the corresponding result was generated with some delays, delays may be different than others.

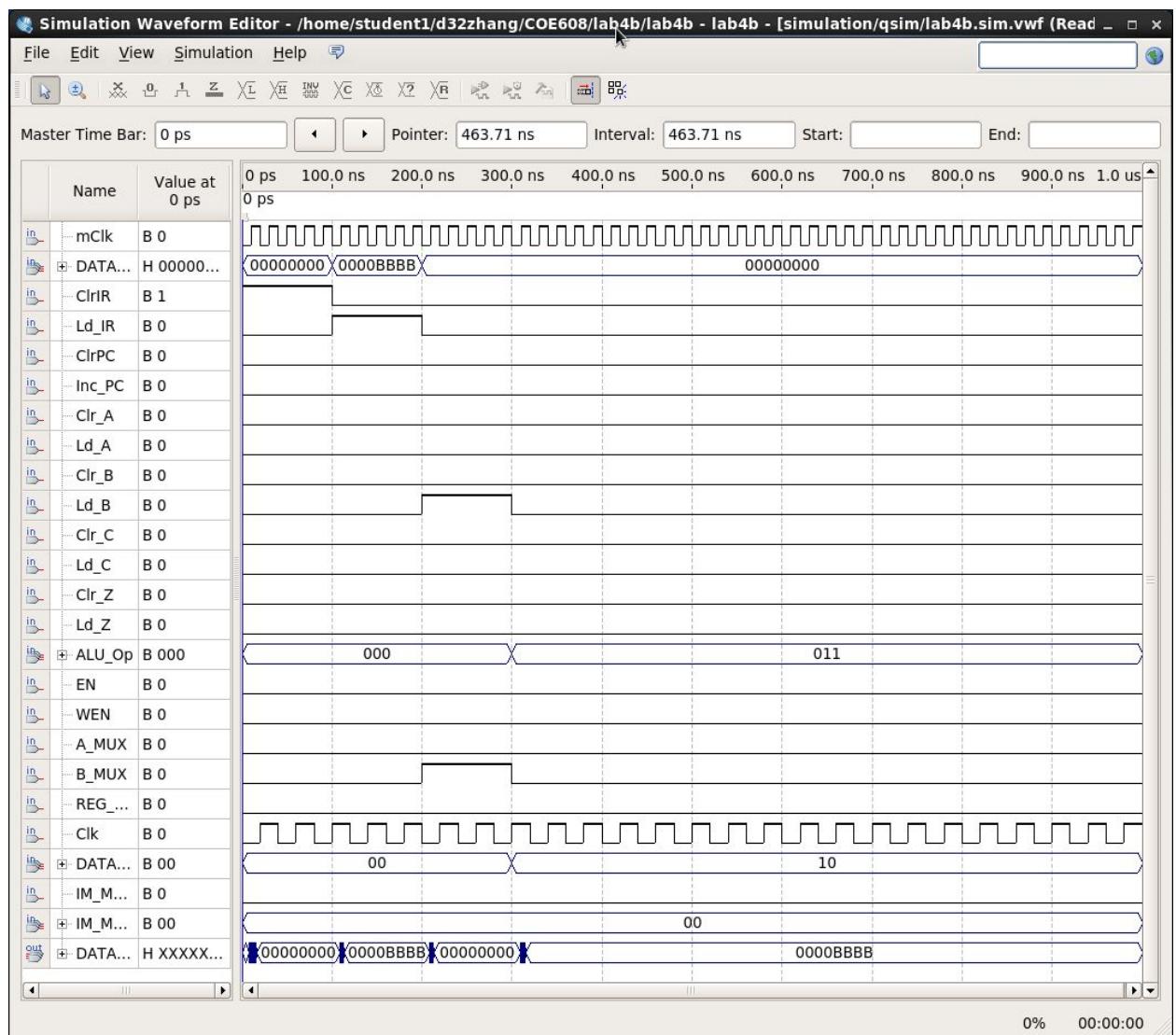
## What is a reliable limit for your data-path clock?

The reliable limit for my data-path clock was 8.806.

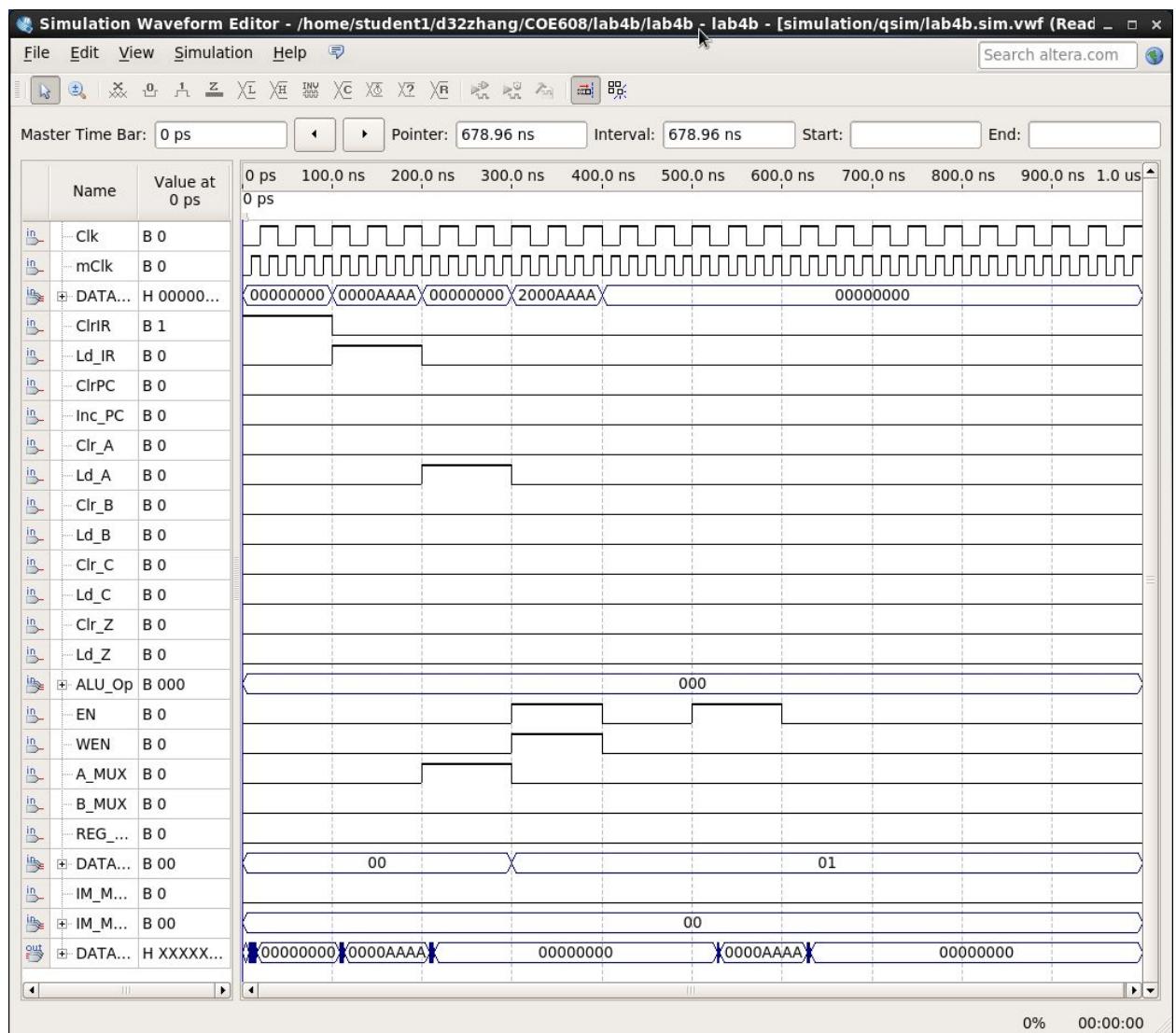
## 16. LDAI.vwf



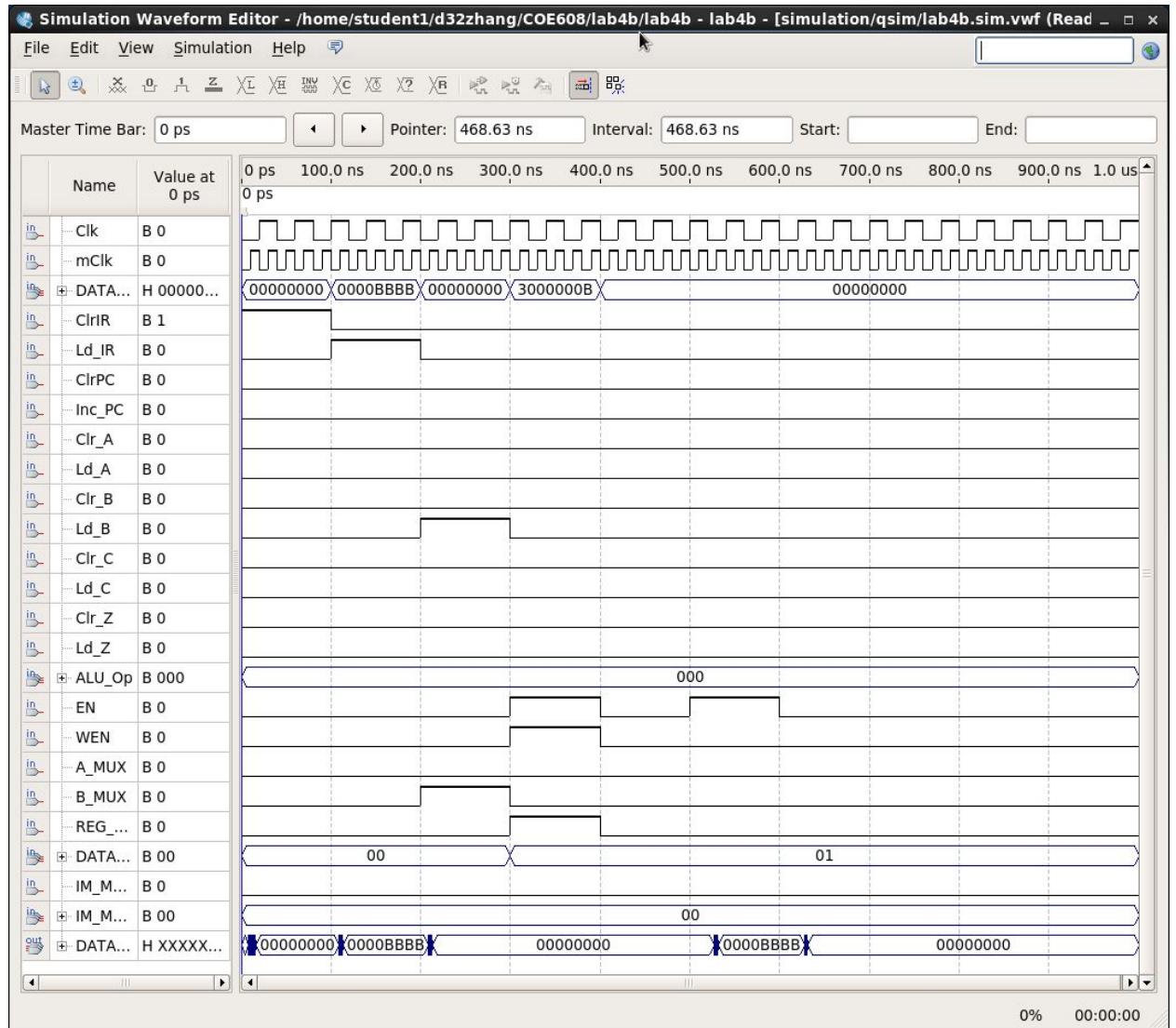
## 17. LDBI.vwf



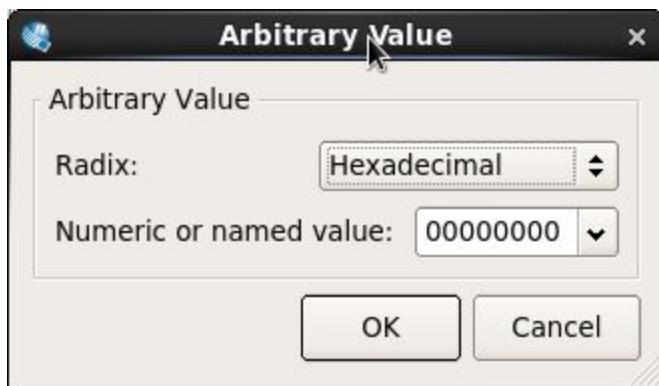
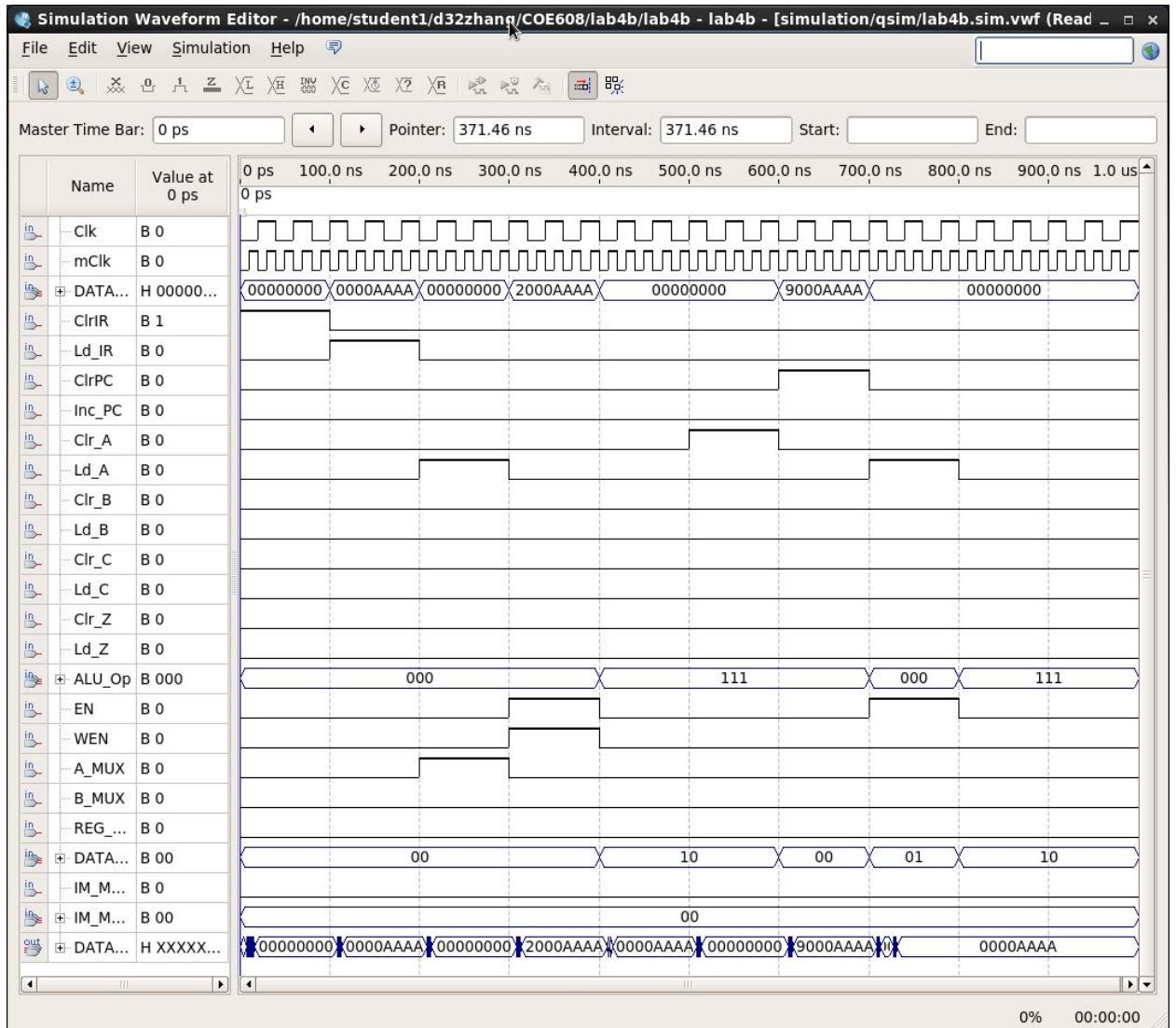
## 18. STA.vwf



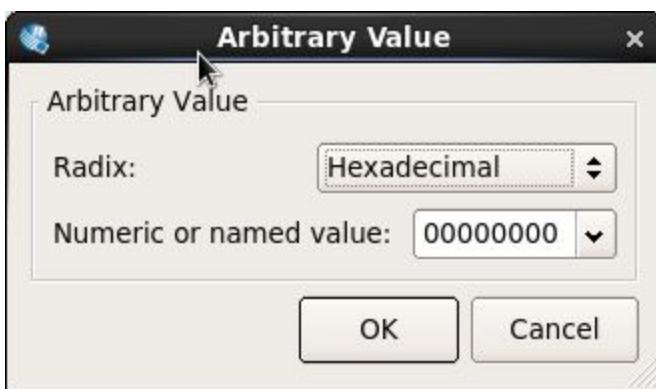
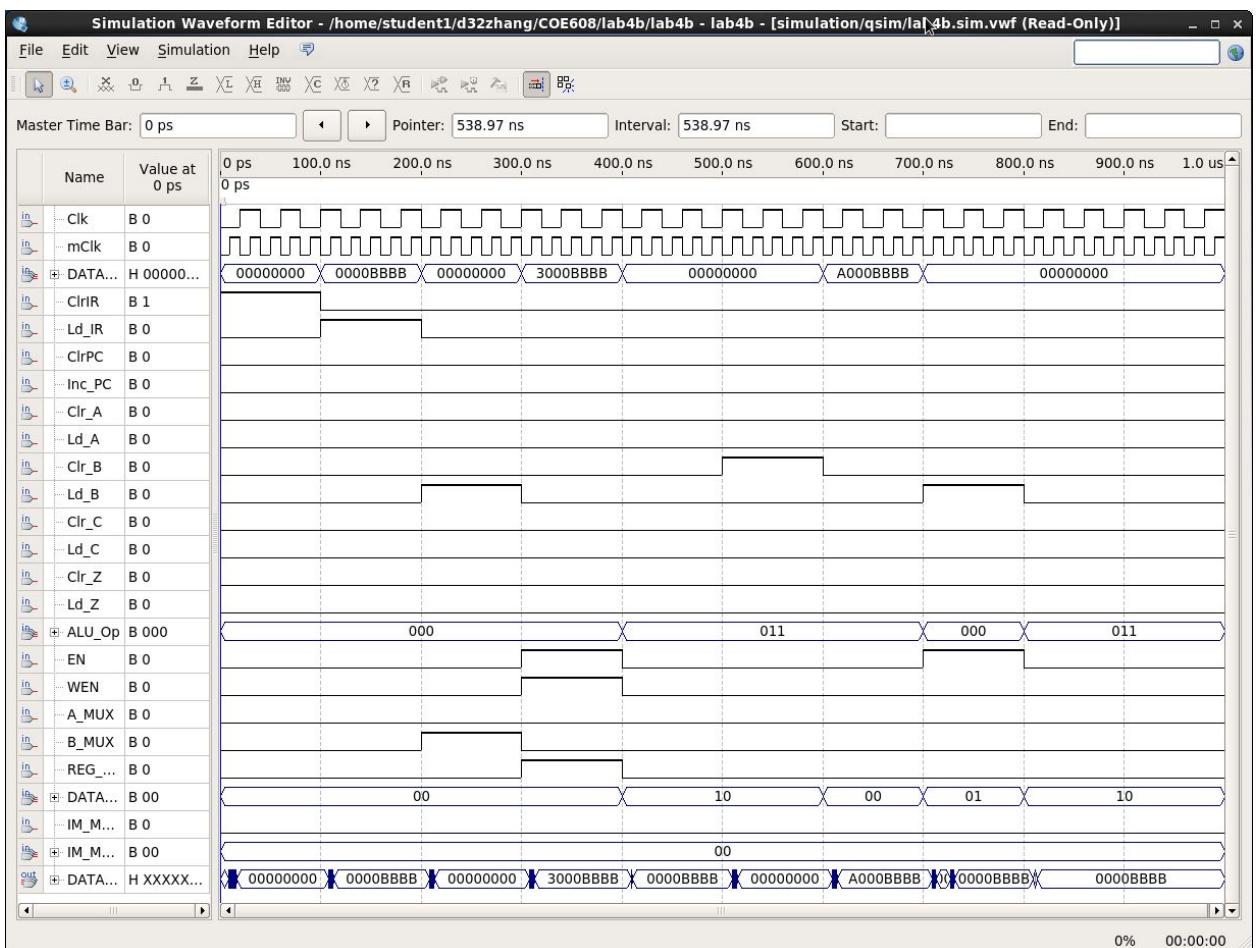
## 19. STB.vwf



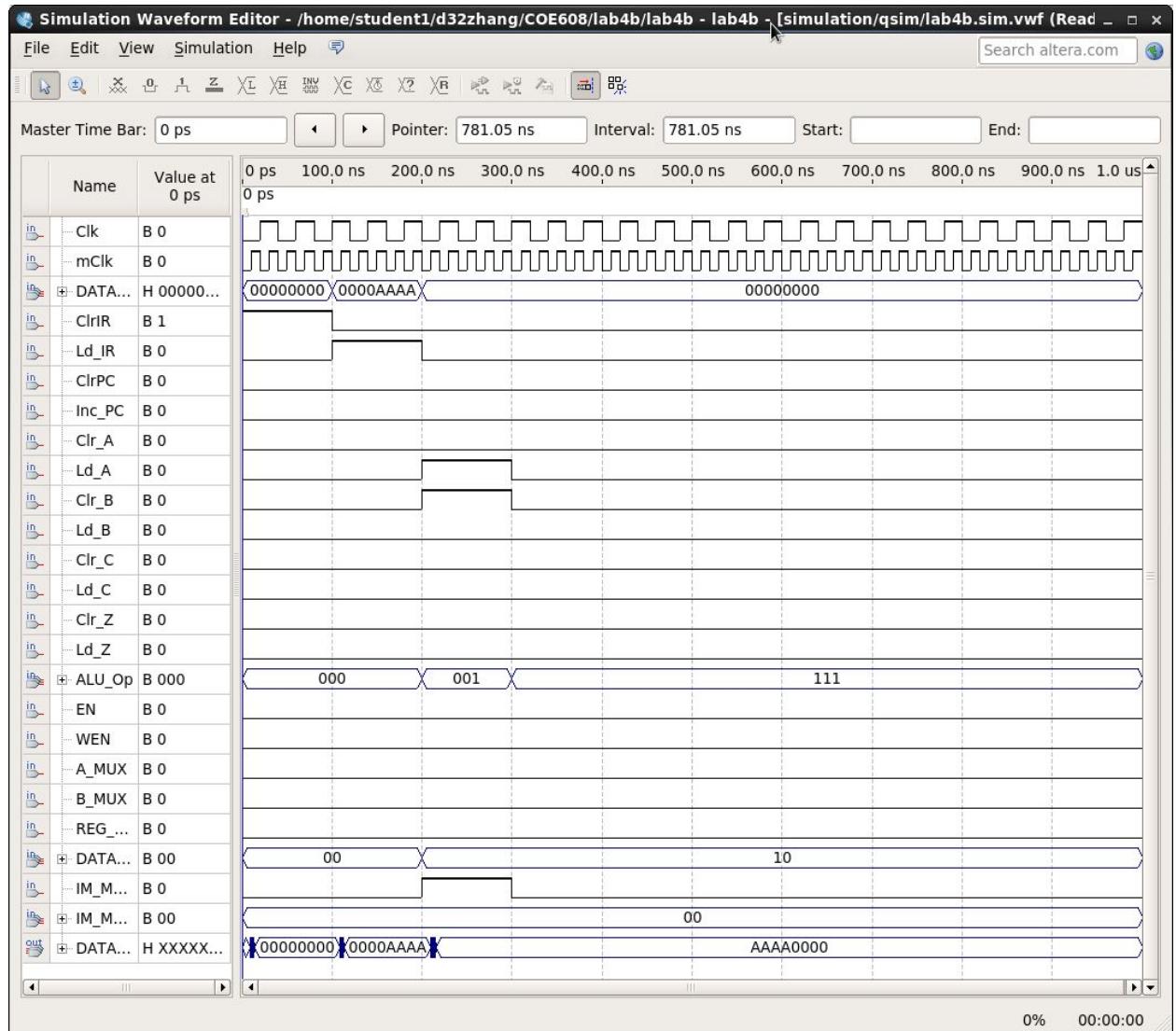
## 20. LDA.vwf



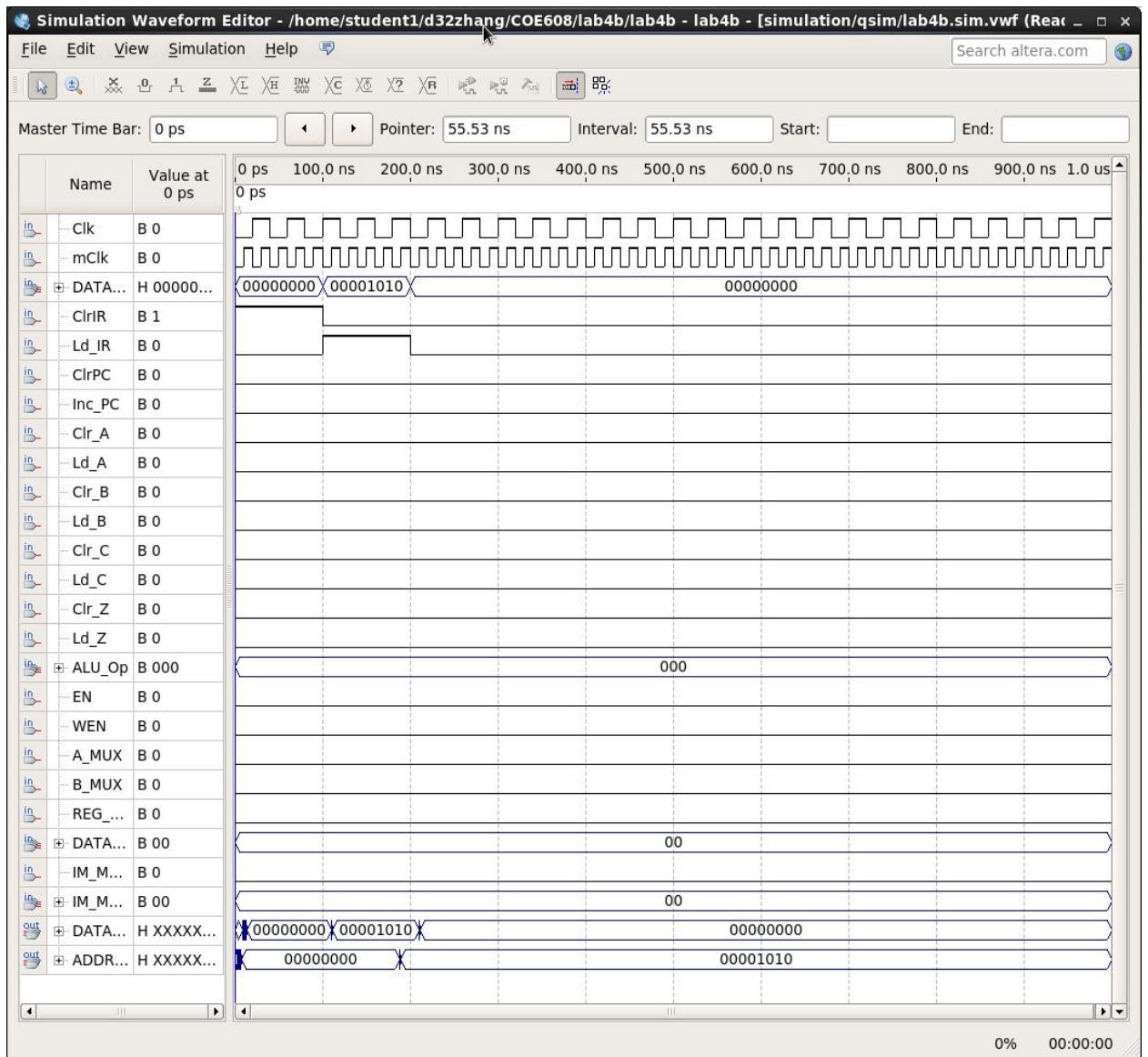
## 21. LDB



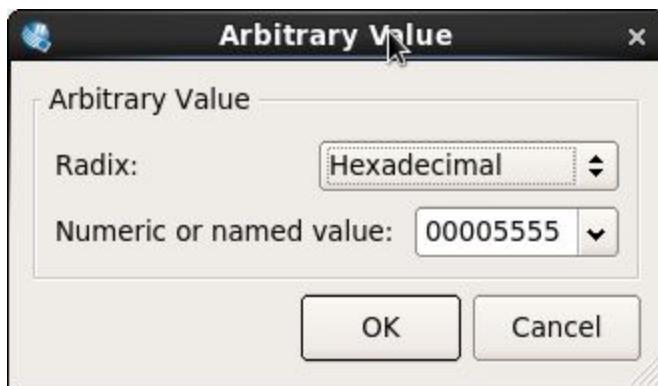
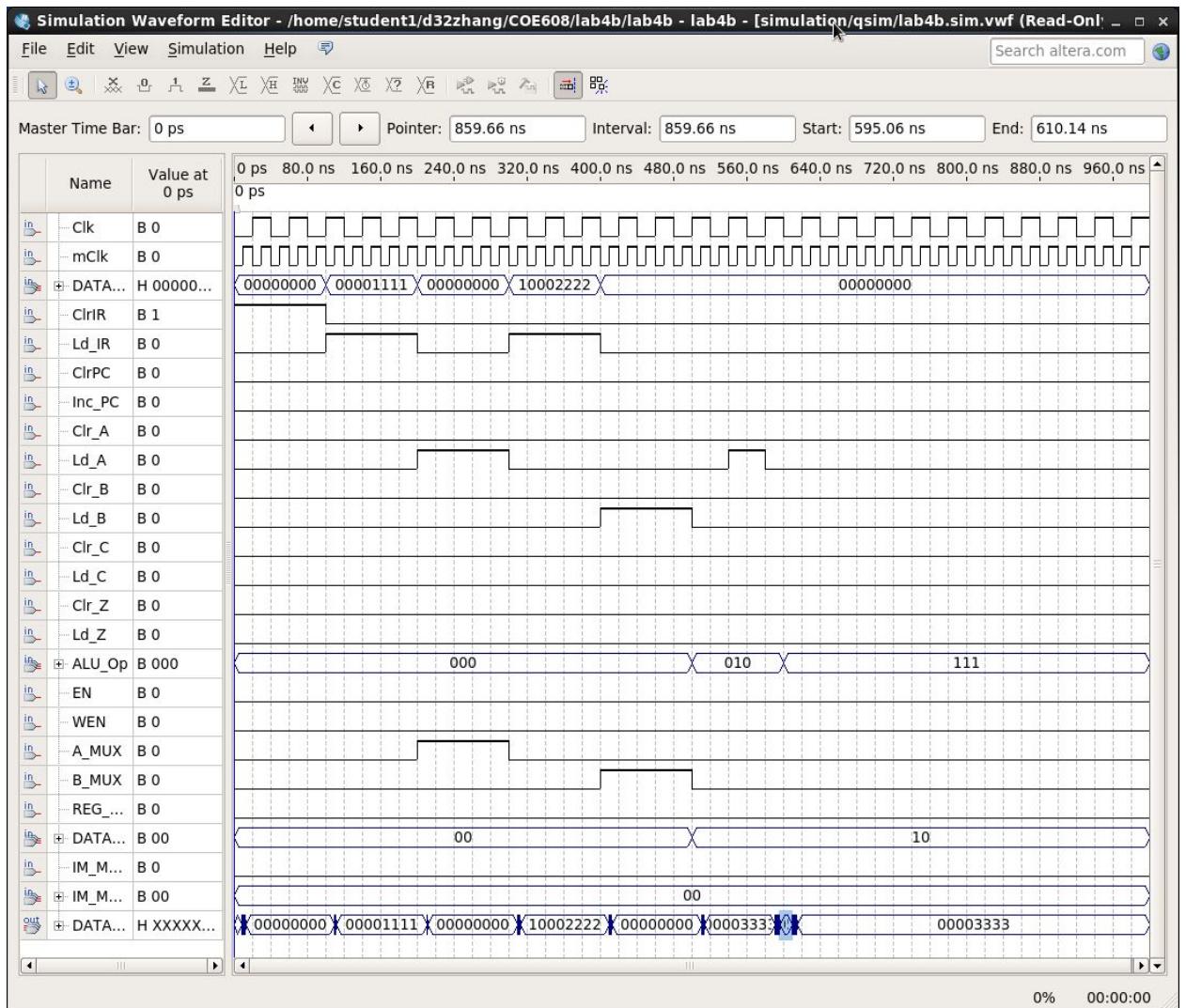
## 22. LUI



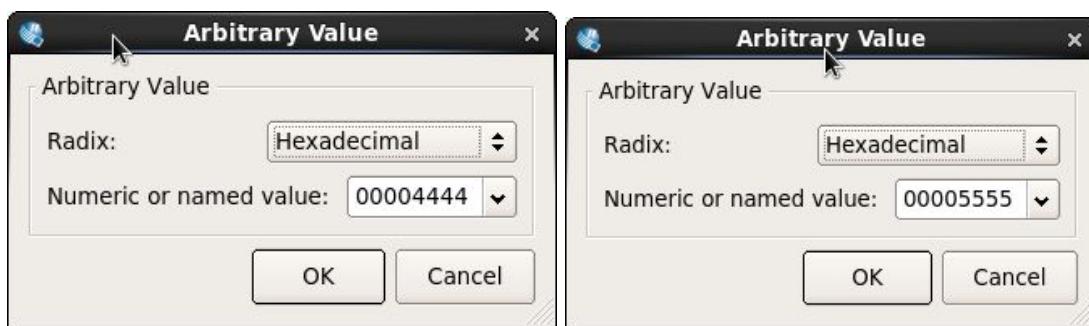
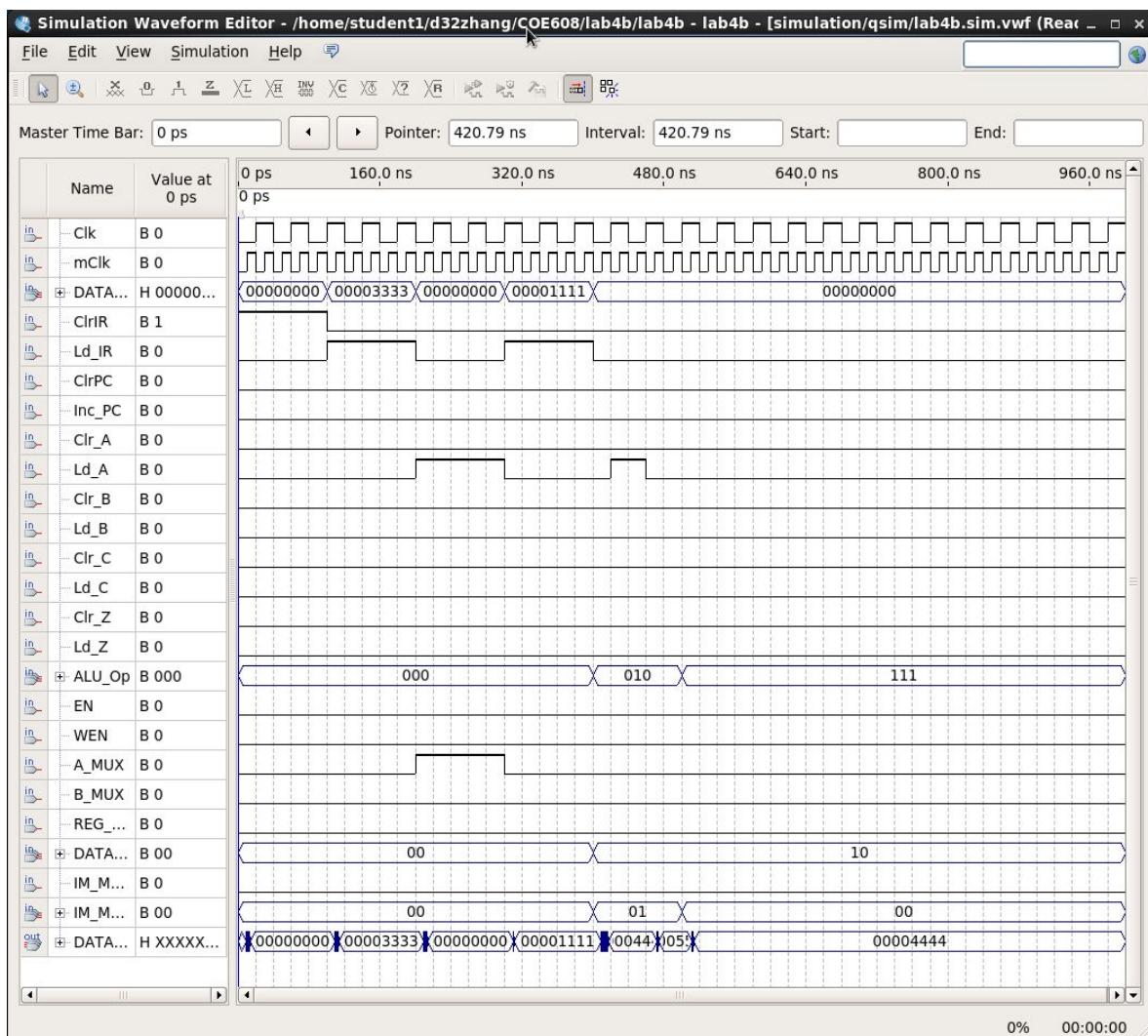
## 23. JMP



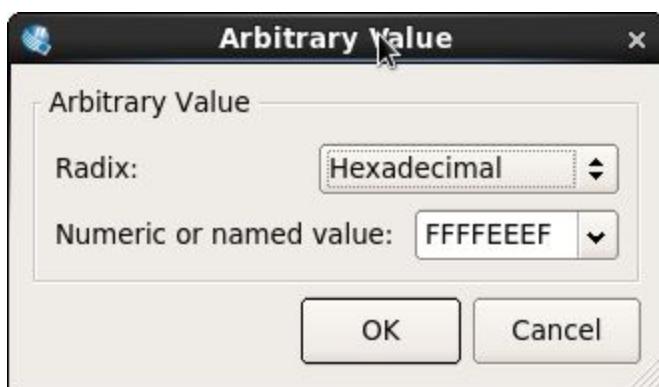
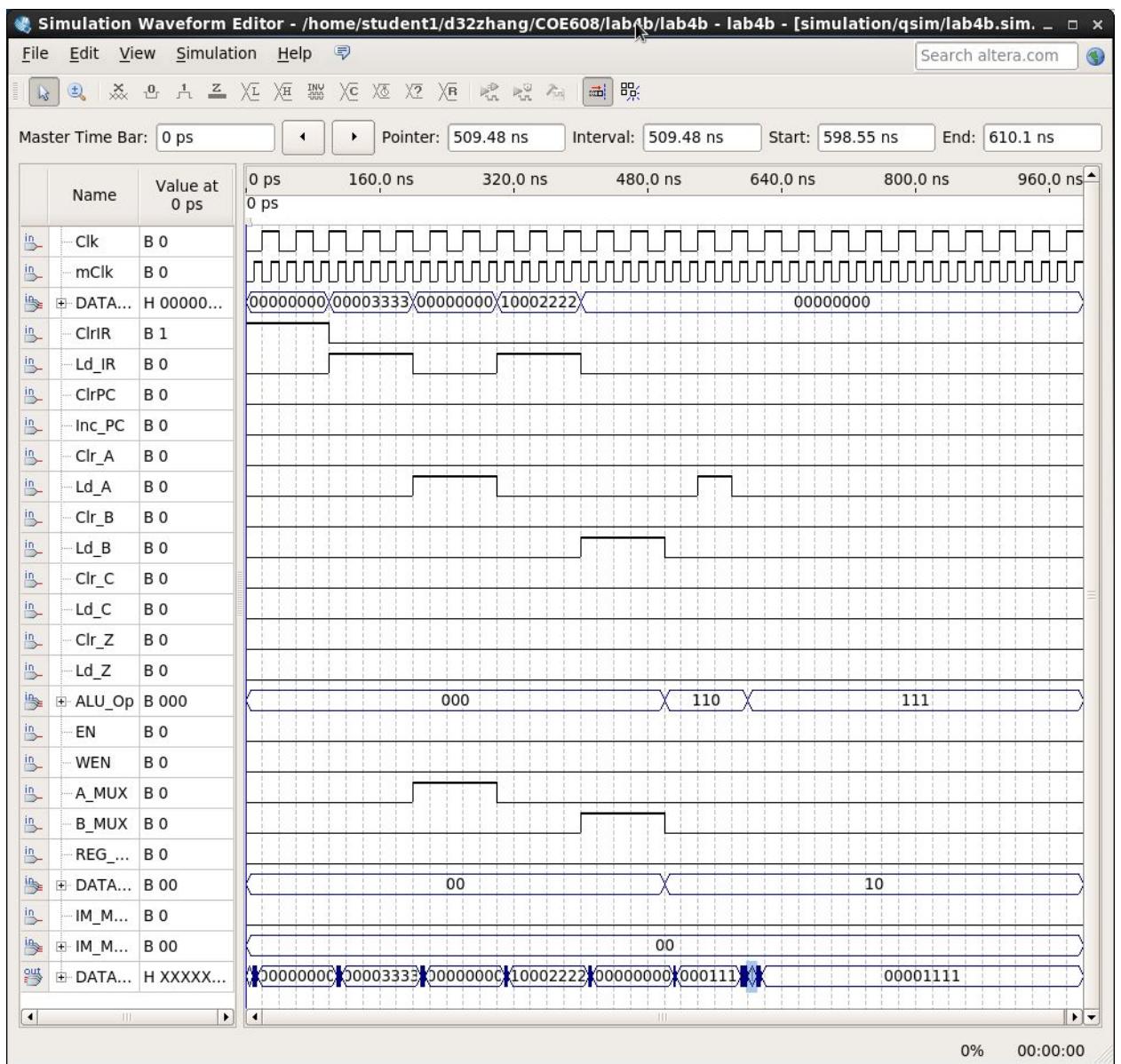
## 24. ADD



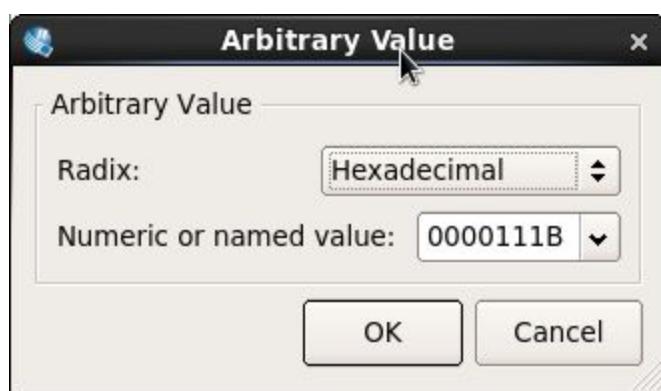
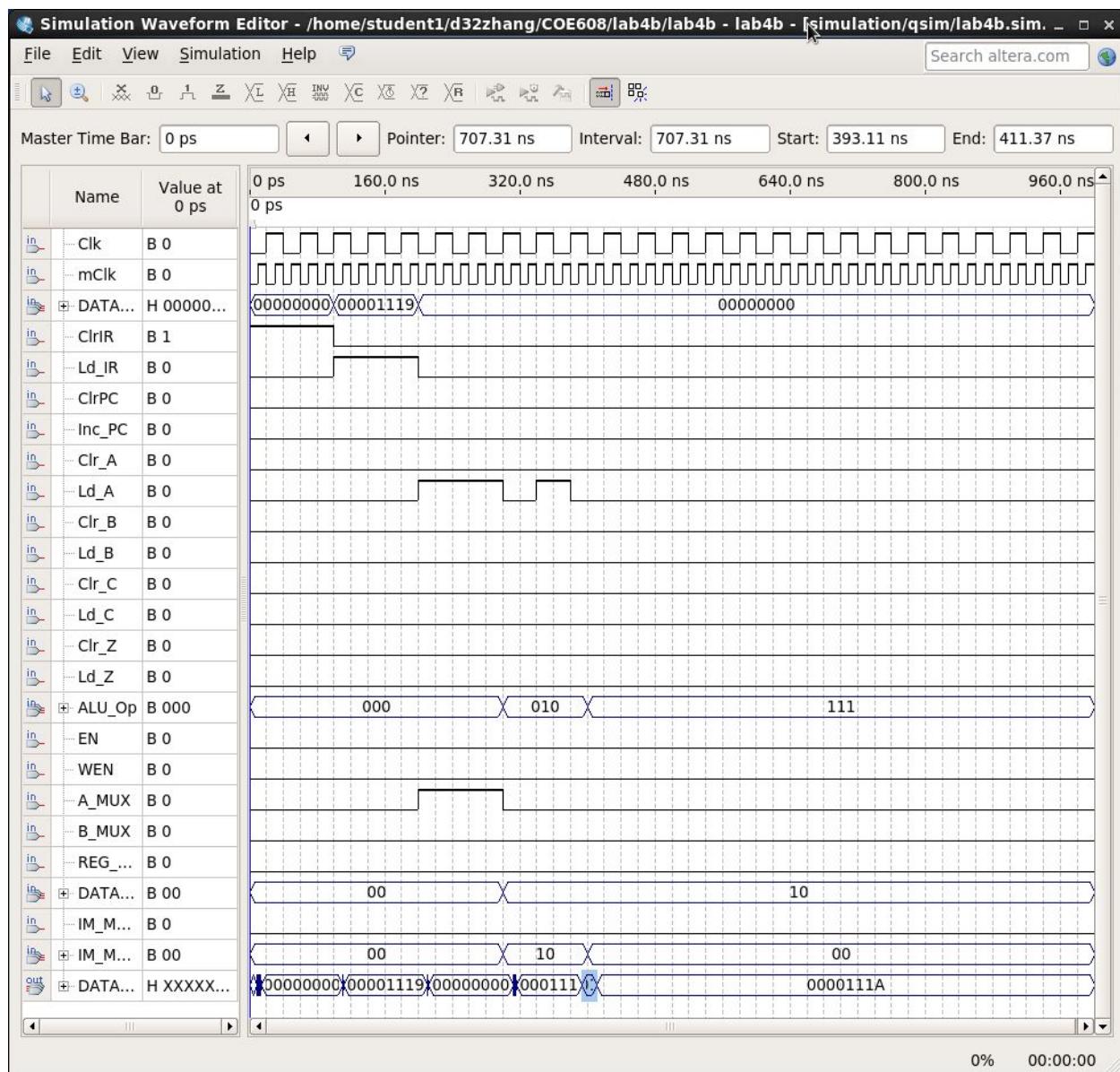
## 25. ADDI



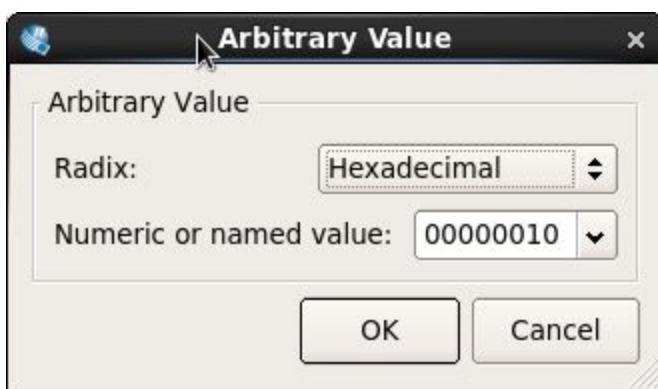
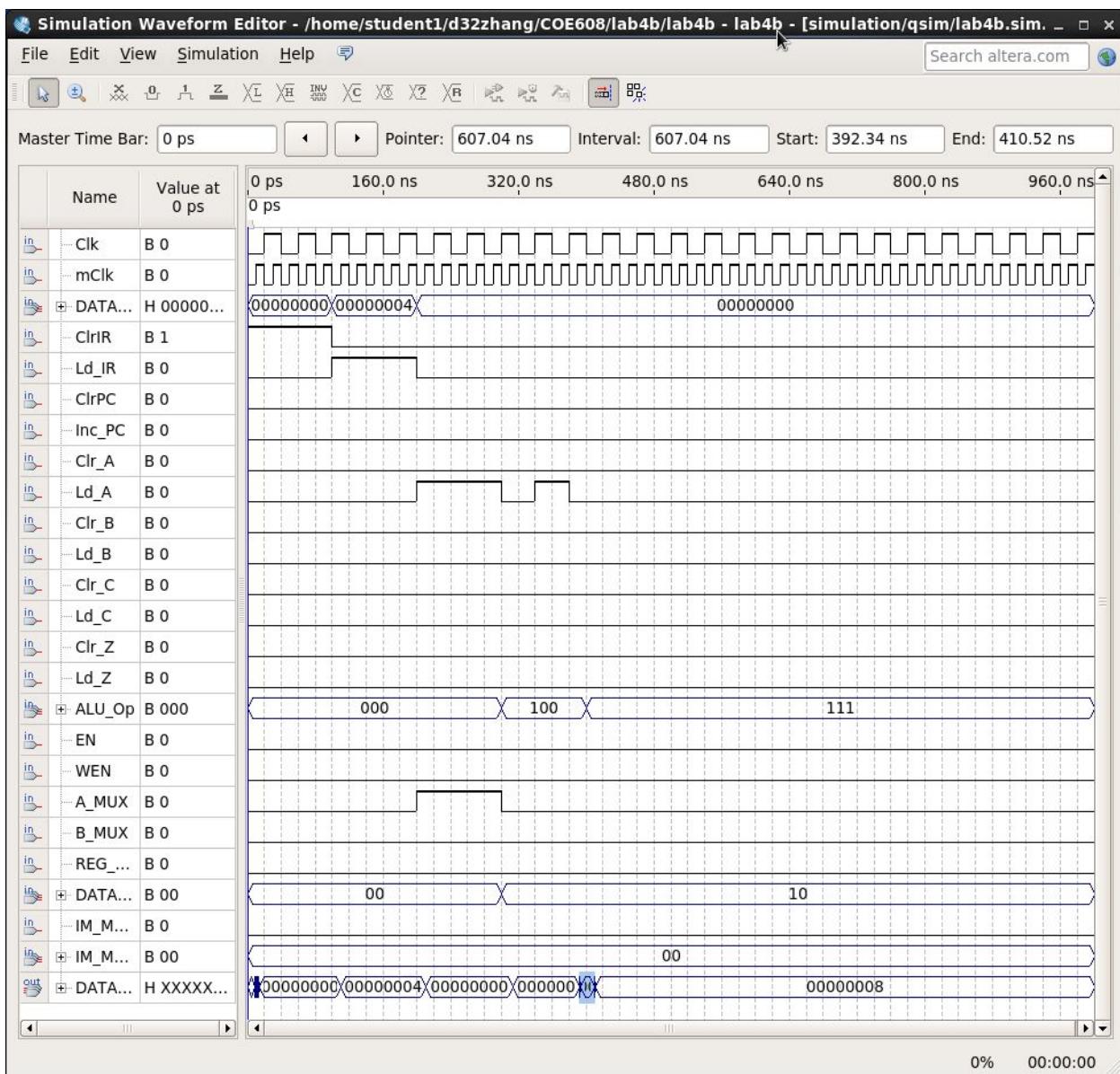
## 26. SUB



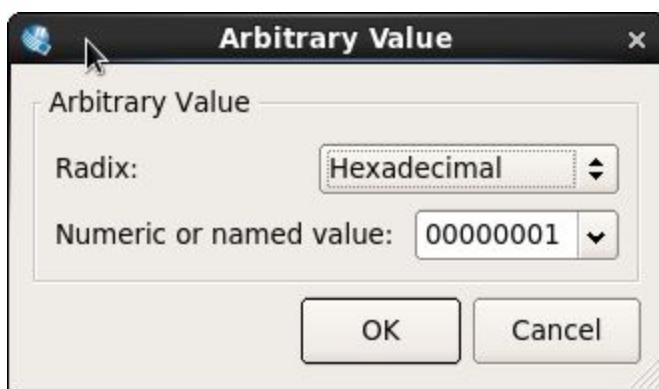
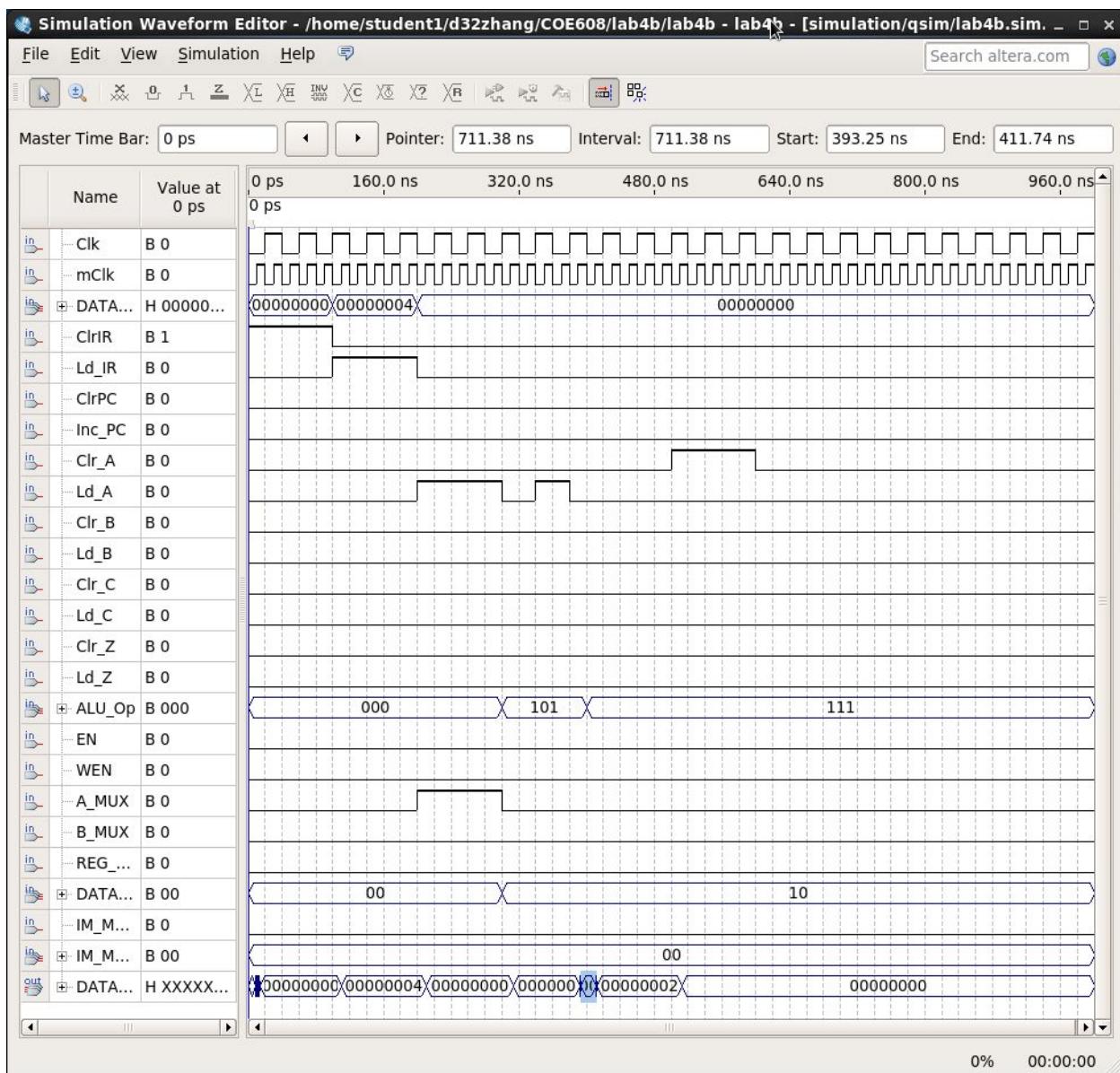
## 27. INCA



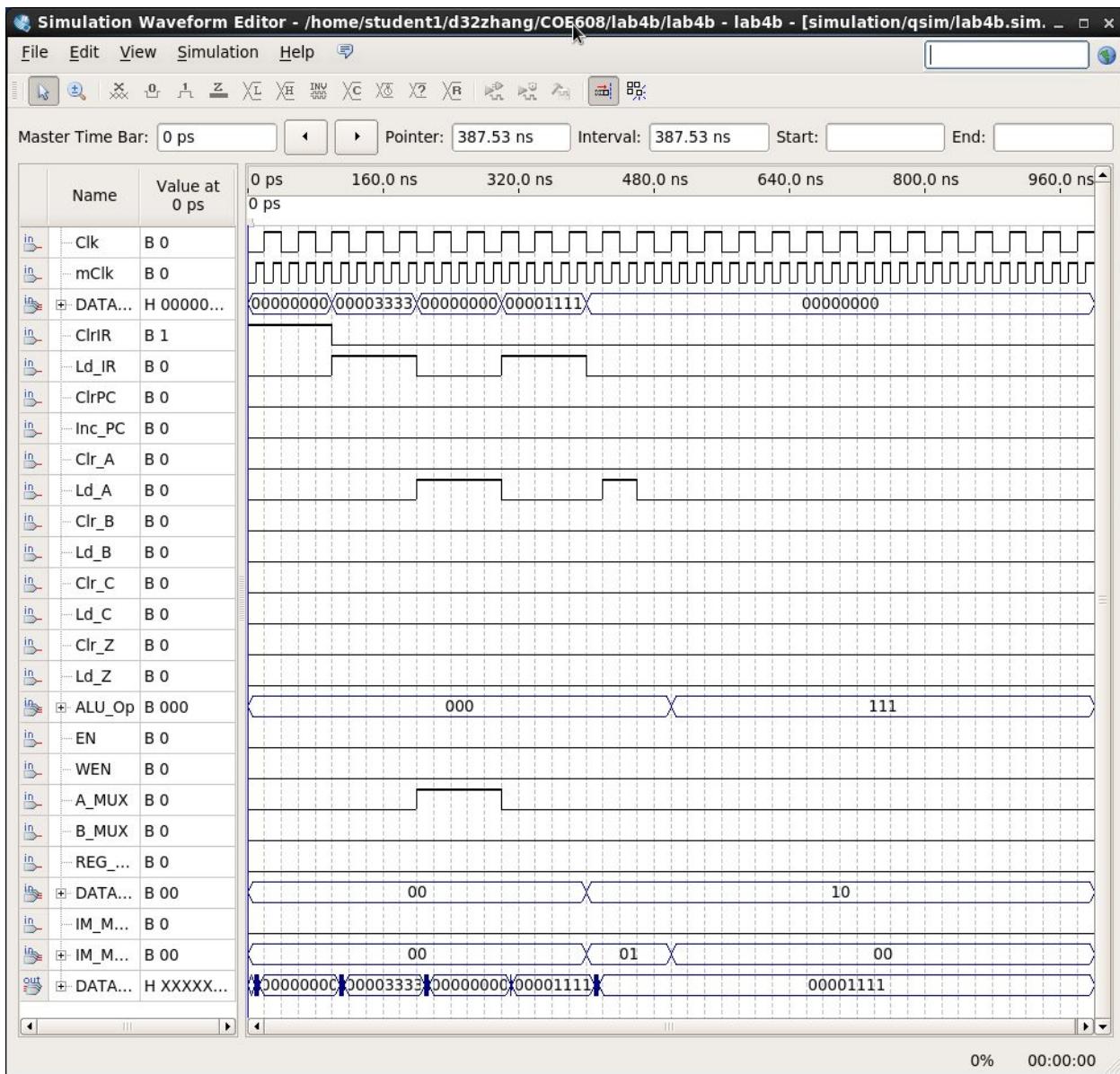
## 28. ROL



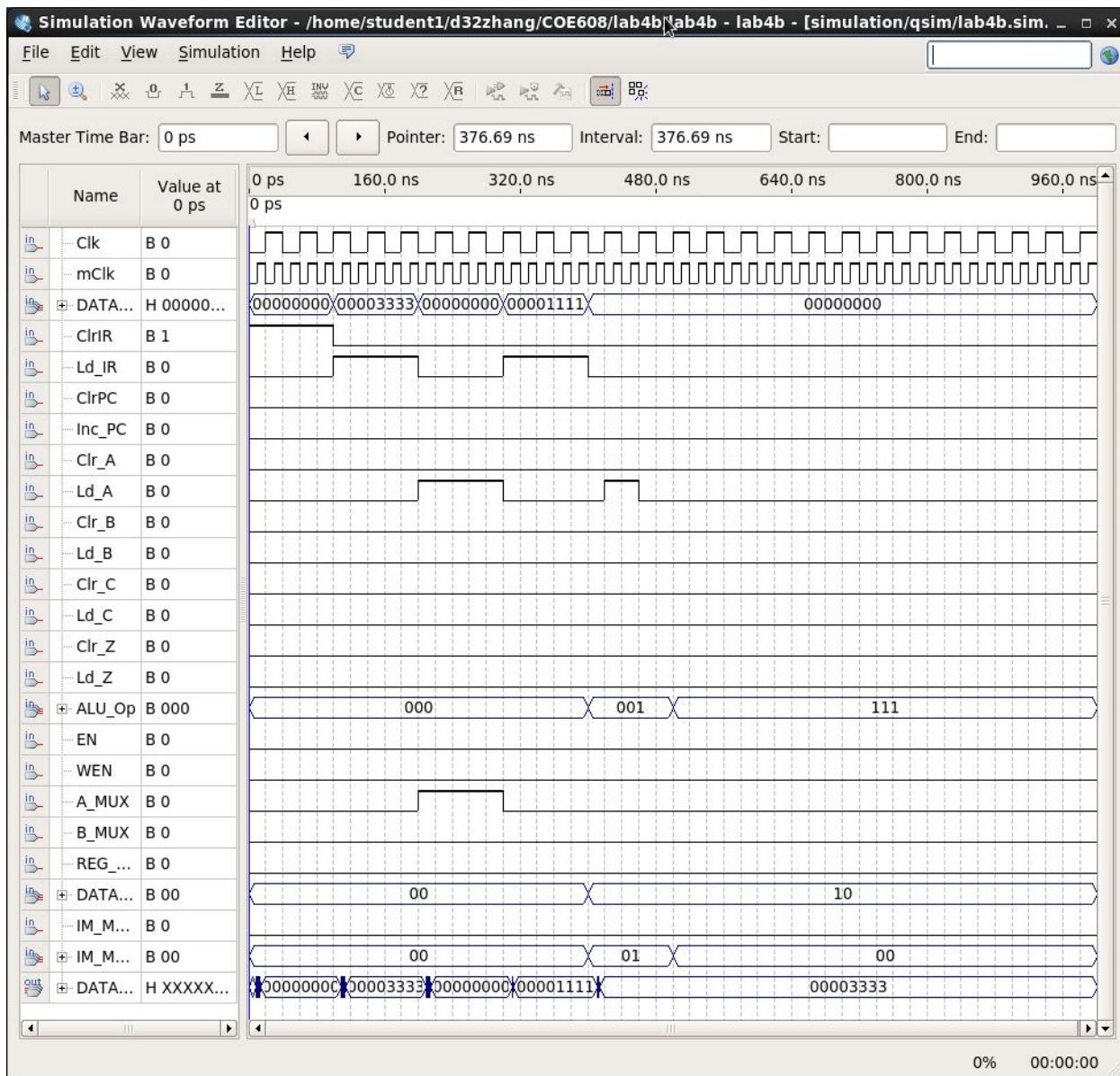
## 29. CLRA



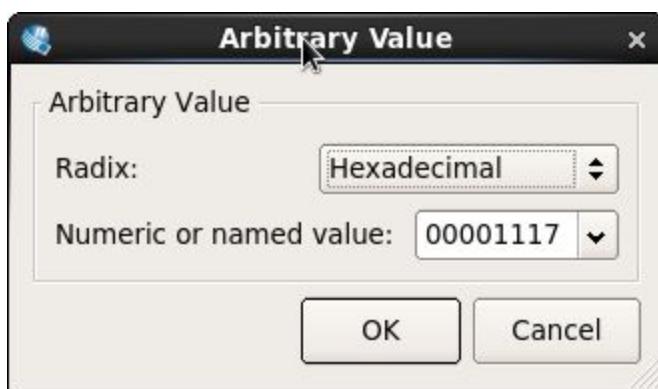
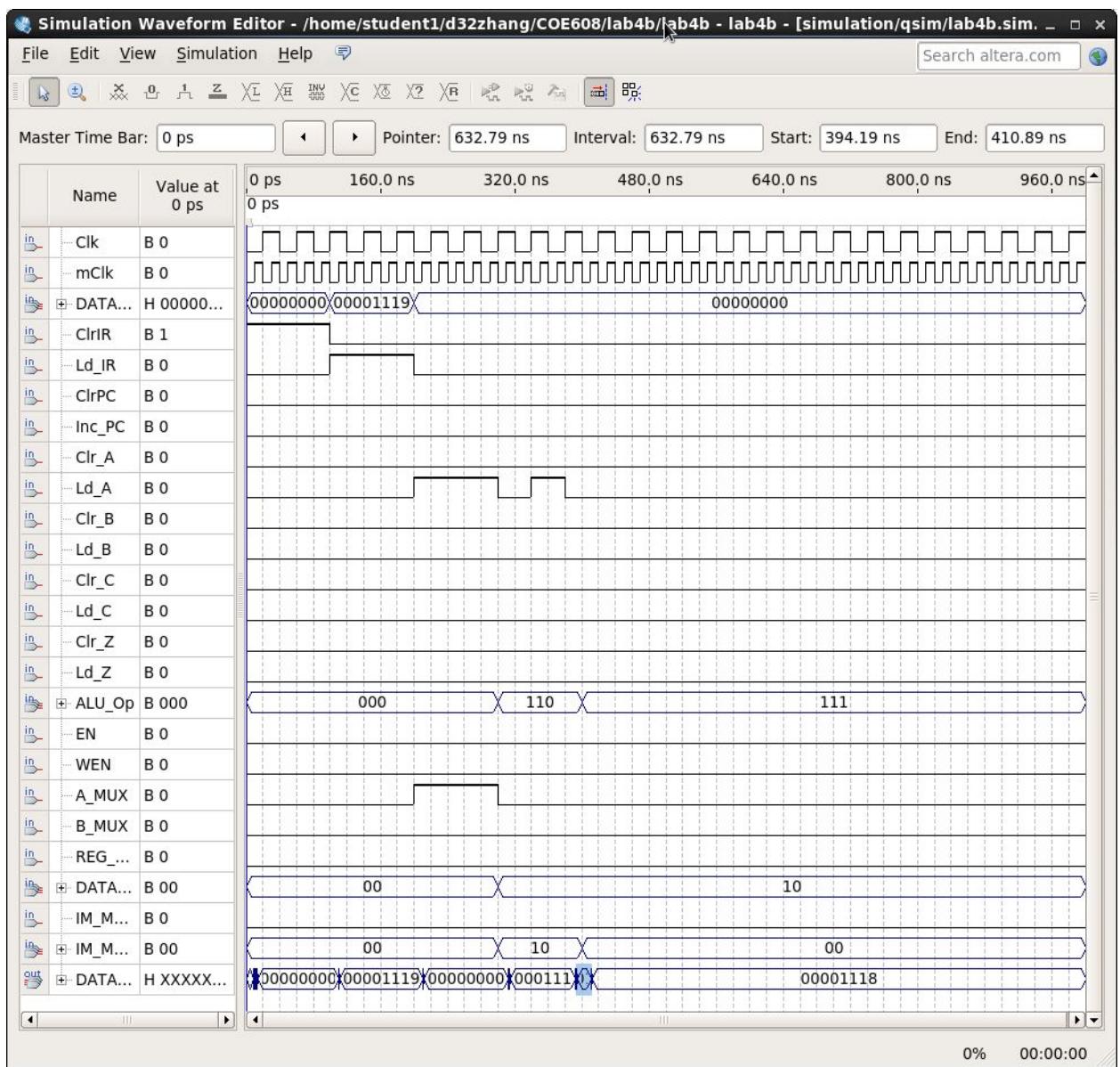
### 30. ANDI



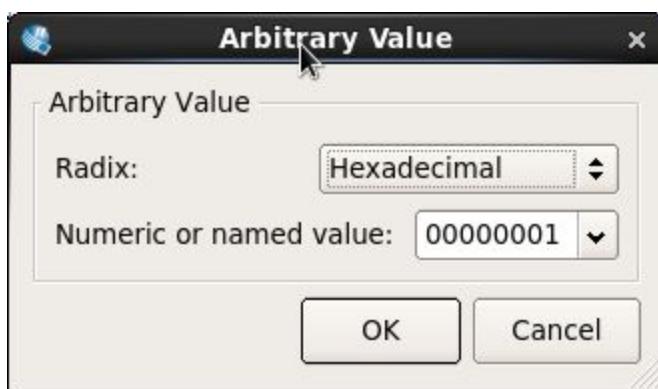
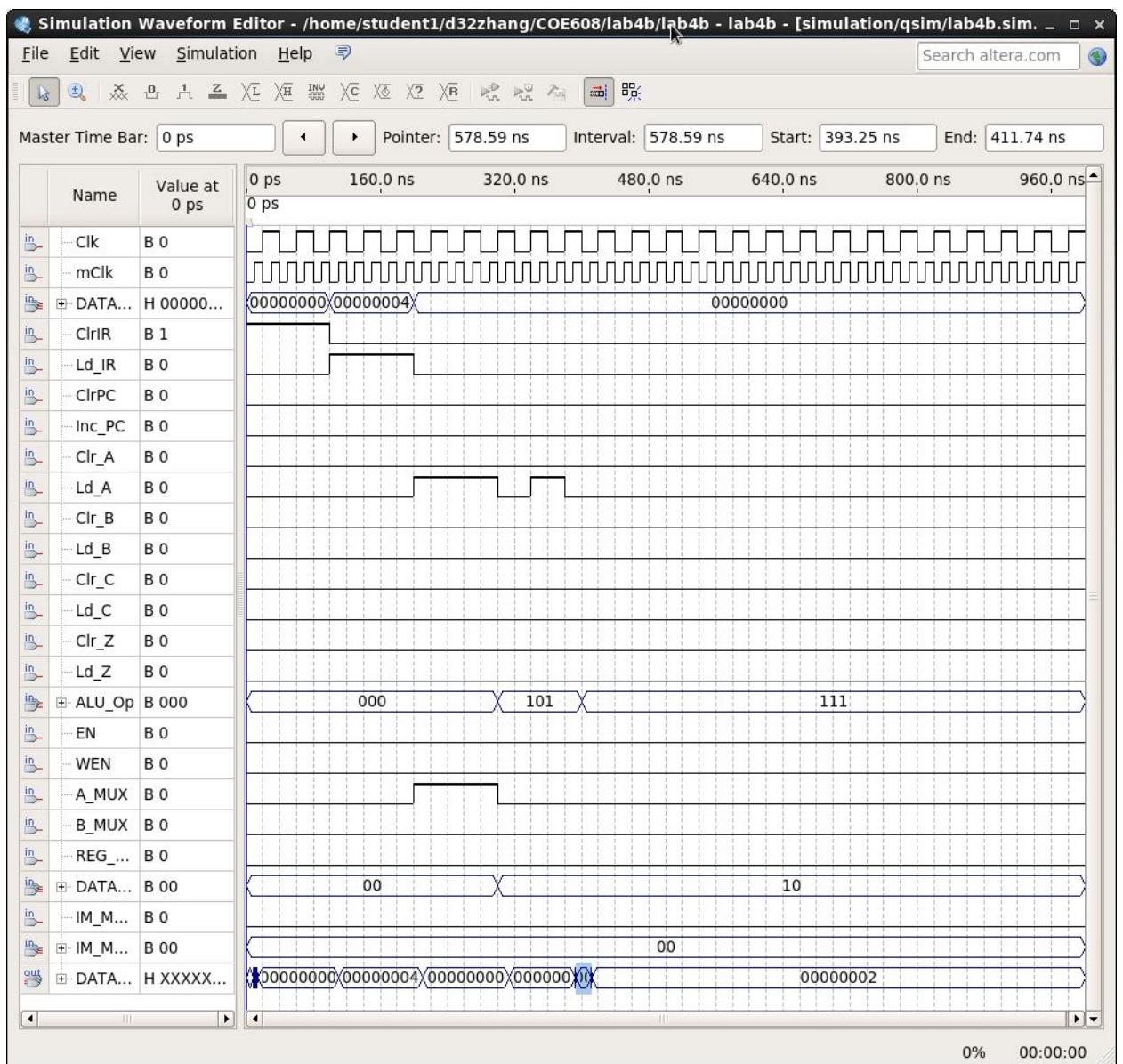
### 31. ORI



## 32. DECA



### 33. ROR



Student Name: Duanwei Zhang Student#: 500824903 Section: 04

INST	CLR_IR LD_IR	LD_PC INC_PC	CLR_A LD_A	CLR_B LD_B	CLR_C LD_C	CLR_Z LD_Z	ALU OP	EN WEN	A/B MUX	REG MUX	Data MUX	IM_MUX1 IM_MUX2
LDA	0/0	0/0	0/1	0/0	0/0	0/0	XXX	1/0	0/X	X	01	X
LDB	0/0	0/0	0/0	0/1	0/0	0/0	XXX	1/0	X/0	X	01	X
STA	0/0	0/0	0/0	0/0	0/0	0/0	XXX	1/1	X	0	X	X
STB	0/0	0/0	0/0	0/0	0/0	0/0	XXX	1/1	X	1	X	X
JMP	0/0	1/0	0/0	0/0	0/0	0/0	XXX	X	X	X	X	X
LDAI	0/0	0/0	0/1	0/0	0/0	0/0	XXX	X	1/X	X	X	X
LDBI	0/0	0/0	0/0	0/1	0/0	0/0	XXX	X	X/1	X	X	X
LUI	0/0	0/0	0/1	1/0	0/0	0/0	001	X	0/X	X	10	1/X
ANDI	0/0	0/0	0/1	0/0	0/1	0/1	000	X	0/X	X	10	0/01
DECA	0/0	0/0	0/1	0/0	0/1	0/1	110	X	0/X	X	10	0/10
ADD	0/0	0/0	0/1	0/0	0/1	0/1	010	X	0/X	X	10	0/00
SUB	0/0	0/0	0/1	0/0	0/1	0/1	110	X	0/X	Y	10	0/00
INCA	0/0	0/0	0/1	0/0	0/1	0/1	010	Y	0/X	X	10	0/10
AND	0/0	0/0	0/1	0/0	0/1	0/1	000	X	0/X	X	10	0/00
ADDI	0/0	0/0	0/1	0/0	0/1	0/1	010	Y	0/X	X	10	0/01
ORI	0/0	0/0	0/1	0/0	0/1	0/1	001	X	0/X	X	10	0/01
ROL	0/0	0/0	0/1	0/0	0/1	0/1	100	Y	0/X	X	10	0/X
ROR	0/0	0/0	0/1	0/0	0/1	0/1	101	Y	0/X	X	10	0/X
CLRA	0/0	0/0	1/0	0/0	0/0	0/0	XXX	X	X	Y	X	X
CLRB	0/0	0/0	0/0	1/0	0/0	0/0	XXX	X	Y	X	X	X
CLRC	0/0	0/0	0/0	0/0	1/0	0/0	XXX	Y	Y	X	X	X
CLRZ	0/0	0/0	0/0	0/0	0/0	1/0	XXX	X	X	X	X	X
PC <= PC+4	0/0	1/1	0/0	0/0	0/0	0/0	XXX	X	Y	X	X	X
IR <= M[INST]	0/1	0/0	0/0	0/0	0/0	0/0	XXX	X	Y	X	00	X
PC <= IR[15..0]	0/0	1/0	0/0	0/0	0/0	0/0	XXX	X	X	X	Y	X

Table 1: Data-Path Control Signals