

1. Introduction

In order to design a microprocessor that supports the MIPS instruction set, fundamental components must first be created in preparation for construction of a datapath and controller. This assignment focuses on creating a 1-bit, 5-bit, and 32-bit multiplexer, a 32-bit register, and a sign and zero extender that will later be connected to read R, I, and J-type instructions.

2. Design and Testing:

A multiplexer is a component that selects one signal from several signals, based on "select" input signals and outputs one signal. Figure 1.1 shows the internal design of a 2-to-1 multiplexer. The "select" input signal is decoded and ANDed with each input signal and then ORed with the outputs of each AND gate, giving the selected output signal. Figure 1.2 displays the truth table for the 2-to-1 multiplexer. Figure 1.3 is the VHDL code for a 1-bit mux and figure 1.4 is the functional simulation waveform. Using the Cyclone II EP2C8T144C8 FPGA, the calculated propagation delay for a 1-bit mux is approximately 10.9 nanoseconds. See appendix for the VHDL code and test benches of a 5-bit and 32-bit multiplexer.



Figure 1.1



-												
1	Rachel Kreynin											
2	2 to 1 Mux											
3	library ieee;											
4	<pre>use ieee.std_logic_1164.all;</pre>											
5	<pre>use ieee.numeric_std.all;</pre>											
6												
7	entity mux1 is											
8	port (
9	<pre>in0, in1 : IN std_logic;</pre>											
10	Sel : IN std_logic;											
11	0 : OUT std_logic											
12);											
13	end mux1;											
14												
15	architecture behavior of mux1 is											
16	-begin											
17	0 <= in0 when Sel = '0' else											
18	L in1;											
19	end behavior;											

_ <mark>∕⊉</mark> +	Msgs										7
🔶 in0	-No Dat										
🔶 in 1	-No Dat										
i 🔶 sel	-No Dat										
🔷 🔷 out0	-No Dati						The sele	-4 :4			
		The select is to zero, so t output takes the value of input 0.	set he on		The select is set to '1', so the mux ouputs input '		ine sele signal se the new of input outputs	ct input ects value 0 and 1'		The select is to '1' so input value is the output = '0'	set 1's
🛎 📰 🟵 Now	200 ns	ilil)ns 20	ns 40	lıl İns	- I - I - 60 ns 80	ns 100	lılı)ns 120	lılı)ns 140	lılı Dıns 16	lılı 60 ns 180	ins 200 i
🙃 🌽 🤤 or 1	436 ns										

Figure 1.4

Figure 1.5 shows the functional and timing simulation of a 5-bit mux, including propagation delays using a Cyclone II EP2C8T144C8 FPGA, which is measured to be about 13.5 nanoseconds. Figure 1.6 shows the functional and timing simulation of a 32-bit mux, with a propagation delay of approximately 14.7 nanoseconds. It is apparent that the 32-bit mux has longer propagation delays than the smaller multiplexer components because of the wider bus size.

∻ -•	Msgs				
. ⊥ → in0	0B	08			
😐 🔶 in 1	1F	1F			
🔷 sel	0				
💶 🔶 out0	0B	0B	1F	(0B	ΪF
		The 5-bit mux selects input 0, so the value 0x0B is the output	The select is now '1' so the output is input 1 = 0x1F	-Repeat-	
🕮 🐺 💿 Now	50600 ns	ins	100) ns	200

∕ ⊶	Msgs								
. → in0	0B	0B							
📕 🔶 in 1	1F	1F							
i 🔷 sel	1								
🛨	1F	0B						1F	
					The propa 5-bit mux FPGA is ~1	gation delay on a Cyclone 3.5	of the II		
🛎 📰 💿 Now	500 ns	40	ns	 50 ns		60 E	ns		
jade Gerne and Sor 1	50 ns			50 ns		13.489 ns			
🔓 🖉 😑 sor 2	1 89 ns						63.4	89 ns	3

Figure 1.5



Figure 1.6

The MIPS microprocessor uses 32 32-bit registers to quickly store and load bits, hence why the field size of the "rs," "rt," and "rd" partition of the instruction are each 5-bits wide (31 in binary is "11111"). Internally, a register consists of D-flip flops, which output previously stored data or new data. The flip flops in the register are reliant on the falling or rising edge of the clock for synchronization and form the basic storage unit in hardware design. See figure 2.1 for the VHDL code for a 32-bit register and figure 2.2 for the simulation. Since the worst-case propagation delay for the register is about 9.2 nanoseconds, the clock rate must exceed approximately 18.4 MHz – 20 MHz. For the test bench used in Altera's ModelSim, see appendix.

```
1 --Rachel Kreynin
    L-- 32-bit Register
2
3
4
   library ieee;
 5 use ieee.std_logic_1164.all;
 6 use ieee.numeric_std.all;
 7
8 entity reg32 is
9 白
         generic (
10
         WIDTH : positive := 32);
11
   白
       port (
12
                 D
                                    : IN std_logic_vector(WIDTH-1 downto 0);
13
                 wr, Clk, clr
                                 : IN std logic;
14
                 Q
                                    : OUT std_logic_vector(WIDTH-1 downto 0)
15
                 );
    L
16
         end reg32;
17
18
    architecture behavior of reg32 is
19
20
     SIGNAL Q_Temp : std_logic_vector(WIDTH-1 downto 0); --internal signal; input and output type
21
22 Ebegin
23 🛱
          process(Clk, clr)
24
             begin
25 🖯
                 if (clr = '0') then --Active-Low Reset
26
27
28
                 Q Temp <= (others => '0'); --32 bits all set to '0'
                 elsif (rising_edge(Clk)) then
                 if (wr = '1') then
29
                     Q_Temp <= D; --Input new data on true 'wr'</pre>
30
    Ē
                  else
31
                     Q_Temp <= Q_Temp; --Hold previous data</pre>
32
                  end if;
33
              end if;
34
          end process;
```

37
38 end behavior;

35 36

```
Figure 2.1
```

Q <= Q_Temp; --Simple assignment; Output Q always equals Q_Temp

\$ ⊒•	Msgs	
 → d	ABCD12	0000000 <u>/12</u> 34ABCD XABCD 1234
😐 🔶 q	000000	0000000 X1234ABCD XABC X00000000
🔶 wr	1	
🔷 dk	1	
🔷 dr	0	Wr is set to '1' -Held for 10 clock cycles- Signal is reset when clr = '0' So Q equals the new value D = (wr = '0') New value of D = 0x1234ABCD on the rising edge of Clk New value of D = 0xABCD1234 is set on the rising edge of clock and held for two cycles
🗳 🐺 🖲 Now	360 ns	ns 200 ns 400 ns 600 ns 800 ns





The MIPS architecture consists of three types of instructions: R-type, I-type, and J-type, and all instructions are 32-bits long. However, I-type and J-type instructions require a method of extending the size of a binary number to 32-bits in order to perform arithmetic with an ALU. Therefore, two components, a zero extender and a sign extender, are implemented to carry out those instructions. Instructions like "immediate OR" require a zero extender to OR a 32-bit number with the 16-bit number immediately addressed and instructions like immediate loading require a sign extender to preserve the value of 16-bit number in the instruction. Therefore, a 32-it mux is used to select the correct value. Figure 3.1 shows the VHDL code for the "extender" component. For the zero extender and sign extender, see appendix. In figure 3.2, the timing simulation represents the function of the extender, including propagation delays. For the test bench used in simulation, see appendix.

```
1
    --Rachel Kreynin
 2
     -- Extender
 3
 4
      library ieee;
 5
    use ieee.std_logic_1164.all;
 6 use IEEE.STD_LOGIC_ARITH.ALL;
 7
    use IEEE.STD_LOGIC_SIGNED.ALL;
 8
 9
    entity extender is -- top level entity
10
11
          port (
    Þ
12
                 in0
                       : in std_logic_vector(15 downto 0);
                        : in std_logic;
13
                 Sel
14
                 out0
                           : out std_logic_vector(31 downto 0)
15
                 );
     L
16
          end extender;
17
18
    architecture behavior of extender is
19
20
          signal zero_out0, sign_out0 : std_logic_vector(31 downto 0); --internal signals to connect output port to input port of mux32
21
22
    -begin
23
24
    U ZERO: entity work.zeroext
25
                 PORT MAP(in0 => in0, out0 => zero out0); --implicit assignment
26
27
    U_SIGN: entity work.signext
28
                 PORT MAP(in0 => in0, out0 => sign_out0);
29
    U_MUX32: entity work.mux32
30
31
                 PORT MAP(zero_out0, sign_out0, Sel, out0); --mux used to select between zero or sign extension
32
33
34
      end behavior;
```

Figure 3.1

_ 🍋 •	Msgs				39 AC	
🛨 - 🔶 in0	7FFF	7FFF)FFFF		(7FFF
💶 🔶 out0	00007FFF	00007FFF		0000FFFF)FFFFFFFF	00007FFF
sel	0	Mux select = '0' so the input is zero extended with 16 '0' bits concantenated in front.	Mux select = '1' so the input is sign extended. The MSE of the input is '0' so 16 '0' bits are concantented	The new input is zero extended, so the output retains the LSB's of the input and concantenates '0' bits.	Mux select = '1' so the input is sign extended. The MSE of the input is a '1,' so 16 '1' bits are concantenated to the front in the output.	
🛎 📰 💿 🛛 Now	161300 ns	ns i i i	40	0 ns	80	l i i i i)0 ns
🔂 🖉 🤤 ursor 1	0.00 ns	0.00 ns				



EL4713C	ASSIGNMENT 2
Chapter 2: 2.1	1.4-6,2.14.1-3, 2.21.1-2
2.11.4) 'a' rep	resents an R-type instruction resents cm 1-type instruction
2.11.5) 'a' - Jubbre the	sub \$VI \$VO \$VI acts values in \$VI - \$VO and stores in into \$VI register
(b'- L loads	W & at \$ VO 4 word from value in register \$ VO +4 registers that
2.11.6) a-00000	26 10 0001 10010 0011,00000,100 0 RS RT RO SHAMT FUR
2.14.1) $i = 22$	j = 5 (const x 1
a) odd g SLL	$\begin{array}{c} \ddagger 1 \\ \ddagger 1 \\ \ddagger 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1$
b) add y SLL d ORI d	HEI Atto Bzero 1219 121 OXFFFFFFFF
2, 14, 2) $z = 4, 5$	= 0
a) add 4 Bll	# E1 \$10 \$2020 \$1 28
b) add a SLL & ORI &	121 120 1700 121 OXFFFFFFF
2.14.3) i= 31	j= 28
a) add \$	ti ato azero
b) add (4 Skl (4 ORI (4	tl 4 tl 0xFFFFFFF

6	2.	2	1.	Ż		0	0 d	x Ø	70	FO X	F7 -	B	a	F3	F)		C Ø1	17	0	0	x	/	0	0	0 7	+ 1	8	0	o gli	0	4	
		M			P Q	Y	nu a a a s l a	1 & & & & & & & & & & & & & & & & & & &	of dad diver	loi i	6a		a a s x a s P V V V V	01 022 0 0	,	12 40 1000	100 S S S S S T 0	Call Property and the second s		1 4 6e 9-	0	0	al									
	6			A		m J°	- 9-	a va	la do d	2a di			s c a				1 5 5	1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1								1 2 2 2 2						
					40			hait ait	udau tau	d		A C F G G G	a Boolar	1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2		m I ai		8	2	2		de je	li.		Y (30			<u></u>	4		

Component Design

5-Bit Multiplexer

```
1
    --Rachel Kreynin
 2
     --2 to 1 Mux
    library ieee;
3
 4
   use ieee.std logic 1164.all;
 5
    use ieee.numeric_std.all;
 6
7
   entity mux5 is
8
   Ė
         generic(
9
         WIDTH : positive := 5); --generic map with default value 5; can change per instance
10
   Ė
         port (
11
                 in0, in1 : IN std logic vector(WIDTH-1 downto 0);
12
                 Sel
                           : IN std logic;
13
                            : OUT std_logic_vector(WIDTH-1 downto 0)
                 0
14
                 );
    15
         end mux5;
16
17
    architecture behavior of mux5 is
18 Degin
         0 <= in0 when Sel = '0' else --conditional assignment
19
20
              in1;
21
    end behavior;
```

32-Bit Multiplexer

```
🚍 -- Rachel Kreynin
1
     ---2 to 1 Mux
2
3
     library ieee;
4
     use ieee.std logic 1164.all;
5
     use ieee.numeric std.all;
6
7
   entity mux32 is
8
          generic(
    9
          WIDTH : positive := 32);
10
    Ġ
          port (
                  in0, in1 : IN std_logic_vector(WIDTH-1 downto 0);
11
12
                  Sel
                             : IN std logic;
                             : OUT std_logic_vector(WIDTH-1 downto 0)
13
                  0
14
                  );
     L
15
          end mux32;
16
17
     architecture behavior of mux32 is
18
    begin
19
20
          0 <= in0 when Sel = '0' else
21
              in1;
22
     end behavior;
```

Zero Extender

```
--Rachel Kreynin
1
2
     -- Zero Extension
3
4
     library ieee;
5
    use ieee.std logic 1164.all;
6
     use ieee.numeric std.all;
7
8
   entity zeroext is
9
    Ė
10
          port (
11
                  in0
                             : in std logic vector(15 downto 0);
12
                  out0
                             : out std logic vector (31 downto 0)
13
                  );
     L
14
          end zeroext;
15
16
    architecture behavior of zeroext is
17
18
   begin
19
20
          out0 <= x"0000" & in0;
21
    <sup>L</sup>end behavior;
22
```

Sign Extender

```
1
    --Rachel Kreynin
    L -- Sign Extension
 2
 3
 4
     library ieee;
 5
     use ieee.std_logic_1164.all;
     use IEEE.STD_LOGIC_ARITH.ALL; --Allows sxt function
 6
 7
     use IEEE.STD_LOGIC_SIGNED.ALL;
 8
9
    entity signext is
10
11
    白
          port (
12
                             : in std logic vector(15 downto 0);
                  in0
                             : out std_logic_vector(31 downto 0)
13
                  out0
14
                  );
     L
15
          end signext;
16
    architecture behavior of signext is
17
18
19
    begin
20
          out0 <= sxt(in0, out0'length); --sign extends to the length of the output signal
21
22
23
    <sup>L</sup>end behavior;
```

5-Bit Multiplexer

```
1
    --Rachel Kreynin
     --Test Bench
2
3
4
     LIBRARY ieee;
5
    USE ieee.std_logic_1164.all;
6
     USE ieee.numeric_std.all;
7
8
   ENTITY mux5_tb IS
    END mux5_tb;
9
10
11
     --Component Declaration
12 ARCHITECTURE behavior OF mux5_tb IS
13
14
      SIGNAL in0, in1, out0 : STD_LOGIC_VECTOR(4 downto 0);
15
     SIGNAL Sel
                          : STD_LOGIC;
16
17 BEGIN
18
    --Component Instatiation
19 📋 UUT:
                ENTITY work.mux5
                 PORT MAP (in0, in1, Sel, out0); --implicit port map
20
21
22
      in0 <= "01011"; --inputs values to test</pre>
23
     in1 <= "111111";
24
25
    process
26
     begin
27
28
    for i in 0 to 200 loop --cycle between Sel = '1' and '0'
29
          Sel <= '0';
30
          wait for 50ns;
31
          Sel <= '1';
32
     end loop;
33
34
     WAIT FOR 500ns;
35
      REPORT "SIMULATION FINISHED!";
36
     WAIT;
37
38
    - END PROCESS;
39
      --End Test Bench
40
     L END;
```

32-Bit Multiplexer

```
1 --Rachel Kreynin
 2 L--Test Bench
 3
 4
    LIBRARY ieee;
 5 USE ieee.std logic 1164.all;
 6 USE ieee.numeric std.all;
 7
 8
   ENTITY mux32 tb IS
    END mux32_tb;
 9
10
11
     --Component Declaration
12 ARCHITECTURE behavior OF mux32 tb IS
13
     SIGNAL in0, in1, out0 : STD_LOGIC_VECTOR(31 downto 0);
14
     SIGNAL Sel
                  : STD LOGIC;
15
16
17
   BEGIN
18
19
     --Component Instatiation
20 📋 UUT:
                ENTITY work.mux32 -- Unit Under Test: mux32
21
                 PORT MAP (in0, in1, Sel, out0);
22
23
     inO <= x"F47CD539"; --32-bit values to be test
24 in1 <= x"C885EFBC";</pre>
25
    process
26
27
    begin
28
   for i in 0 to 200 loop
29
30
        Sel <= '0';
31
         wait for 50ns;
         Sel <= '1';
32
33
    end loop;
34
35
36
     WAIT FOR 500ns;
37
    REPORT "SIMULATION FINISHED!";
38
     WAIT;
39
40
   - END PROCESS;
     --End Test Bench
41
42
    LEND;
```

```
32-Bit Register
```

```
--Rachel Kreynin
1
 2 L--Test Bench
 3
 4
    LIBRARY ieee;
 5 USE ieee.std_logic_1164.all;
    USE ieee.numeric_std.all;
 6
 7
 8 ENTITY reg32_tb IS
9
   LEND reg32_tb;
10
11
    --Component Declaration
12 ARCHITECTURE behavior OF reg32_tb IS
13
14
     SIGNAL D, Q
                      : STD LOGIC VECTOR(31 downto 0);
    SIGNAL wr, Clk, clr : STD_LOGIC;
15
16
17
18 BEGIN
19
20
    --Component Instatiation
21
   ĖUUT:
               ENTITY work.reg32
22
                PORT MAP (D => D, wr => wr, Clk => Clk, clr => clr, Q => Q);
23
24
25
    process
26
    begin
```

```
27
28
    for i in 0 to 500 loop --loop rising and falling clock
29
         clk <= '0';
30
         wait for 20 ns;
31
         clk <= '1';
32
          wait for 20 ns; --40 ns per cycle
33
   end loop;
34
    end process;
35
36 process
37
38
     begin
39
40
     clr
            <= '1'; --active-low asynchronous clear/reset false</pre>
41
     wr
            <= '0'; --write false
42
             <= (others => '0'); --D is cleared
     D
43
44
     wait for 80 ns;
45
            <= '1';
46
     clr
            <= '1'; --D value set
47
     wr
48
      D
             <= x"1234ABCD"; --new D value to be written
49
50
     wait for 40 ns;
51
     wr <= '0'; --hold value
52
53
54
     wait for 400 ns; --holds value for 10 cycles
55
56
      wr
             <= '1';
57
      D
             <= x"ABCD1234"; --new value set
58
59
     wait for 80 ns; --hold value for two cycles
60
61
      clr <= '0'; --reset
62
63
      wait for 80 ns;
64
65
      WAIT FOR 500 ns;
      REPORT "SIMULATION FINISHED!";
66
67
      WAIT;
68
     - END PROCESS;
69
      --End Test Bench
70
71
72 LEND;
```

Extender

```
--Rachel Kreynin
1
2
   --Test Bench
3
4
5
    LIBRARY ieee;
 6
    USE ieee.std logic 1164.all;
7
    USE ieee.numeric_std.all;
8
9
   ENTITY extender_tb IS
10
   LEND extender_tb;
11
12
     --Component Declaration
13
   ARCHITECTURE behavior OF extender_tb IS
14
15
                        : STD_LOGIC_VECTOR(15 downto 0);
     SIGNAL in0
                        : STD_LOGIC_VECTOR(31 downto 0);
16
     SIGNAL out0
    SIGNAL Sel
17
                        : STD_LOGIC;
18
19
   BEGIN
20
21
22
     --Component Instatiation
   ĖUUT:
23
                ENTITY work.extender
24
                 PORT MAP (in0 => in0, Sel => Sel, out0 => out0);
25
26
   process
27
     begin
28
29
30 📋
          for k in 0 to 200 loop -- loops select values and input values
31
              in0 <= x"7FFF"; --sign extend 0's</pre>
32
              Sel <= '0';
33
             wait for 200ns;
             Sel <= '1';
34
35
             wait for 200ns;
36
             in0 <= x"FFFF"; --sign extend 1's</pre>
37
             Sel <= '0';
38
              wait for 200ns;
39
              Sel <= '1';
40
              wait for 200ns;
41
          end loop;
42
      WAIT FOR 500ns;
      REPORT "SIMULATION FINISHED!";
43
44
      WAIT;
45
46
   - END PROCESS;
```

48 49 **END**;

--End Test Bench

47