Rachel Kreynin

1/29/2012

EEL4713C:

Assignment 2

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.

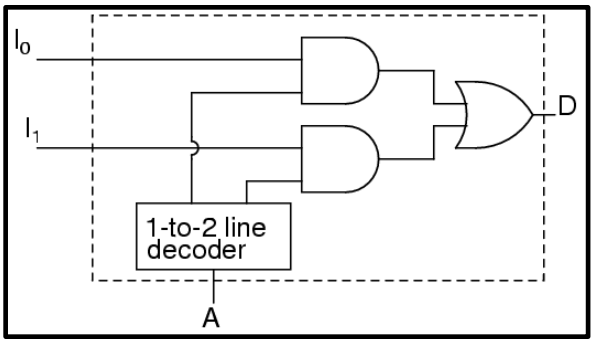


Figure 1.1

I_1	I_0	A	D
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	1

Figure 1.2

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

Figure 1.3

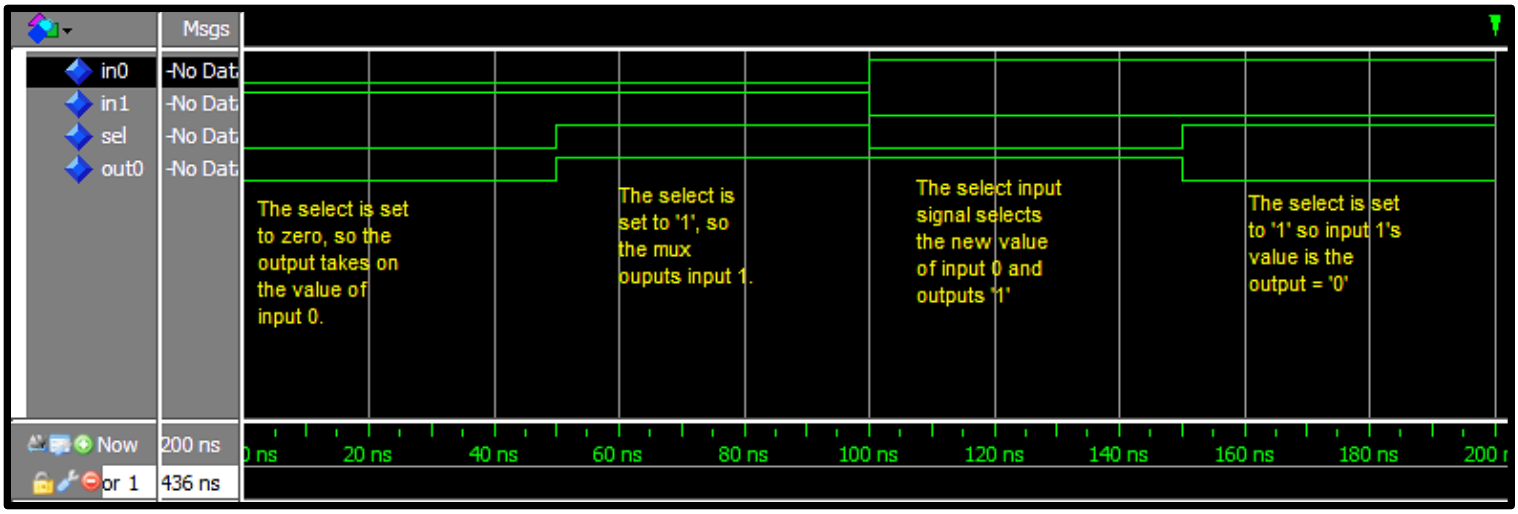
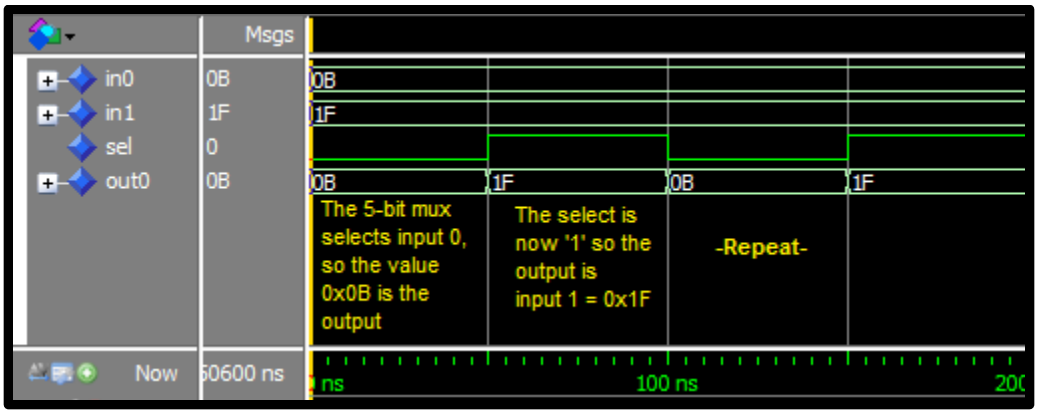


Figure 1.4

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.



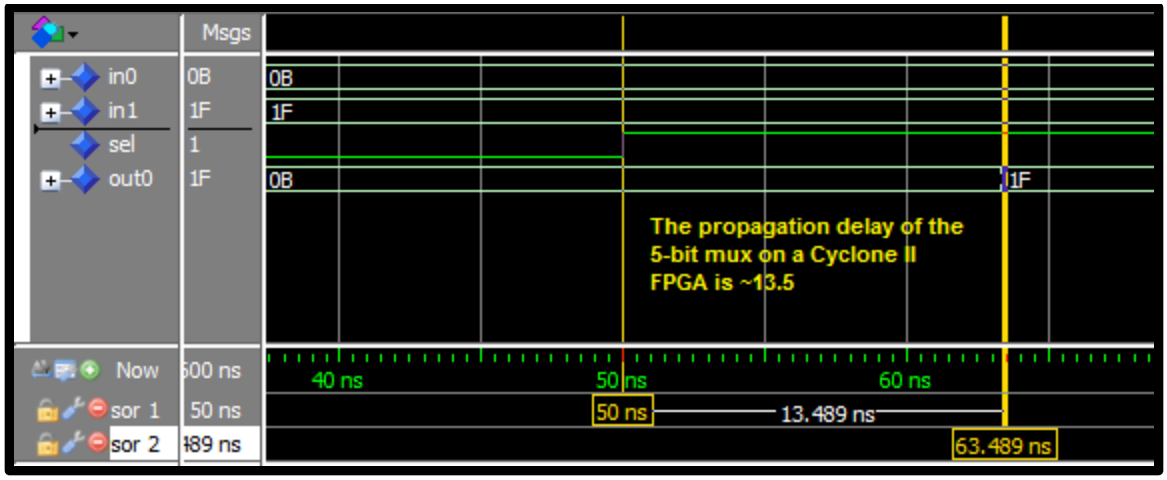


Figure 1.5

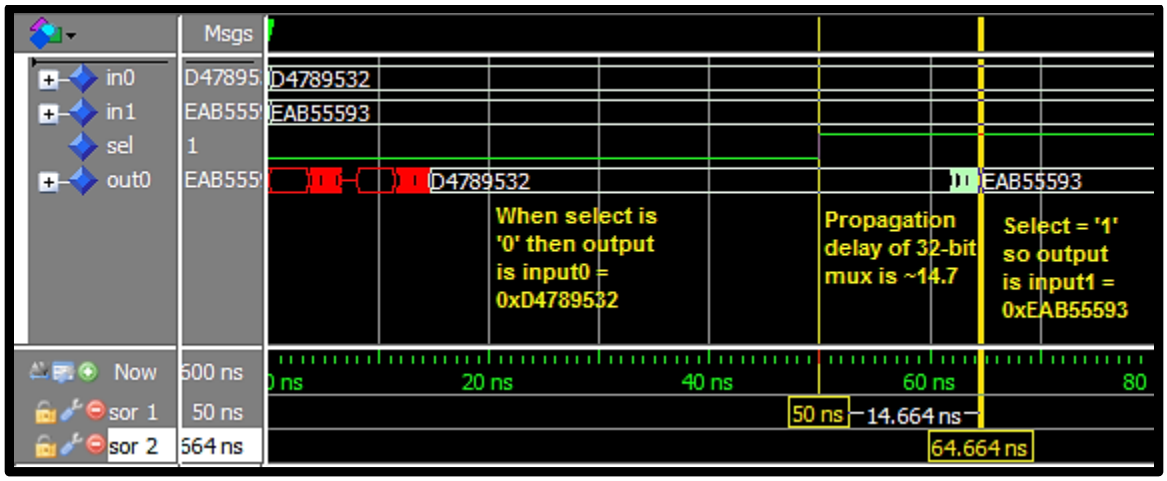


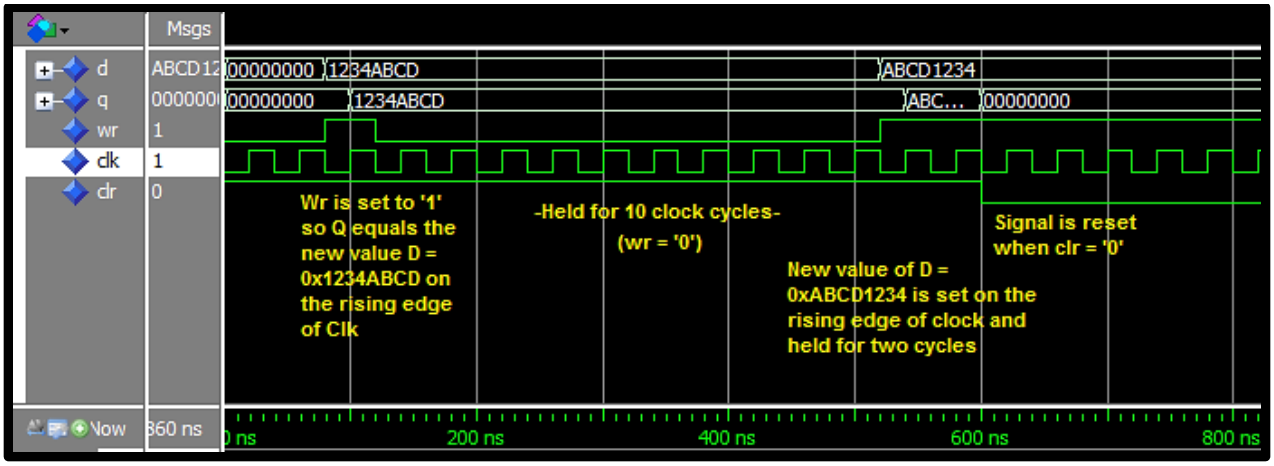
Figure 1.6

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
2  -- 32-bit Register
3
4  library ieee;
5  use ieee.std_logic_1164.all;
6  use ieee.numeric_std.all;
7
8  entity reg32 is
9      generic (
10         WIDTH : positive := 32);
11     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             );
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 begin
23     process(Clk, clr)
24         begin
25             if (clr = '0') then --Active-Low Reset
26             Q_Temp   <= (others => '0'); --32 bits all set to '0'
27             elsif (rising_edge(Clk)) then
28             if (wr = '1') then
29                 Q_Temp <= D; --Input new data on true 'wr'
30             else
31                 Q_Temp <= Q_Temp; --Hold previous data
32             end if;
33         end if;
34     end process;
35
36     Q <= Q_Temp; --Simple assignment; Output Q always equals Q_Temp
37
38 end behavior;
```

Figure 2.1



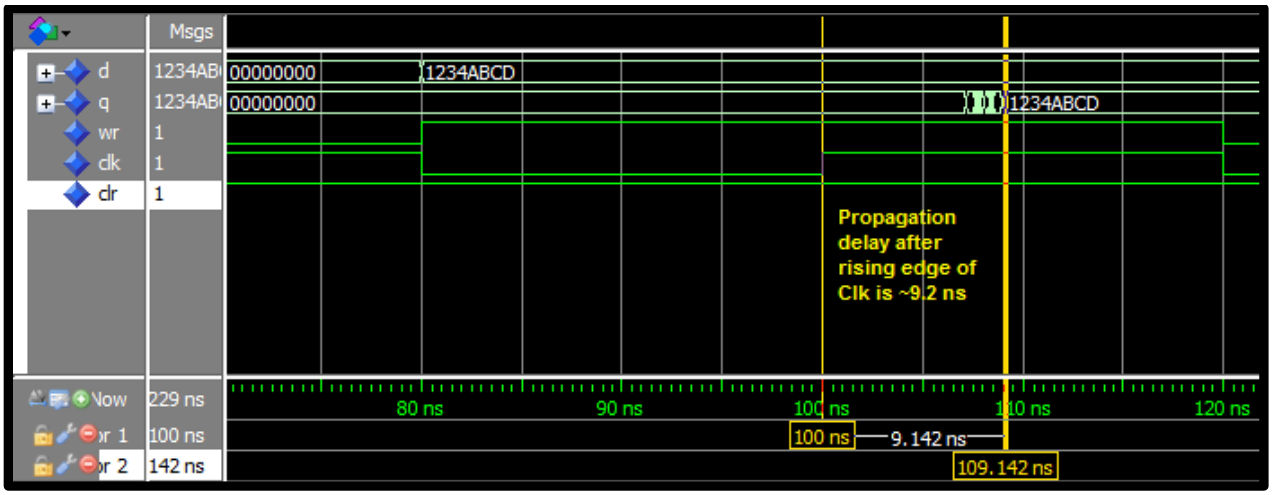


Figure 2.2

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.

```vhdl
--Rachel Kreynin
-- Extender

library ieee;
use ieee.std_logic_1164.all;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_SIGNED.ALL;

entity extender is --top level entity

    port (
            in0     : in std_logic_vector(15 downto 0);
            Sel     : in std_logic;
            out0        : out std_logic_vector(31 downto 0)
            );
    end extender;

architecture behavior of extender is

    signal zero_out0, sign_out0 : std_logic_vector(31 downto 0); --internal signals to connect output port to input port of mux32

begin

U_ZERO: entity work.zeroext
            PORT MAP(in0 => in0, out0 => zero_out0); --implicit assignment

U_SIGN: entity work.signext
            PORT MAP(in0 => in0, out0 => sign_out0);

U_MUX32: entity work.mux32
            PORT MAP(zero_out0, sign_out0, Sel, out0); --mux used to select between zero or sign extension


end behavior;
```

Figure 3.1

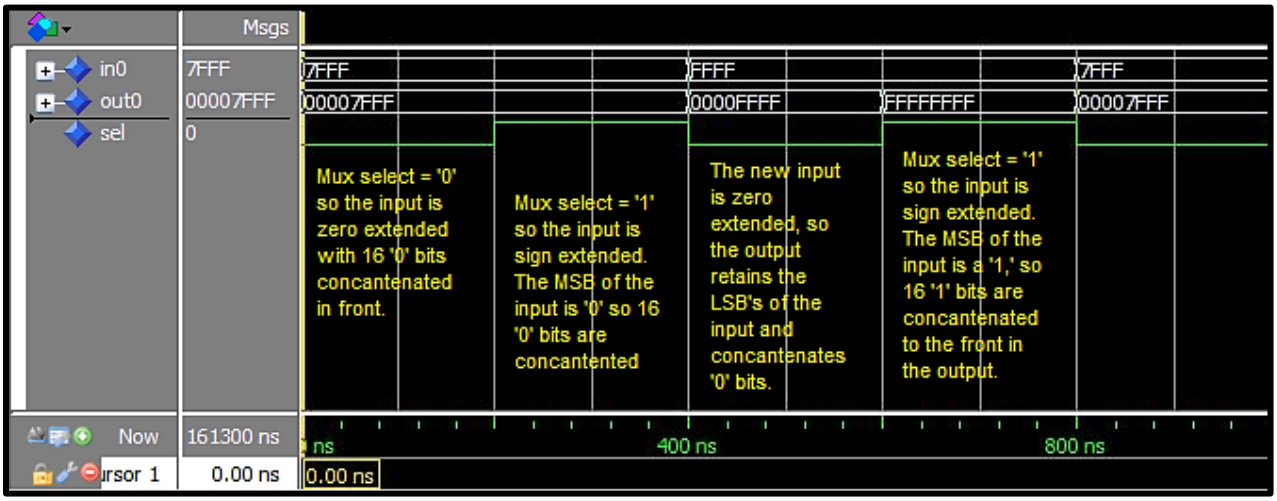


Figure 3.2

Textbook Questions

EL4713C | ASSIGNMENT 2

Chapter 2: 2.11.4-6, 2.14.1-3, 2.21.1-2

2.11.4) 'a' represents an R-type instruction
'b' represents an I-type instruction

2.11.5) 'a' - SUB $v1 $v0 $v1
subtracts values in $v1 - $v0 and stores them into $v1 register

'b' - LW $at $v0 4
loads word from value in register $v0 +4 to registers $at

2.11.6) 'a' - 31 26
0000 00 00011 00010 00011 00000 1000
OP RS RT RD SHAMT FUN

'b' - 100011 00001 00100 0000000000000
Const x 4

2.14.1) i = 22, j = 5

31 - (i-j) 17 14
FIELD
31 - 17 = 14

a) add $t1 $t0 $zero
SLL $t1 9

b) add $t1 $t0 $zero
SLL $t1 9
ORI $t1 0xFFFFFFFF

2.14.2) i = 4, j = 0

a) add $t1 $t0 $zero
SLL $t1 28

b) add $t1 $t0 $zero
SLL $t1 10
ORI $t1 0xFFFFFFFF

2.14.3) i = 31 j = 28

a) add $t1 $t0 $zero

b) add $t1 $t0 $zero
SRL $t1 4
ORI $t1 0xFFFFFFFF

2.21.1) 0x7FFF FFFC 0x1000 8000
$a0 - $a3, $v0 x-y+my_global

a)

```
        .data
my_global:  .word 100

MAIN:   addi  $a0  10
        addi  $a1  20
        addi  $sp, $sp, -4
        sw    $ra, ($sp)
        lw    $a2  my_global
        addi  $sp  $sp 4          -----> jr $ra
FUNC:   ADD   $v0  $a1  my_global
        SUB   $v0  $a0  $v0
        jr    $ra
```

b)

```
my_global:  .word 100

MAIN:   addi  $sp, $sp 8
        sw    $ra ($sp)
        addi  $a0  0
        lw    $a1  my_global
        addi  $a1  1
        jal   FUNC
        lw    $ra ($sp)           ----> addi $sp $sp 4
FUNC:                                   jr $ra
        addi  $v0 $a1 1
        jr    $ra
```

Appendix

Component Design

5-Bit Multiplexer

```
--Rachel Kreynin
--2 to 1 Mux
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity mux5 is
    generic(
    WIDTH : positive := 5); --generic map with default value 5; can change per instance
    port (
            in0, in1    : IN std_logic_vector(WIDTH-1 downto 0);
            Sel         : IN std_logic;
            O           : OUT std_logic_vector(WIDTH-1 downto 0)
            );
    end mux5;

architecture behavior of mux5 is
begin
    O <= in0 when Sel = '0' else --conditional assignment
         in1;
end behavior;
```

32-Bit Multiplexer

```
--Rachel Kreynin
--2 to 1 Mux
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity mux32 is
    generic(
    WIDTH : positive := 32);
    port (
            in0, in1    : IN std_logic_vector(WIDTH-1 downto 0);
            Sel         : IN std_logic;
            O           : OUT std_logic_vector(WIDTH-1 downto 0)
            );
    end mux32;

architecture behavior of mux32 is
begin

    O <= in0 when Sel = '0' else
         in1;
end behavior;
```

Zero Extender

```
--Rachel Kreynin
-- Zero Extension

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

entity zeroext is

    port (
            in0         : in std_logic_vector(15 downto 0);
            out0        : out std_logic_vector(31 downto 0)
            );
    end zeroext;

architecture behavior of zeroext is

begin

    out0 <= x"0000" & in0;

end behavior;
```

Sign Extender

```
--Rachel Kreynin
-- Sign Extension

library ieee;
use ieee.std_logic_1164.all;
use IEEE.STD_LOGIC_ARITH.ALL; --Allows sxt function
use IEEE.STD_LOGIC_SIGNED.ALL;

entity signext is

    port (
            in0         : in std_logic_vector(15 downto 0);
            out0        : out std_logic_vector(31 downto 0)
            );
    end signext;

architecture behavior of signext is

begin

    out0 <= sxt(in0, out0'length); --sign extends to the length of the output signal

end behavior;
```

Test Benches

5-Bit Multiplexer

```
--Rachel Kreynin
--Test Bench

LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.numeric_std.all;

ENTITY mux5_tb IS
END mux5_tb;

--Component Declaration
ARCHITECTURE behavior OF mux5_tb IS

SIGNAL in0, in1, out0 : STD_LOGIC_VECTOR(4 downto 0);
SIGNAL Sel            : STD_LOGIC;

BEGIN
--Component Instatiation
UUT:        ENTITY work.mux5
            PORT MAP (in0, in1, Sel, out0); --implicit port map

in0 <= "01011"; --inputs values to test
in1 <= "11111";
```

```

process
begin

for i in 0 to 200 loop --cycle between Sel = '1' and '0'
    Sel <= '0';
    wait for 50ns;
    Sel <= '1';
end loop;

WAIT FOR 500ns;
REPORT "SIMULATION FINISHED!";
WAIT;

END PROCESS;
--End Test Bench
END;
```

32-Bit Multiplexer

```
1  --Rachel Kreynin
2  --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
9  END mux32_tb;
10
11 --Component Declaration
12 ARCHITECTURE behavior OF mux32_tb IS
13
14 SIGNAL in0, in1, out0 : STD_LOGIC_VECTOR(31 downto 0);
15 SIGNAL Sel            : STD_LOGIC;
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 in0 <= x"F47CD539"; --32-bit values to be test
24 in1 <= x"C885EFBC";
```

```
25
26 process
27 begin
28
29 for i in 0 to 200 loop
30     Sel <= '0';
31     wait for 50ns;
32     Sel <= '1';
33
34 end loop;
35
36 WAIT FOR 500ns;
37 REPORT "SIMULATION FINISHED!";
38 WAIT;
39
40 END PROCESS;
41 --End Test Bench
42 END;
```

32-Bit Register

```
--Rachel Kreynin
--Test Bench

LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.numeric_std.all;

ENTITY reg32_tb IS
END reg32_tb;

--Component Declaration
ARCHITECTURE behavior OF reg32_tb IS

SIGNAL D, Q             : STD_LOGIC_VECTOR(31 downto 0);
SIGNAL wr, Clk, clr     : STD_LOGIC;


BEGIN

--Component Instatiation
UUT:        ENTITY work.reg32
            PORT MAP (D => D, wr => wr, Clk => Clk, clr => clr, Q => Q);



process
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
41  wr      <= '0'; --write false
42  D       <= (others => '0'); --D is cleared
43
44  wait for 80 ns;
45
46  clr     <= '1';
47  wr      <= '1'; --D value set
48  D       <=  x"1234ABCD"; --new D value to be written
49
50  wait for 40 ns;
51
52  wr <= '0'; --hold value
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;
66  REPORT "SIMULATION FINISHED!";
67  WAIT;
68
69  END PROCESS;
70  --End Test Bench
71
72  END;
```

Extender

```
--Rachel Kreynin
--Test Bench


LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.numeric_std.all;

ENTITY extender_tb IS
END extender_tb;

--Component Declaration
ARCHITECTURE behavior OF extender_tb IS

SIGNAL in0          : STD_LOGIC_VECTOR(15 downto 0);
SIGNAL out0         : STD_LOGIC_VECTOR(31 downto 0);
SIGNAL Sel          : STD_LOGIC;


BEGIN

--Component Instatiation
UUT:        ENTITY work.extender
            PORT MAP (in0 => in0, Sel => Sel, out0 => out0);


```

```
process
begin

    for k in 0 to 200 loop --loops select values and input values
        in0 <= x"7FFF"; --sign extend 0's
        Sel <= '0';
        wait for 200ns;
        Sel <= '1';
        wait for 200ns;
        in0 <= x"FFFF"; --sign extend 1's
        Sel <= '0';
        wait for 200ns;
        Sel <= '1';
        wait for 200ns;
    end loop;
WAIT FOR 500ns;
REPORT "SIMULATION FINISHED!";
WAIT;

END PROCESS;
--End Test Bench

END;
```