Experiment 3: A Universal Shifter

Karrthik Arya Roll Number 200020068
EE-214, WEL, IIT Bombay
September 8, 2021

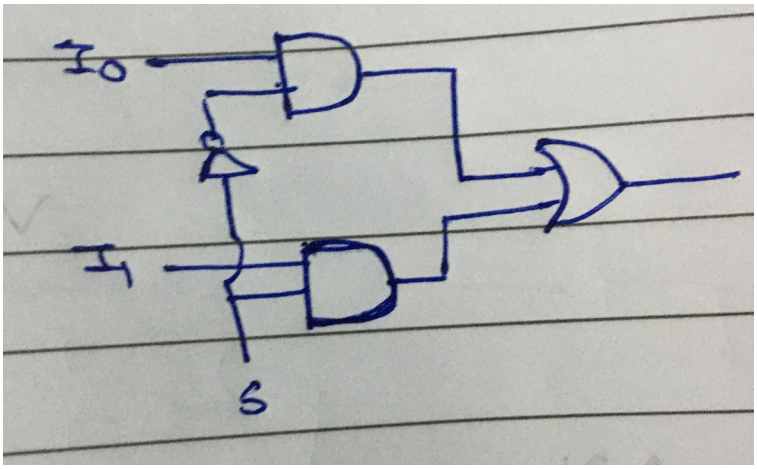
Overview of the experiment:

In this experiment we were asked to design a Universal Shifter. The circuit was supposed to shift an 8 bit number to by the specified bits. It should have been able to shift the bits to both left and right according to the input given, thus a universal shifter.

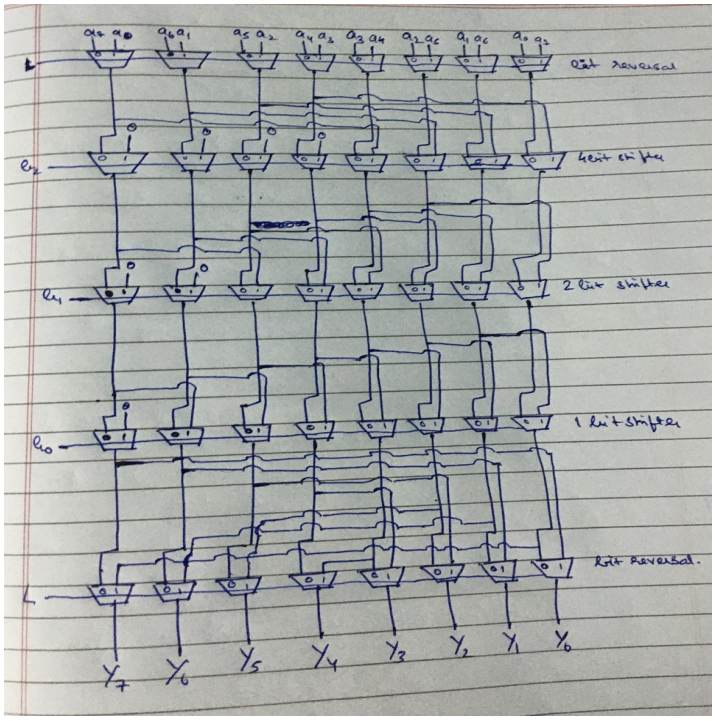
We had to first design the circuit then implement it using structural modelling in vhdl and then test it on the CPLD board using scanchain.

Approach to the experiment:

The basic block diagram for the circuit was given to us. The circuit to right shift the by 4 bits was also given to us. We just needed shifters to shift the bits by 4,2 and 1 bits. Since with the combination of these we can shift the bits by any number from 0 to 7. The bit reversal is needed so that we can shift the bits to left as well. It changes the ordering of the bits i.e. it takes b7b6b5...b2b1b0 and makes it b0b1b2...b5b6b7. If we reverse the bits of a number, then right shift by the specified bits and then reverse the bits again we would get the number left shifted by the specified bits. Thus bit reversal is used in the design.
The design for the 2x1 Multiplexer I used is this:



The circuit for right shift for 4 bits that they had given us used 8 2x1 multiplexers. In a similar way I designed the circuits for right shifting by 2 bits, 1 bits and bit reversal. Here's my design:



Design document and VHDL code if relevant:

I then implemented the design that I described above with VHDL. Most of the circuit uses multiplexers so I first specified the multiplexer as a component in Gates.vhdl. Here's the code describing the multiplexer:

```
entity MUX_2x1  is
  port (I0,I1,S: in std_logic; Y: out std_logic);
end entity MUX_2x1;
architecture Struct of MUX_2x1 is
  signal s1,s2,s3: std_logic;
begin
  -- component instances
  n1: INVERTER port map (A => S, Y=> s1);
  a1: AND_2 port map (A => I0, B => s1, Y => s2);
  a2: AND_2  port map (A => I1, B => S, Y => s3);
  o1: OR_2 port map (A => s2, B => s3, Y => Y);
end Struct;
```

To describe the main shifter circuitI used the generate statement to describe each block, bit reversal, 4 bit shifter etc. The code for 4 bit shifter was also given to us. I described the other blocks in a similar way. Here's the code:

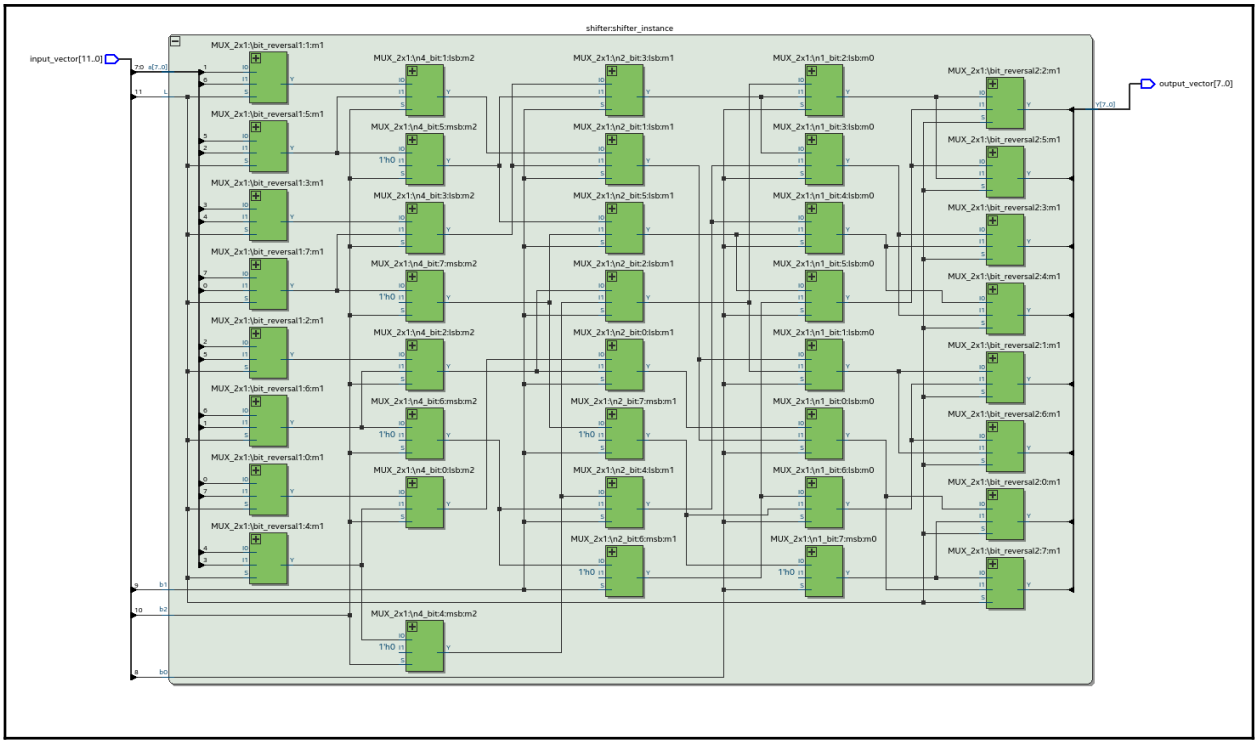
```
entity shifter is
  port (a: in std_logic_vector(7 downto 0); L,b0,b1,b2: in std_logic; Y: out
std_logic_vector(7 downto 0));
end entity shifter;

architecture Struct of shifter is
  signal s1,s2,s3,s4: std_logic_vector(7 downto 0);
begin
      bit_reversal1 : for i in 0 to 7 generate
       m1: MUX_2x1 port map(I0=>a(i), I1=>a(7-i), S=>L, Y=>s1(i) );
    end generate ;

       n4_bit : for i in 0 to 7 generate

        lsb: if i < 4 generate
            m2: MUX_2x1 port map(I0 => s1(i), I1 => s1(i+4), S => b2, Y =>
s2(i));
        end generate lsb;

        msb: if i > 3 generate
            m2: MUX_2x1 port map(I0 => s1(i), I1 => '0', S => b2, Y =>
s2(i));
        end generate msb;

    end generate ;

       n2_bit : for i in 0 to 7 generate

        lsb: if i < 6 generate
            m1: MUX_2x1 port map(I0 => s2(i), I1 => s2(i+2), S => b1, Y =>
s3(i));
        end generate lsb;

        msb: if i > 5 generate
            m1: MUX_2x1 port map(I0 => s2(i), I1 => '0', S => b1, Y =>
s3(i));
        end generate msb;

    end generate ;

       n1_bit : for i in 0 to 7 generate

        lsb: if i < 7 generate
            m0: MUX_2x1 port map(I0 => s3(i), I1 => s3(i+1), S => b0, Y =>
s4(i));
        end generate lsb;
```

```
        msb: if i > 6 generate
            m0: MUX_2x1 port map(I0 => s3(i), I1 => '0', S => b0, Y =>
s4(i));
        end generate msb;

    end generate ;

       bit_reversal2 : for i in 0 to 7 generate
       m1: MUX_2x1 port map(I0=>s4(i), I1=>s4(7-i), S=>L, Y=>Y(i) );
    end generate ;

end Struct;
```

The a is the input 8 bit vector which it is supposed to shift, L decides whether to shift right or left while b0,b1,b2 specify the no. of bits by which to shift and Y is the output bit vector. The signals s1,s2,s3,s4 are the intermediate outputs from each of the blocks.

Each of the blocks are specified using generate statements. Each of them instantiates a number of multiplexers needed for that block. In some of them if statements are needed since some of the multiplexers require different inputs than the others in the block. This is how I implemented the circuit in VHDL. After this using the files given for textbench and the tracefile given I tested it on RTL and Gate Level Simulations.

RTL View:

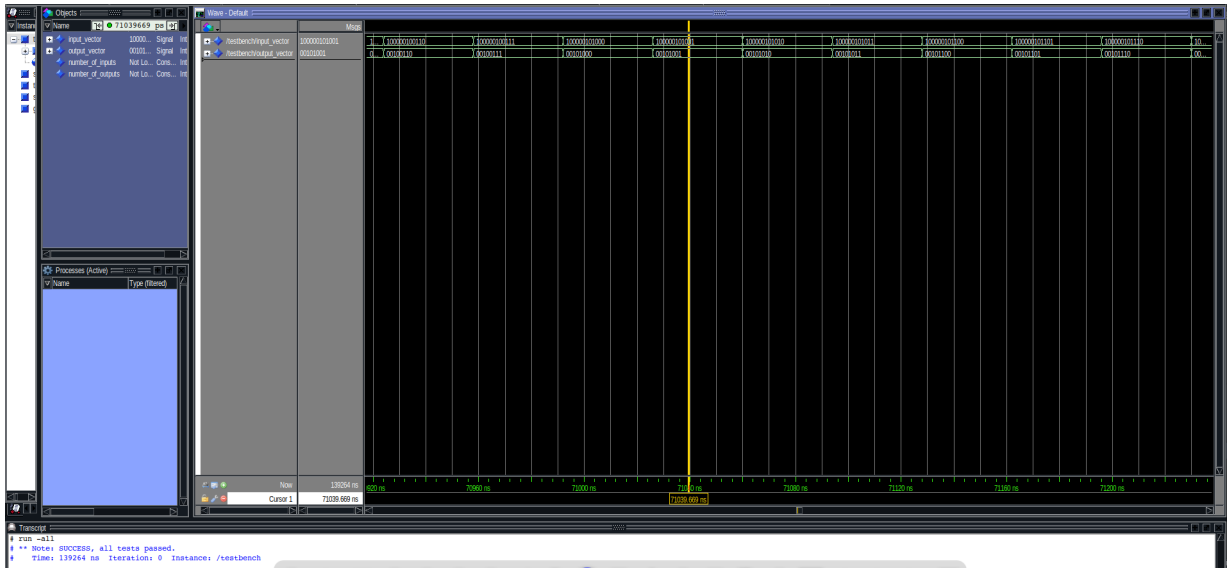


DUT Input/Output Format:

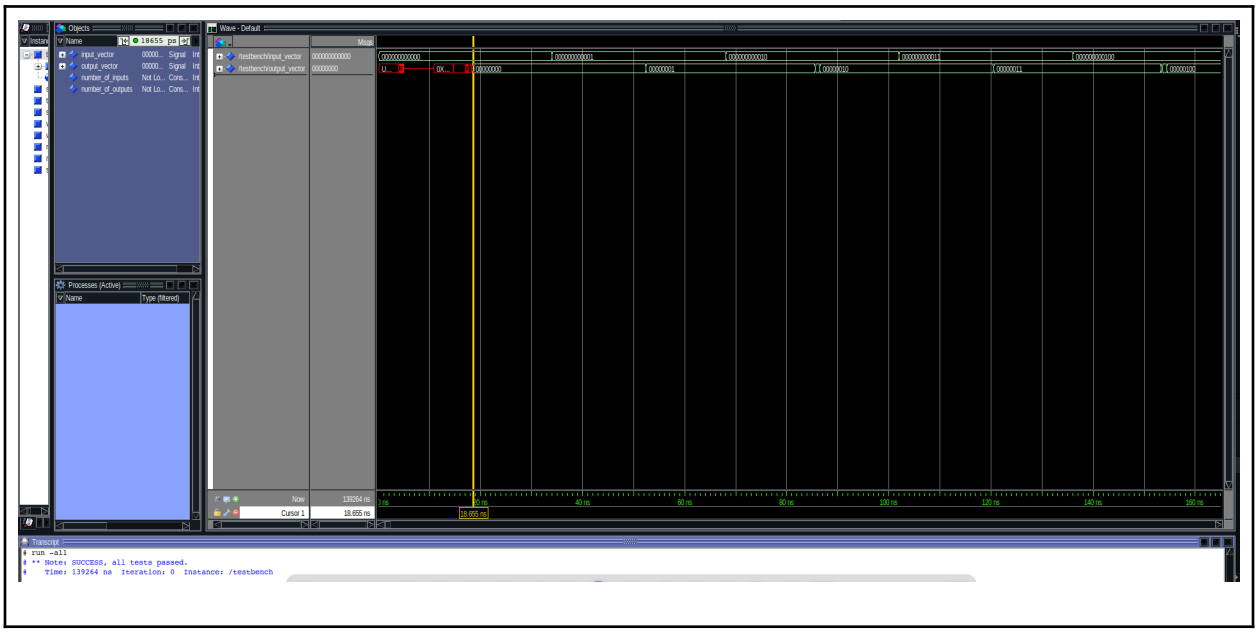
The input format of the trace file looks like
LB2B1B0A7A6A5A4A3A2A1A0 S7S6S5S4S3S2S1S0 11111111

L decides whether to shift right or left while B0,B1,B2 specify the no. of bits by which to shift. A7 to A0 specify the bits of the input vector which is to be shifted and S7 to S0 is the output vector. 111...1 represent the masking bits to tell whether a particular test case is to be verified or not.
Here are a few examples:
100011101001 11101001 11111111
100011101010 11101010 11111111
101101110001 10001000 11111111
101101110010 10010000 11111111
111000111101 01000000 11111111
111000111110 10000000 11111111

RTL Simulation:



Gate-level Simulation:



Krypton board*:

I used Scanchain to test my design on the Krypton board. I used the files given, set TopLevel.vhd as the top level entity, changed the no. of inputs and outputs to the correct value and generated the svf file. I then uploaded the svf file on the krypton board and tested it using scanchain with the tracefile given.

Observations*:

After testing with scanshain I found that in out.txt all the test cases were success. Thus the design worked perfectly and the experiment was successful. Here are few cases from out.txt:

000000000000 00000000 Success
000000000001 00000001 Success
000000000010 00000010 Success
000000000011 00000011 Success

100110110010 01100100 Success
100110110011 01100110 Success
100110110100 01101000 Success
100110110101 01101010 Success
100110110110 01101100 Success
100110110111 01101110 Success

References:

You may include the references if any.

* To be submitted after the tutorial on ”Using Krypton.