

**Department of Electrical Engineering**

**Lab Report 6: Universal Shift Register**

Name: Socheath Sok, ID: 014470701

Partner: Ashley Tran

Group: 11

Professor: Dr. Ray Wang

Class: EE 301

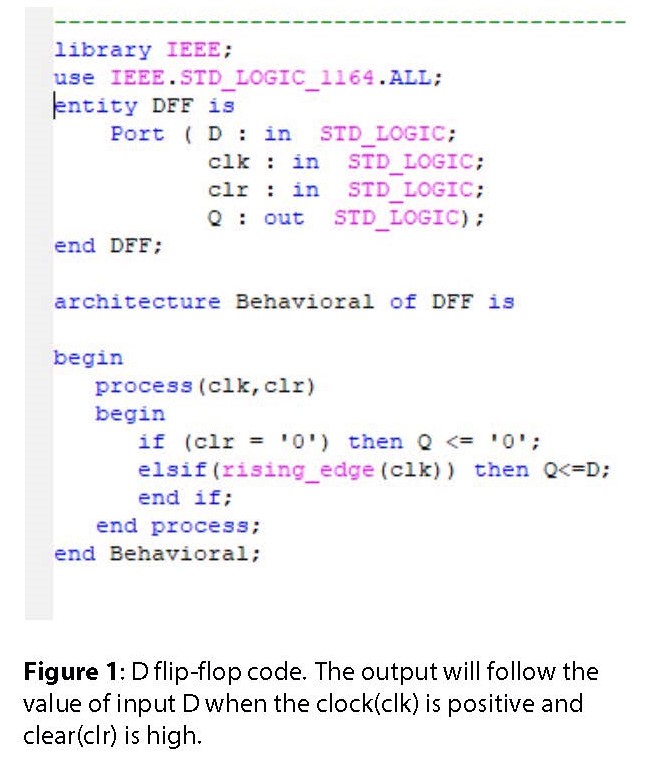
Date: December 6, 2017

**Abstract**

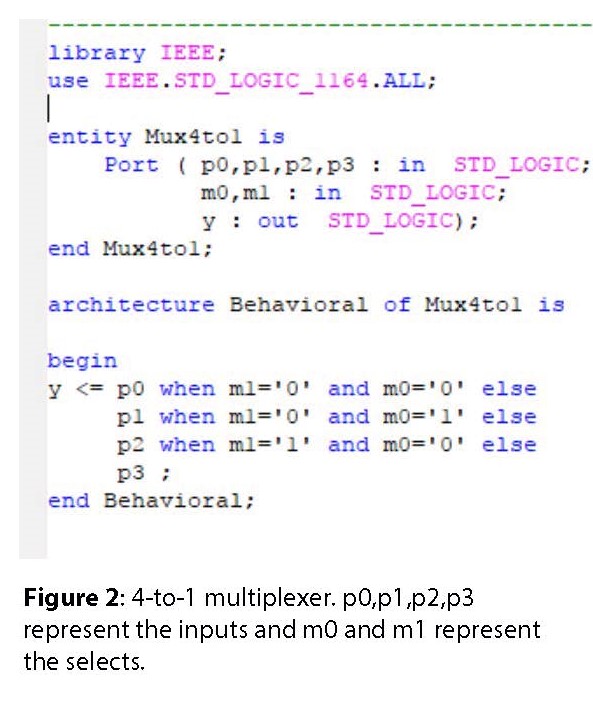
The goal of this lab is to design a 4-bit universal shift register that can perform right shift, left shift, and parallel loading. This shift register will be constructed using four D flip-flops and four 4-to-1 multiplexers. The overall combinational circuit will consist of eight inputs (P(3:0), Serial\_in, Q(3:0), clock, M(1:0), and clear) and four outputs (Q(3:0)). The input M(1:0) will decide the functionality of the register and the others inputs and outputs will have to be connected appropriately to the flip-flops and the multiplexers to generate different modes.

**Introduction**

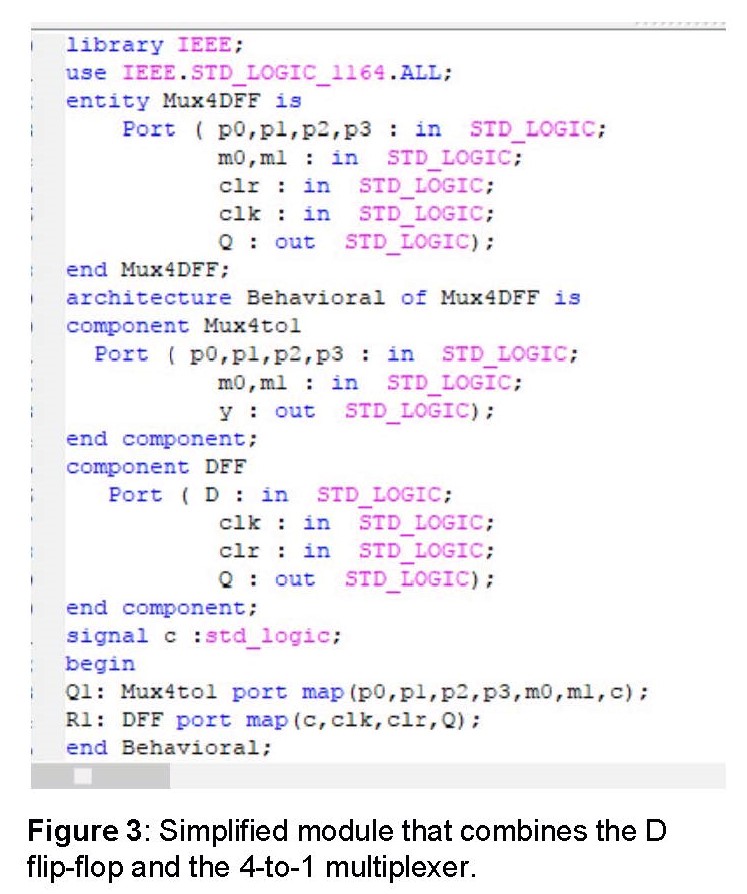
To create a structural VHDL code for the shift register, we, first, had to create the behavioral codes for the D flip-flop module and the multiplexer module. For the D flip-flop code, we defined the behavior of flip-flop based on its truth table using the if-elsif-else commands and made the output triggered at the positive-edge of the clock. The code for the D flip-flop is shown in **Figure 1**.



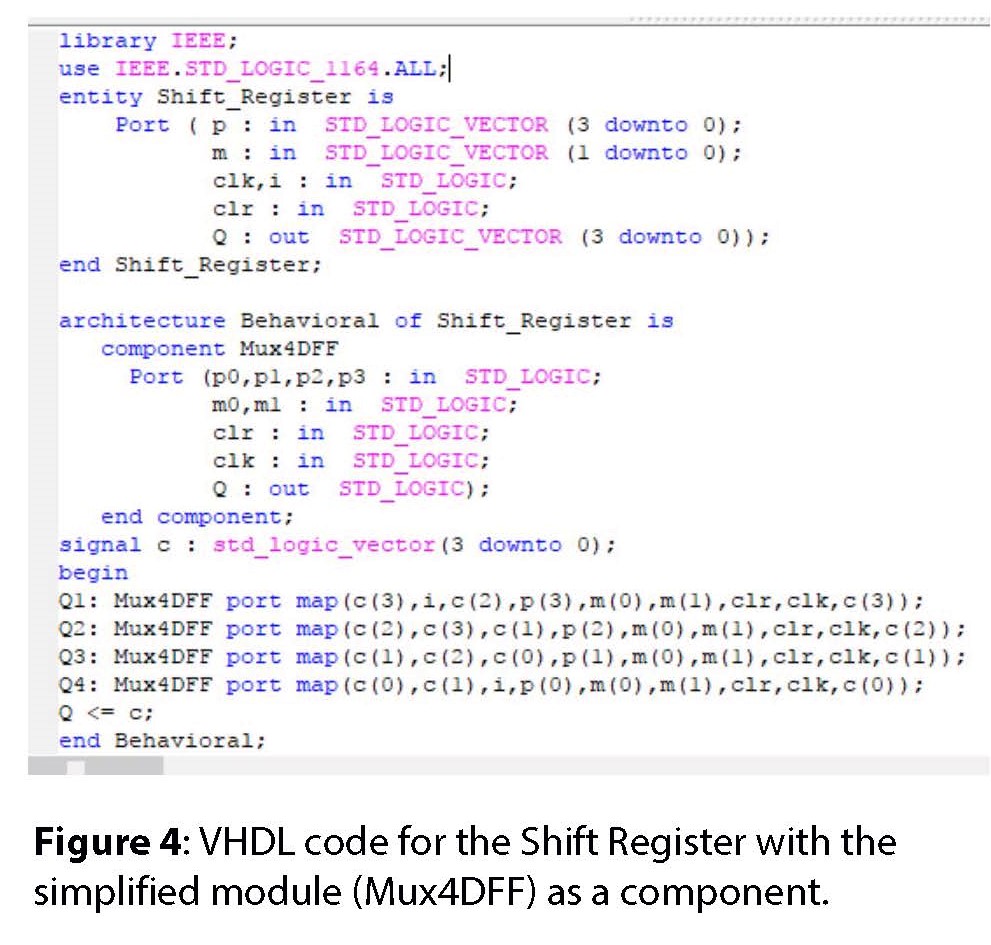
As for the 4-to-1 multiplexer, we used the when-else commands to define the outputs when the selects changes. When the selects is ‘00’, for example, the output is the value of the first input. When selects is ‘01’, the output will follow the second inputs and the rests of the combinations will follow the same pattern. The code of the multiplexer is shown in **Figure 2**.



With the two codes completed, we can create the structural VHDL code for our shift register by instantiating the D flip-flops and the 4-to-1 multiplexers using the component/portmap statements. However, we felt that the structural code would be too long and many signals would be needed to connect everything together. To simplify our code, we decided to create another module, which connected the multiplexer to the D flip-flop. This module will, then, be used as a component in the Shift Register structural code. The module that connect the 4-to-1 multiplexer to the D flip-flop can be seen in **Figure 3**.

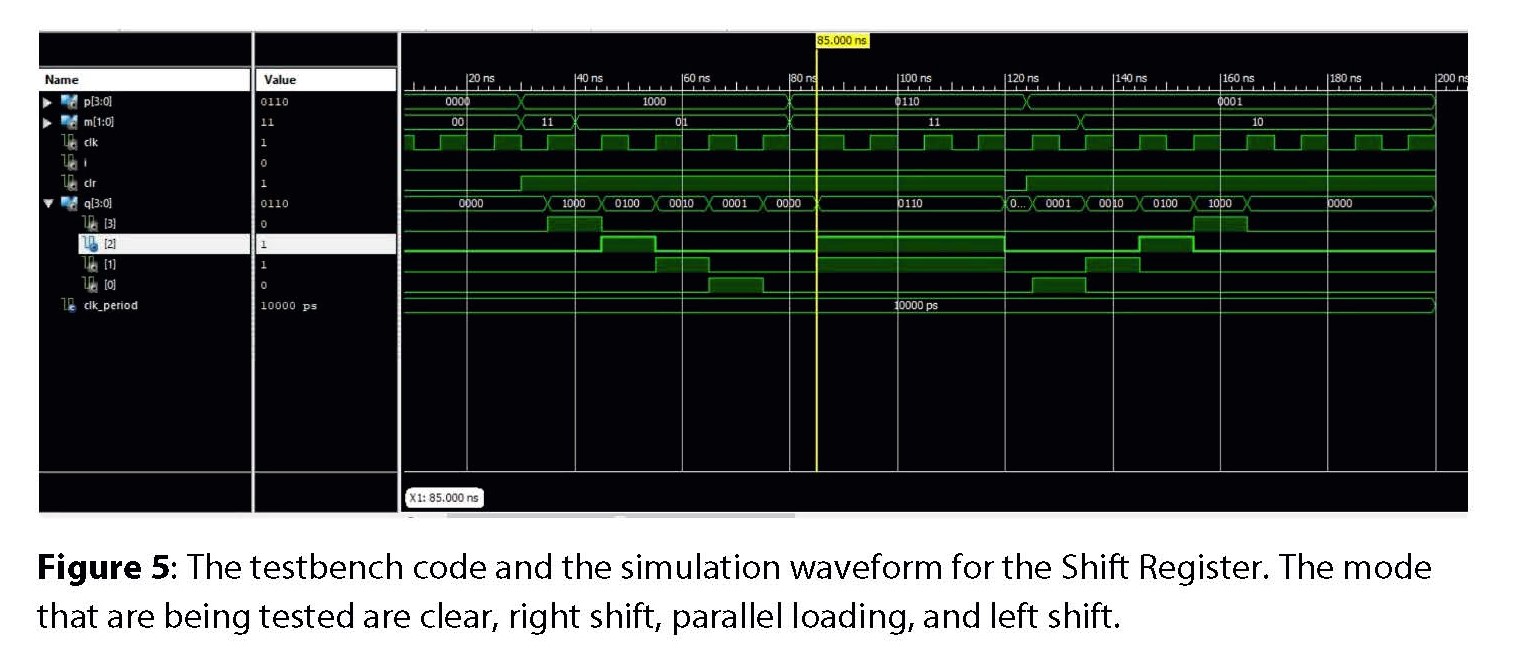


With the component simplified, all we had to do left was assign the inputs and outputs to their appropriate places. The same input clock and input clear was connected to all four D flip-flops of the simplified module. When input M (selects) is ‘00’, there is no change to the output of the register, so the outputs have to be connected to the inputs, thus the first input of each of the four simplified module are Q3, Q2, Q1, and Q0. When M is ‘01’, the mode is Right Shift and the sequence that identified the specific input of each modules will be Serial\_in → Q3, Q3 → Q2, Q2 → Q1, Q1 → Q0. From the sequence, we know that at that specific M (‘01’), the second input are Serial\_in, Q3, Q2, and Q1. When M is ‘10’, the mode is Left Shift and the sequence will be Q2 → Q3, Q1 → Q2, Q0 → Q1, Serial\_in → Q0, so the input are Q2, Q1, Q0, and Serial\_in. Lastly, when M is ‘11’, the mode is Parallel Loading, and the sequence is P3 → Q3, P2 → Q2, P1 → Q1, P0 → Q0, so the input are P3, P2, P1 and P0. After deciding how everything is connected, we created the shift register module with signal “i” represents the Serial\_in and vector “c” represents Qs. The result is shown in **Figure 4**.

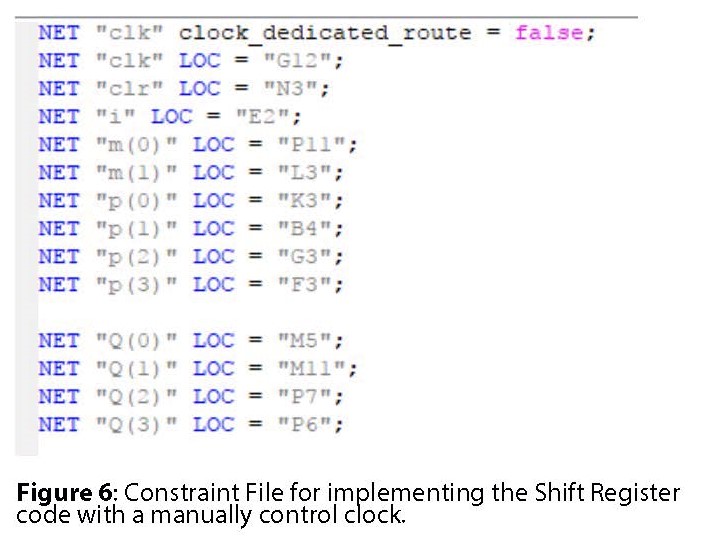


Next, we created the testbench code to test our design. In ‘stimulus process’, we wrote a code to test the input clear, and the right shift, followed by the the parallel loading and the left shift. To test the input clear, we set it equal to zero for 30 ns. To perform parallel loading, we set input ‘m’ (selects) equal to ‘11’ and P equal to ‘0110’. For the left and right shifts, we had to set ‘m’ to ‘11’ first to put in the starting point. Then, we made P to be ‘1000” and ‘m’ to ‘01’ to perform Right Shift. For the left shift, we loaded in P equal ‘0001’and ‘m’ to be ‘10’. In both cases, the Serial\_in was set to low, so, at the end, the output for both is “0000”. If we were to set the Serial\_in to high, as the shift occurs, a high output will start coming in from the opposite direction and will resulted in the final output of “1111”. The testbench code for the shift register along with the waveform is shown in **Figure 5**.

| ENTITY Shift\_Register\_test IS  END Shift\_Register\_test;    ARCHITECTURE behavior OF Shift\_Register\_test IS    COMPONENT Shift\_Register  PORT(  p : IN std\_logic\_vector(3 downto 0);  m : IN std\_logic\_vector(1 downto 0);  clk : IN std\_logic;  i : IN std\_logic;  clr : IN std\_logic;  Q : OUT std\_logic\_vector(3 downto 0) );  END COMPONENT;  signal p : std\_logic\_vector(3 downto 0) := (others => '0');  signal m : std\_logic\_vector(1 downto 0) := (others => '0');  signal clk : std\_logic := '0';  signal i : std\_logic := '0';  signal clr : std\_logic := '0';  signal Q : std\_logic\_vector(3 downto 0);  constant clk\_period : time := 10 ns;  BEGIN  uut: Shift\_Register PORT MAP (  p => p,  m => m,  clk => clk,  i => i,  clr => clr,  Q => Q);  clk\_process :process  begin  clk <= '0';  wait for clk\_period/2;  clk <= '1';  wait for clk\_period/2;  end process; | -- Stimulus process  stim\_proc: process  begin  -- test CLR  clr<='0';  wait for 30ns;  -- test loading  clr<='1';  m<="11";  P<="1000";  wait for 10ns;  -- test shift right  m<="01";  i<='0';  wait for 40ns;  -- test parallel loading  clr<='1';  m<="11";  P<="0110";  wait for 40ns;  clr<='0';  wait for 4ns;  -- test loading  clr<='1';  m<="11";  P<="0001";  wait for 10ns;  -- test shift left  m<="10";  i<='0';  wait for 40ns;  wait;  end process; |
| --- | --- |
|  |  |



After checking the waveform, we moved on to the implementing process by creating the constraint file. We included the statement “NET “clk” clock\_dedicated\_route = false ;” and connected the input clock to a push button, so that we can control it manually. Next, we bound the remaining inputs to the switches on the board and the outputs to the first four LEDs. The result is shown in **Figure 6**.



**Discussion**

For this lab, we started by creating VHDL modules for the D flip-flop and the 4-to-1 multiplexer, using the program build-in functions and the truth tables of each components. To avoid a long structural code, we combined the D flip-flop and the 4-to-1 multiplexer into a single module and use it as a component in the Shift Register code. With the code completed, we created the testbench code and generate a waveform to check the functionality of our register. Once everything is proven to be correct, we implemented the code onto the board and performed all the operations, which were parallel loading, right shift, left shift, no change and clear.