ECS527U:
Digital System Design Lab 4: LIFO Implementation

LIFO Model

We designed our LIFO by first creating declaring our top-level ports that were defined in the buffer design. This was done by first creating our entity in which we first declared all our input and output ports. We did this by using the keywords **IN/OUT STD_LOGIC** depending on whether or not the port created is an output or an input. Being that the ports D_IN(shift register input), EN(enable) and RST(reset), PUSH_POPn and CLK(clock) are all buffer inputs. We must then declare all these ports as in std_logic and the vice versa for D_out(output of the shift register), full and empty since they are our outputs in this design. Another element we need to consider when declaring these ports is how many bits the ports are expected to output or input. Being that this LIFO buffer design has 4 shift register it means that our D_IN input is expected to have 4 bits, for this reason, we need to ensure that we not only declare the port as std_logic but as a std_logic_vector to make sure that we let the compiler know that this port is expected to have 4 bits. Similarly, D_OUT would also have to be declared as std_logic_vector output. To ensure that the compiler knows that these ports need 4 bits we need to also declare the number bits by stating “(3 downto 0)” which highlights 4 bits starting from 0 to 3 are expected in both the inputs/outputs from the shift register.

Once these ports are created we can end the entity and start the architecture, the architecture is used to describe the underlying functionality of the entity. Since the shift register and counter were both previously created for us we needed to declare these VHDL codes as components in the architecture. This is done so we can connect all the shift registers and the counter blocks together to produce our LIFO buffer. So we declared the counter as components by copying the entity that was already written and changing the keywords entity to components and making sure to end the component with the syntax “end component” and similarly we do the same thing for the shift register. Next, we create signals that will be needed to port map the components to the LIFO buffer. In this buffer design, many of the components port will be port mapped to the port already declared in the entity. However, those that will not be will need to be connected to signals that are created in the architecture. In this design, we have created 3 signals QC, QS and NP. The QC signals will be port mapped to the Q output of the counter. This Q output has been declared as having 4 bits, the signal created also needs to be declared as a vector with 4 bits. Similarly, the QS signal was created to connect to the output of the shift register with 8 bits. The NP signal was created to port map the output of the not gate connected to the L_Rn port in the counter. We then port map all the signals according to the specification design, we port map the shift register 4 times each port map for each shift register.

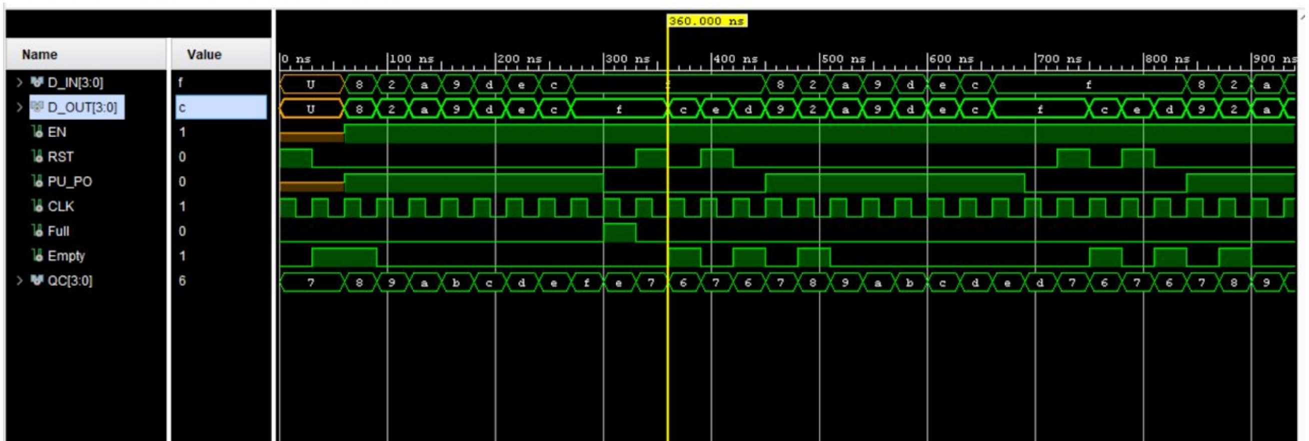
Once we finish port mapping all the signals we then create a process that is sensitive to the clock by putting CLK in the sensitivity list and then describe the logic that makes the output full and empty. The outputs empty and full are described using a series of if conditions that are nested in another if statement that highlights the condition of the common clock signal that synchronises the process, we use this if statement to suggest that these logic operations only happen at the positive clock edge. Indicated in the specification when the counter is reset, the counter holds the value “0111” and so, therefore, the buffer would be empty and the empty output would be 1 and when the counter is at “1111” the buffer would be full and the full output would be 1. Once this is described we can end both if statements, the process and lastly the architecture.

LIFO Buffer testbench

In the design of the LIFO Buffer testbench, we used the LIFO48 model as a component in order to simulate the test runs of the LIFO Buffer. In order to map these component values to the signal values, we use the **port map.** Since the buffer is a synchronous process, we simulate the clock positive and negative edges using a process; this is used in unison with the LIFO48 model to trigger the **LIFO_proc** process and initiate changes to the values of Full and Empty.

In order to test the cases of the stimuli a process other than the **stim_clk** used to simulate the clock must be used. So we use the **stim_proc** to produce these cases. For the first case in order to show that the first codeword in will be the first out, we have pushed values onto the stack until it has been filled by writing ‘1’ to the **PU_PO** whilst varying the input **D_IN.** Since the LIFO Buffer is a destructive read operation, when we write ‘0’ to **PU_PO** the input does not have to be changed, since the value for **Q(0)** is lost after being popped off the stack.

In order to simulate the second case the Empty and Full signals must be tested, where **QC** at ‘0111’ represents empty and ‘1111’ represents full. Initially to test the empty case we found that QC was undefined initially, so we used to reset to initialise the value of QC to ‘0111’. And in order to test the full case, we knew that the counter increments by 1 for each data word added to the stack, so we added 8 data words to fill the stack and satisfy the full case. Lastly, to show that the reset can be used to initialise and clear the stack we wrote a ‘1’ to reset, this changed the value of QC from ‘1111’ to ‘0111’ to show 0 data words present in the stack.



```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity LIFO_tb is
end LIFO_tb;

architecture Behavioral of LIFO_tb is
component LIFO48 is
 port(
  D_IN : in std_logic_vector (3 downto 0);
  EN, RST, PU_PO: in std_logic;
  CLK: in std_logic;
  D_OUT: out std_logic_vector(3 downto 0);
  Full, Empty: out std_logic );
end component;
signal D_IN, D_OUT : std_logic_vector (3 downto 0);
signal EN, RST, PU_PO, CLK, Full, Empty:  std_logic;
begin
 LIFO_map: LIFO48 port map ( D_IN => D_IN , D_OUT => D_OUT, EN => EN, RST =>
RST, PU_PO => PU_PO, CLK => CLK, Full => Full, Empty => Empty);

 stim_clk: process
 begin
 CLK <= '1'; wait for (15 ns);
 CLK <= '0'; wait for (15 ns);
 end process;

 stim_proc: process
 begin
 RST <= '1'; wait for 30ns;
 RST <= '0'; wait for 30ns;
 D_IN <= "1000"; PU_PO <= '1'; EN <= '1'; wait for 30ns;
 D_IN <= "0010"; PU_PO <= '1'; EN <= '1'; wait for 30ns;
 D_IN <= "1010"; PU_PO <= '1'; EN <= '1'; wait for 30ns;
 D_IN <= "1001"; PU_PO <= '1'; EN <= '1'; wait for 30ns;
 D_IN <= "1101"; PU_PO <= '1'; EN <= '1'; wait for 30ns;
 D_IN <= "1110"; PU_PO <= '1'; EN <= '1'; wait for 30ns;
 D_IN <= "1100"; PU_PO <= '1'; EN <= '1'; wait for 30ns;
 D_IN <= "1111"; PU_PO <= '1'; EN <= '1'; wait for 30ns;
 RST <= '1'; wait for 30ns;
 RST <= '0'; wait for 30ns;
 end process;


end Behavioral;
```

```
library IEEE;
use IEEE.std_logic_1164.all;

entity LIFO48 is
port(
  D_IN : in std_logic_vector (3 downto 0);
  EN, RST, PU_PO: in std_logic;
  CLK: in std_logic;
  D_OUT: out std_logic_vector(3 downto 0);
  Full, Empty: out std_logic );
end LIFO48;

architecture LIFO of LIFO48 is
component Counter4 is
port (
          EN, RST, U_Dn: in std_logic;
          CLK: in std_logic;
          Q: out std_logic_vector(3 downto 0));
end Component;
Component ShiftReg8 is
port (
    IN_R, IN_L, S_Hn, L_Rn: in std_logic;
    CLK: in  std_logic;
    Q:   out std_logic_vector(7 downto 0));
end component;
signal QC: std_logic_vector(3 downto 0);
signal QS: std_logic_vector(7 downto 1);
signal NP: std_logic;
begin
 NP <= not(PU_PO);
Counter: Counter4 port map( EN => EN, RST => RST, U_Dn => PU_PO, CLK => CLK, Q =>
QC);
ShiftReg1: ShiftReg8 port map (IN_R => D_IN(0) , IN_L => '0' , S_Hn => EN, L_Rn => NP,
CLK => CLK , Q(0) => D_OUT(0), Q(7 downto 1) => QS);
ShiftReg2: ShiftReg8 port map (IN_R => D_IN(1) , IN_L => '0' , S_Hn => EN, L_Rn => NP,
CLK => CLK , Q(0) => D_OUT(1), Q(7 downto 1) => QS);
ShiftReg3: ShiftReg8 port map (IN_R => D_IN(2) , IN_L => '0' , S_Hn => EN, L_Rn => NP,
CLK => CLK , Q(0) => D_OUT(2), Q(7 downto 1) => QS);
ShiftReg4: ShiftReg8 port map (IN_R => D_IN(3) , IN_L => '0' , S_Hn => EN, L_Rn => NP,
CLK => CLK , Q(0) => D_OUT(3), Q(7 downto 1) => QS);

 LIFO_proc: process(CLK)
   begin
       if CLK'event and CLK = '1' then
        if(QC = "1111") then
           Full <= '1';
           empty <='0';
        elsif ( QC = "0111") then
           empty <= '1';
           full <= '0';
        else
           full <= '0';
           empty <= '0';
       end if;

      end if;
   end process;
end LIFO;
```