Digital Design Lab #8: Registers and Register Banks

Charlie Coleman
Lab Partner: Luke Taylor
2016 October 27

Objective: The objective of this lab was to create registers and register banks using HDL.

Design:

Part one:

The goal of part one was to design a four bit edge triggered register. The register acts as 4 edge triggered flip-flops all driven by the same clock. The register is implemented using vectors for D and Q, which are the input and output respectively, and a single input for the clock.

Truth Table 1:

CLK	D3	D2	D1	D0	Q3	Q2	Q1	Q0
POSEDGE	0	0	0	0	0	0	0	0
POSEDGE	0	0	0	1	0	0	0	1
POSEDGE	0	0	1	0	0	0	1	0
POSEDGE	0	0	1	1	0	0	1	1
POSEDGE	0	1	0	0	0	1	0	0
POSEDGE	0	1	0	1	0	1	0	1
POSEDGE	0	1	1	0	0	1	1	0
POSEDGE	0	1	1	1	0	1	1	1
POSEDGE	1	0	0	0	1	0	0	0
POSEDGE	1	0	0	1	1	0	0	1
POSEDGE	1	0	1	0	1	0	1	0
POSEDGE	1	0	1	1	1	0	1	1
POSEDGE	1	1	0	0	1	1	0	0
POSEDGE	1	1	0	1	1	1	0	1
POSEDGE	1	1	1	0	1	1	1	0
POSEDGE	1	1	1	1	1	1	1	1

Code:

```
entity fourbitedgetrigger is
    Port ( CLK : in  STD_LOGIC;
           D : in  STD_LOGIC_VECTOR(3 DOWNTO 0);
           Q : out  STD_LOGIC_VECTOR(3 DOWNTO 0));
end fourbitedgetrigger;

architecture Behavioral of fourbitedgetrigger is
begin
      process(CLK)
      begin
            if(CLK' event and clk = '1') then
                  Q <= D;
            end if;
      end process;
      end Behavioral;
```

Part two:

The goal for part two was to implement a 4x4Bit register bank using HDL. The 4x4Bit register bank was implemented using four of the registers created in part one, a 2 to 4 decoder, and a quad 4:1 multiplexer. The 2:4 decoder was used to activate the clock on each of the 4 registers, and the multiplexer was used to select the correct outputs to display. The select lines and the outputs of each register were fed into the multiplexer inputs.

2:4 Decoder:

```
entity twofourdecoder is
    Port ( a : in STD_LOGIC_VECTOR(1 downto 0);
           b : out STD_LOGIC_VECTOR(3 downto 0);
                   EN : in STD_LOGIC);
end twofourdecoder;

architecture Behavioral of twofourdecoder is
begin

            b(0) <= not a(1) and not a(0) and EN;
            b(1) <= not a(1) and  a(0) and EN;
            b(2) <=  a(1) and not a(0) and EN;
            b(3) <=  a(1) and  a(0) and EN;

end Behavioral;
```

Quad 4:1 Multiplexer:

```
entity multiplexer4_1 is
    Port ( s0 : in  STD_LOGIC;
           s1 : in  STD_LOGIC;
           a0 : in  STD_LOGIC;
           a1 : in  STD_LOGIC;
           a2 : in  STD_LOGIC;
           a3 : in  STD_LOGIC;
           b0 : in  STD_LOGIC;
           b1 : in  STD_LOGIC;
           b2 : in  STD_LOGIC;
           b3 : in  STD_LOGIC;
           c0 : in  STD_LOGIC;
           c1 : in  STD_LOGIC;
           c2 : in  STD_LOGIC;
           c3 : in  STD_LOGIC;
           d0 : in  STD_LOGIC;
           d1 : in  STD_LOGIC;
           d2 : in  STD_LOGIC;
           d3 : in  STD_LOGIC;
           y0 : out  STD_LOGIC;
           y1 : out  STD_LOGIC;
           y2 : out  STD_LOGIC;
           y3 : out  STD_LOGIC);
end multiplexer4_1;

architecture Behavioral of multiplexer4_1 is

begin
      y0 <= a0 when (s1 = '0' and s0 = '0') else
```

```
                  bØ when (s1 = 'Ø' and sØ = '1') else
                  cØ when (s1 = '1' and sØ = 'Ø') else
                  dØ when (s1 = '1' and sØ = '1');
      y1 <= a1 when (s1 = 'Ø' and sØ = 'Ø') else
                  b1 when (s1 = 'Ø' and sØ = '1') else
                  c1 when (s1 = '1' and sØ = 'Ø') else
                  d1 when (s1 = '1' and sØ = '1');
      y2 <= a2 when (s1 = 'Ø' and sØ = 'Ø') else
                  b2 when (s1 = 'Ø' and sØ = '1') else
                  c2 when (s1 = '1' and sØ = 'Ø') else
                  d2 when (s1 = '1' and sØ = '1');
      y3 <= a3 when (s1 = 'Ø' and sØ = 'Ø') else
                  b3 when (s1 = 'Ø' and sØ = '1') else
                  c3 when (s1 = '1' and sØ = 'Ø') else
                  d3 when (s1 = '1' and sØ = '1');
end Behavioral;
```

4x4Bit Register:

Truth Table 2:

S1	SØ	CLK	D3	D2	D1	DØ	Q3	Q2	Q1	QØ
Ø	Ø	POSEDGE	Ø	Ø	1	1	Ø	Ø	1	1
Ø	Ø	X	X	X	X	X	Ø	Ø	1	1
Ø	1	POSEDGE	1	Ø	1	Ø	1	Ø	1	Ø
Ø	1	X	X	X	X	X	1	Ø	1	Ø
1	Ø	POSEDGE	Ø	1	Ø	1	Ø	1	Ø	1
1	Ø	X	X	X	X	X	Ø	1	Ø	1
1	1	POSEDGE	1	1	Ø	Ø	1	1	Ø	Ø
1	1	X	X	X	X	X	1	1	Ø	Ø
Ø	Ø	X	X	X	X	X	Ø	Ø	1	1
Ø	1	X	X	X	X	X	1	Ø	1	Ø
1	Ø	X	X	X	X	X	Ø	1	Ø	1
1	1	X	X	X	X	X	1	1	Ø	Ø

```
entity fourbyfour is
    Port ( S : in  STD_LOGIC_VECTOR(1 downto Ø);
           w : in  STD_LOGIC;
           clock : in  STD_LOGIC;
                    Din : in STD_LOGIC_VECTOR(3 downto Ø);
           Y : out  STD_LOGIC_VECTOR(3 downto Ø));

end fourbyfour;

architecture Behavioral of fourbyfour is
COMPONENT twofourdecoder
      PORT(
            a : IN std_logic_vector(1 downto Ø);
            EN : IN std_logic;
            b : OUT std_logic_vector(3 downto Ø)
            );
      END COMPONENT;
COMPONENT fourbitedgetrigger
      PORT(
            CLK : IN std_logic;
            D : IN std_logic_vector(3 downto Ø);
            Q : OUT std_logic_vector(3 downto Ø)
```

```
        );
    END COMPONENT;

    COMPONENT multiplexer4_1
        PORT(
            s0 : IN std_logic;
            s1 : IN std_logic;
            a0 : IN std_logic;
            a1 : IN std_logic;
            a2 : IN std_logic;
            a3 : IN std_logic;
            b0 : IN std_logic;
            b1 : IN std_logic;
            b2 : IN std_logic;
            b3 : IN std_logic;
            c0 : IN std_logic;
            c1 : IN std_logic;
            c2 : IN std_logic;
            c3 : IN std_logic;
            d0 : IN std_logic;
            d1 : IN std_logic;
            d2 : IN std_logic;
            d3 : IN std_logic;
            y0 : OUT std_logic;
            y1 : OUT std_logic;
            y2 : OUT std_logic;
            y3 : OUT std_logic
            );
        END COMPONENT;

    signal asig: STD_LOGIC_VECTOR(3 downto 0);
    signal bsig: STD_LOGIC_VECTOR(3 downto 0);
    signal csig: STD_LOGIC_VECTOR(3 downto 0);
    signal dsig: STD_LOGIC_VECTOR(3 downto 0);
    signal decodesig: STD_LOGIC_VECTOR(3 downto 0);
    signal enablesig: STD_LOGIC;
begin

enablesig <= W and CLOCK;
    Inst_twofourdecoder: twofourdecoder PORT MAP(
        a => S,
        b => decodesig,
        EN => enablesig
    );
    Inst_fourbitedgetriggerA: fourbitedgetrigger PORT MAP(
            CLK => decodesig(0),
            D => Din,
            Q => asig
    );
        Inst_fourbitedgetriggerB: fourbitedgetrigger PORT MAP(
            CLK => decodesig(1),
            D => Din,
            Q => bsig
    );
        Inst_fourbitedgetriggerC: fourbitedgetrigger PORT MAP(
            CLK => decodesig(2),
            D => Din,
```

```
                Q => csig
        );
              Inst_fourbitedgetriggerD: fourbitedgetrigger PORT MAP(
                    CLK => decodesig(3),
                    D => Din,
                    Q => dsig
        );
        Inst_multiplexer4_1: multiplexer4_1 PORT MAP(
              sØ => S(Ø),
              s1 => S(1),
              aØ => asig(Ø),
              a1 => asig(1),
              a2 => asig(2),
              a3 => asig(3),
              bØ => bsig(Ø),
              b1 => bsig(1),
              b2 => bsig(2),
              b3 => bsig(3),
              cØ => csig(Ø),
              c1 => csig(1),
              c2 => csig(2),
              c3 => csig(3),
              dØ => dsig(Ø),
              d1 => dsig(1),
              d2 => dsig(2),
              d3 => dsig(3),
              yØ => Y(Ø),
              y1 => Y(1),
              y2 => Y(2),
              y3 => Y(3)
        );

    end Behavioral;
```

Procedure:

For the first part of the lab, a 4 bit positive edge triggered register was designed using VHDL. This register was tested for inputs of Ø, 2, 1Ø, 12, and 3. The register was then downloaded and tested on the board. A VHDL code was then written for a 4x4 bit register bank using 4 of the registers made just prior, as well as the 2:4 decoder and 16:4 MUX. The register bank was the compiled and simulated in the exact order below. By testing it in this order, the bank was able to save the outputs for certain inputs.

W = Ø run 8ns
(W = Ø) S1SØ = ØØ D3D2 D1DØ = ØØ11 run 8ns
W = 1 run 8ns
W = Ø run 8ns
(W = Ø) S1SØ = Ø1 D3D2 D1DØ = 1Ø1Ø run 8ns
W = 1 run 8ns W = Ø run 8ns
(W = Ø) S1SØ = 1Ø D3D2 D1DØ = Ø1Ø1 run 8ns
W = 1 run 8ns W = Ø run 8ns
(W = Ø) S1SØ = 11 D3D2 D1DØ = 11ØØ run 8ns
W = 1 run 8ns W = Ø run 8ns

(W = Ø) S1SØ = ØØ run 8ns Check the outputs Y3 through YØ
(W = Ø) S1SØ = Ø1 run 8ns Check the outputs Y3 through YØ
(W = Ø) S1SØ = 1Ø run 8ns Check the outputs Y3 through YØ
(W = Ø) S1SØ = 11 run 8ns Check the outputs Y3 through YØ

Data: See attached Xilinx simulation results.

Data Analysis: The data collected using Xilinx simulation and the on-board testing matched with the data expected. The simulation of the 4 bit register matched Truth Table 1 and the 4x4Bit register matched Truth Table 2. The 2:4 decoder was not tested, but is assumed to have worked correctly as the 4x4Bit register bank worked correctly. The quad 4:1 multiplexer was used in Lab 7, and worked correctly.

Conclusion: In this lab, the objective was to create registers and register banks using the HDL coding language. The register bank allowed more practice with implementing smaller designs as components into a larger design. The lab also provided practice implementing sequential circuits instead of combinational circuits. This is important as many designs are based on time instead of simply individual inputs. A problem encountered was that our register bank would not simulate properly, and many different potential solutions were tested. When the design was simply copied and pasted into a new project, the design simulated properly and could be put on a board.