Sequential Logic Design using VHDL

Brihat Ratna Bajracharya

*Department of Electronics and Computer Engineering, IOE Central Campus, Pulchowk  
Lalitpur, Nepal*

070bct513@ioe.edu.np

***Abstract*—VHDL is a language for describing digital electronic systems.** **VHDL is designed to fill a number of needs in the design process. Firstly, it allows description of the structure of a design, i.e. how it is decomposed into sub-designs, and how those sub-designs are interconnected. Secondly, it allows the specification of the function of designs using familiar programming language forms. Thirdly, as a result, it allows a design to be simulated before being manufactured, so that designers can quickly compare alternatives and test for correctness without the delay and expense of hardware prototyping.**

1. Introduction

VHDL stands for VHSIC (Very High Speed Integrated Circuit) Hardware Descriptive Language. It is used in electronic design automation to describe digital and mixed-signal systems such as field-programmable gate arrays and integrated circuits. VHDL can also be used as a general programming language. VHDL was originally developed at the behest of the US Department of Defence in order to document the behaviour of the ASICs that supplier companies were including in equipment. VHDL is influenced from Ada and Pascal. The initial release of VHDL was designed as per IEEE 1076 standard in 1987. The most commonly used VHDL version supported by CAD tools is IEEE 1076 1993.

1. *Features of VHDL*

* It is a hardware descriptive language used for design entry and simulation of digital circuits.
* It is an event-driven language: i.e. whenever an event occurs on signals in VHDL, it triggers the execution of a statement.
* It allows both concurrent as well as sequential modelling.
* It gives the flexibility to define data types that are specific to user needs apart from predefined types.
* It supports code reusability and code sharing via packages and user defined libraries.
* It is case-insensitive i.e. it does not differentiate between lowercase and uppercase letters.
* It is strongly typed language i.e. it does not support implicit conversion between data types.

1. *Levels of representation and abstraction*

A digital system can be represented at different levels of abstraction. This keeps the description and design of complex systems manageable.

1. Behavioral – The highest level of abstraction that describes a system in terms of what it does (or how it behaves) rather than in terms of its components and interconnection between them. A behavioural description specifies the relationship between the input and output signals. This could be a Boolean expression or a more abstract description such as the Register Transfer or Algorithmic level.
2. Structural – The structural level, on the other hand, describes a system as a collection of gates and components that are interconnected to perform a desired function. A structural description could be compared to a schematic of interconnected logic gates. It is a representation that is usually closer to the physical realization of a system.
3. Sequential Statements

The logic circuits in which the values of the output depends not only on the present values of the inputs but also on the past behaviour of the circuit are referred to as sequential circuits. A sequential circuit is described in terms of logic conditions called logic states. Sequential circuits include memory elements that store the values of the logic states. When the circuit's input changes values, the new input values either leave the circuit in the same state or cause it to change into a new state. Over time the circuit changes through a sequence of states as a result of changes in the inputs. Circuits that behave in this way are referred to as sequential circuits. The general structure of a sequential circuit is shown in figure.

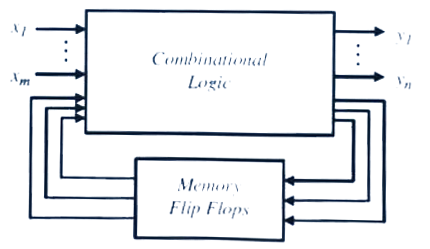


Fig. Sequential circuit block diagram

A stable state of a sequential circuit is described as previous, present, or next. The present state is the present logic output of the circuit. The previous state is the logic output of the circuit before the present state. The next state is the logic output of the circuit after the present state. The previous state cannot jump to the next state without going through the present state.

1. Activity I

Write VHDL code to implement a JK flip-flop using a D flip-flop and the characteristic equation given. The D flip-flop needs to be implemented using a structural architecture style using the circuit shown in figure. The JK flip-flop must be constructed using a component declaration for a D flip-flop.

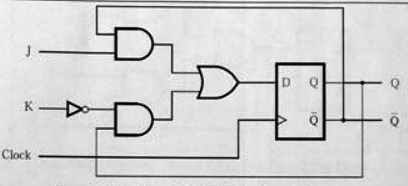




Fig. JK flip-flop using D flip-flop and characteristic equation

Write a VHDL test bench to verify the operation of the JK flip-flop and provide waveform.

**VHDL Code**

**[comp1.vhd]**

LIBRARY IEEE;

USE IEEE.STD\_LOGIC\_1164.ALL;

ENTITY comp1 IS PORT (

i1, i2, a3, a4: IN STD\_LOGIC;

o1: OUT STD\_LOGIC

);

END comp1;

ARCHITECTURE dataflow OF comp1 IS

BEGIN

o1 <= ((i1 AND a4) or ((not i2) and a3));

END dataflow;

**[nand1.vhd]**

LIBRARY IEEE;

USE IEEE.STD\_LOGIC\_1164.ALL;

ENTITY NAND1 IS

PORT(a , b: IN std\_logic;

o: OUT std\_logic);

END NAND1;

ARCHITECTURE DATAFLOW OF NAND1 IS

BEGIN

o <= a NAND b;

END DATAFLOW;

**[nand2.vhd]**

LIBRARY IEEE;

USE IEEE.STD\_LOGIC\_1164.ALL;

ENTITY NAND\_2 IS

PORT(a, b, c: IN std\_logic;

o: OUT std\_logic);

END NAND\_2;

ARCHITECTURE DATAFLOW OF NAND\_2 IS

BEGIN

o <= NOT(a AND b AND c);

END DATAFLOW;

**[dff.vhd]**

LIBRARY IEEE;

USE IEEE.STD\_LOGIC\_1164.ALL;

ENTITY dff IS

PORT( CLK , DATA: IN std\_logic;

Q, QBAR: OUT std\_logic);

END dff;

ARCHITECTURE STRUC OF dff IS

SIGNAL A, I1, I2, I3, I4: STD\_LOGIC;

SIGNAL I5: STD\_LOGIC := '0';

SIGNAL I6: STD\_LOGIC := '1';

COMPONENT NAND1

PORT(a,b: IN std\_logic;

o : OUT std\_logic

);

END COMPONENT;

COMPONENT NAND\_2

PORT(a,b,c: IN std\_logic;

o : OUT std\_logic

);

END COMPONENT;

BEGIN

Q <= I5;

QBAR <= I6;

O1: NAND1 PORT MAP (a => I2,

b => I4, o=> I1);

O2: NAND1 PORT MAP (a => CLK,

b => I1, o=> I2);

O4: NAND\_2 PORT MAP (a => I2,

b => CLK, c => I4, o=> I3)

O3: NAND1 PORT MAP (a => I3,

b => DATA, o=> I4);

O5: NAND1 PORT MAP (a => I2,

b => I6, o=> I5);

O6: NAND1 PORT MAP (a => I5,

b => I3, o=> I6);

END STRUC;

**[jkff.vhd]**

LIBRARY IEEE;

USE IEEE.STD\_LOGIC\_1164.ALL;

ENTITY jkff IS

PORT(CLOCK, J , K: IN std\_logic;

QT, QTBAR: INOUT std\_logic);

END jkff;

ARCHITECTURE STRUC OF jkff IS

SIGNAL A, A1: STD\_LOGIC;

SIGNAL A5: STD\_LOGIC := '0';

SIGNAL A6: STD\_LOGIC := '1';

COMPONENT dff

PORT( CLK,DATA: IN std\_logic;

Q, QBAR : INOUT std\_logic

);

END COMPONENT;

COMPONENT comp1

PORT(i1,i2,a3,a4:IN std\_logic;

o1: OUT std\_logic

);

END COMPONENT;

BEGIN

QT <= A5;

QTBAR <= A6;

O1: comp1 PORT MAP(i1 => J, i2 => K, a3 => A5, a4 => A6, o1 => A1);

D1: dff PORT MAP(CLK => CLOCK, DATA => A1, Q => A5, QBAR => A6);

END STRUC;

**Test bench**

LIBRARY IEEE;

USE IEEE.STD\_LOGIC\_1164.ALL;

USE IEEE.NUMERIC\_STD.ALL;

ENTITY jkff\_tb IS

END jkff\_tb;

ARCHITECTURE behavioral OF jkff\_tb IS

COMPONENT jkff

PORT(

CLOCK: IN STD\_LOGIC;

J: IN STD\_LOGIC;

K: IN STD\_LOGIC;

QT: INOUT STD\_LOGIC;

QTBAR: INOUT STD\_LOGIC

);

END COMPONENT;

SIGNAL clock: STD\_LOGIC;

SIGNAL input\_vector: STD\_LOGIC\_VECTOR (1 DOWNTO 0) := "00";

SIGNAL out1: STD\_LOGIC:= '0';

SIGNAL out2: STD\_LOGIC:= '1';

CONSTANT clk\_p: time:= 10 ns;

BEGIN

uut: jkff PORT MAP(

CLOCK => clock,

J => input\_vector(1),

K => input\_vector(0),

QT => out1,

QTBAR => out2

);

clkproc: PROCESS

BEGIN

clock <= '0';

WAIT FOR clk\_p/2;

clock <= '1';

WAIT FOR clk\_p/2;

END PROCESS clkproc;

stim\_proc: PROCESS

BEGIN

FOR index IN 0 TO 3 LOOP

input\_vector <= std\_logic\_vector (to\_unsigned(index,2));

WAIT FOR 37 ns;

END LOOP;

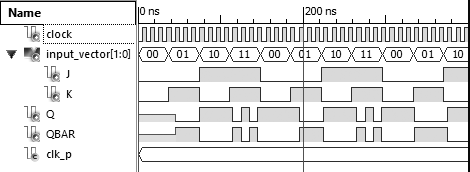
END PROCESS;

END behavioral;

Discussion:

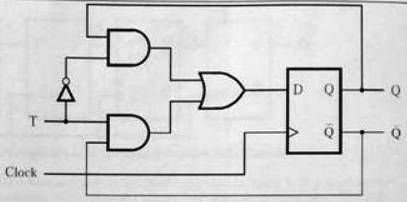
In this activity, we need to implement JK flip-flop using D flip-flop. Both JK and D flip-flop are implemented using component declaration. Separate components are made in separate files like nand1, nand2, comp1, dff and finally integrated in jkff. A test bench is written to verify the design of JK flip-flop.

Output:



1. Activity II

Write VHDL code to implement a T flip-flop using a D flip-flop and the characteristic equation given. The D flip-flop needs to be implemented using a structural architecture style using the circuit shown in figure. The T flip-flop must be constructed using a component declaration for a D flip-flop.



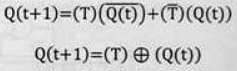


Fig. T flip-flop using D flip flop and its characteristic equation

Write a VHDL test bench to verify the operation of the T flip-flop and provide waveform

**VHDL Code**

**[comp1.vhd]**

LIBRARY IEEE;

USE IEEE.STD\_LOGIC\_1164.ALL;

ENTITY comp1 IS PORT (

i1, i2, a3, a4: IN STD\_LOGIC;

o1: OUT STD\_LOGIC

);

END comp1;

ARCHITECTURE dataflow OF comp1 IS

BEGIN

o1 <= ((not i2 and a4) or (i1 and a3));

END dataflow;

**[nand1.vhd]**

LIBRARY IEEE;

USE IEEE.STD\_LOGIC\_1164.ALL;

ENTITY NAND1 IS

PORT(a , b: IN std\_logic;

o: OUT std\_logic);

END NAND1;

ARCHITECTURE DATAFLOW OF NAND1 IS

BEGIN

o <= a NAND b;

END DATAFLOW;

**[nand2.vhd]**

LIBRARY IEEE;

USE IEEE.STD\_LOGIC\_1164.ALL;

ENTITY NAND\_2 IS

PORT(a, b, c: IN std\_logic;

o: OUT std\_logic);

END NAND\_2;

ARCHITECTURE DATAFLOW OF NAND\_2 IS

BEGIN

o <= NOT(a AND b AND c);

END DATAFLOW;

**[dff.vhd]**

LIBRARY IEEE;

USE IEEE.STD\_LOGIC\_1164.ALL;

ENTITY dff IS

PORT( CLK , DATA: IN std\_logic;

Q, QBAR: OUT std\_logic);

END dff;

ARCHITECTURE STRUC OF dff IS

SIGNAL A, I1, I2, I3, I4: STD\_LOGIC;

SIGNAL I5: STD\_LOGIC := '0';

SIGNAL I6: STD\_LOGIC := '1';

COMPONENT NAND1

PORT(a,b: IN std\_logic;

o : OUT std\_logic

);

END COMPONENT;

COMPONENT NAND\_2

PORT(a,b,c: IN std\_logic;

o : OUT std\_logic

);

END COMPONENT;

BEGIN

Q <= I5;

QBAR <= I6;

O1: NAND1 PORT MAP (a => I2,

b => I4, o=> I1);

O2: NAND1 PORT MAP (a => CLK,

b => I1, o=> I2);

O4: NAND\_2 PORT MAP (a => I2,

b => CLK, c => I4, o=> I3)

O3: NAND1 PORT MAP (a => I3,

b => DATA, o=> I4);

O5: NAND1 PORT MAP (a => I2,

b => I6, o=> I5);

O6: NAND1 PORT MAP (a => I5,

b => I3, o=> I6);

END STRUC;

**[tff.vhd]**

LIBRARY IEEE;

USE IEEE.STD\_LOGIC\_1164.ALL;

ENTITY tff IS

PORT( CLOCK, T, TNOT: IN STD\_LOGIC;

QT, QTBAR: INOUT STD\_LOGIC);

END tff;

ARCHITECTURE STRUC OF tff IS

SIGNAL A, A1: STD\_LOGIC;

SIGNAL A5: STD\_LOGIC := '0';

SIGNAL A6: STD\_LOGIC := '1';

COMPONENT dff

PORT( CLK, DATA: IN STD\_LOGIC;

Q, QBAR : INOUT STD\_LOGIC

);

END COMPONENT;

COMPONENT comp1

PORT(i1,i2,a3,a4:IN STD\_LOGIC;

o1: OUT STD\_LOGIC

);

END COMPONENT;

BEGIN

QT <= A5;

QTBAR <= A6;

O1: comp1 PORT MAP(i1 => T, i2 => TNOT, a3 => A6, a4 => A5, o1 => A1);

D1: dff PORT MAP(CLK => CLOCK, DATA => A1, Q => A5, QBAR => A6);

END STRUC;

**Test Bench**

LIBRARY IEEE;

USE IEEE.STD\_LOGIC\_1164.ALL;

USE IEEE.NUMERIC\_STD.ALL;

ENTITY tff\_tb IS

END tff\_tb;

ARCHITECTURE behavioral OF tff\_tb IS

COMPONENT tff

PORT(

CLOCK: IN STD\_LOGIC;

T: IN STD\_LOGIC;

TNOT: IN STD\_LOGIC;

QT: INOUT STD\_LOGIC;

QTBAR: INOUT STD\_LOGIC

);

END COMPONENT;

SIGNAL clk: STD\_LOGIC;

SIGNAL input, ninput: STD\_LOGIC;

SIGNAL out1: STD\_LOGIC:= '1';

SIGNAL out2: STD\_LOGIC:= '1';

CONSTANT clk\_p: time:= 10 ns;

BEGIN

uut: tff PORT MAP(

CLOCK => clk,

T => input,

TNOT => ninput,

QT => out1,

QTBAR => out2

);

clkproc: PROCESS

BEGIN

clk <= '0';

WAIT FOR clk\_p/2;

clk <= '1';

WAIT FOR clk\_p/2;

END PROCESS clkproc;

stim\_proc: PROCESS

BEGIN

if out1 = 'U' then

input <= '0';

ninput <= '0';

WAIT FOR 37 ns;

end if;

input <= '0';

ninput <= not input;

WAIT FOR 37 ns;

input <= '1';

ninput <= not input;

WAIT FOR 37 ns;

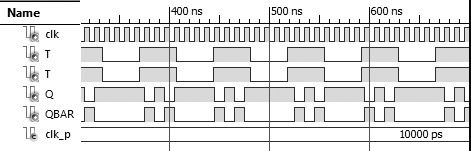
END PROCESS stim\_proc;

END behavioral;

Discussion:

As in first activity, D flip-flop is to be implemented as component for designing T flip-flop. So, component required for D flip-flop is same as in previous activity. Only small portion of code in JK flip-flop is changed for T flip-flop as T flip-flop has only one input and behaves same as JK flip-flop while giving same value in J and K. A test bench is written to verify the operation of T flip-flop.

Output:



1. Activity III

Use VHDL to implement a 4-bit serial in serial out (SISO) right-shift register. Determine the output of the shift register after the input sequence 01010101 has been shifted eight times starting with the most significant bit. Assume that the output of the shift register is reset initially to 0000. The shift-register must be constructed with D flip-flops using a component declaration for a D flip-flop.

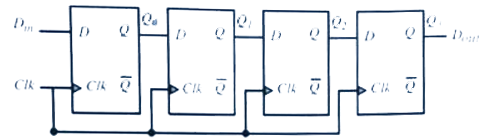


Fig. SISO right shift register using D flip-flop

Write a VHDL test bench to verify the operation of the 4-bit SISO and provide appropriate waveforms.

**VHDL Code**

**[nand1.vhd]**

LIBRARY IEEE;

USE IEEE.STD\_LOGIC\_1164.ALL;

ENTITY NAND1 IS

PORT(a , b: IN std\_logic;

o: OUT std\_logic);

END NAND1;

ARCHITECTURE DATAFLOW OF NAND1 IS

BEGIN

o <= a NAND b;

END DATAFLOW;

**[nand2.vhd]**

LIBRARY IEEE;

USE IEEE.STD\_LOGIC\_1164.ALL;

ENTITY NAND\_2 IS

PORT(a, b, c: IN std\_logic;

o: OUT std\_logic);

END NAND\_2;

ARCHITECTURE DATAFLOW OF NAND\_2 IS

BEGIN

o <= NOT(a AND b AND c);

END DATAFLOW;

**[dff.vhd]**

LIBRARY IEEE;

USE IEEE.STD\_LOGIC\_1164.ALL;

ENTITY dff IS

PORT( CLK , DATA: IN std\_logic;

Q, QBAR: OUT std\_logic);

END dff;

ARCHITECTURE STRUC OF dff IS

SIGNAL A, I1, I2, I3, I4: STD\_LOGIC;

SIGNAL I5: STD\_LOGIC := '0';

SIGNAL I6: STD\_LOGIC := '1';

COMPONENT NAND1

PORT( a,b: IN std\_logic;

o : OUT std\_logic

);

END COMPONENT;

COMPONENT NAND\_2

PORT( a,b,c: IN std\_logic;

o : OUT std\_logic

);

END COMPONENT;

BEGIN

Q <= I5;

QBAR <= I6;

O1: NAND1 PORT MAP (a => I2,

b => I4, o=> I1);

O2: NAND1 PORT MAP (a => CLK,

b => I1, o=> I2);

O4: NAND\_2 PORT MAP (a => I2,

b => CLK, c => I4, o=> I3)

O3: NAND1 PORT MAP (a => I3,

b => DATA, o=> I4);

O5: NAND1 PORT MAP (a => I2,

b => I6, o=> I5);

O6: NAND1 PORT MAP (a => I5,

b => I3, o=> I6);

END STRUC;

**[siso.vhd]**

LIBRARY IEEE;

USE IEEE.STD\_LOGIC\_1164.ALL;

ENTITY siso IS

PORT( CLOCK, Din: IN STD\_LOGIC;

QA, QB, QC: INOUT STD\_LOGIC;

Dout, QTBAR: INOUT STD\_LOGIC);

END siso;

ARCHITECTURE STRUC OF siso IS

SIGNAL Q0, Q0B, Q1, Q1B, Q2, Q2B, Q3, Q3B, A1: STD\_LOGIC;

SIGNAL A5: STD\_LOGIC := '0';

SIGNAL A6: STD\_LOGIC := '1';

COMPONENT dff

PORT( CLK, DATA: IN STD\_LOGIC;

Q, QBAR : INOUT STD\_LOGIC

);

END COMPONENT;

COMPONENT comp1

PORT(i1,i2,a3,a4:IN STD\_LOGIC;

o1: OUT STD\_LOGIC

);

END COMPONENT;

BEGIN

Dout <= A5;

QTBAR <= A6;

QA <= Q0;

QB <= Q1;

QC <= Q2;

D1: dff PORT MAP (CLK => CLOCK, DATA => Din, Q => Q0, QBAR => Q0B);

D2: dff PORT MAP (CLK => CLOCK, DATA => Q0, Q => Q1, QBAR => Q1B);

D3: dff PORT MAP (CLK => CLOCK, DATA => Q1, Q => Q2, QBAR => Q2B);

D4: dff PORT MAP (CLK => CLOCK, DATA => Q2, Q => A5, QBAR => A6);

END STRUC;

**Test Bench**

LIBRARY IEEE;

USE IEEE.STD\_LOGIC\_1164.ALL;

USE IEEE.NUMERIC\_STD.ALL;

ENTITY siso\_tb IS

END siso\_tb;

ARCHITECTURE behavioral OF siso\_tb IS

COMPONENT siso

PORT(

CLOCK: IN STD\_LOGIC;

Din: IN STD\_LOGIC;

QA, QB, QC: INOUT STD\_LOGIC;

Dout: INOUT STD\_LOGIC;

QTBAR: INOUT STD\_LOGIC

);

END COMPONENT;

SIGNAL clk: STD\_LOGIC;

SIGNAL input: STD\_LOGIC;

signal input\_seq: STD\_LOGIC\_VECTOR(7 downto 0):= "10101010";

SIGNAL qa, qb, qc: std\_logic;

SIGNAL out1: STD\_LOGIC:= '1';

CONSTANT clk\_p: time:= 10 ns;

BEGIN

uut: siso PORT MAP(

CLOCK => clk,

QA => qa,

QB => qb,

QC => qc,

Din => input,

Dout => out1

);

clkproc: PROCESS

BEGIN

clk <= '0';

WAIT FOR clk\_p/2;

clk <= '1';

WAIT FOR clk\_p/2;

END PROCESS clkproc;

stim\_proc: PROCESS

BEGIN

for index in 1 to 8 loop

input <= input\_seq(7);

input\_seq(7 downto 1) <=

input\_seq(6 downto 0);

input\_seq(0) <= '0';

WAIT FOR 15 ns;

end loop;

wait;

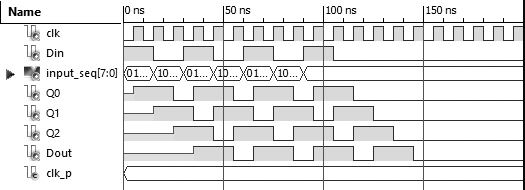
END PROCESS stim\_proc;

END behavioral;

Discussion:

This is another example of using D flip-flop as component. Here D flip-flop is implemented to design shift register. Four D flip-flop is required to design four bit serial in serial out right shift register. D flip-flop code in vhdl is borrowed from previous activity and structural declaration of shift register is made using D flip-flop as its major component. A test bench was made and checked for input sequence of ‘10101010’ whose output is given below.

Output:



1. Activity IV

Use VHDL to implement a 4-bit synchronous up counter. In a synchronous counter all flip-flop receive a common clock signal and change their states at the same time. The shift-register must be constructed with T flip-flops using a component declaration for a T flip-flop.



Fig. Synchronous up counter using T flip-flop

Write a VHDL test bench to verify the operation of the 4-bit synchronous up counter and provide appropriate waveforms.

**VHDL Code**

**[tff.vhd]**

LIBRARY IEEE;

USE IEEE.STD\_LOGIC\_1164.ALL;

ENTITY tff IS

PORT(CLOCK, RESET, T: IN STD\_LOGIC;

QT: OUT STD\_LOGIC);

END tff;

ARCHITECTURE behav OF tff IS

SIGNAL output: STD\_LOGIC;

BEGIN

PROCESS(RESET, CLOCK)

BEGIN

IF RESET = '1' THEN

output <= '0';

ELSIF CLOCK'EVENT AND CLOCK='1' THEN

IF T = '0' THEN

output <= output;

ELSIF T = '1' THEN

output <= NOT output;

ELSE output <= 'U';

END IF;

END IF;

END PROCESS;

QT <= output;

END BEHAV;

**[syncup.vhd]**

LIBRARY IEEE;

USE IEEE.STD\_LOGIC\_1164.ALL;

ENTITY syncup IS

PORT(clock,reset,input:IN STD\_LOGIC;

data: out std\_logic\_vector(3 downto 0) );

END syncup;

architecture struc of syncup is

COMPONENT tff

PORT( CLOCK, RESET, T: IN STD\_LOGIC;

QT: OUT STD\_LOGIC

);

END COMPONENT;

signal temp: std\_logic\_vector(3 downto 0) := "0000";

signal and\_1, and\_2: std\_logic;

begin

and\_1 <= temp(0) and temp(1);

and\_2 <= temp(2) and and\_1;

d0 : tff port map (CLOCK => clock, RESET => reset, T => '1', QT => temp(0));

d1 : tff port map (CLOCK => clock, RESET => reset, T => temp(0), QT => temp(1));

d2 : tff port map (CLOCK => clock, RESET => reset, T => and\_1, QT => temp(2));

d3 : tff port map (CLOCK => clock, RESET => reset, T => and\_2, QT => temp(3));

data <= temp;

end struc;

**Test Bench**

LIBRARY ieee;

USE ieee.std\_logic\_1164.ALL;

USE ieee.std\_logic\_unsigned.all;

USE ieee.numeric\_std.ALL;

ENTITY syncup\_tb IS

END syncup\_tb;

ARCHITECTURE behavior OF syncup\_tb IS

COMPONENT syncup

PORT(

clock: IN std\_logic;

input: IN std\_logic;

reset: IN std\_logic;

data: out std\_logic\_vector(3 downto 0) );

END COMPONENT;

SIGNAL clock: std\_logic := '0';

SIGNAL reset: std\_logic := '1';

SIGNAL input: std\_logic := '1';

SIGNAL data: std\_logic\_vector(3 downto 0);

BEGIN

uut: syncup PORT MAP(

clock => clock,

input => input,

reset => reset,

data => data

);

clk: PROCESS

BEGIN

wait for 5ns;

clock <= not clock;

END PROCESS clk;

main: PROCESS

BEGIN

wait for 7 ns;

reset <= '0';

wait for 20ns;

input <= '1';

wait;

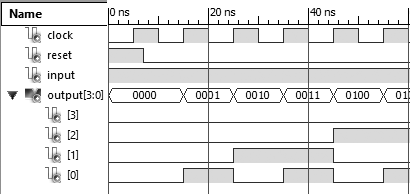
END PROCESS main;

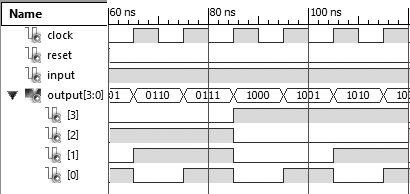
END behavior;

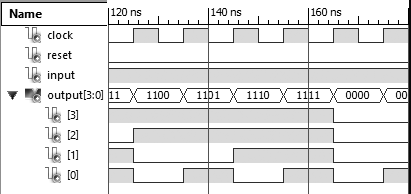
Discussion:

Synchronous up counter is made using component declaration for T flip-flop. For simplicity T flip-flop is made in behavioural model. While declaring component for synchronous counter, block diagram given with the question provided great help in proper connection of components. A test bench is written to verify the operation of this activity.

Output :







1. Activity V

Use VHDL to design an asynchronous decade counter. The 10 states of a decade counter represent the BCD numbers from 0 to 9. In asynchronous counters, only the first flip-flop is clocked by an external clock signal, and each successive flip-flop is clocked by the output of the preceding flip-flop. Asynchronous counters are also referred to as ripple counters because the information ripples from the less significant bit to the more significant bit, one bit at a time. The asynchronous decade counter must be constructed with T flip-flops using a component declaration for a T flip-flop.

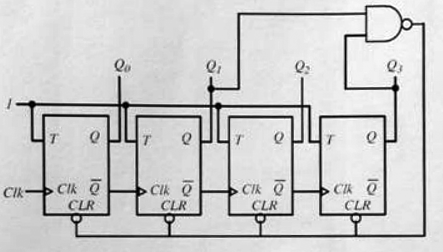


Fig. Asynchronous Decade Counter

Write a VHDL test bench to verify the operation of the decade counter and provide appropriate waveforms.

**VHDL Code**

**[tff.vhd]**

LIBRARY IEEE;

USE IEEE.STD\_LOGIC\_1164.ALL;

ENTITY tff IS

PORT(CLOCK, RESET, T: IN STD\_LOGIC;

QT: OUT STD\_LOGIC);

END tff;

ARCHITECTURE behav OF tff IS

SIGNAL output: STD\_LOGIC;

BEGIN

PROCESS(RESET, CLOCK)

BEGIN

IF RESET = '1' THEN

output <= '0';

ELSIF CLOCK'EVENT AND CLOCK='1' THEN

IF T = '0' THEN

output <= output;

ELSIF T = '1' THEN

output <= NOT output;

ELSE output <= 'U';

END IF;

END IF;

END PROCESS;

QT <= output;

END BEHAV;

**[asyncdec.vhd]**

LIBRARY IEEE;

USE IEEE.STD\_LOGIC\_1164.ALL;

ENTITY asyncdec IS

PORT( clock, input: IN STD\_LOGIC;

data : out std\_logic\_vector(3 downto 0) );

END asyncdec;

architecture struc of asyncdec is

COMPONENT tff

PORT( CLOCK, RESET, T: IN STD\_LOGIC;

QT, QTBAR: OUT STD\_LOGIC

);

END COMPONENT;

signal temp : std\_logic\_vector(3 downto 0) := "0000";

signal ntemp : std\_logic\_vector(3 downto 0) := "1111";

signal and\_1 : std\_logic := '1';

begin

and\_1 <= temp(1) and temp(3);

d0 : tff port map (CLOCK => clock, RESET => and\_1, T => input, QT => temp(0), QTBAR => ntemp(0));

d1 : tff port map (CLOCK => ntemp(0), RESET => and\_1, T => input, QT => temp(1), QTBAR => ntemp(1));

d2 : tff port map (CLOCK => ntemp(1), RESET => and\_1, T => input, QT => temp(2), QTBAR => ntemp(2));

d3 : tff port map (CLOCK => ntemp(2), RESET => and\_1, T => input, QT => temp(3), QTBAR => ntemp(3));

data <= temp;

end struc;

1. **Test Bench**

LIBRARY ieee;

USE ieee.std\_logic\_1164.ALL;

USE ieee.std\_logic\_unsigned.all;

USE ieee.numeric\_std.ALL;

ENTITY asyncdec\_tb IS

END asyncdec\_tb;

ARCHITECTURE behavior OF asyncdec\_tb IS

COMPONENT asyncdec

PORT( clock : IN std\_logic;

input : IN std\_logic;

data : out std\_logic\_vector(3 downto 0) );

END COMPONENT;

SIGNAL clock : std\_logic := '1';

SIGNAL input : std\_logic;

SIGNAL data : std\_logic\_vector(3 downto 0) := "0000";

BEGIN

uut: asyncdec PORT MAP(

clock => clock,

input => input,

data => data

);

clk: PROCESS

BEGIN

wait for 5ns;

clock <= not clock;

END PROCESS clk;

main: PROCESS

BEGIN

input <= '1';

wait;

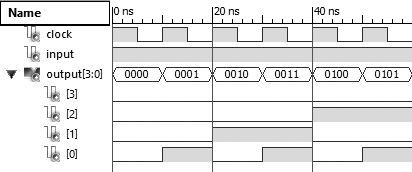
END PROCESS main;

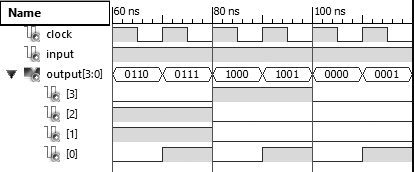
END behavior;

Discussion:

Asynchronous counter are those in which all flip-flops do not receive same clock pulse. Rather output from preceding flip-flop act as clock for next flip-flop. In this activity, we implemented T flip-flop as main component and designed an asynchronous decade counter. The operation of this counter was verified using a test bench simulation. Output waveform is shown below.

Output:





1. Activity VI

Use VHDL to design a four bit Johnson coounter. A Johnson counter generates a sequence of the binary numbers where only one bit position changes between two consecutive numbers. Starting with an initial value of Q0Q1Q2Q3 equal to 0000, the Johnson counter periodically generates the sequence 0000, 1000, 1100, 1110, 1111, 0111, 0011, 0001, and 0000. Notice that only one bit position changes between two consecutive numbers, as in the case of the Gray code. For a Johnson coounter to work properly, it must be reset initially to 0000. The Johnson counter must be constructed using a component declaration for a D flip-flops.

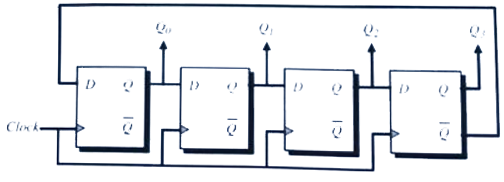


Fig. Johnson Counter

Write a VHDL test bench to verify the operation of the Johnson counter and provide appropriate waveforms.

**VHDL Code:**

**[dff.vhd]**

LIBRARY IEEE;

USE IEEE.STD\_LOGIC\_1164.ALL;

use IEEE.STD\_LOGIC\_ARITH.ALL;

use IEEE.STD\_LOGIC\_UNSIGNED.ALL;

ENTITY dff IS

PORT( CLK , DATA: IN std\_logic;

Q, QBAR: OUT std\_logic);

END dff;

ARCHITECTURE STRUC OF dff IS

BEGIN

PROCESS (CLK, DATA)

BEGIN

IF(CLK'EVENT AND CLK = '1') THEN Q <= DATA;

QBAR <= NOT DATA;

END IF;

END PROCESS;

END STRUC;

**[johnson.vhd]**

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

use IEEE.STD\_LOGIC\_ARITH.ALL;

use IEEE.STD\_LOGIC\_UNSIGNED.ALL;

entity johnson is Port (

clock : in std\_logic;

reset : in std\_logic;

data : out std\_logic\_vector(3 downto 0) );

end johnson;

architecture struc of johnson is

COMPONENT dff

PORT( clock : in std\_logic;

reset : in std\_logic;

d : in std\_logic;

q : out std\_logic

);

END COMPONENT;

signal temp : std\_logic\_vector(3 downto 0) := "0000";

signal wir : std\_logic := '0';

begin

wir <= not temp(0);

d0 : dff port map (reset => reset, clock => clock, d => wir, q => temp(3));

d1 : dff port map (reset => reset, clock => clock, d => temp(3), q => temp(2));

d2 : dff port map (reset => reset, clock => clock, d => temp(2), q => temp(1));

d3 : dff port map (reset => reset, clock => clock, d => temp(1), q => temp(0));

data <= temp;

end struc;

**Test Bench**

LIBRARY ieee;

USE ieee.std\_logic\_1164.ALL;

ENTITY johnson\_tb IS

END johnson\_tb;

ARCHITECTURE behavior OF johnson\_tb IS

COMPONENT johnson

PORT( clock : IN std\_logic;

reset : IN std\_logic;

data : out std\_logic\_vector(3 downto 0) );

END COMPONENT;

SIGNAL clock : std\_logic := '0';

SIGNAL reset : std\_logic := '1';

SIGNAL data : std\_logic\_vector(3 downto 0);

BEGIN

uut: johnson PORT MAP(

clock => clock,

reset => reset,

data => data

);

clk: PROCESS

BEGIN

wait for 5ns;

clock <= not clock;

END PROCESS clk;

main: PROCESS

BEGIN

reset <= '1';

wait for 20ns;

reset <= not reset;

wait;

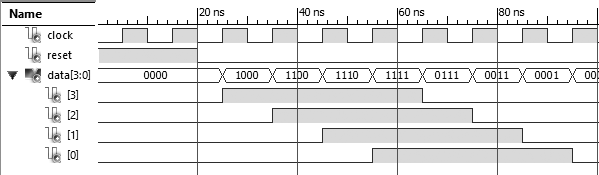
END PROCESS main;

END behavior;

Discussion:

Johnson counter is also called ring counter in which there is change in only one bit in one clock pulse. In above VHDL code, Johnson counter is made using component declaration for D flip-flop as instructed. A test bench is used for output waveform which is shown below.

Output:



1. Activity VII

Use VHDL to create a 2-bit BCD counter as shown. The BCD counter consists of two 1 bit BCD counters cascaded to form a 2-bit BCD counter. The 2-bit BCD counter counts from 00 to 99. Notice that both 1-bit BCD counters are initialized by a load data input of 0. The 2-bit BCD counter must be constructed using component declarations for the D flip-flops.

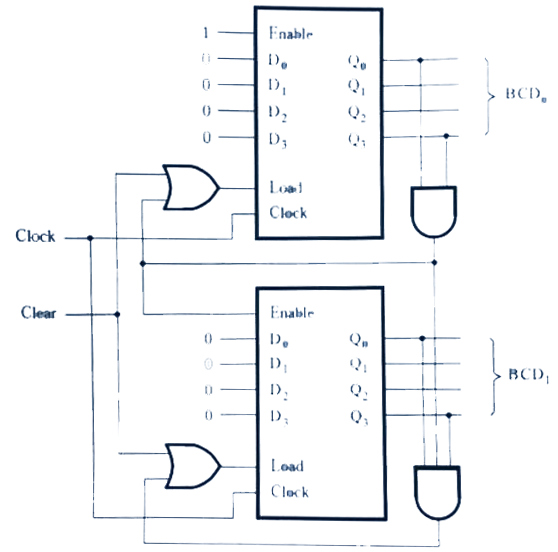


Fig. Block diagram for 2-bit BCD counter

**VHDL Code:**

**[dff.vhd]**

library ieee;

use IEEE.STD\_LOGIC\_1164.ALL;

use IEEE.STD\_LOGIC\_ARITH.ALL;

use IEEE.STD\_LOGIC\_UNSIGNED.ALL;

entity dff is

port (

enable: in STD\_LOGIC;

d : in STD\_LOGIC;

clock : in STD\_LOGIC;

reset : in STD\_LOGIC;

q : out STD\_LOGIC;

qb : out STD\_LOGIC

);

end dff;

architecture behave of dff is

begin

process(reset,clock,enable)

begin

if (enable = '1') then

if (reset = '1') then

q <= '0';

qb <= '1';

elsif (clock'event and clock = '1') then

q <= d;

qb <= not d;

end if;

end if;

end process;

end behave;

**[bcd1.vhd]**

LIBRARY IEEE;

USE IEEE.STD\_LOGIC\_1164.ALL;

ENTITY bit2dec IS

PORT(clock, reset: IN STD\_LOGIC;

output1 : out std\_logic\_vector

(3 downto 0);

output2 : out std\_logic\_vector

(3 downto 0) );

END bit2dec;

architecture struc of bit2dec is

COMPONENT dff

PORT(clock, enable, reset, d:

IN STD\_LOGIC;

q, qb : out std\_logic

);

END COMPONENT;

signal temp1: std\_logic\_vector (3 downto 0) := "0000";

signal temp2: std\_logic\_vector (3 downto 0) := "0101";

signal ntemp1: std\_logic\_vector (3 downto 0) := "1111";

signal ntemp2: std\_logic\_vector (3 downto 0) := "1010";

signal and\_1: std\_logic := '1';

signal and\_2: std\_logic := '1';

begin

d0 : dff port map (clock => clock, enable => '1', reset => and\_1, d => ntemp1(0), q => temp1(0), qb => ntemp1(0));

d1 : dff port map (clock => ntemp1(0), enable => '1', reset => and\_1, d => ntemp1(1), q => temp1(1), qb => ntemp1(1));

d2 : dff port map (clock => ntemp1(1), enable => '1', reset => and\_1, d => ntemp1(2), q => temp1(2), qb => ntemp1(2));

d3 : dff port map (clock => ntemp1(2), enable => '1', reset => and\_1, d => ntemp1(3), q => temp1(3), qb => ntemp1(3));

and\_1 <= temp1(1) and temp1(3);

output1 <= temp1;

d4 : dff port map (clock => and\_1, enable => '1' , reset => and\_2, d => ntemp2(0), q => temp2(0), qb => ntemp2(0));

d5 : dff port map (clock => ntemp2(0), enable => '1', reset => and\_2, d => ntemp2(1), q => temp2(1), qb => ntemp2(1));

d6 : dff port map (clock => ntemp2(1), enable => '1', reset => and\_2, d => ntemp2(2), q => temp2(2), qb => ntemp2(2));

d7 : dff port map (clock => ntemp2(2), enable => '1', reset => and\_2, d => ntemp2(3), q => temp2(3), qb => ntemp2(3));

and\_2 <= temp2(1) and temp2(3);

output2 <= temp2;

end struc;

**Test bench**

LIBRARY ieee;

USE ieee.std\_logic\_1164.ALL;

USE ieee.std\_logic\_unsigned.all;

USE ieee.numeric\_std.ALL;

ENTITY bit2dec\_tb IS

END bit2dec\_tb;

ARCHITECTURE behavior OF bit2dec\_tb IS

COMPONENT bit2dec

PORT( clock, reset: IN STD\_LOGIC;

output1 : out std\_logic\_vector(3 downto 0);

output2 : out std\_logic\_vector(3 downto 0)

);

END COMPONENT;

SIGNAL clock : std\_logic := '1';

SIGNAL reset : std\_logic := '1';

SIGNAL out1 : std\_logic\_vector (3 downto 0) := "0000";

SIGNAL out2 : std\_logic\_vector (3 downto 0) := "0101";

BEGIN

uut: bit2dec PORT MAP(

clock => clock,

reset => reset,

output1 => out1,

output2 => out2

);

clk: PROCESS

BEGIN

wait for 5ns;

clock <= not clock;

END PROCESS clk;

main: PROCESS

BEGIN

wait for 5 ns;

reset <= '0';

wait;

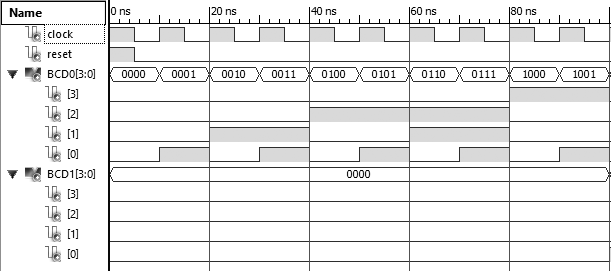
END PROCESS main;

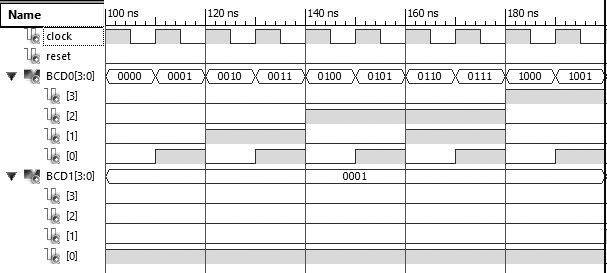
END behavior;

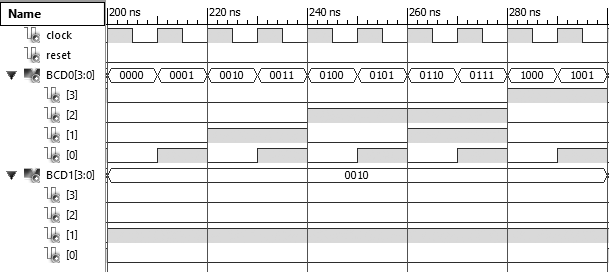
Discussion:

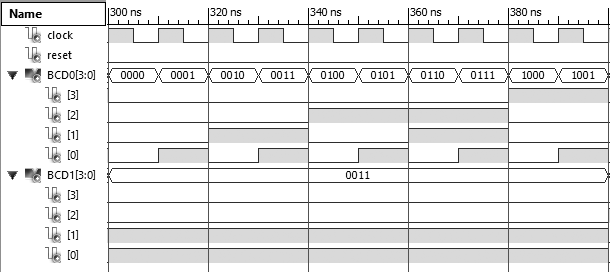
In this activity, we need to make 2 bit bcd counter using D flip-flop. This can be done using two one bit decade counter that can count from 0 to 9. For each cycle in least significant bit, one clock pulse is send to most significant bit. Hence clock of second 1 bit counter is connected to output of condition that trigger next bit in first 1 bit counter. VHDL code along with test bench is given above. BCD counter is implemented using D flip-flop which is defined in behavioural model for simplicity.

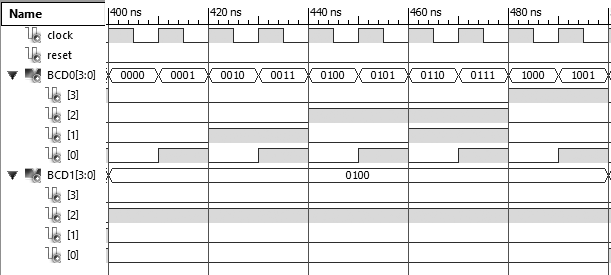
Output:

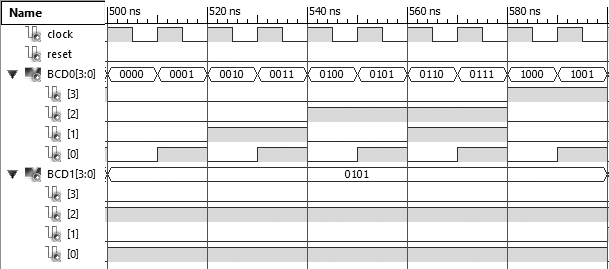


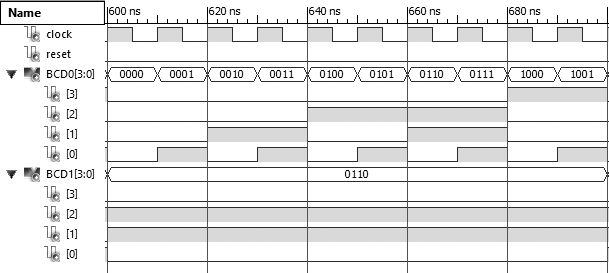


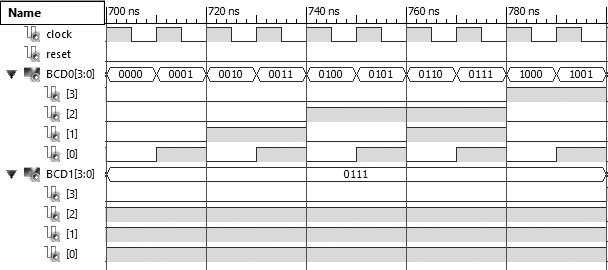


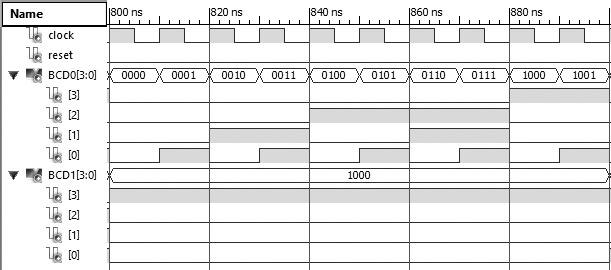


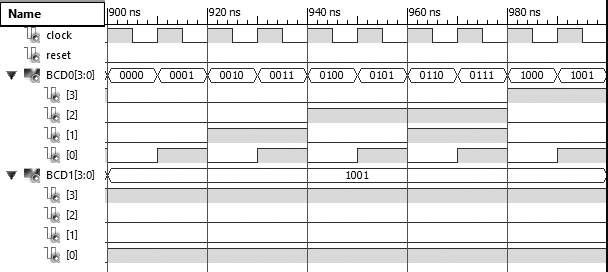












Appendix

**Appendix A**

Logic circuit for positive edge triggered D flip-flop

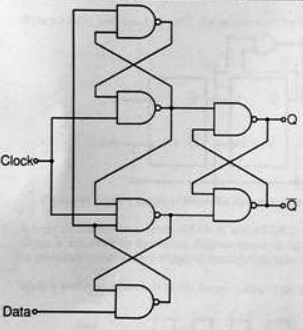


Figure showing positive edge triggered logic circuit using NAND Gate. This diagram is followed to implement D flip-flop in structural model

Conclusion

Various activities concerned with sequential logic design using VHDL programming were done. Different logic circuits were designed mostly using flip-flops mostly using structural model where components are defined in behavioral model for simplicity. Test bench for each activity is made to generate

output waveforms. Simulation was done in Xilinx ISE Design Suite. VHDL codes and waveform to all activities are included in this report.

Acknowledgment

This lab report is prepared as a document for activities done that is concerned with sequential logic design using VHDL programming. This report is made accurate and professional as far as possible. I would like to express our deepest gratitude to our teacher, Mr. Dinesh Baniya Kshatri, for guiding us in the practical. I am very grateful to the Department of Electronics and Computer Engineering (DoECE) of IOE Central Campus, Pulchowk for arranging such a schedule on our academic side.

References

[1] (2016) The Wikipedia website. [Online]. Available: <https://en.m.wikipedia.org/wiki/VHDL/>

[2] P. J. Ashenden, The VHDL Cookbook [portable document]. Available: <http://www.cs.adelaide.edu.au/>

[3] (2016) The edaboard website. [Online]. Available: <http://www.edaboard.com/thread/77216.html/>

[4] (2016) The Stack Overflow website. [Online]. Available: <http://www.stackflow.com/>

[5] (2016) The asic-world website. [Online]. Available: <http://www.asic-world.com/examples/vhdl/t_ff.html/>

[6] (2016) The ohio.edu website. [Online]. Available: <http://www.ohio.edu/people/starzykj/webcad/EE4143/packages.html/>

[7] (2016) DFF Test bench [Online]. Available: <http://esd.cs.ucr.edu/labs/tutorial/tb_dff.vhd/>

[8] (2016) The Wikibooks website. [Online]. Available: <http://en.m.wikibooks/wiki/VHDL_for_FPGA_Design/T_Flip_Flop/>

[9] S. Brown and Z. Vranesic, “Flip-flops, registers, counters and a simple processor” in Fundamental of *Digital Logic with VHDL Design,* 2nd ed., the United States: McGraw-Hill, 2005.

[10] D. Kshatri, Sequential Logic Design using VHDL [scanned document]. Available: Offline.

[11] S. Shrestha, VHDL [presentation]. Available: Offline