Edinburgh Napier University

Programmable Logic Design

ELE10105

Lab Logbook.

40292302

Table of Content

Task for week 1: Tutorial 1 Concurrent Design (Combinational Logic).

Task Date: 20.01.20

Objectives: Write behavioural code for example applications.
Familiarisation with Vivado.
Revision of Boolean simplification.

1. Combinational Logic.

Circuit Logic Simplification:
Truth table for output F:

A	B	C	D	F
0	0	0	0	0
0	0	0	1	0
0	0	1	0	1
0	0	1	1	0
0	1	0	0	0
0	1	0	1	0
0	1	1	0	0
0	1	1	1	1
1	0	0	0	0
1	0	0	1	0
1	0	1	0	1
1	0	1	1	0
1	1	0	0	0
1	1	0	1	0
1	1	1	0	1
1	1	1	1	0

F = (NOT A . NOT B . C . NOT D) + (A . NOT B . C . NOT D) + (A . B . C . NOT D) + (NOT A . B . C . D)

Karnaugh Map:

AB / **CD**	**00**	**01**	**11**	**10**
00	0	0	0	0
01	0	0	0	0
11	0	1	0	0
10	1	0	1	1

F = (C . NOT D. NOT B) + (A . B . C . NOT D) + (NOT A . B . C .D)

Using DeMorgan theorem and boolean algebra simplification:

F = (C . NOT D. NOT B) + [(B . C) . (A XOR D)]

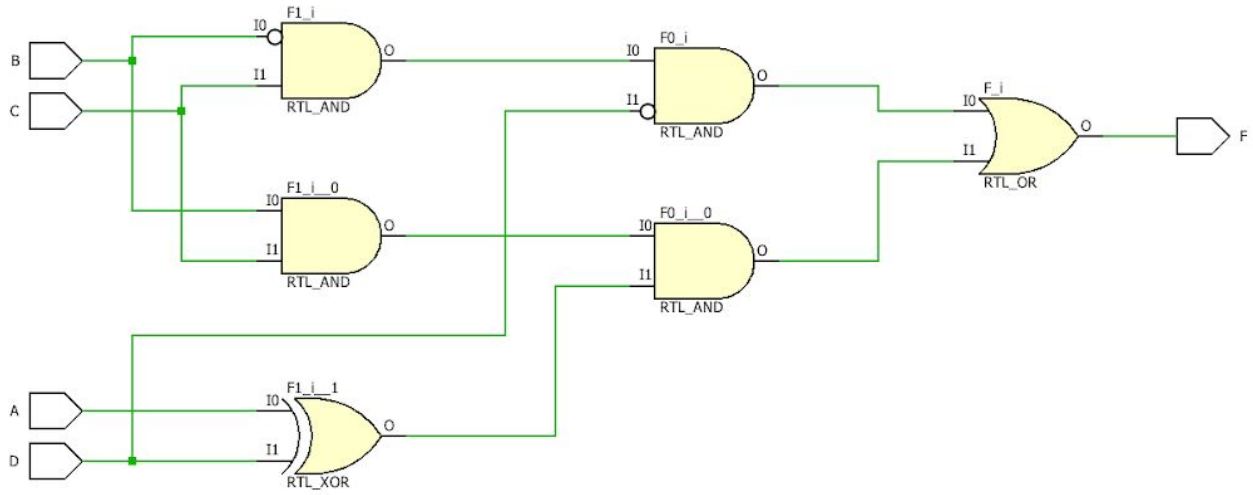
Vivado - VHDL code Implementation of F:

```
----------------------------------------------------------------------------------

library IEEE;
use IEEE.STD_LOGIC_1164.ALL; -- Main Libraries
-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx leaf cells in this code.
--library UNISIM;
--use UNISIM.VComponents.all;
----------------------------------------------------------------------------------
entity Tutorial1_1 is --Description of I/O
 Port ( A, B, C, D: in std_logic; -- Set Inputs
        F: out std_logic);-- Set Output
end Tutorial1_1;
----------------------------------------------------------------------------------
architecture Behavioral of Tutorial1_1 is
begin
```

F <= ((not B and C and not D) or ((B and C) and (A xor D))); -- Function of Input to output for F

end Behavioral;

Vivado - VHDL Schematic of F:

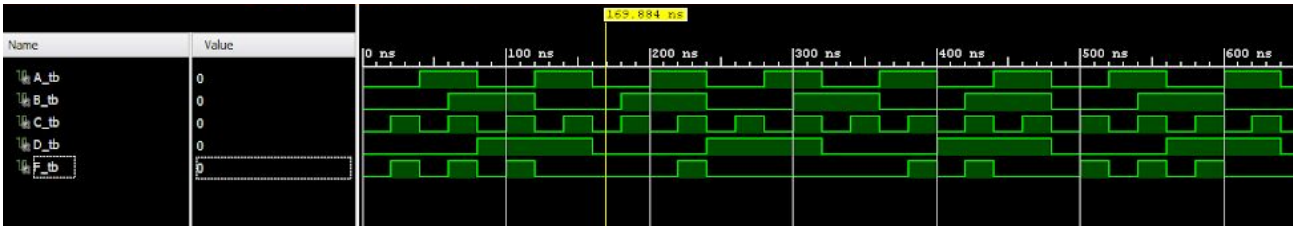


Vivado - VHDL code Test Bench for F (Added on 10.02.20):

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

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx leaf cells in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity Tutorial1_1tb is
end Tutorial1_1tb;

architecture testbench of Tutorial1_1tb is
------ DUT Declaration:------
    component Tutorial1_1 is
        Port ( A, B, C, D: in std_logic; -- Set Inputs
          F: out std_logic);-- Set Output
```

```
    end component;
    ------Signal Declaration:------
    signal A_tb: std_logic := '0';
    signal B_tb: std_logic := '0';
    signal C_tb: std_logic := '0';
    signal D_tb: std_logic := '0';
    signal F_tb : std_logic;
begin
    ------ DUT Instantiation:------
        dut: Tutorial1_1 Port map (A => A_tb, B => B_tb, C => C_tb, D => D_tb, F => F_tb);
    ------ Stimuli Generation:------
        A_tb <= NOT A_tb AFTER 40ns;
        B_tb <= NOT B_tb AFTER 60ns;
        C_tb <= NOT C_tb AFTER 20ns;
        D_tb <= NOT D_tb AFTER 80ns;

end testbench;
```

Vivado - Simulation output of F (Added on 10.02.20):



Discussion: On the first attempt errors were made in the circuit simplification which caused an unwanted state. At the first testbench attempt, the signal were not initialised at low so the output would not return any values

2. Half and Full Adder.

A. Half adder.

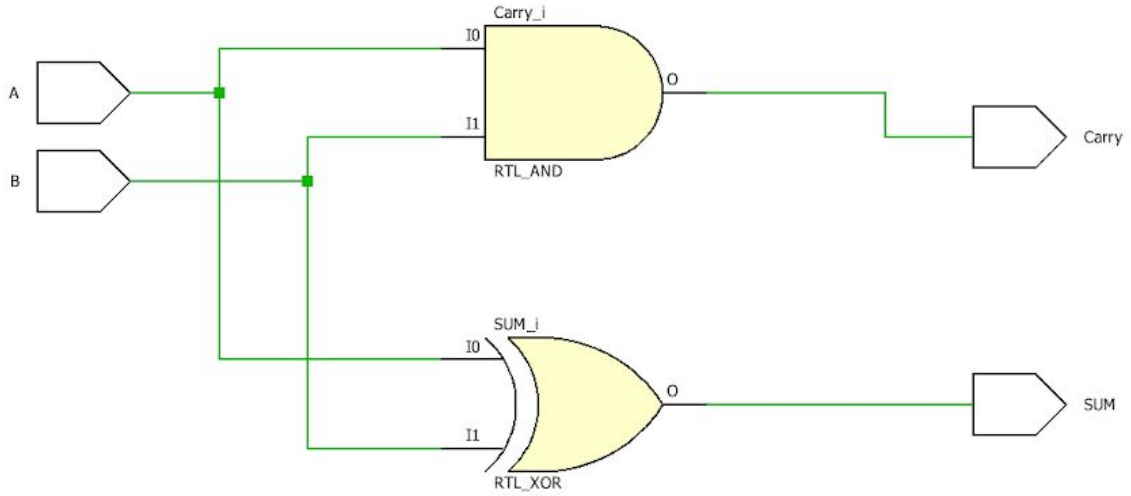
Truth table readings:

From the truth table we can deduce there are 2 inputs (A and B) and 2 outputs (SUM and Carry). From the table we can observe the system has therefore two gates: a XOR for SUM and a AND for Carry.

VHDL Code - Half Adder:

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

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx leaf cells in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity Tutorial1_2 is -- I/O Port setting
    Port ( A : in STD_LOGIC; -- A set as input
           B : in STD_LOGIC; -- B set as input
           SUM : out STD_LOGIC; -- SUM set as output
           Carry : out STD_LOGIC); -- Carry set as output
end Tutorial1_2;

architecture Behavioral of Tutorial1_2 is

begin -- Function behaviour description

    SUM <= A xor B; -- SUM output set to A exclusive or B
    Carry <= A and B; -- Carry output set to A and B

end Behavioral;

----------------------------------------------------------------------------------------------------
```

Vivado - VHDL Schematic of Half Adder:

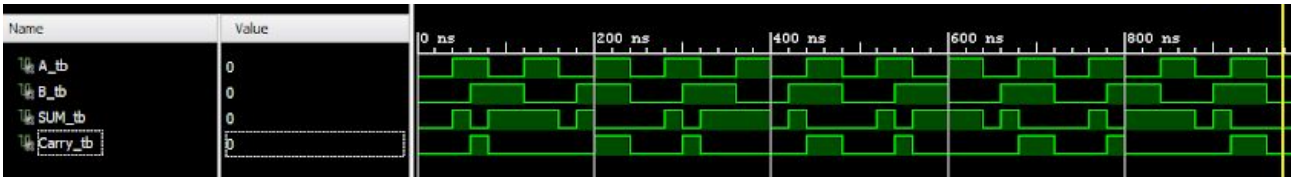


Vivado - VHDL code Test Bench for Half Adder (Added on 15.02.20):

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

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx leaf cells in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity Tutorial1_2tb is
end Tutorial1_2tb;

architecture testbench of Tutorial1_2tb is
------ DUT Declaration:------
    component Tutorial1_2 is
        Port ( A, B : in std_logic; -- Set Inputs
          SUM, Carry : out std_logic);-- Set Output
    end component;
    ------Signal Declaration:------
    signal A_tb: std_logic := '0';
    signal B_tb: std_logic := '0';
    signal SUM_tb : std_logic;
```

```
    signal Carry_tb : std_logic;
begin
    ------ DUT Instantiation:------
        dut: Tutorial1_2 Port map (A => A_tb, B => B_tb, Carry => Carry_tb, SUM => SUM_tb);
    ------ Stimuli Generation:------
        A_tb <= NOT A_tb AFTER 40ns;
        B_tb <= NOT B_tb AFTER 60ns;

end testbench;
```

Vivado - Simulation output of Half Adder (Added on 15.02.20):



Discussion: The truth table being given it was fairly simple to identify the required gates to implement. The test bench was functional from the first attempt and returned appropriate outputs.

B. Full Adder.

Karnaugh Map for SUM output:

AB / **Cin**	00	01	11	10
0	0	1	0	1
1	1	0	1	0

SUM = A XOR B XOR Cin

Karnaugh Map for Cout:

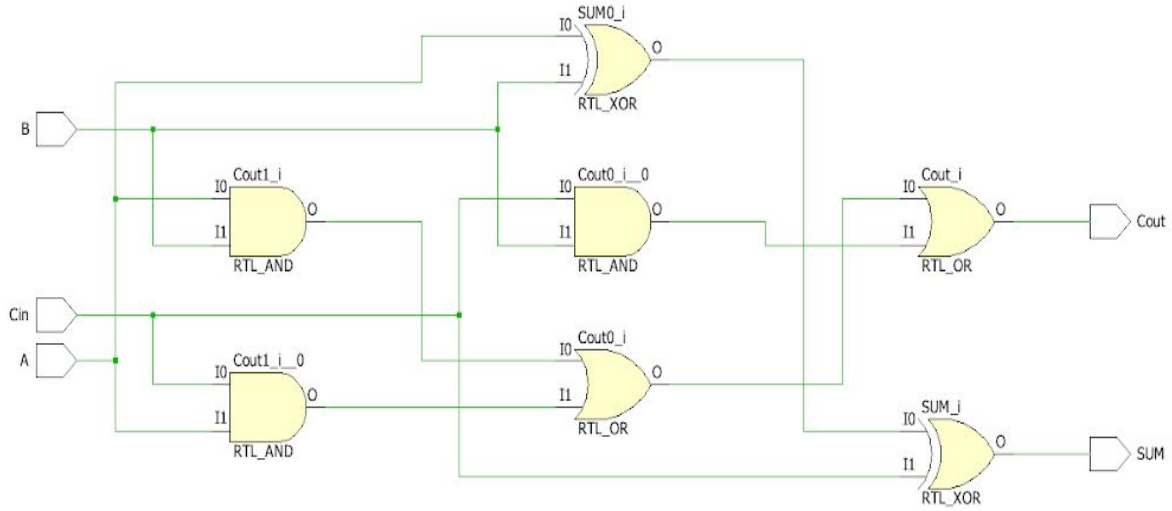
AB **Cin**	00	01	11	10
0	0	0	1	0
1	0	1	1	1

Cout = (A AND B) OR (Cin AND A) OR (Cin AND B)

VHDL Code - Full Adder:

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

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx leaf cells in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity Tutorial1_2b is -- I/O Settings
    Port ( A : in STD_LOGIC; -- Set A as input
           B : in STD_LOGIC; -- Set B as input
           Cin : in STD_LOGIC; -- Set Cin as input
           SUM : out STD_LOGIC; -- Set SUM as output
           Cout : out STD_LOGIC); -- Set Cout as output
end Tutorial1_2b;

architecture Behavioral of Tutorial1_2b is

begin -- Outputs behaviour
    SUM <=  A XOR B XOR Cin; -- Behaviour for output SUM
    Cout <= (A AND B) OR (Cin AND A) OR (Cin AND B) ; -- Behaviour for output Cout

end Behavioral;
----------------------------------------------------------------------------------------------------
```

Vivado - VHDL Schematic of Full Adder:

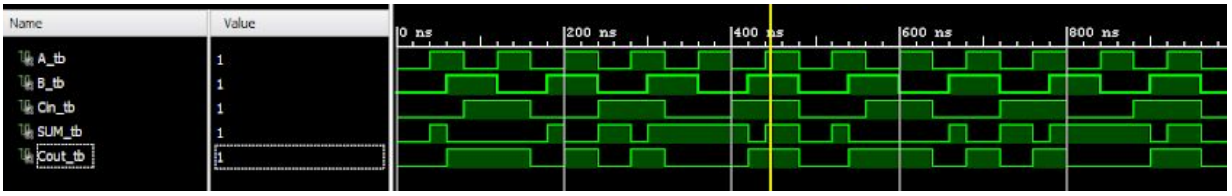


Vivado - VHDL code Test Bench for Full Adder (Added on 15.02.20):

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

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx leaf cells in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity Tutorial1_2btb is
end Tutorial1_2btb;

architecture testbench of Tutorial1_2btb is
------ DUT Declaration:------
    component Tutorial1_2b is
        Port ( A, B, Cin : in std_logic; -- Set Inputs
          SUM, Cout : out std_logic);-- Set Output
    end component;
    ------Signal Declaration:------
    signal A_tb: std_logic := '0';
    signal B_tb: std_logic := '0';
    signal Cin_tb: std_logic := '0';
    signal SUM_tb : std_logic;
```

```
    signal Cout_tb : std_logic;
begin
    ------ DUT Instantiation:------
        dut: Tutorial1_2b Port map (A => A_tb, B => B_tb, Cin => Cin_tb,Cout => Cout_tb,
SUM => SUM_tb);
    ------ Stimuli Generation:------
        A_tb <= NOT A_tb AFTER 40ns;
        B_tb <= NOT B_tb AFTER 60ns;
        Cin_tb <= NOT Cin_tb AFTER 80ns;

end testbench;
```

Vivado - Simulation output of Full Adder (Added on 15.02.20):



Discussion: Using Karnaugh maps it was possible to simplify the SUM and Cout outputs. No issues with setting up the testbench.

3. Multiplexor 4 to 1.

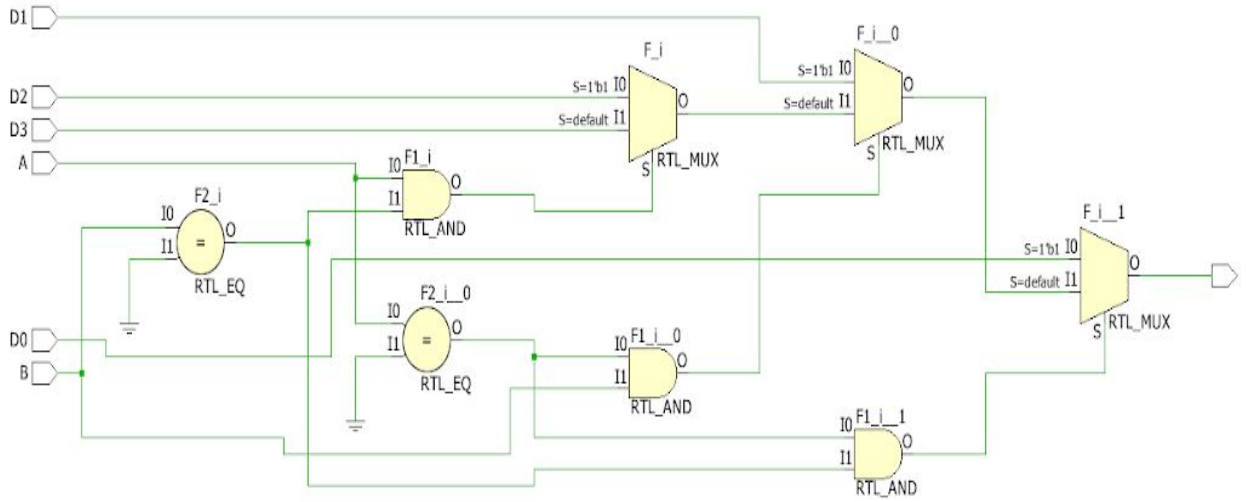
Truth table:

A	B	Output
0	0	D0
0	1	D1
1	0	D2
1	1	D3

Multiplexor 4 to 1 VHDL Code:

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

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx leaf cells in this code.
--library UNISIM;
--use UNISIM.VComponents.all;
----------------------------------------------------------------------------------
entity mux4to1 is
port( D0,D1,D2,D3 : std_logic_vector(3 downto 0);
      A,B: in std_logic;
      F: out std_logic_vector(3 downto 0));
end mux4to1;

Architecture behaviour of mux4to1 is
Begin

   F<= D0 when (A='0' and B='0') else
      D1 when (A='0' and B='1') else
      D2 when (A='1' and B='0') else
      D3;

end behaviour;
----------------------------------------------------------------------------------
```

Vivado - VHDL Schematic of Multiplexer:

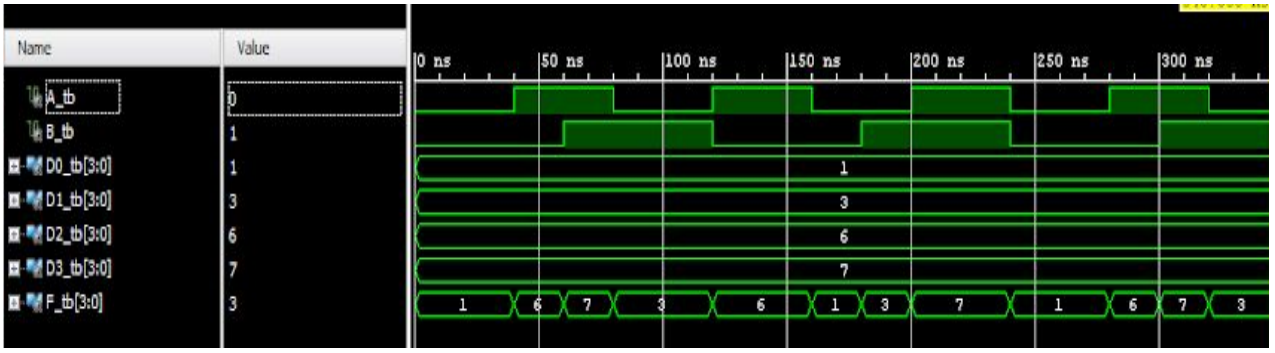


Vivado - VHDL code Test Bench for Multiplexor 4 to 1 (Added on 15.02.20):

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

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx leaf cells in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity mux4to1tb is
end mux4to1tb;

architecture testbench of mux4to1tb is
------ DUT Declaration:------
    component mux4to1 is
        Port ( A, B: in std_logic; -- Set Inputs
           D0, D1, D2, D3 : in std_logic_vector(3 downto 0);
          F : out std_logic_vector(3 downto 0));-- Set Output
    end component;
    ------Signal Declaration:------
    signal A_tb: std_logic := '0';
```

```
    signal B_tb: std_logic := '0';
    signal D0_tb: std_logic_vector(3 downto 0) := "0001";
    signal D1_tb : std_logic_vector(3 downto 0):= "0011";
    signal D2_tb : std_logic_vector(3 downto 0):= "0110";
    signal D3_tb : std_logic_vector(3 downto 0):= "0111";
    signal F_tb : std_logic_vector(3 downto 0);
begin
    ------ DUT Instantiation:------
        dut: mux4to1 Port map (A => A_tb, B => B_tb, D0 => D0_tb, D1 => D1_tb, D2 =>
D2_tb, D3 => D3_tb, F=>F_tb);
    ------ Stimuli Generation:------
        A_tb <= NOT A_tb AFTER 40ns;
        B_tb <= NOT B_tb AFTER 60ns;

end testbench;
-------------------------------------------------------------------------------------------------
```

Vivado - Simulation output of Multiplexor 4 to 1 (Added on 15.02.20):



Discussion: Multiplexor was implemented using when-else statements. Values were set in the testbench for each D-output and displayed in the simulation to identify the sequence easily.

4. Encoder.

Logic of the design:

The system feeds in 8 input combinations and returns 3 outputs combinations.

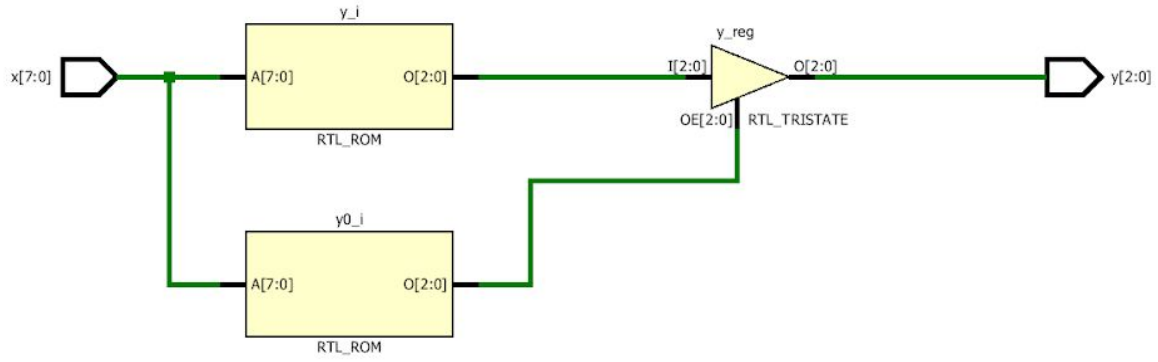
Input x0 to x7	Output y0 to y2
"00000001"	"000"
"00000010"	"001"
"00000100"	"010"
"00001000"	"011"
"00010000"	"100"
"00100000"	"101"
"01000000"	"110"
"10000000"	“111”
others	“zzz” all other undesirable cases

Encoder 8 to 3 VHDL Code:

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

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx leaf cells in this code.
--library UNISIM;
--use UNISIM.VComponents.all;
----------------------------------------------------------------------------------
entity Tutorial1_4 is
    Port ( x : in STD_LOGIC_VECTOR (7 DOWNTO 0); -- 8 inputs x
           y : out STD_LOGIC_VECTOR (2 DOWNTO 0)); -- 3 outputs y
end Tutorial1_4;
----------------------------------------------------------------------------------
```

```
architecture Encoder of Tutorial1_4 is

begin
   with x select
      y <= "000" when "00000001", -- select output y for input x
           "001" when "00000010",
           "010" when "00000100",
           "011" when "00001000",
           "100" when "00010000",
           "101" when "00100000",
           "110" when "01000000",
           "111" when "10000000",
           "ZZZ" when others;


end Encoder;
```

Vivado - VHDL Schematic of Encoder:

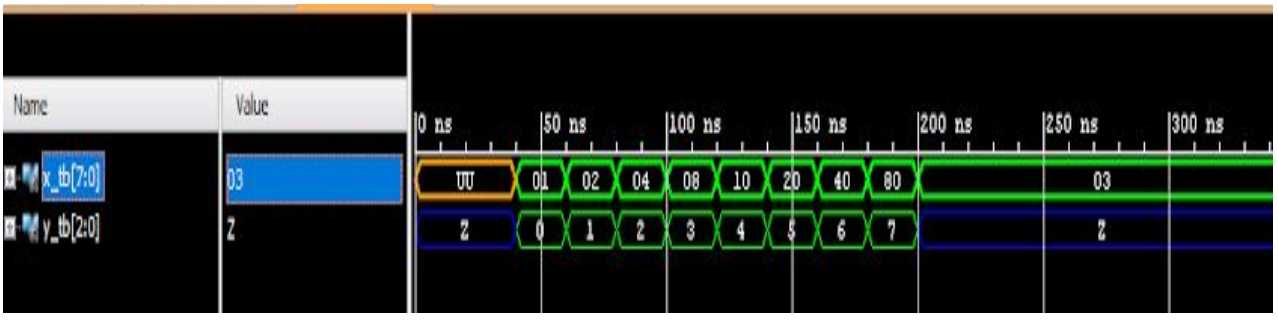


Vivado - VHDL code Test Bench for Encoder (Added on 15.02.20):

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

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx leaf cells in this code.
--library UNISIM;
--use UNISIM.VComponents.all;
```

```
entity Tutorial1_4tb is
end Tutorial1_4tb;

architecture testbench of Tutorial1_4tb is
------ DUT Declaration:------
    component Tutorial1_4 is
        Port ( x : in STD_LOGIC_VECTOR (7 DOWNTO 0):="00000000"; -- Set Input
               y : out STD_LOGIC_VECTOR (2 DOWNTO 0));-- Set Output
    end component;
    ------Signal Declaration:------
    signal x_tb: STD_LOGIC_VECTOR (7 DOWNTO 0); --Input signal
    signal y_tb: STD_LOGIC_VECTOR (2 DOWNTO 0); --Output signal

begin
    ------ DUT Instantiation:------
        dut: Tutorial1_4 Port map (x => x_tb, y => y_tb);
    ------ Stimuli Generation:------
        x_tb <= "00000001" AFTER 40ns, "00000010" AFTER 60ns, "00000100" AFTER 80ns,
"00001000" AFTER 100ns, "00010000" AFTER 120ns, "00100000" AFTER 140ns,
"01000000" AFTER 160ns, "10000000" AFTER 180ns, "00000011" AFTER 200ns;

end testbench;
```

Vivado - Simulation output of Encoder (Added on 15.02.20):



Discussion: The encoder was implemented using “with select when” statements. The encoder takes in 8 bit sequences and returns 3 bit sequences (displayed as numerical values in the simulation).

Task for week 2: Tutorial 2 Sequential Design (Flip Flops and Registers)

Task Date: 27.01.20

Objectives: Create and test Flip-flops in VHDL

Use Flip-flops to create registers.

1. D Type Flip-flop.

D-type flip flop truth table:

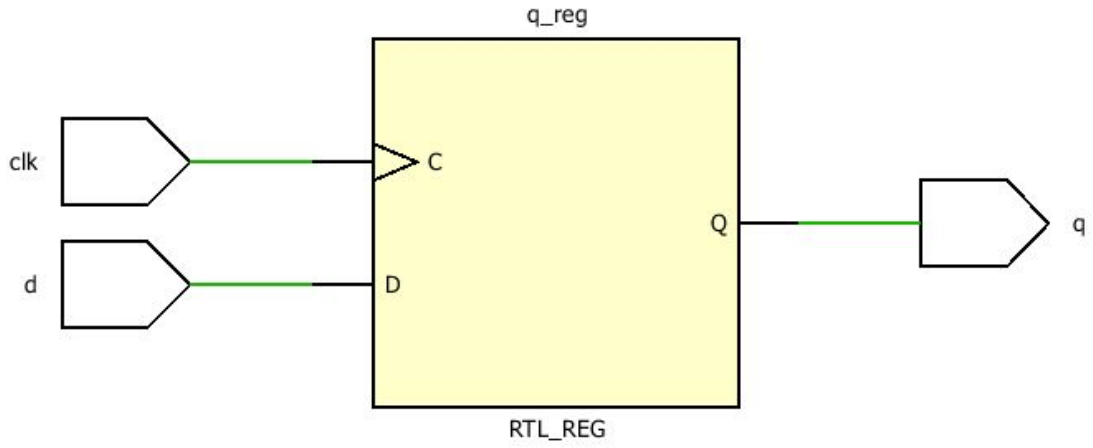
clock	Input	Output	
	D	Q	Q’
Low	x	0	1
High	0	0	1
High	1	1	0

D-Type Flip flop - VHDL Code:

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

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx leaf cells in this code.
--library UNISIM;
--use UNISIM.VComponents.all;
----------------------------------------------------------------------------------------------------
entity D_FF is -- I/O Settings description
    Port ( clk : in STD_LOGIC; -- clock set as input
           d : in STD_LOGIC; -- clock set as input
           q : out STD_LOGIC); -- q set as output
end D_FF;
----------------------------------------------------------------------------------------------------
```

```
architecture Behavioral of D_FF is -- description of D-Flip Flop
begin
    process(clk) -- run process for every rising edge of the clock.
    begin
        if (clk' event and clk='1') Then -- change output if clk rises and clk is 1
            q <= d; -- output q changed to d
        end if;
    end process;

end Behavioral;
```

Vivado - VHDL Schematic of D-type Flip Flop:



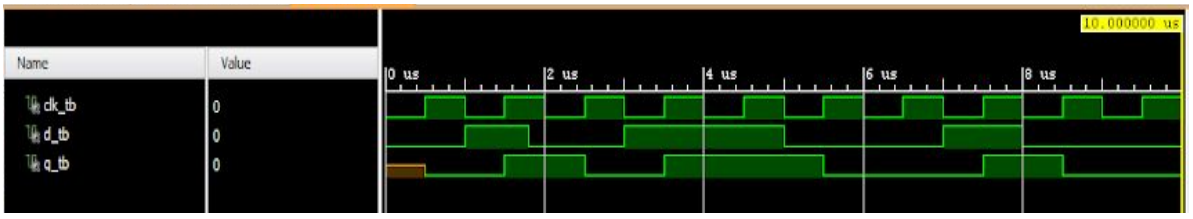
B. Vivado - VHDL code Test Bench for D Flip Flop (Added on 16.02.20):

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

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx leaf cells in this code.
--library UNISIM;
--use UNISIM.VComponents.all;
```

```
entity Tutorial2_1atb is
end Tutorial2_1atb;
```

```
architecture testbench of Tutorial2_1atb is
------ DUT Declaration:------
    component D_FF is
         Port ( clk, d : in std_logic; -- Set Inputs
           q : out std_logic);-- Set Output
    end component;
    ------Signal Declaration:------
    signal clk_tb: std_logic := '0';
    signal d_tb: std_logic := '0';
    signal q_tb : std_logic;

begin
    ------ DUT Instantiation:------
        dut: D_FF Port map (clk => clk_tb, d => d_tb, q => q_tb);
    ------ Stimuli Generation:------
        clk_tb <= NOT clk_tb AFTER 0.5us;
        d_tb <= '1' AFTER 1us, '0' AFTER 1.8us, '1' AFTER 3us, '0' AFTER 5us, '1' AFTER
7us, '0' AFTER 8us;


end testbench;
```

Vivado - Simulation output of D Flip Flop (Added on 16.02.20):



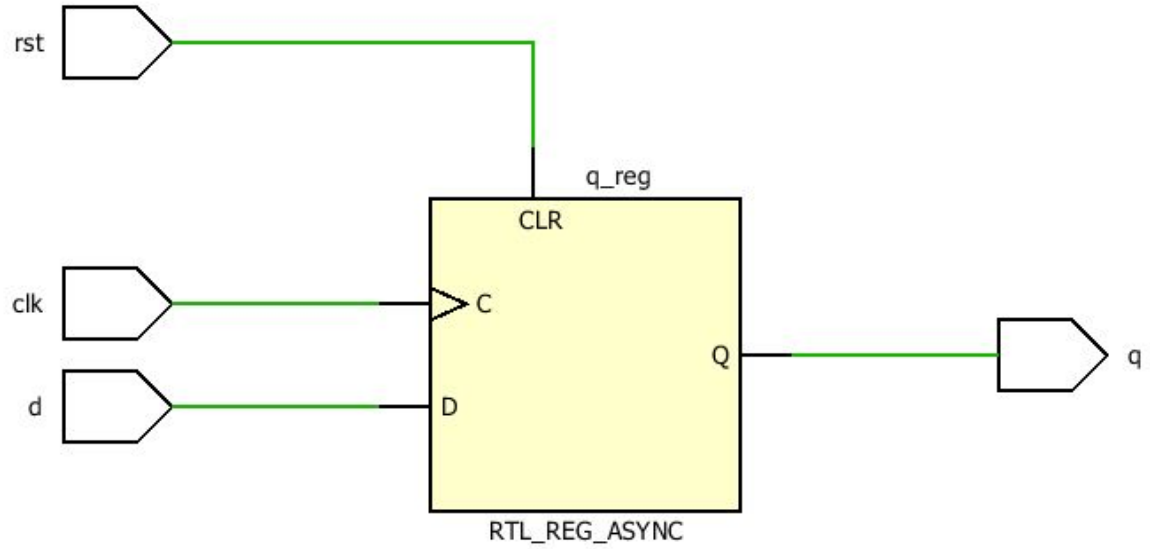
Discussion: As shown in the output time diagram, the D flip flop performs as indicated in the truth table. For each rising edge of the clock the D flip flop returns the input value as an output.

C. Asynchronous reset D type flip flop.

Asynchronous D-Type Flip flop - VHDL Code:

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

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx leaf cells in this code.
--library UNISIM;
--use UNISIM.VComponents.all;
----------------------------------------------------------------------------------------------------
entity D_FFa is -- Description of I/0
    Port ( clk : in STD_LOGIC; -- clock set as an input.
           d : in STD_LOGIC; -- d set as an input
           rst : in STD_LOGIC; -- reset as input
           q : out STD_LOGIC); -- q  as an input
end D_FFa;
----------------------------------------------------------------------------------------------------
architecture Behavioral of D_FFa is -- description of D-Flip Flop
begin
    process(clk, rst) -- run process for every rising edge of the clock or reset.
    begin
        if rst= '1' then -- if reset is enabled.
            q <= '0'; -- set output to 0.
        elsif (clk' event and clk='1') then -- change output if clk rises and clk is 1.
            q <= d; -- output q changed to d.
        end if;
    end process;

end Behavioral;
----------------------------------------------------------------------------------------------------
```

Vivado - VHDL Schematic of D-type Flip Flop:

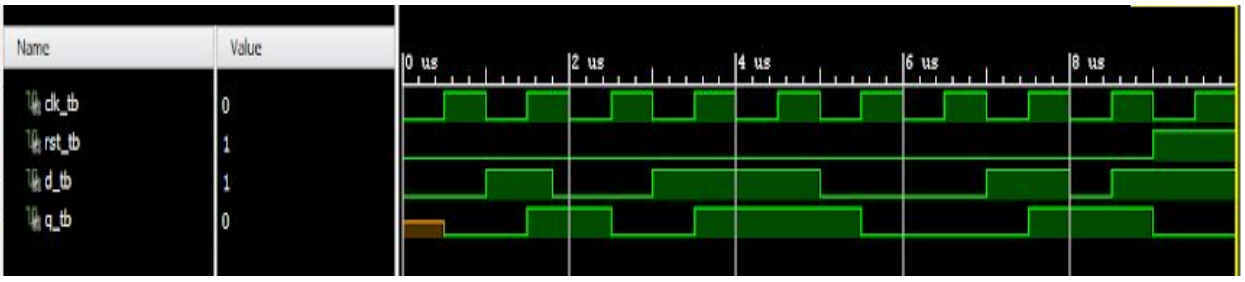


D. Vivado - VHDL code Test Bench for D Flip Flop asynchronous reset (Added on 16.02.20):

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

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx leaf cells in this code.
--library UNISIM;
--use UNISIM.VComponents.all;
----------------------------------------------------------------------------------
entity Tutorial2_1ctb is
end Tutorial2_1ctb;

architecture testbench of Tutorial2_1ctb is
------ DUT Declaration:------
    component D_FFa is
         Port (rst,clk, d : in std_logic; -- Set Inputs
           q : out std_logic);-- Set Output
    end component;
    ------Signal Declaration:------
    signal clk_tb: std_logic := '0';
     signal rst_tb: std_logic := '0';
    signal d_tb: std_logic := '0';
```

```
    signal q_tb : std_logic;

begin
    ------ DUT Instantiation:------
        dut: D_FFa Port map (clk => clk_tb, d => d_tb, q => q_tb, rst => rst_tb);
    ------ Stimuli Generation:------
        clk_tb <= NOT clk_tb AFTER 0.5us;
        rst_tb <= NOT rst_tb AFTER 9us;
        d_tb <= '1' AFTER 1us, '0' AFTER 1.8us, '1' AFTER 3us, '0' AFTER 5us, '1' AFTER
7us, '0' AFTER 8us, '1' AFTER 8.5us;


end testbench;
-------------------------------------------------------------------------------------------------
```

Vivado - Simulation output of D Flip Flop asynchronous reset (Added on 16.02.20):



Discussion: This circuit integrates the reset function. When the reset is triggered the output is reset to 0 independently from the clock cycle.

2. Combinational Logic to Sequential using D Flip Flop.

Logic of the circuit:

buzzer <= (not door and ignition) or (not sbelt and ignition)
buzzer <= (not door or not sbelt) and ignition.

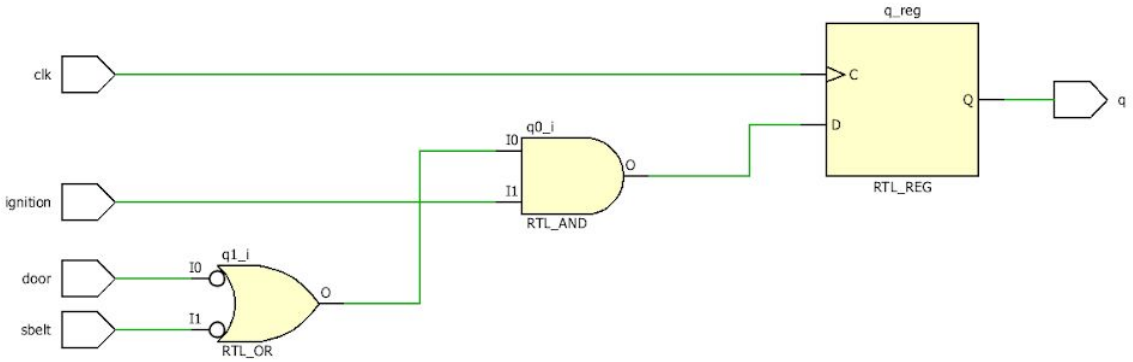
Buzzer will be the d input of the flip flop which will pass its state to the output at a clock turn and clock 1.

Circuit - VHDL Code:

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

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx leaf cells in this code.
--library UNISIM;
--use UNISIM.VComponents.all;
----------------------------------------------------------------------------------------------------
entity DFF_comb2seq is
    Port ( clk : in STD_LOGIC; -- clock set as an input.
           door : in STD_LOGIC; -- door set as an input.
           ignition : in STD_LOGIC;-- ignition set as an input.
           sbelt : in STD_LOGIC;-- sbelt set as an input.
           q : out STD_LOGIC); -- q set as output

end DFF_comb2seq;
----------------------------------------------------------------------------------------------------

architecture Behavioral of DFF_comb2seq is
signal d: STD_LOGIC;
begin
    process(clk) -- run process for every rising edge of the clock.
    begin
        if (clk' event and clk='1') Then -- change output if clk rises and clk is 1
            q <= d; -- output of D flipflop updated to output through logic.
        end if;
    end process;
```

```
  d <= (not door or not sbelt) and ignition;
end Behavioral;
----------------------------------------------------------------------------------------------------
```

Vivado - VHDL Schematic of Circuit:

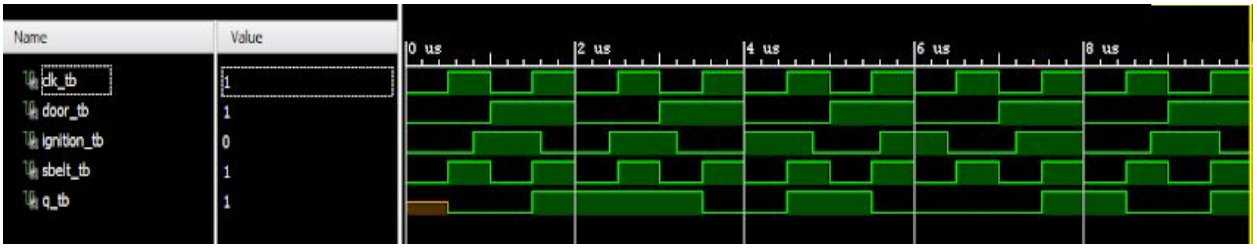


Vivado - VHDL code Test Bench for D Flip Flop output control (Added on 16.02.20):

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

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx leaf cells in this code.
--library UNISIM;
--use UNISIM.VComponents.all;
----------------------------------------------------------------------------------------------------
entity Tutorial2_2tb is
end Tutorial2_2tb;

architecture testbench of Tutorial2_2tb is
------ DUT Declaration:------
    component DFF_comb2seq is
        Port ( clk, door, ignition, sbelt : in std_logic; -- Set Inputs
          q : out std_logic);-- Set Output
    end component;
    ------Signal Declaration:------
    signal clk_tb: std_logic := '0';
    signal door_tb: std_logic := '0';
    signal ignition_tb: std_logic := '0';
```

```
    signal sbelt_tb: std_logic := '0';
    signal q_tb : std_logic;

begin
    ------ DUT Instantiation:------
        dut: DFF_comb2seq Port map (clk => clk_tb, door => door_tb, q => q_tb, ignition =>
ignition_tb, sbelt => sbelt_tb);
    ------ Stimuli Generation:------
        clk_tb <= NOT clk_tb AFTER 0.5us;
        door_tb <= NOT door_tb AFTER 1us, NOT door_tb AFTER 1.8us, NOT door_tb AFTER
3us, NOT door_tb AFTER 5us, NOT door_tb AFTER 7us, NOT door_tb AFTER 8us;
        ignition_tb <= NOT ignition_tb AFTER 0.8us, NOT ignition_tb AFTER 1.3us, NOT
ignition_tb AFTER 2.5us, NOT ignition_tb AFTER 4.5us, NOT ignition_tb AFTER 6.5us, NOT
ignition_tb AFTER 8.5us;
        sbelt_tb <= NOT sbelt_tb AFTER 0.5us, NOT sbelt_tb AFTER 1.5us, NOT sbelt_tb
AFTER 2.5us, NOT sbelt_tb AFTER 3.8us, NOT sbelt_tb AFTER 6us, NOT sbelt_tb AFTER
7us, NOT sbelt_tb AFTER 9.5us;
end testbench;
-------------------------------------------------------------------------------------------------
```

Vivado - Simulation output of D Flip Flop output control (Added on 16.02.20):



Discussion: The circuit builds on the example of the first tutorial. The output will only be changed at the rise of the clock giving time for the outputs to settle and avoid in between states.

3. T type Flip flop.

T-type flip flop truth table:

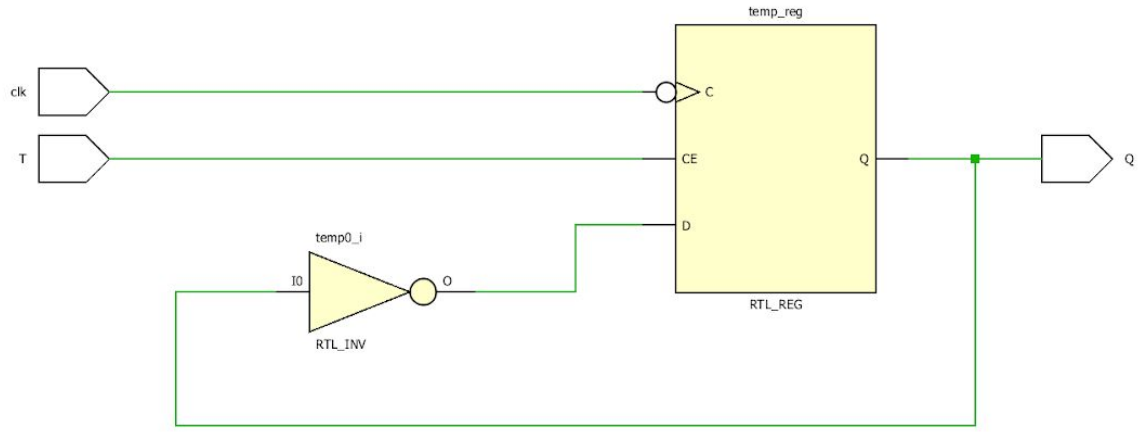
T	Q	Qpos
0	0	0
1	0	1
0	1	1
1	1	0

T-Type Flip flop - VHDL Code:

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

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx leaf cells in this code.
--library UNISIM;
--use UNISIM.VComponents.all;
----------------------------------------------------------------------------------------------------
entity T_FF is
  port(
    clk, T : in  std_logic;
    Q      : out std_logic);
end entity T_FF;
----------------------------------------------------------------------------------------------------
architecture behaviour of T_FF is
signal temp: STD_LOGIC :='0';
begin
    process(clk) is
    begin
        if falling_edge(clk) then
```

```
        if T = '1' then
        temp <= not (temp);
        elsif T ='0' then
        temp <= temp;
        end if;
      end if;
    Q <= temp;
  end process;
end behaviour;
```

Vivado - VHDL Schematic of T Flip flop:



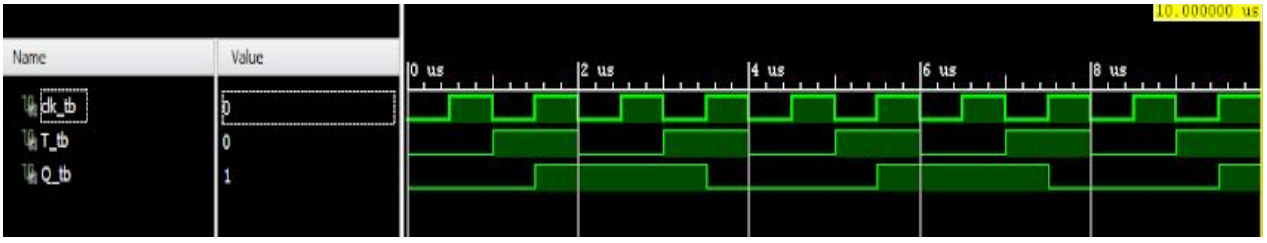
Vivado - VHDL code Test Bench for T Flip flop (Added on 16.02.20):

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

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx leaf cells in this code.
--library UNISIM;
--use UNISIM.VComponents.all;
```

```
entity Tutorial2_3tb is

end Tutorial2_3tb;
```

```
architecture testbench of Tutorial2_3tb is
------ DUT Declaration:------
    component T_FF is
         Port ( clk, T : in std_logic; -- Set Inputs
           Q : out std_logic);-- Set Output
    end component;
    ------Signal Declaration:------
    signal clk_tb: std_logic := '0';
    signal T_tb: std_logic := '0';
    signal Q_tb : std_logic;

begin
    ------ DUT Instantiation:------
        dut: T_FF Port map (clk => clk_tb, T => T_tb, Q => Q_tb);
    ------ Stimuli Generation:------
        clk_tb <= NOT clk_tb AFTER 0.5us;
        T_tb <= NOT T_tb AFTER 1us, NOT T_tb AFTER 1.8us, NOT T_tb AFTER 3us, NOT
T_tb AFTER 5us, NOT T_tb AFTER 7us, NOT T_tb AFTER 8us;


end testbench;
-----------------------------------------------------------------------------------------------
```

Vivado - Simulation output of T Flip flop (Added on 16.02.20):



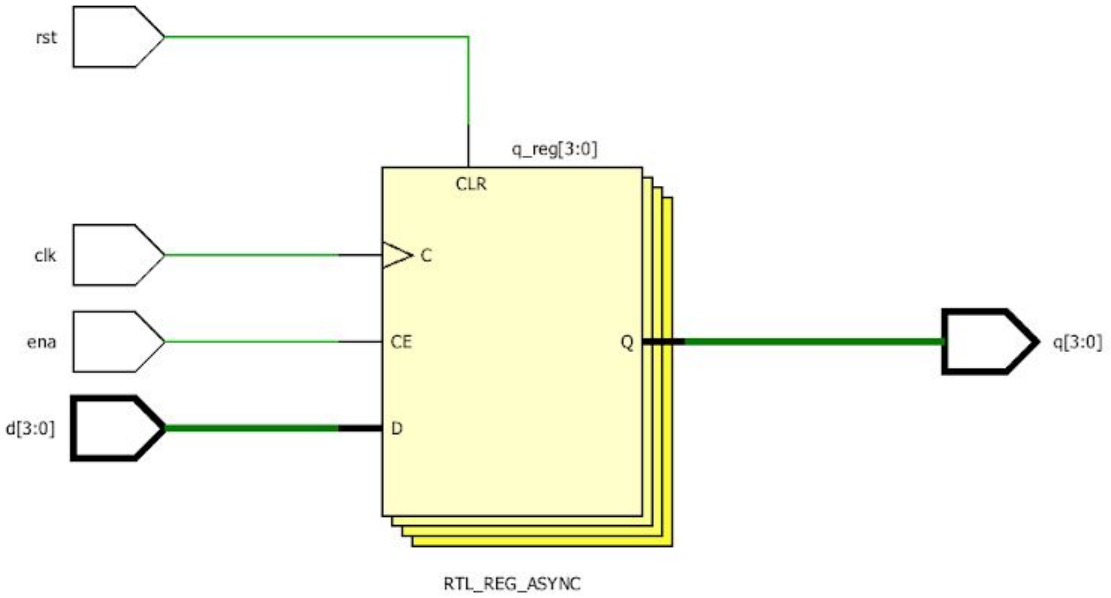
Discussion: In this example, a temporary signal was created to hold the current state. This allows the toggle effect of the T flip flop. Moreover this example is controlled by the falling edge of the clock.

4. 4-bit register with D Flipflop (asynchronous clear and synchronous enable).

D-Type Flip flop 4 bit register - VHDL Code:

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

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx leaf cells in this code.
--library UNISIM;
--use UNISIM.VComponents.all;
----------------------------------------------------------------------------------
entity D_FF_4bit_reg is -- I/O Settings description
    Port ( clk : in STD_LOGIC; -- clock set as input
           d : in STD_LOGIC_VECTOR (3 DOWNTO 0); -- d set as input
           ena : in STD_LOGIC; -- ena as an input
           rst : in STD_LOGIC; -- rst as an input
           q : out STD_LOGIC_VECTOR (3 DOWNTO 0)); -- q set as output
end D_FF_4bit_reg;
----------------------------------------------------------------------------------
architecture Behavioral of D_FF_4bit_reg is -- description of D-Flip Flop
begin
    process(clk) -- run process for every rising edge of the clock.
    begin
        if rst = '1' then -- if reset triggered
            q <= "0000"; --set outputs to 0
        elsif (clk' event and clk='1') Then -- if clock turns to high
            if ena = '1' then -- and enable is triggered
            q <= d; -- outputs q changed to d
            end if;
        end if;
    end process;
end Behavioral;
----------------------------------------------------------------------------------
```

Vivado - VHDL Schematic of 4 bit register:

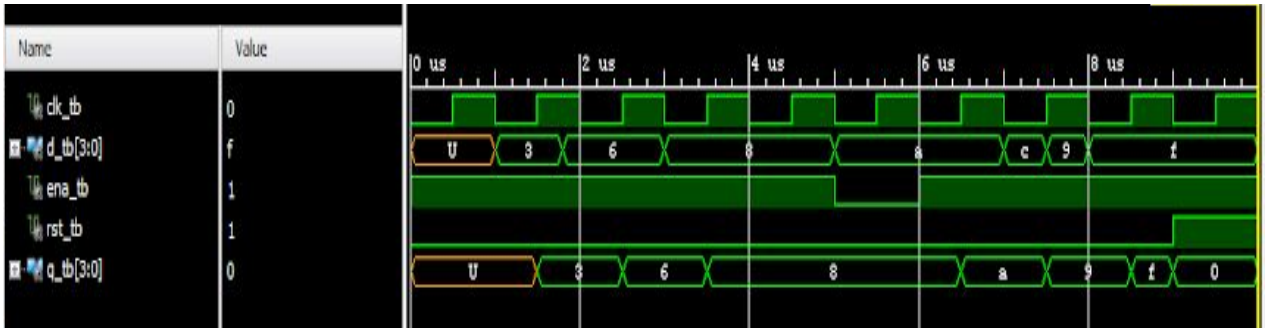


Vivado - VHDL code Test Bench for 4-bit register with D Flipflop (Added on 16.02.20):

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

entity Tutorial2_4tb is
end Tutorial2_4tb;

architecture testbench of Tutorial2_4tb is
------ DUT Declaration:------
    component D_FF_4bit_reg is
    Port ( clk : in STD_LOGIC; -- clock set as input
           d : in STD_LOGIC_VECTOR (3 DOWNTO 0); -- clock set as input
           ena : in STD_LOGIC; -- ena as an input
           rst : in STD_LOGIC; -- rst as an input
           q : out STD_LOGIC_VECTOR (3 DOWNTO 0)); -- q set as output
    end component;
    ------Signal Declaration:------
    signal clk_tb: std_logic := '0';
    signal d_tb: std_logic_vector(3 DOWNTO 0);
    signal ena_tb : std_logic := '1';
    signal rst_tb : std_logic := '0';
    signal q_tb : std_logic_vector(3 DOWNTO 0);
```

```
begin
    ------ DUT Instantiation:------
        dut: D_FF_4bit_reg Port map (clk => clk_tb, d => d_tb, q => q_tb, ena => ena_tb, rst =>
rst_tb);
    ------ Stimuli Generation:------
        clk_tb <= NOT clk_tb AFTER 0.5us;
        rst_tb <= NOT rst_tb AFTER 9us;
        ena_tb <=  NOT ena_tb AFTER 5us, ena_tb AFTER 6us;
        d_tb <= "0011" AFTER 1us, "0110" AFTER 1.8us, "1000" AFTER 3us, "1010" AFTER
5us, "1100" AFTER 7us, "1001" AFTER 7.5us, "1111" AFTER 8us;

end testbench;
```

Vivado - Simulation output of 4-bit register with D Flipflop (Added on 16.02.20):



Discussion: The register is composed of four D flip flops one for each data input line (here a vector input). The D flip flop will only pass on the values at the turn of the clock and if the output is enabled.

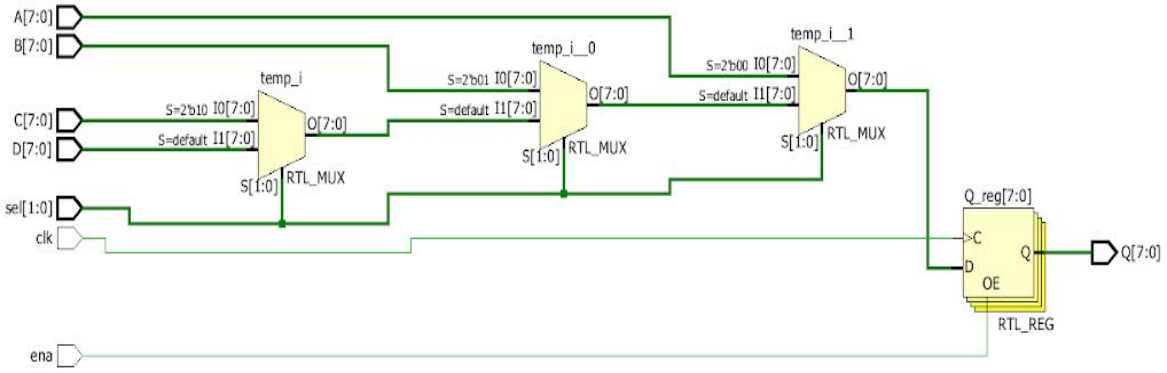
5. 8 to 8 bit Mux with 2 bit selector with flip flop output control.

Mux with flip flop controlled output - VHDL Code:

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

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx leaf cells in this code.
--library UNISIM;
--use UNISIM.VComponents.all;
----------------------------------------------------------------------------------
entity Mux8to8 is
    Port ( A, B, C, D : in STD_LOGIC_VECTOR (7 DOWNTO 0); -- 4 data inputs of 8 bit
           sel : in STD_LOGIC_VECTOR (1 DOWNTO 0); -- 2 bit
           clk : in STD_LOGIC; -- clock input
           ena : in STD_LOGIC; -- output enable input
           Q : out STD_LOGIC_VECTOR (7 DOWNTO 0)); -- system output

end Mux8to8;
----------------------------------------------------------------------------------
architecture Behavioral of Mux8to8 is
-- Shift register behaviour----------------------------------------------------
    signal temp: STD_LOGIC_VECTOR (7 DOWNTO 0);
begin
    process(clk, ena) -- enable can be activated at any time.
        begin
        if rising_edge(clk) then
            if ena = '1' then -- if enable activated
               Q <= temp; -- output will be set to temp signal used to store mux output.
            else
               Q <= "ZZZZZZZZ"; -- If not enable the output set to high impedance Z
            end if;
         end if;
    end process;
 --  8 to 8 bit Mux behaviour-------------------------------------------------
 temp <= A when sel="00" else -- A is output if selector is at 00
         B when sel="01" else -- A is output if selector is at 01
```

```
        C when sel="10" else -- C is output if selector is at 10
        D; -- D is output otherwise
end Behavioral;
```

Vivado - VHDL Schematic of Mux:

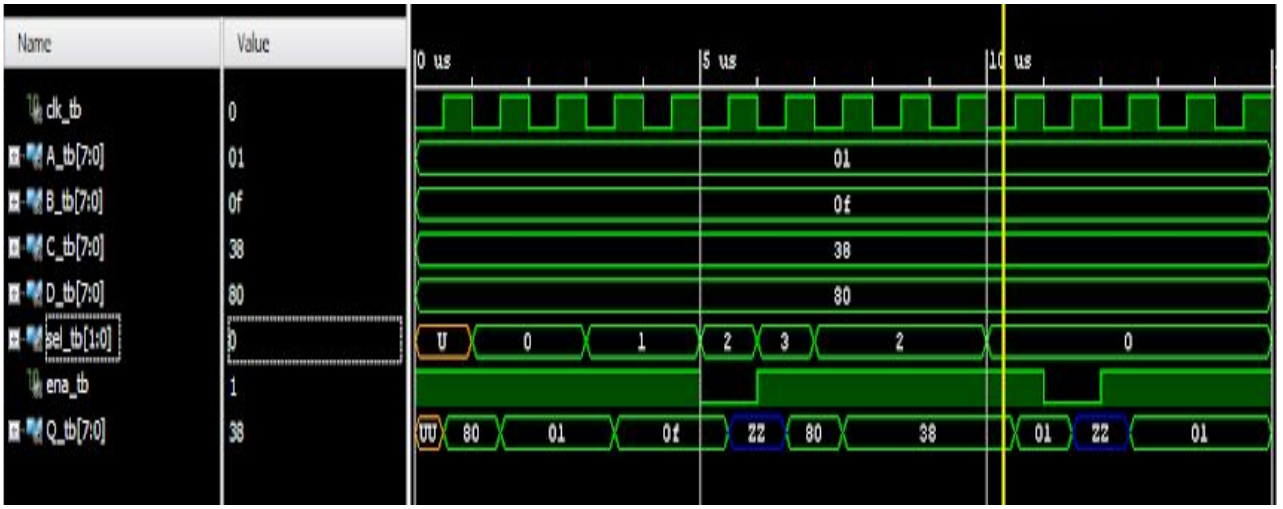


Vivado - VHDL code Test Bench for mux with 2 bit selector with flip flop output control (Added on 16.02.20):

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

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx leaf cells in this code.
--library UNISIM;
--use UNISIM.VComponents.all;
----------------------------------------------------------------------------------
entity Tutorial2_5tb is
end Tutorial2_5tb;
----------------------------------------------------------------------------------
architecture testbench of Tutorial2_5tb is
------ DUT Declaration:------
    component Mux8to8 is
        Port ( A, B, C, D : in STD_LOGIC_VECTOR (7 DOWNTO 0); -- 4 data inputs of 8 bit
          sel : in STD_LOGIC_VECTOR (1 DOWNTO 0); -- 2 bit
          clk : in STD_LOGIC; -- clock input
          ena : in STD_LOGIC; -- output enable input
          Q : out STD_LOGIC_VECTOR (7 DOWNTO 0)); -- system output
```

```
    end component;
    ------Signal Declaration:------
    signal clk_tb: std_logic := '0';
    signal A_tb: std_logic_vector(7 DOWNTO 0):= "00000001";
    signal B_tb: std_logic_vector(7 DOWNTO 0):= "00001111";
    signal C_tb: std_logic_vector(7 DOWNTO 0):= "00111000";
    signal D_tb: std_logic_vector(7 DOWNTO 0):= "10000000";
    signal sel_tb: std_logic_vector(1 DOWNTO 0);
    signal ena_tb : std_logic := '1';
    signal Q_tb : std_logic_vector(7 DOWNTO 0);
--------------------------------------------------------------------------------
begin
    ------ DUT Instantiation:------
        dut: Mux8to8 Port map (clk => clk_tb, A => A_tb, B => B_tb, C => C_tb, D => D_tb, ena
=> ena_tb, sel => sel_tb, Q => Q_tb);
    ------ Stimuli Generation:------
        clk_tb <= NOT clk_tb AFTER 0.5us;
        ena_tb <=  NOT ena_tb AFTER 5us, ena_tb AFTER 6us;
        sel_tb <= "00" AFTER 1us,
                  "01" AFTER 3us,
                  "10" AFTER 5us,
                  "11" AFTER 6us,
                  "10" AFTER 7us,
                  "00" AFTER 10us;

end testbench;
```

Vivado - Simulation output of mux with 2 bit selector with flip flop output control (Added on 16.02.20):



Discussion: The circuits perform as an 8to 8 Mux with synchronous enable-controlled output. It would be an improvement to make the output enable asynchronous. Data for each port was created in the testbench to easily recognise the sequence.

Task for week 3: Tutorial 3 Test Benches, ALU and Memory.

Task Date: 11.02.20

Objectives: Implement testbenches.

Arithmetic Logic Units.

ROM and RAM.

1. Test benches updated in related tutorials and questions.

2. 8-bit Arithmetic Logic Unit.

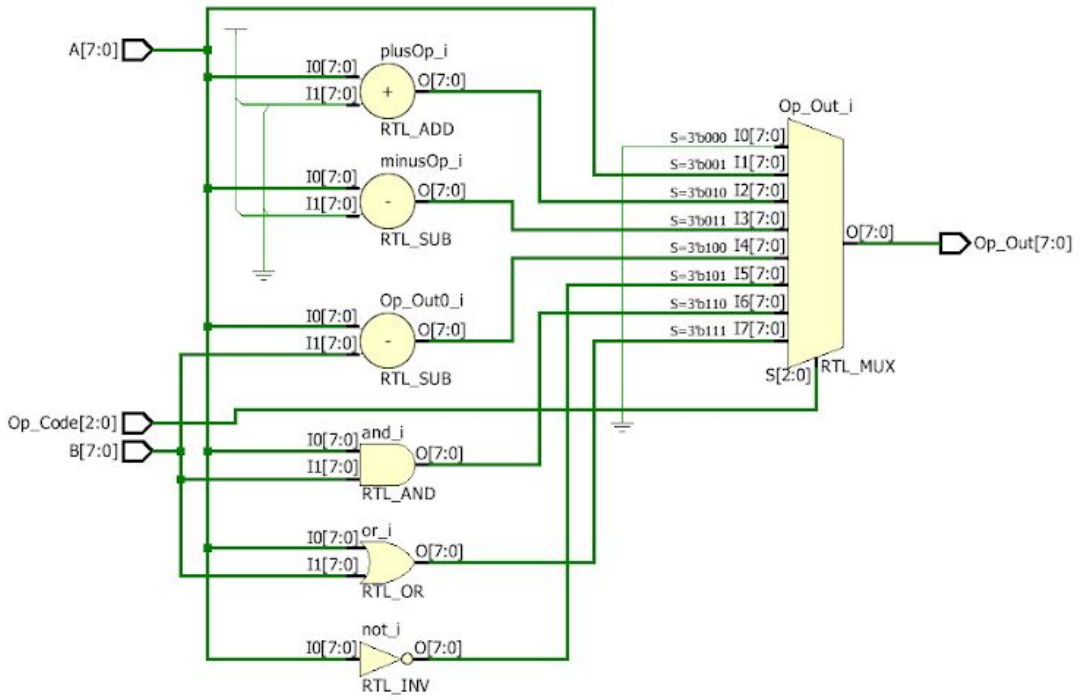
Circuit Logic:

Data In	Input Op_Code	Op_Out
-	"000"	reset
A	"001"	A
A	"010"	A+1
A	"011"	A-1
A, B	"100"	A-B
A	"101"	Not A
A, B	"110"	A and B
A,B	"111"	A or B
-	others	“zzz” all other undesirable cases

8-bit Arithmetic Logic Unit - VHDL Code:

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

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx leaf cells in this code.
--library UNISIM;
--use UNISIM.VComponents.all;
----------------------------------------------------------------------------------
entity ALU is
  Port (A : in signed(7 downto 0);
        B : in signed(7 downto 0);
        Op_Code : in STD_LOGIC_VECTOR (2 downto 0);
        Op_Out : out signed(7 downto 0));

end ALU;
----------------------------------------------------------------------------------
architecture Behavioral of ALU is

begin
    with Op_Code select
    Op_Out <= "00000000" when "000",
              A when "001",
              (A + 1) when "010",
              (A - 1) when "011",
              (A - B) when "100",
              NOT A when "101",
              (A AND B) when "110",
              (A OR B) when "111",
              "ZZZZZZZZ" when others;
end Behavioral;
----------------------------------------------------------------------------------
```

Vivado - VHDL Schematic of 8-bit Arithmetic Logic Unit :

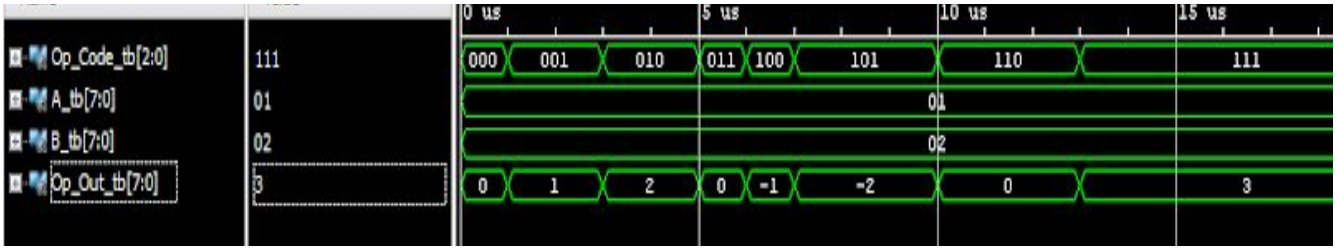


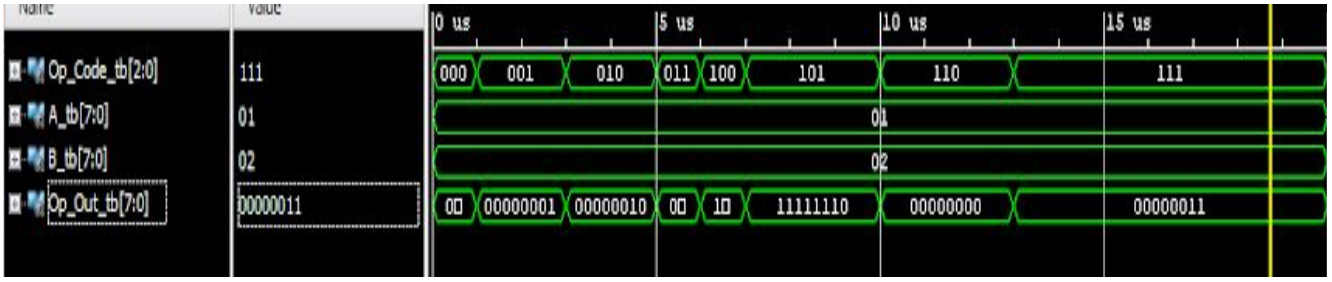
Vivado - VHDL code Test Bench for 8-bit Arithmetic Logic Unit (Added on 23.02.20):

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

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx leaf cells in this code.
--library UNISIM;
--use UNISIM.VComponents.all;
----------------------------------------------------------------------------------
entity Tutorial3_2tb is
end Tutorial3_2tb;
----------------------------------------------------------------------------------
architecture testbench of Tutorial3_2tb is
------ DUT Declaration:------
    component ALU is
        Port ( A, B: in signed (7 DOWNTO 0); -- 4 data inputs of 8 bit
          Op_Code : in STD_LOGIC_VECTOR (2 DOWNTO 0); -- 3 bit Selector
          Op_Out : out signed (7 DOWNTO 0)); -- system output
    end component;
```

```
    ------Signal Declaration:------
    signal Op_Code_tb: STD_LOGIC_VECTOR (2 DOWNTO 0):="000";
    signal A_tb: signed(7 DOWNTO 0):= "00000001";
    signal B_tb: signed(7 DOWNTO 0):= "00000010";
    signal Op_Out_tb: signed(7 DOWNTO 0);

begin
    ------ DUT Instantiation:------
        dut: ALU Port map (Op_Code => Op_Code_tb, A => A_tb, B => B_tb, Op_Out =>
Op_Out_tb);
    ------ Stimuli Generation:------
        Op_Code_tb <= "001" AFTER 1us,
                "010" AFTER 3us,
                "011" AFTER 5us,
                "100" AFTER 6us,
                "101" AFTER 7us,
                "110" AFTER 10us,
                "111" AFTER 13us;

end testbench;
```

Vivado - Simulation output of 8-bit Arithmetic Logic Unit (Added on 23.02.20):





Discussion: The circuit performs as a mux controlled by the op_code. Actions are performed according to this code with the related inputs A and B. The first output simulation diagram shows the arithmetic decimal output. The second, demonstrates how the 101 operation (NOT A) functions. In the bench A is 00000001 therefore NOT A = 11111110.

3. ROM and RAM Circuits.

A. 16 x 4 bit ROM.

Circuit Logic: The circuit returns the data stored according to the address index.

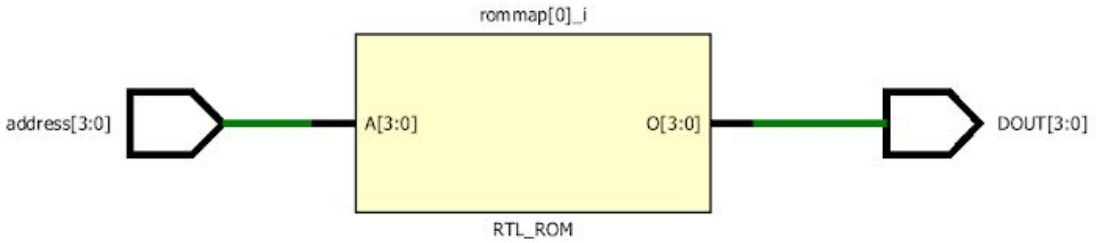
Address index	Memory Location Content
0	0100
1	1001
2	0010
3	0011
4	0110
5	0101
6	0110
7	0111
8	1010
9	1001
10	1010
11	1011
12	1100
13	1101
14	1110
15	1111

16 x 4 bit ROM - VHDL Code:

```
--------------------------------------------------------------------------------
 library ieee;
 use ieee.std_logic_1164.all;
 use ieee.numeric_std.all;
--------------------------------------------------------------------------------
entity rom16x4 is

port(address : in std_logic_vector (3 downto 0); -- address input
```

```
    DOUT : out std_logic_vector (3 downto 0)); -- data output

end rom16x4;
--------------------------------------------------------------------------------
architecture behavior of rom16x4 is
type memory is array (0 to 15) of std_logic_vector (3 downto 0); -- 16 memory locations each
4 bits

    constant rommap : memory := ("0100",  -- data stored at address 0
                                 "1001", -- data stored at address 1
                                 "0010", -- data stored at address 2
                                 "0011", -- data stored at address 3
                                 "0110", -- data stored at address 4
                                 "0101", -- data stored at address 5
                                 "0110", -- data stored at address 6
                                 "0111", -- data stored at address 7
                                 "1010", -- data stored at address 8
                                 "1001", -- data stored at address 9
                                 "1010", -- data stored at address 10
                                 "1011", -- data stored at address 11
                                 "1100", -- data stored at address 12
                                 "1101", -- data stored at address 13
                                 "1110", -- data stored at address 14
                                 "1111"); -- data stored at address 15

begin
    process (address)
    begin

        DOUT <= rommap(to_integer(unsigned(address))); -- convert address to decimal
integer  and assign the related memory location to the output.

    end process;
end behavior;
--------------------------------------------------------------------------------
```

Vivado - VHDL Schematic of 16 x 4 bit ROM:

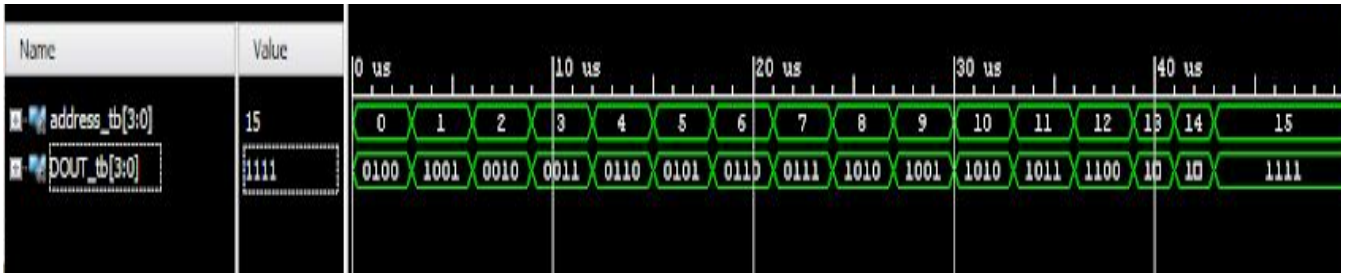


Vivado - VHDL code Test Bench for 16 x 4 bit ROM:

```
--------------------------------------------------------------------------------

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx leaf cells in this code.
--library UNISIM;
--use UNISIM.VComponents.all;
--------------------------------------------------------------------------------
entity Tutorial3_3atb is
end Tutorial3_3atb;
--------------------------------------------------------------------------------
architecture testbench of Tutorial3_3atb is
------ DUT Declaration:------
   component rom16x4 is
      Port (address : in std_logic_vector (3 downto 0); -- address input
            DOUT : out std_logic_vector (3 downto 0)); -- data output
   end component;
   ------Signal Declaration:------
   signal address_tb : std_logic_vector (3 downto 0):= "0000";
   signal DOUT_tb : std_logic_vector (3 downto 0);


begin
   ------ DUT Instantiation:------
      dut: rom16x4 Port map (address => address_tb, DOUT => DOUT_tb);
   ------ Stimuli Generation:------
      address_tb <= "0001" AFTER 3us,
                "0010" AFTER 6us,
                "0011" AFTER 9us,
```

```
        "0100" AFTER 12us,
        "0101" AFTER 15us,
        "0110" AFTER 18us,
        "0111" AFTER 21us,
        "1000" AFTER 24us,
        "1001" AFTER 27us,
        "1010" AFTER 30us,
        "1011" AFTER 33us,
        "1100" AFTER 36us,
        "1101" AFTER 39us,
        "1110" AFTER 42us,
        "1111" AFTER 45us;

end testbench;
--------------------------------------------------------------------------------
```

Vivado - VHDL simulation output for 16 x 4 bit ROM:



Discussion: The simulation demonstrates the data being retrieved from memory location. The rommap data from each memory location in the array is assigned to the output according to the memory index called.

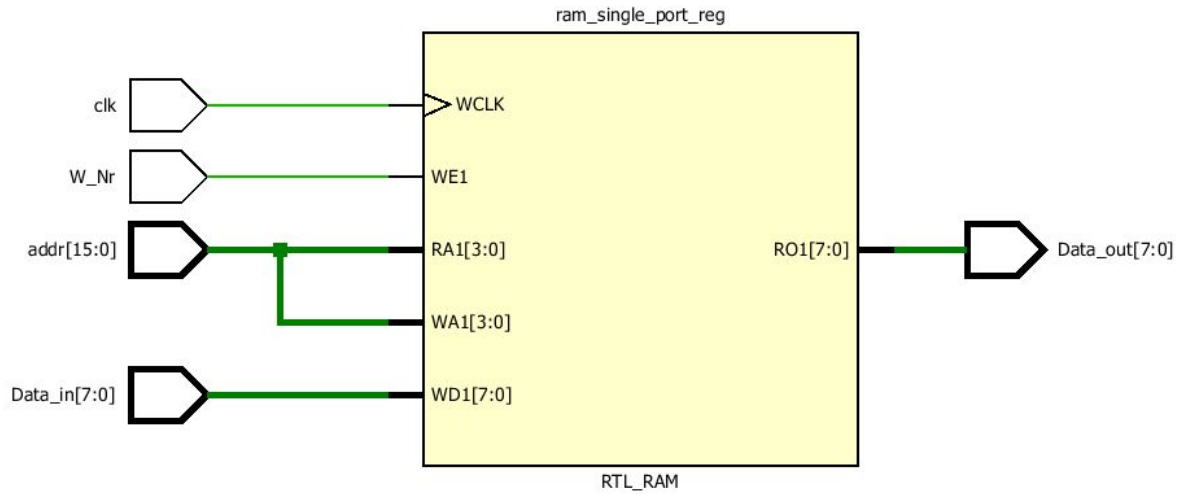
B. 16 x 8 bit RAM.

Circuit Logic: The circuit allows to pick to write or read to the memory location. For a write the input is assigned to the memory location. For a read the memory location is assigned to the output.

16 x 8 bit RAM - VHDL Code:

```
--------------------------------------------------------------------------------
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
--------------------------------------------------------------------------------
entity single_port_RAM is

  port(
    clk: in std_logic; -- clock signal
    W_Nr : in std_logic; -- write not read when high
    addr : in std_logic_vector(15 downto 0); -- ram address
    Data_in : in std_logic_vector(7 downto 0); -- data port
    Data_out : out std_logic_vector(7 downto 0) -- data port
    );
end single_port_RAM;
--------------------------------------------------------------------------------
architecture behaviour of single_port_RAM is
 type ram_type is array (15 downto 0) of std_logic_vector (7 downto 0); -- 16 addresses of 8
bit each
 signal ram_single_port : ram_type; -- ram signal
begin
  process(clk)
  begin
    if (clk'event and clk='1') then -- when the clock is high
      if (W_Nr='1') then -- if write not read is high write data to address 'addr'
        --convert 'addr' type to integer from std_logic_vector
        ram_single_port(to_integer(unsigned(addr))) <= Data_in; -- assign the data to the
current ram address.
      end if;
   end if;
  end process;
Data_out <= ram_single_port(to_integer(unsigned(addr))); -- read data out
end behaviour;
--------------------------------------------------------------------------------
```

Vivado - VHDL Schematic of 16 x 8 bit RAM:

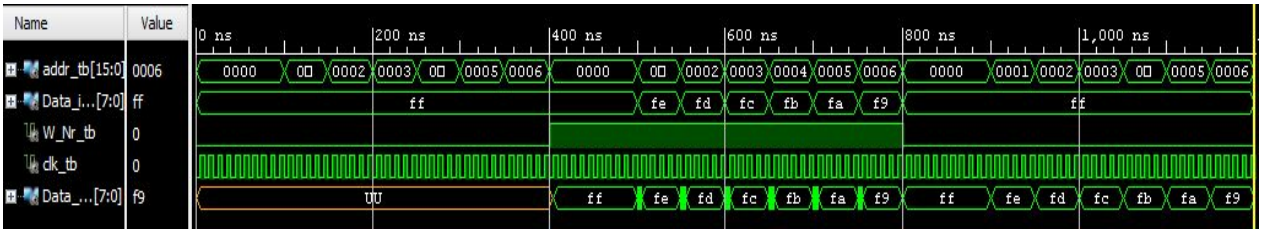


Vivado - VHDL code Test Bench for 16 x 8 bit RAM:

```
library ieee;
use ieee.std_logic_1164.ALL;
use ieee.std_logic_unsigned.ALL;

-- VHDL testbench code for the single-port RAM
entity Tutorial3_3btb is
end Tutorial3_3btb;
------ DUT Declaration:------
architecture testbench OF Tutorial3_3btb is

    -- Component Declaration for the single-port RAM in VHDL

    component single_port_RAM
    PORT(
        clk: in std_logic;
        W_Nr : in std_logic;
        addr : in std_logic_vector(15 downto 0);
        Data_in : in std_logic_vector(7 downto 0);
        Data_out : out std_logic_vector(7 downto 0)
        );
    end component;
    ------Signal Declaration:------
   signal addr_tb : std_logic_vector(15 downto 0) := (others => '0');
   signal Data_in_tb : std_logic_vector(7 downto 0) := (others => '0');
   signal W_Nr_tb : std_logic := '0';
```

```
   signal clk_tb : std_logic := '0';
   signal Data_out_tb : std_logic_vector(7 downto 0);

begin

     ------ DUT Instantiation:------
    dut: single_port_RAM Port map(addr =>addr_tb, Data_in => Data_in_tb, W_Nr =>
W_Nr_tb, clk => clk_tb, Data_out => Data_out_tb);

     ----- Stimuli Generation:------
    clk_tb <= NOT clk_tb AFTER 5ns;
    stim_proc: process
       begin
      W_Nr_tb <= '0';
      addr_tb <= "0000000000000000";
      Data_in_tb <= x"FF";
         wait for 100 ns;
      -- Read data from RAM to check if blank.
      for i in 0 to 5 loop
      addr_tb <= addr_tb + "000000000000001";
         wait for 50 ns;
      end loop;
      addr_tb <= "0000000000000000";
      W_Nr_tb <= '1';
      -- Write Data to RAM
      wait for 100 ns;
      for i in 0 to 5 loop
      addr_tb <= addr_tb + "000000000000001";
      Data_in_tb <= Data_in_tb-x"01";
         wait for 50 ns;
      end loop;
      W_Nr_tb <= '0';
           addr_tb <= "0000000000000000";
      Data_in_tb <= x"FF";
         wait for 100 ns;
      -- Read data stored in RAM.
      for i in 0 to 5 loop
      addr_tb <= addr_tb + "000000000000001";
         wait for 50 ns;
      end loop;
         wait;
       end process;
end testbench;
```

Vivado -Vivado - VHDL simulation output for 16 x 8 bit RAM:



Discussion: Similar to the ROM except an input can be assigned to an index location to write. The Write Not Read allows to select which action to perform. The test bench incorporates functions to generate random data to be written.

Task for week 4: Tutorial 4 Sequential Finite State Machine.

Task Date: 25.03.20

Objectives: Create Finite State Machines which can recognise a sequence.

1. Synchronous, sequential circuit detecting the non overlapping serial binary sequence 1101.

Circuit Logic: At each high of the clock, the programme will compute the current state, next state by comparing the input to the logic.

Input	Current State	Next State	Comment
0	s0	s0	Stay at the start
1	s0	s1	Start Sequence Detection (1)
0	s1	s0	Restart
1	s1	s2	Continue Sequence detection (11)
0	s2	s3	Continue Sequence detection (110)
1	s2	s1	Restart Sequence Detection at (1)

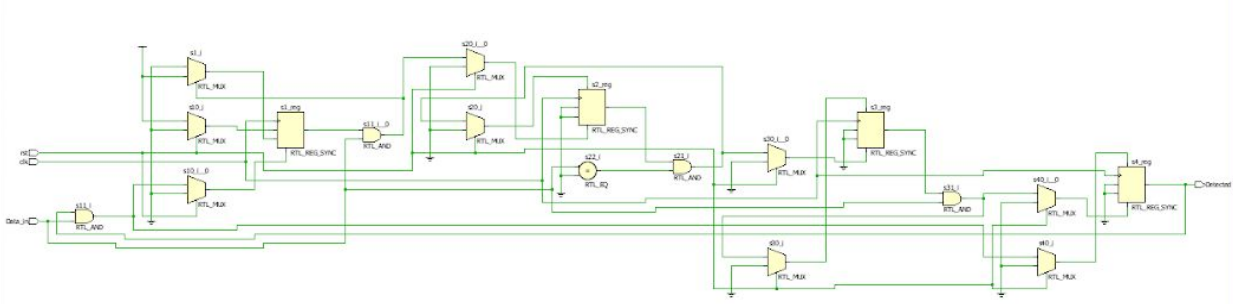
0	s3	s0	Restart
1	s3	s4	Sequence detected (1101)
0	s4	s0	Restart
1	s4	s1	Restart Sequence Detection at (1)

FSM 1101 detector - VHDL Code:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
----------------------------------------------------------------
entity FSM_Sequence_Detector is
port (
 clk: in std_logic; -- clock signal
 rst: in std_logic; -- reset input
 Data_in: in std_logic; -- binary sequence input
 Detected: out std_logic -- output of the VHDL sequence detector
);
end FSM_Sequence_Detector;
----------------------------------------------------------------
architecture behaviour of FSM_Sequence_Detector is
signal s0,s1,s2,s3,s4: std_logic:= '0'; -- state signals

begin
-- Sequential control of the Sequence Detector
process(clk,rst,Data_in)
    begin
     if(rst='1') then -- at reset zero the states
        s0<='1';
        s1<='0';
        s2<='0';
        s3<='0';
        s4<='0';
     elsif(rising_edge(clk)) then -- change state at turn of the clock
          if s0='1' and Data_in='1' then -- if state 0 high and first bit a 1 = state 1
            s1 <= '1';
            s0 <= '0';
          else --back to state 0
            s0<='1';
          end if;
          if s1='1' and Data_in='1' then -- if state 1 high and data bit a 1 = state 2
            s2 <= '1';
            s1 <= '0';
```

```
        else --back to state 1
        s1<='1';
        end if;
        if s2='1' and Data_in='0' then -- if state 2 high and data bit a 1 = state 3
          s3 <= '1';
          s2 <= '0';
        else --back to state 0
          s0<='1';
        end if;
        if s3='1' and Data_in='1' then -- if state 3 high and data bit a 1 = state 4
          s4 <= '1';
          s3 <= '0';
        else -- back to state 0
          s0<='1';
        end if;
        if s4='1' and Data_in='1' then -- if state 4 high and data bit a 1 = back to state 1
          s1 <='1';
        else  --back to state 0
          s0 <='1';
        end if;
       end if;
   end process;
-------------------------------------------------------------
-- Output logic of the Sequence Detector
Detected <= '1' when s4='1' else -- if s4 high then sequence detected
           '0'; -- otherwise 0

end behaviour;
-------------------------------------------------------------
```

Vivado - VHDL Schematic of FSM 1101 detector:

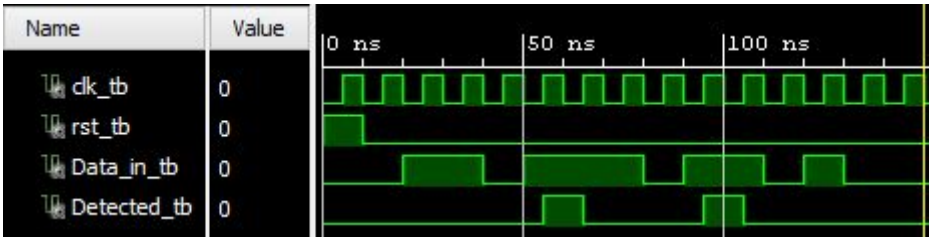


Vivado - VHDL code Test Bench for FSM 1101 detector:

```
library ieee;
use ieee.std_logic_1164.ALL;
 ----------------------------------------------------------------
entity Tutorial4_1tb is
end Tutorial4_1tb;
----------------------------------------------------------------
architecture testbench of Tutorial4_1tb is

------ DUT Declaration:------
    component FSM_Sequence_Detector
       Port(
            clk : IN  std_logic;
            rst : IN  std_logic;
            Data_in : IN  std_logic;
            Detected : OUT  std_logic
           );
       end component;
       ------Signal Declaration:------
      signal clk_tb : std_logic := '0'; -- test bench signal for the clock
      signal rst_tb : std_logic := '0'; -- test bench signal for the reset
      signal Data_in_tb : std_logic := '0'; -- test bench signal for the data input
      signal Detected_tb : std_logic; -- test bench signal for the output

begin

------ DUT Instantiation:------
  uut: FSM_Sequence_Detector Port map (clk => clk_tb, rst => rst_tb, Data_in =>
Data_in_tb, Detected => Detected_tb); -- connect testbench to circuit

----- Stimuli Generation:------
  clk_tb <= not clk_tb after 5ns; -- switch clock every 5ns
  stim_proc: process -- After reset switch input every 10ns
  begin
    Data_in_tb <= '0';
    rst_tb <= '1'; -- reset input
       wait for 10 ns;
    rst_tb <= '0'; -- reset off
      wait for 10 ns;
    Data_in_tb <= '1';
      wait for 10 ns;
    Data_in_tb <= '1';
      wait for 10 ns;
    Data_in_tb <= '0';
      wait for 10 ns;
```

```
      Data_in_tb <= '1';
        wait for 10 ns;
      Data_in_tb <= '1';
        wait for 10 ns;
      Data_in_tb <= '1';
        wait for 10 ns;
      Data_in_tb <= '0';
        wait for 10 ns;
      Data_in_tb <= '1';
        wait for 10 ns;
      Data_in_tb <= '1';
        wait for 10 ns;
      Data_in_tb <= '0';
        wait for 10 ns;
      Data_in_tb <= '1';
        wait for 10 ns;
      Data_in_tb <= '0';
        wait for 10 ns;
      Data_in_tb <= '0';
        wait;
   end process;
end testbench;
-------------------------------------------------------------
```

Vivado - VHDL simulation output for FSM 1101 detector:



Discussion: How many D Flip Flops are used?

Five D-flip flops are used showing the five states the circuits go through before reaching the output.

2. With enumerated data type.

Circuit Logic: At each high of the clock, the programme will compute the current state, next state by comparing the input to the logic.

Input	Current State	Next State	Comment
0	Zero	Zero	Stay at the start
1	Zero	One	Start Sequence Detection (1)
0	One	Zero	Restart
1	One	OneOne	Continue Sequence detection (11)
0	OneOne	OneOneZero	Continue Sequence detection (110)
1	OneOne	One	Restart Sequence Detection at (1)
0	OneOneZero	Zero	Restart
1	OneOneZero	OneOneZero One	Sequence detected (1101)
0	OneOneZero One	Zero	Restart
1	OneOneZero One	One	Restart Sequence Detection at (1)

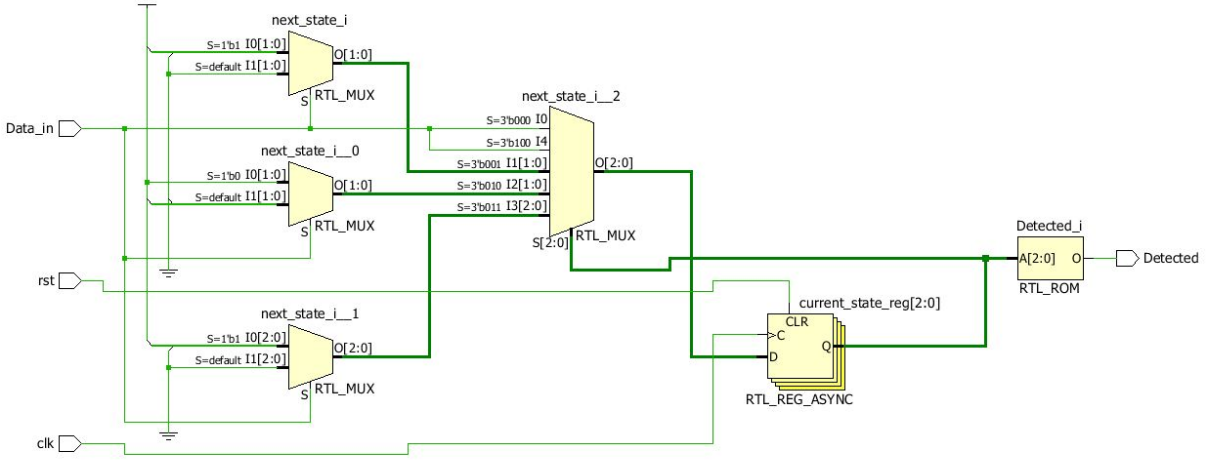
FSM 1101 detector - VHDL Code:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
----------------------------------------------------------
entity FSM_Sequence_Detector is
port (
 clk: in std_logic; -- clock signal
 rst: in std_logic; -- reset input
 Data_in: in std_logic; -- binary sequence input
 Detected: out std_logic -- output of the VHDL sequence detector
);
end FSM_Sequence_Detector;
----------------------------------------------------------
```

```
architecture behaviour of FSM_Sequence_Detector is
type Sequence_Detector is (Zero, One, OneOne, OneOneZero, OneOneZeroOne); --
Sequence State Names
signal current_state, next_state: Sequence_Detector; -- State Flags

begin
-- Sequential control of the Sequence Detector
process(clk,rst)
   begin
    if(rst='1') then -- at reset zero the current state
     current_state <= Zero;
    elsif(rising_edge(clk)) then -- change state at turn of the clock
     current_state <= next_state;
    end if;
end process;
-----------------------------------------------------------------
-- Next state logic of the Sequence Detector
process(current_state,Data_in)
   begin
      case(current_state) is
       when Zero => -- previous bit sequence
        if(Data_in='1') then -- required bit
         -- "1"
         next_state <= One;
        else
         next_state <= Zero; -- restart sequence
        end if;
       when One => -- previous bit sequence
        if(Data_in='1') then -- required bit
         -- "11"
         next_state <= OneOne;
        else
         next_state <= Zero; -- restart sequence
        end if;
       when OneOne => -- previous bit sequence
        if(Data_in='0') then -- required bit
         -- "110"
         next_state <= OneOneZero;
        else
         next_state <= One; -- start sequence check again
        end if;
       when OneOneZero => -- previous bit sequence
        if(Data_in='1') then -- required bit
         -- "1101"
         next_state <= OneOneZeroOne;
        else
```

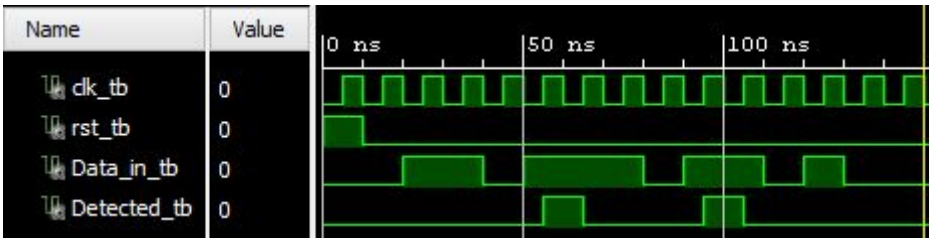
```
          next_state <= Zero; -- restart sequence
         end if;
        when OneOneZeroOne => -- previous bit sequence
         if(Data_in='1') then -- start sequence check again
          next_state <= One;
         else
          next_state <= Zero; -- restart sequence
         end if;
       end case;
end process;
-------------------------------------------------------------
-- Output logic of the Sequence Detector
process(current_state)
   begin
     case current_state is -- Detected is low until correct Sequence State Name
       when Zero =>
        Detected <= '0';
       when One =>
        Detected <= '0';
       when OneOne =>
        Detected <= '0';
       when OneOneZero =>
        Detected <= '0';
       when OneOneZeroOne =>
        Detected <= '1';
      end case;
   end process;
end behaviour;
-------------------------------------------------------------
```

Vivado - VHDL Schematic of FSM 1101 detector:



Vivado - VHDL code Test Bench for FSM 1101 detector:
Same as previous

Vivado - VHDL simulation output for FSM 1101 detector:



Discussion: Using an enumerated data type makes the programme code simpler to implement which is reflected in the schematics implementation being much clearer and compact.

3. 111 sequence detector with Z1 and Z2 outputs.

Circuit Logic: At each high of the clock, the programme will compute the current state, next state by comparing the input to the logic. If the sequence “00” is detected the output Z2 will go to high. If “00” has not been detected yet and the sequence “111” is detected, Z1 will go to high.

Input	Current State	Next State	Comment
0	Zero	ZeroZero	Z2 Sequence detected (00)
1	Zero	One	Start Z1 Sequence Detection (1)
0	One	Zero	Restart Z2 Sequence detection (0)
1	One	OneOne	Continue Z1 Sequence detection (11)
0	OneOne	Zero	Restart Z2 Sequence detection (0)
1	OneOne	OneOneOne	Z1 Sequence detected (111)
0	OneOneOne	Zero	Restart Z2 Sequence detection (0)
1	OneOneOne	One	Restart Z1 Sequence Detection (1)
0	ZeroZero	Zero	Z2 trigger flat is high

			Restart Z2 Sequence detection (0)
1	ZeroZero	One	Z2 trigger flat is high Restart Z1 Sequence detection (1)

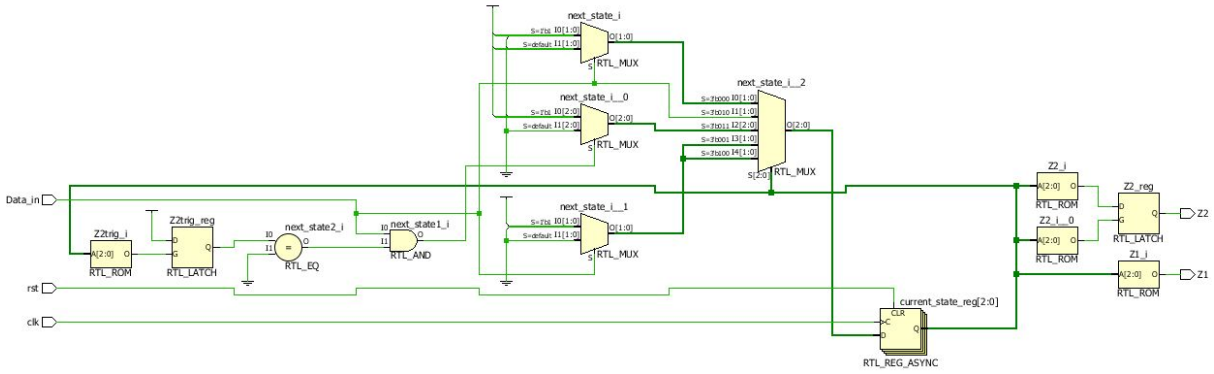
FSM 111 detector - VHDL Code:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
----------------------------------------------------------------
entity FSM_Sequence_Detector is
port (
 clk: in std_logic; -- clock signal
 rst: in std_logic; -- reset input
 Data_in: in std_logic; -- binary sequence input
 Z1: out std_logic; -- Z1 output of the VHDL sequence detector
 Z2: out std_logic -- Z2 output of the VHDL sequence detector
);
end FSM_Sequence_Detector;
----------------------------------------------------------------
architecture behaviour of FSM_Sequence_Detector is
type Sequence_Detector is (Zero, ZeroZero, One, OneOne, OneOneOne); -- Sequence
State Names
signal current_state, next_state: Sequence_Detector; -- State Flags
signal Z2trig : std_logic := '0'; -- Z2 triggered state
begin
-- Sequential control of the Sequence Detector
process(clk,rst)
   begin
    if(rst='1') then -- at reset zero the current state and Z2 Trigger state
     current_state <= Zero;
     Z2trig <= '0';
    elsif(rising_edge(clk)) then -- change state at turn of the clock
     current_state <= next_state;
    end if;
end process;
----------------------------------------------------------------
-- Next state logic of the Sequence Detector
process(current_state,Data_in)
   begin
      case(current_state) is
       when Zero => -- previous 0 bit
        if(Data_in='1') then -- required bit for Z1
         -- "1"
         next_state <= One;
```

```
         else
          next_state <= ZeroZero; -- required bit for Z2
         end if;
        when One => -- previous bit sequence
         if(Data_in='1') then -- required bit for Z1
          -- "11"
          next_state <= OneOne;
         else
          next_state <= Zero; -- restart sequence
         end if;
        when OneOne => -- previous bit sequence
         if(Data_in='1') and Z2trig='0' then -- required bit for Z1 and Z2 ondition
          -- "111"
          next_state <= OneOneOne;
         else
          next_state <= Zero; -- start sequence check again
         end if;
        when ZeroZero => -- to restart sequence from 00
         if(Data_in='1') then
          -- "001"
          next_state <= One; -- restart sequence at 1
         else
          next_state <= Zero; -- restart sequence at 0
         end if;
        when OneOneOne => -- to restart sequence from 111
         if(Data_in='1') then
          next_state <= One; -- restart sequence at 1
         else
          next_state <= Zero; -- restart sequence at 0
         end if;
       end case;
end process;
-------------------------------------------------------------
-- Output logic of the Sequence Detector
process(current_state)
   begin
      case current_state is -- Z1 is low until OneOneOne Valid, Z2 and Z2 trigger low until
ZeroZero
        when Zero =>
         Z1 <= '0';
         Z2 <= '0';
        when One =>
         Z1 <= '0';
         Z2 <= '0';
        when OneOne =>
         Z1 <= '0';
```

```
        Z2 <= '0';
       when OneOneOne =>
        Z1 <= '1';
       when ZeroZero =>
        Z2 <= '1';
        Z1 <= '0';
        Z2trig <='1';
    end case;
  end process;
end behaviour;
----------------------------------------------------------------
```

Vivado - VHDL Schematic of FSM 111 sequence detector



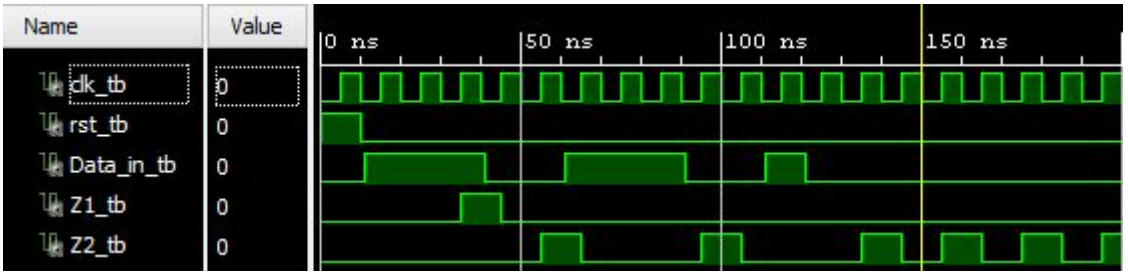
Vivado - VHDL code Test Bench for 111 Sequence detector:

```
library ieee;
use ieee.std_logic_1164.ALL;
 ----------------------------------------------------------------
entity Tutorial4_2tb is
end Tutorial4_2tb;
----------------------------------------------------------------
architecture testbench of Tutorial4_2tb is

------ DUT Declaration:------
    component FSM_Sequence_Detector
       Port(
            clk : IN  std_logic;
            rst : IN  std_logic;
            Data_in : IN  std_logic;
            Z1 : OUT  std_logic;
            Z2 : OUT  std_logic
           );
       end component;
```

```
        ------Signal Declaration:------
       signal clk_tb : std_logic := '0'; -- test bench signal for the clock
       signal rst_tb : std_logic := '0'; -- test bench signal for the reset
       signal Data_in_tb : std_logic := '0'; -- test bench signal for the data input
       signal Z1_tb : std_logic := '0'; -- test bench signal for the Z1 output
       signal Z2_tb : std_logic := '0'; -- test bench signal for the Z2 output


begin

------ DUT Instantiation:------
   uut: FSM_Sequence_Detector Port map (clk => clk_tb, rst => rst_tb, Data_in =>
Data_in_tb, Z1 => Z1_tb, Z2 => Z2_tb); -- connect testbench to circuit

----- Stimuli Generation:------
   clk_tb <= not clk_tb after 5ns; -- switch clock every 5ns
   stim_proc: process -- After reset switch input every 10ns
   begin
      Data_in_tb <= '0';
      rst_tb <= '1'; -- reset input
         wait for 10 ns;
      rst_tb <= '0'; -- reset off
        wait for 1 ns;
      Data_in_tb <= '1';
        wait for 10 ns;
      Data_in_tb <= '1';
        wait for 10 ns;
      Data_in_tb <= '1';
        wait for 10 ns;
      Data_in_tb <= '0';
        wait for 10 ns;
      Data_in_tb <= '0';
        wait for 10 ns;
      Data_in_tb <= '1';
        wait for 10 ns;
      Data_in_tb <= '1';
        wait for 10 ns;
      Data_in_tb <= '1';
        wait for 10 ns;
      Data_in_tb <= '0';
        wait for 10 ns;
      Data_in_tb <= '0';
        wait for 10 ns;
      Data_in_tb <= '1';
        wait for 10 ns;
      Data_in_tb <= '0';
```

```
        wait;
    end process;
end testbench;
-------------------------------------------------------------------
```

Vivado -Vivado - VHDL simulation output for Z1 and Z2



Discussion: Z1 triggers at “111” if the Z2 triggered flag is low until the whole system is reset. Z2 triggers every time “00” is triggered.