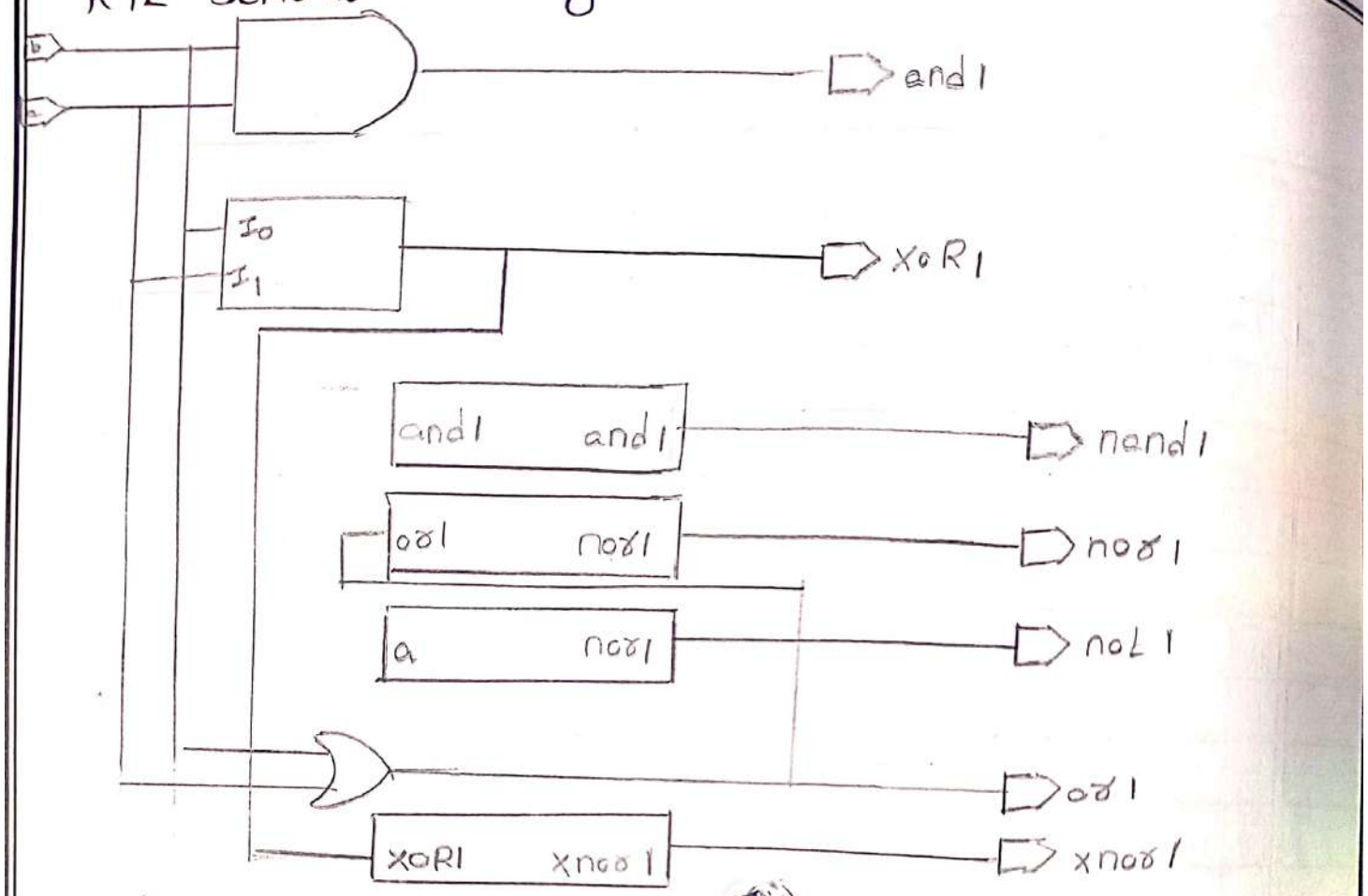
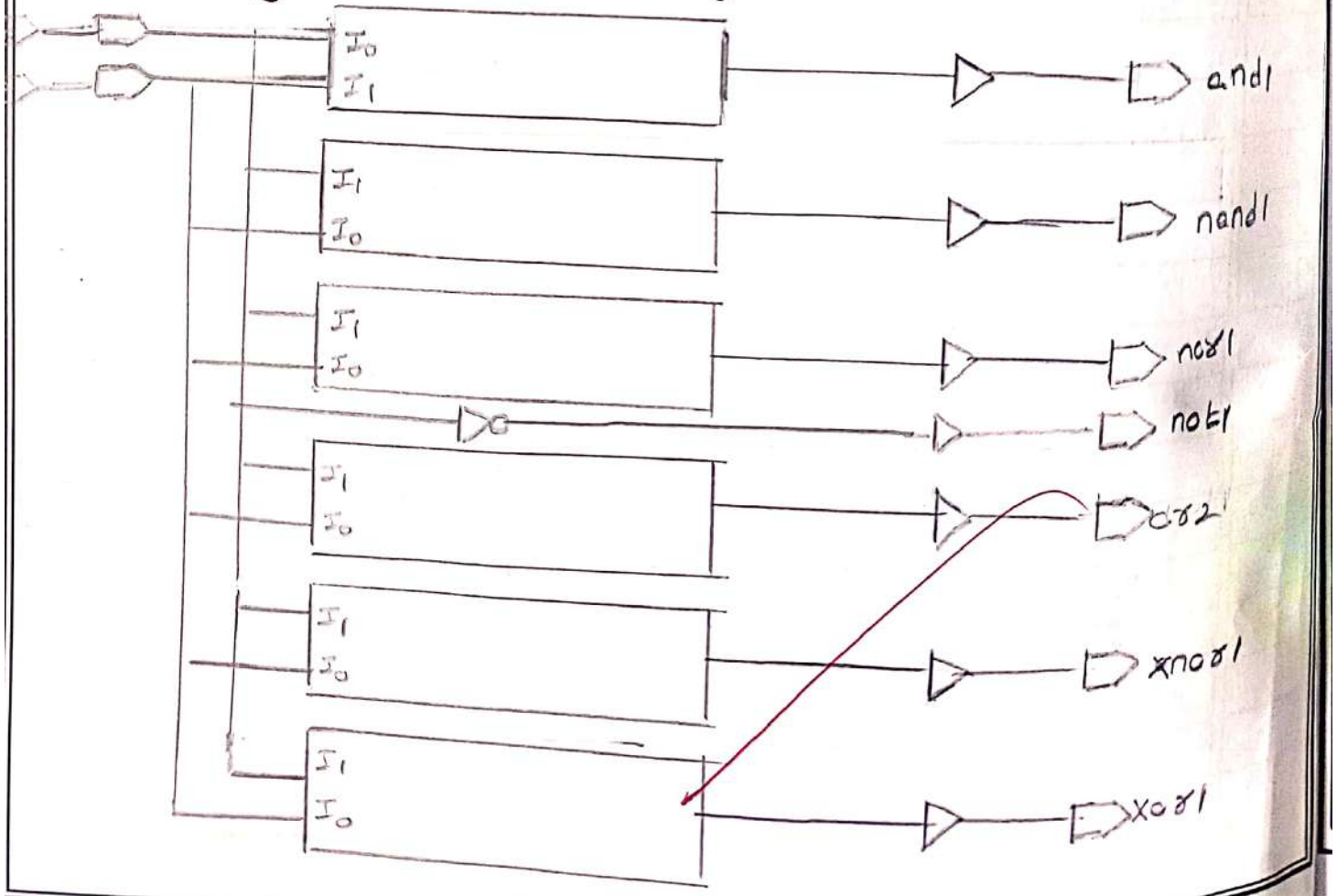
RTL schematic Design:



Technologic schematic design:



1. Realization of logic Gates.

Page No. 1
Date 12/7/19

Aim:
To write a verilog and VHDL source code for realization of logic gates and generate RTL schematic technology map and simulation synthesis report.

Hardware Required:
Digital IC

Software Required:
Xilinx ISE 8.1

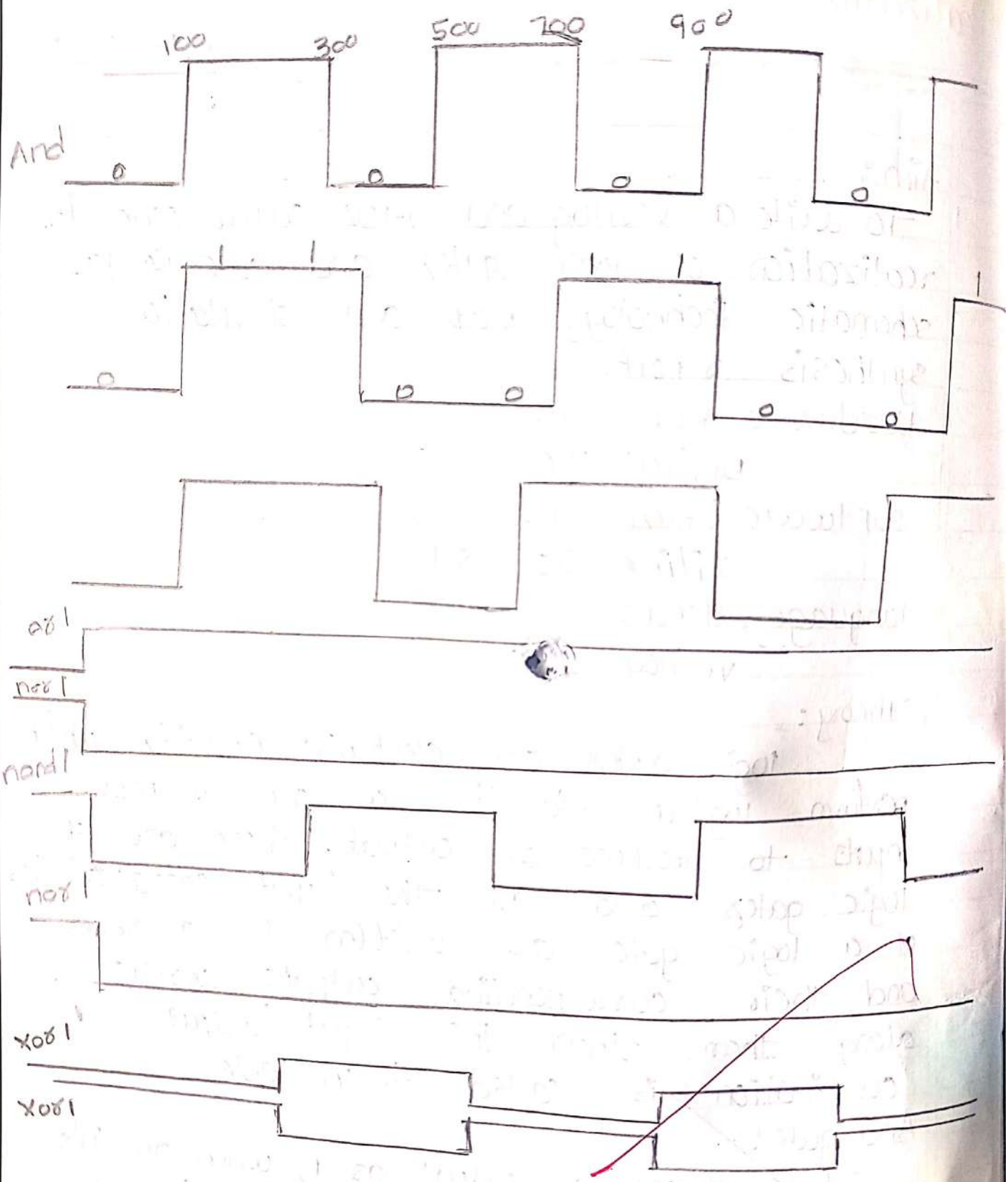
language used:
verilog, VHDL

Theory:
logic gates are electronic circuits which perform logical functions on one or more inputs to produce one output. There are 7 logic gates. when all the input characteristics of a logic gate are written in a series and their corresponding outputs written along them, then this input/output combination is called truth table.

And gate:
It produces an output as 1, when all it's inputs are '1' otherwise output is '0'.

OR gate:
It produces an output as 1, when any or all it's inputs are 1, otherwise the output is '0'.

simulation wave forms:



NOT gate:

It produces the component of its input. It is also called inverter.

NAND gate:

It is actually series of AND gate with NOT gate. Its i/p is 1 when any or all inputs are '0' otherwise 1.

NOR gate:

It is actually a series of OR gate with NOT gate. Its o/p is 0, when any of all inputs are 1.

XOR gate:

It produces an o/p as 1, when number of 1's at its i/p's is odd otherwise 0

XNOR gate:

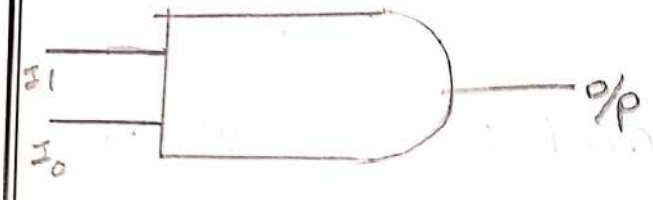
It produces an o/p as 1, when number of 1's i/p's is not odd otherwise '0'.

procedure:

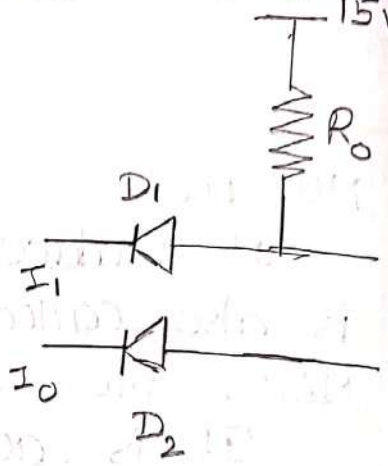
* Double click on xilinx ISE 8.1 software.
* Goto file menu, close the project & start new project.
* Enter project name and click on next.
* product category : All
  family : spartan 3E
  package : VQ100
  speed : -5
  Device : XC3S100E

logic gates hardware circuit diagram with logic gate:

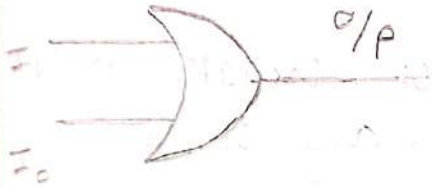
AND Gate:



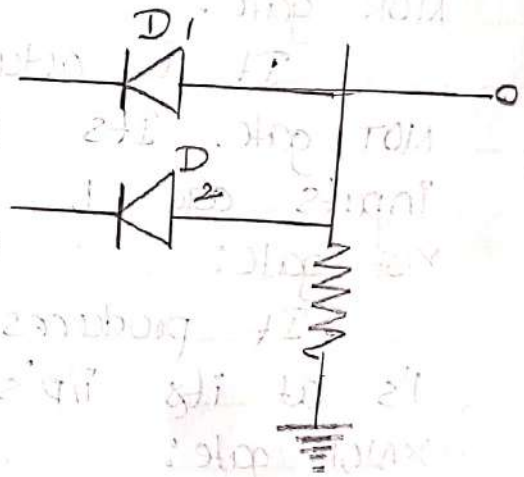
I_1	I_0	o/p
0	0	0
0	1	0
1	0	0
1	1	1



OR Gate:

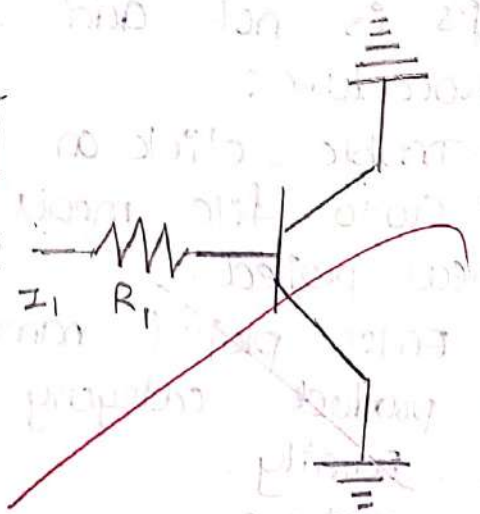


I_1	I_0	o/p
0	0	0
0	1	1
1	0	1
1	1	1



NOT Gate:

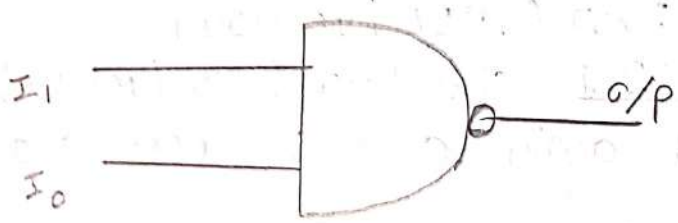
I	o/p
0	1
1	0



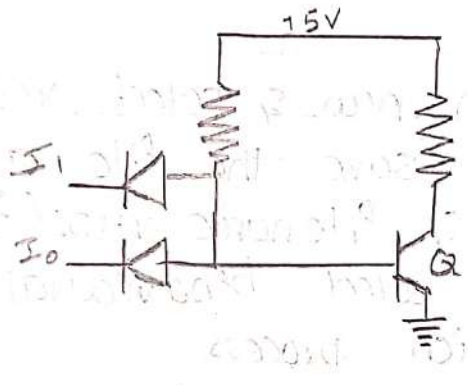
Top level source type : VHDL
synthesis tool : XST (VHDL/verilog)
simulator : ISE simulator (VHDL/verilog)

* click on next and again click on next and then finish.
* Go to file menu, click on new & select text file and ok.
* In text file click on new & select verilog code/VHDL code then save the file as filename.v (for verilog) & filename.vhdl (for VHDL)
* click on source and select behavioural simulation for simulation process.
* To add source file of the device xc3s100, right click, select add source & add filename.v to the device.
* Go to click on filename.bh we get xilinx ISE simulator. Now click 't' which shows the check syntax to check the program error.
* under process double click on create new source In that select test bench waveform and give filename. Then click on next → next → finish
* Then a window displays as filename.tbw
* Now click on process behind the hierachy, click 't' of xilinx ISE, in that double click generate expected simulation & then click on yes, open simulation window to verify o/p i.e gate & synthesis report.

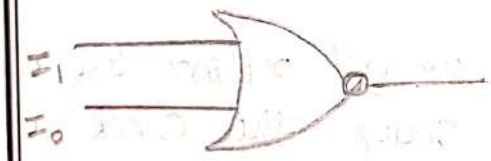
NAND Gate:



I_1	I_0	O/P
0	0	1
0	1	1
1	0	1
1	1	0

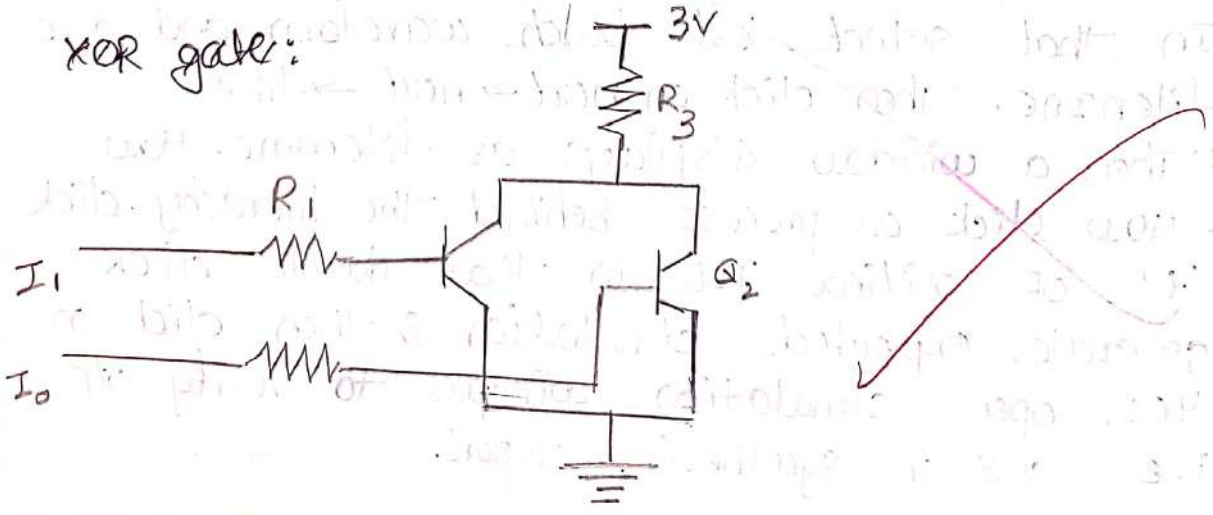


NOR gate:



I_1	I_0	O/P
0	0	0
0	1	1
1	0	1
1	1	1

XOR gate:



* Again click on simulation implementation source and select synthesis / implementation.
* Go to process, double click synthesis XST, view synthesis report.
* click on view technology schematic to generate technology schematic.

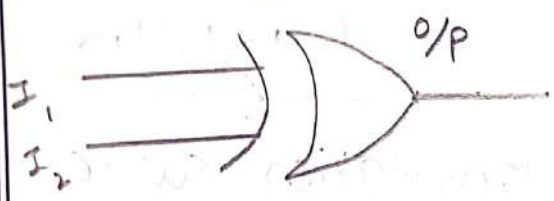
verilog source code:

```
Module gate_bh(and1, or1, not1, nand1, nor1,
    xor1, xnor1, a, b);
input a,b;
output and1, or1, not1, nand1, nor1, xor1, xnor1;
reg and1, or1, not1, nand1, nor1, xor1, xnor1;
always @(a,b)
begin
and1 = a&b;
or1 = a|b;
not1 = ~a;
nand1 = ~(a&b);
nor1 = ~(a|b);
xor1 = a^b;
xnor1 = ~(a^b);
 end
   end
```

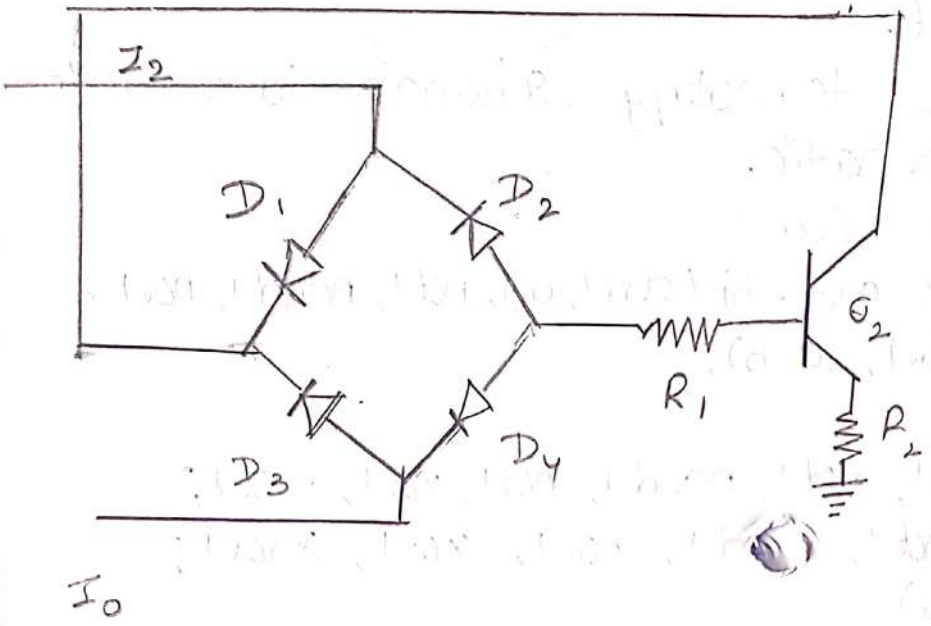
VHDL code:

```
library IEEE;
use IEEE.std_logic_1164.all;
Entity logic gates is
```

XOR - gate:

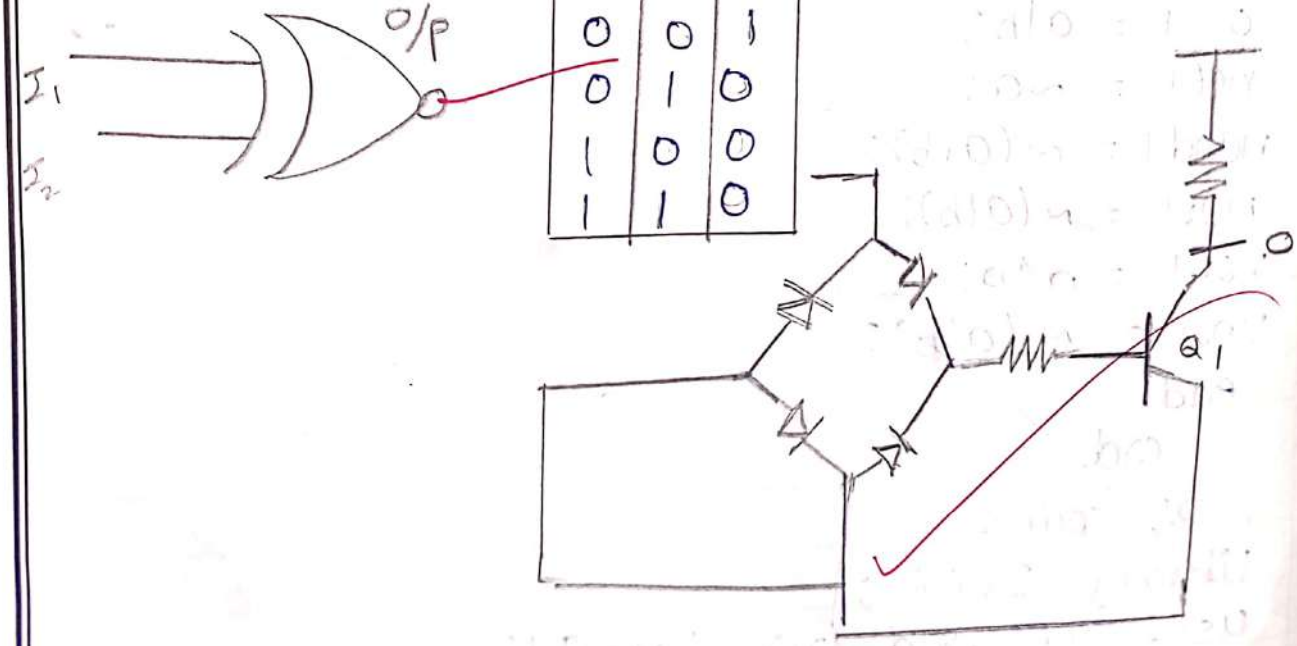


I_1	I_0	O/P
0	0	0
0	1	1
1	0	1
1	1	1



XNOR gate:

I_1	I_0	O/P
0	0	1
0	1	0
1	0	0
1	1	0

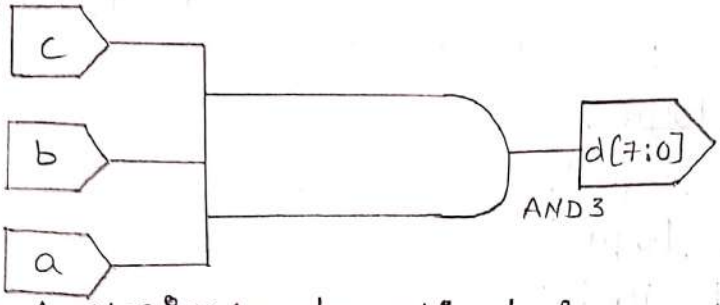


```
port(a: in std_logic;
     b: in std_logic;
     e: out std_logic;
     f: out_std_logic;
     g: out_std_logic;
     h: out_std_logic;
     i: out_std_logic;
     j: out_std_logic;
     k: out_std_logic;
end logic gates.
Architecture dataflow of logic gates is
begin
   e<= a and b;
   f<= a or b;
   g<= not a;
   h<= a nand b;
   i<= a nor b;
   j<= a xor b;
   k<= a xnor b;
end dataflow
```

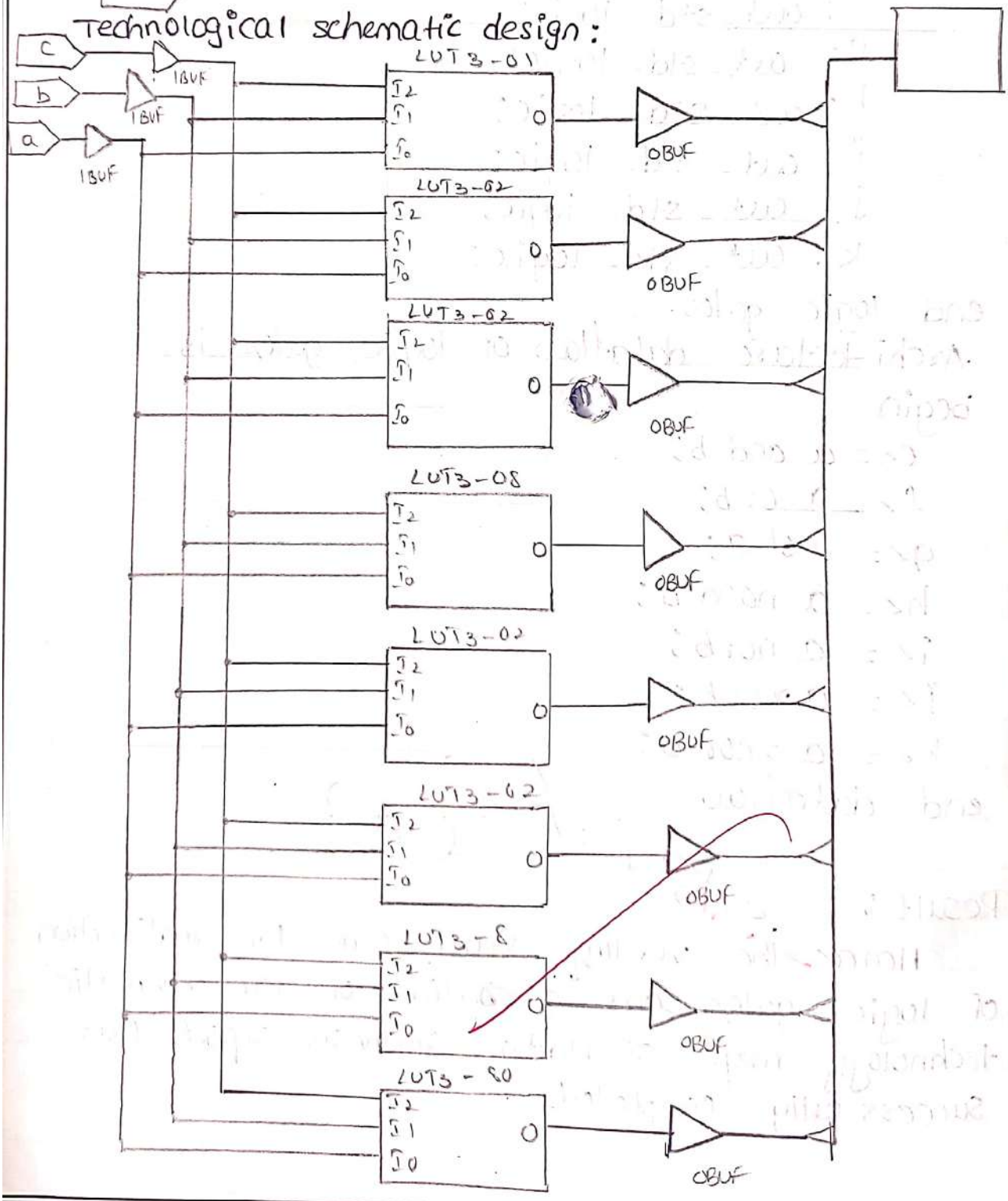
Result:

Hence the verilog source code for realization of logic gates and generation of RTL schematic technology map simulation synthesis report was successfully completed.

RTL-schematic:



Technological schematic design:



2. 3 TO 8 Decoder - 74138

Page No. 6
Date 19/7/19

Aim:
To write a verilog source code for 3-to-8 decoder 74138 and generate RTL schematic technology map simulation synthesis Report.

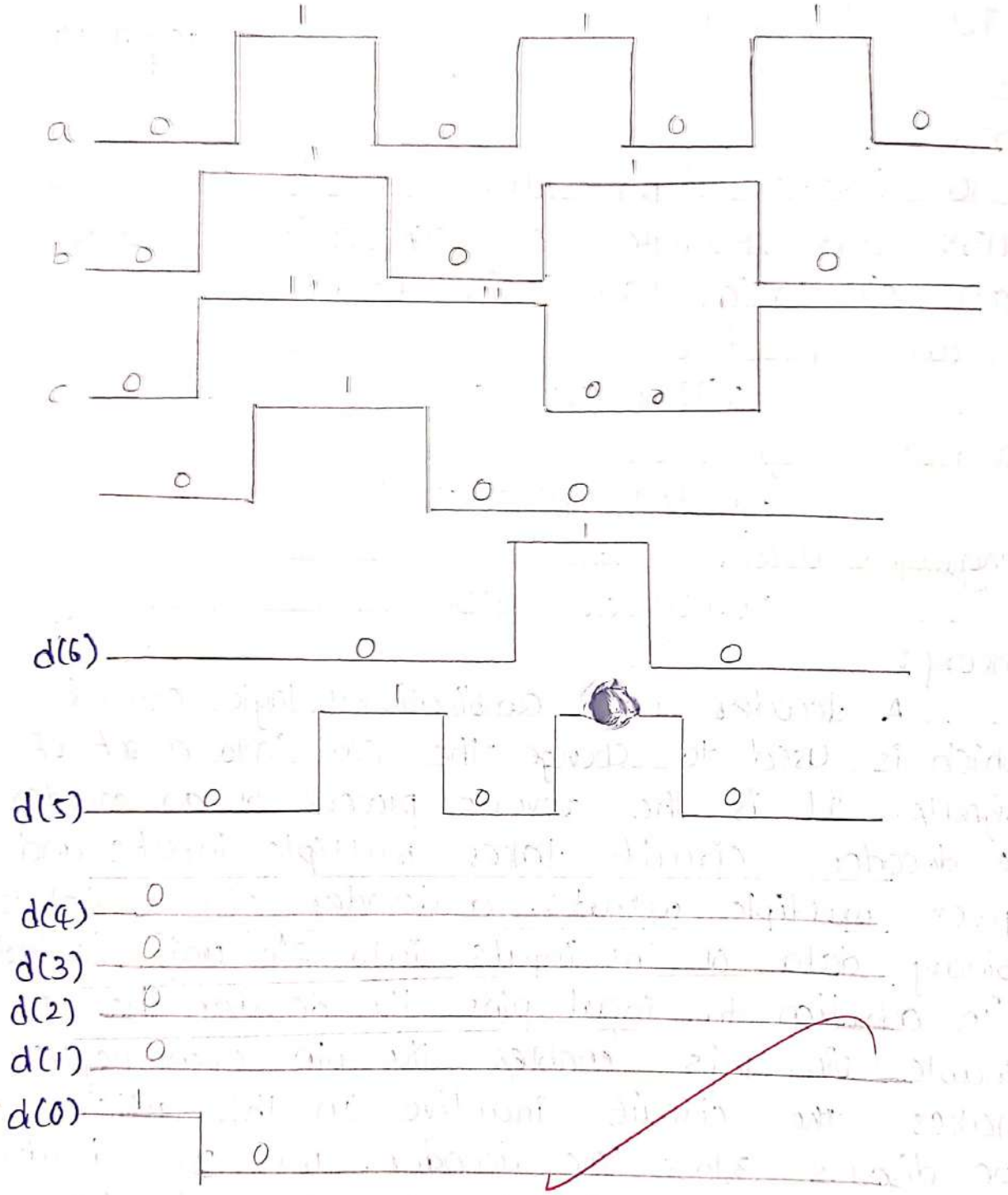
Software Required:
Xilinx 8.1

Hardware Required:
Digital IC 74138

languages used:
verilog or VHDL

Theory:
A decoder is a combinational logic circuit which is used to change the code into a set of signals. It is the reverse process of an encoder. A decoder circuit takes multiple inputs and gives multiple outputs. a decoder circuit takes binary data of 'n' inputs into 2^n unique outputs. In addition to input pins the decoder has a enable pin. This enables the pin when negated makes the circuit inactive. In this article, we discuss 3-to-8 line decoder. This 3-to-8 decoder circuit gives 8 logic outputs For 3 inputs and has a enable pin. The circuit is designed with AND and NAND logic gates. It takes 3 binary inputs and activates one of the eight output. 3-to-8 line decoder circuit is also called as binary to an octal decoder.

simulation wave-form:

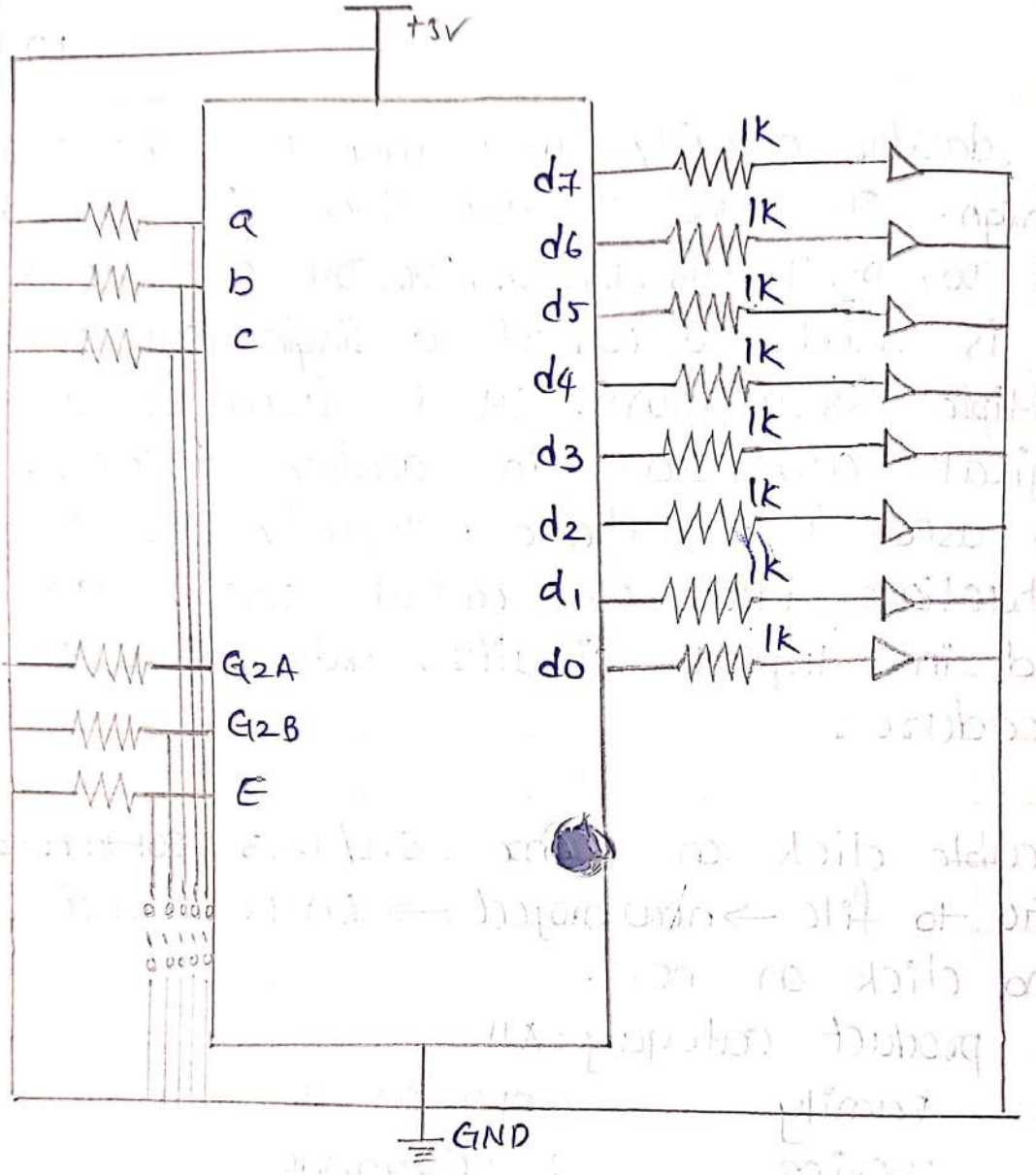


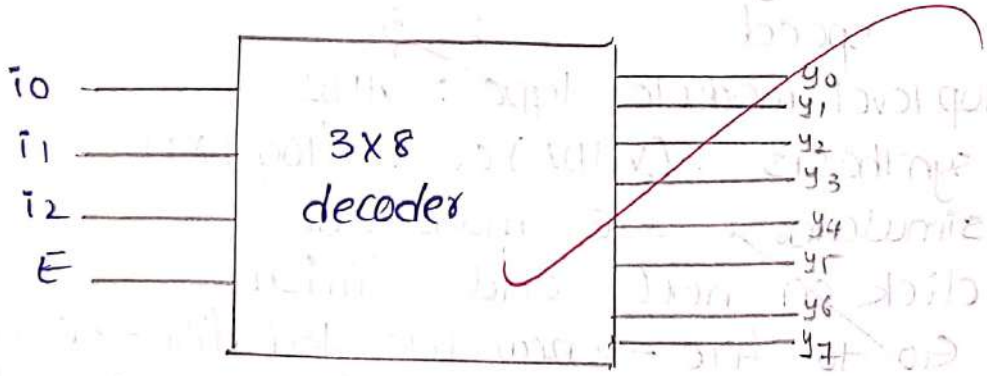
The decoder circuits work only when the enable pin is high. So S_1 and S_2 are three different inputs and D_0, D_1, D_2, D_3, D_4, D_5, D_6, D_7 are the 8 outputs. It is used to connect a single source to multiple destinations. It is used in analog to digital conversion in analog decoders. It is also used in electronic circuits to convert instructions into cpu control signals and mainly used in logical circuits, data transfer.

procedure:

* Double click on Xilinx 8.1/12.3 software
* Go to file → new project → Enter project name and click on next.
  - product category : All
  - Family : spartan 3E
  - Device : CC3550E
  - package : FG.320
  - speed : 5
  - Top level module type : VHDL
  - synthesis : (VHDL) or verilog) XST
  - simulator : ISE modulator.
* click on next and finish
* Go to file → new file text file → click ok.
* Enter verilog code → file save as → file name.v
* Add source file to the device
* Right click on the parameter available at

Hardware circuit:





the left panel → select and source → select filename → click open → ok.

* click synthesis → XST → check syntax
* one source window select behavioural simulation under process → select create new folder → select test bench waveform → sign file name.
* click next and then finish.
* input and values in the test bench waveform and save the values.
* under process → click xilinx ISE simulation → click generate expected simulation → click yes → yes
* expected output waveform as proceded and observe the output values.
* click on source window select synthesis implementation under process → click on view RTL schematic.
* click on under process → click Technology schematic.

Verilog source code:

```
module data [d,a,b,c];
 input a,b,c;
output [7:0] d;
assign d[0]: ~a &~b &~c;
assign d[1]: ~a &~b &c;
assign d[2]: ~a &b &~c;
assign d[3]: ~a &b &c;
assign d[4]: a &~b &~c;
```

Truth table:

E_0	i_0	i_1	i_2	y_7	y_6	y_5	y_4	y_3	y_2	y_1	y_0
0	X	X	X	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	1
1	0	0	0	0	0	0	0	0	0	1	0
1	0	0	0	0	0	0	0	0	1	0	0
1	0	1	1	0	0	0	0	1	0	0	0
1	1	0	0	0	0	0	1	0	0	0	0
1	1	0	1	0	0	1	0	0	0	0	0
1	1	1	0	0	1	0	0	0	0	0	0
1	1	1	1	1	0	0	0	0	0	0	0

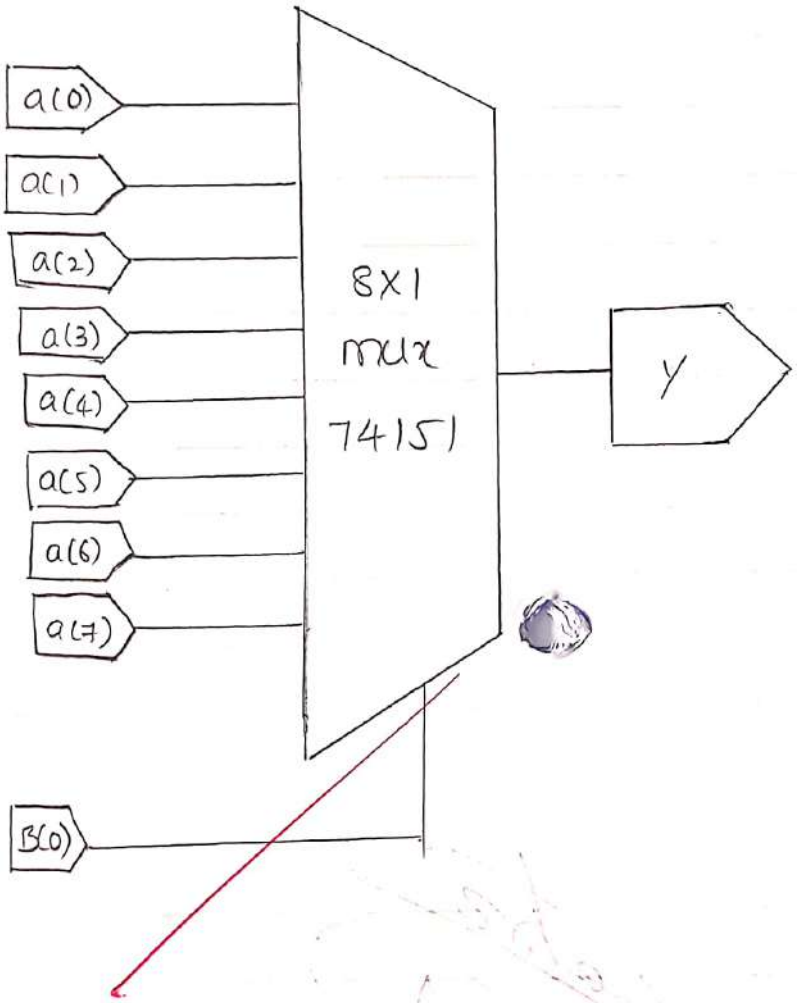
assign d[5] : ~b & c
assign d[6] : a & b & c;
assign d[7] : a & b & c;
VHDL code :

```
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL
use IEEE.STD_LOGIC_unsigned.ALL;
entity decoder is
port(a: in STD_LOGIC_unsigned vector(2 down to 0);
     y: STD_LOGIC_vector (7 down to 0));
end decoder.
Architecture behavioral of decoder is begin
 process(a)
  begin a
case a is
when "000" => y <= "00000001";
when "001" => y <= "00000010";
when "010" => y <= "00000100";
when "011" => y <= "00001000";
when "100" => y <= "00010000";
when "101" => y <= "00100000";
when "110" => y <= "01000000";
when "111" => y <= "10000000";
 End case;
 end process;
 end behavioral;
```

Result:

Hence the verilog source code for 3 to 8 decoder 74138 and generate RTL schematic technology map simulation synthesis report was successfully completed.

RTL- schematic:



3(a). 8X1 Multiplexer

Page No. 11
Date 9/8/19

Aim:

To write a verilog source code for 8x1 multiplexer 74151 and generate RTL schematic technology map, simulation synthesis report.

Software Required:

Xilinx 8.1i

Hardware Required:

Digital IC 74151

language used:

verilog and VHDL

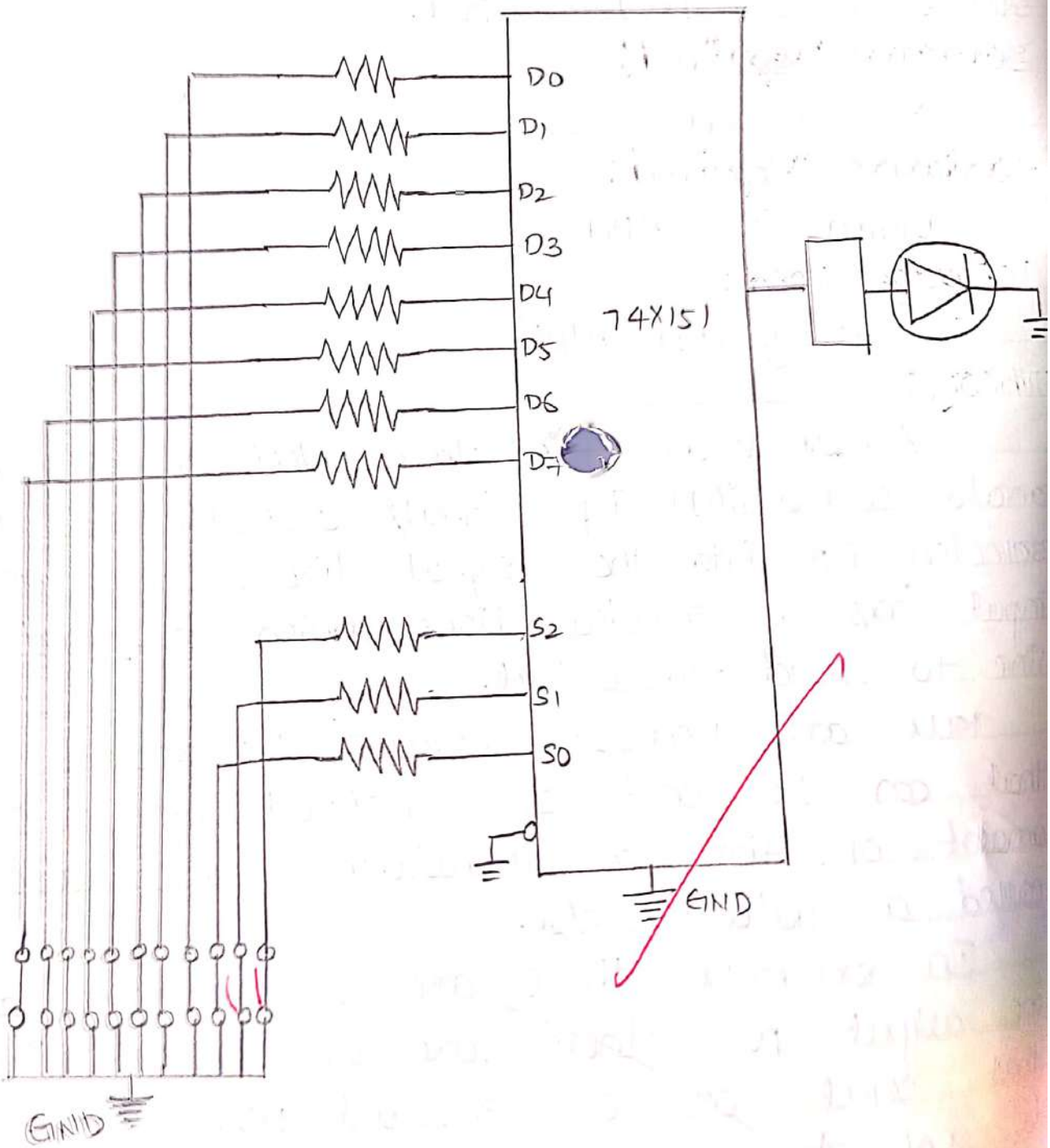
Theory:

A mux is a device that select one of several analog (or) digital i/p signals and forwards the selected i/p into the signal line. A mux of 2n input has n selected lines which are used input line to send to output.

Mux are used to increase the amount of data that can be sent over network with a certain amount of time & bandwidth. A mux also called a data selector.

In 8x1 mux there are 8 input ports and one output port there are also 3 digit input that select one of 8 input port signals to be sent to the output, the particular one selecting depending binary code at 3 select input mux come in the both the purely digital forms and the analog forms.

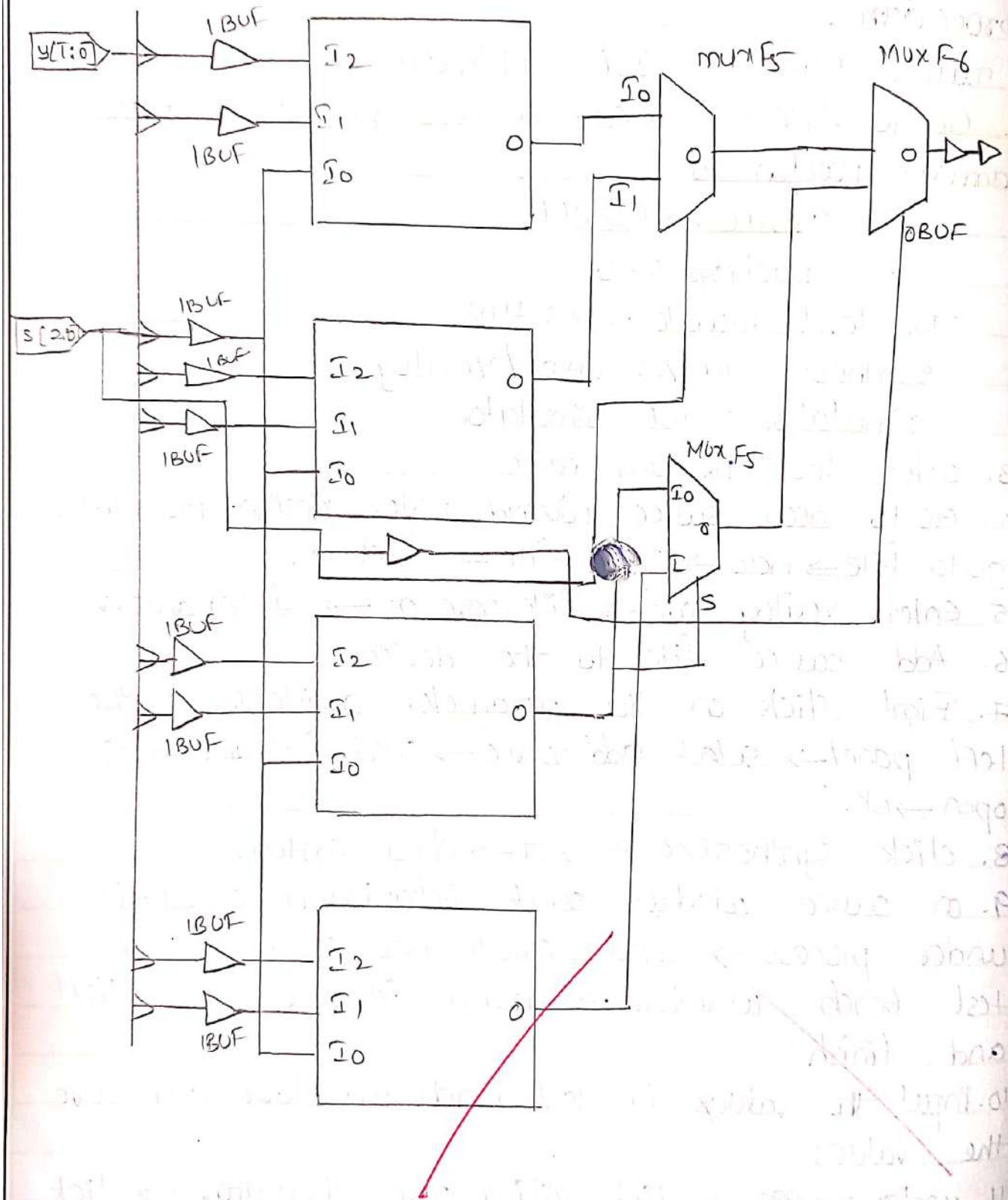
Equivalent circuit for 8x1 mux - 74151:



procedure:

1. Double click on ISE software
2. Go to file → select a new project navigator → Family: spartan 3S
   Device: XC3100E
   package :-5
   Top level module type: HDL
   synthesis tools: VHDL/verilog
   simulator : ISE simulator
3. enter the program name.
4. Go to new source wizard enter design program, goto file → new → T[redacted] file → click ok.
5. Enter verilog code → file save as → filename.v
6. Add source file to the device.
7. Right click on the parameter available at the left panel → select add source → select filename → click open → ok.
8. click synthesize → XST → check syntax.
9. in source window select behavioural simulation under process → select create new source → select test bench waveform → assign filename → click Next and finish
10. Input the values in test bench waveform and save the values.
11. under process → click xilinx ISE simulator → click generate Expect simulation → click yes
12. expected output waveform is processed and observe the o/p values.

Technological schematic:



13. click on source window select synthesis / implementation

14. click under process → click Technological schematic.

verilog code:

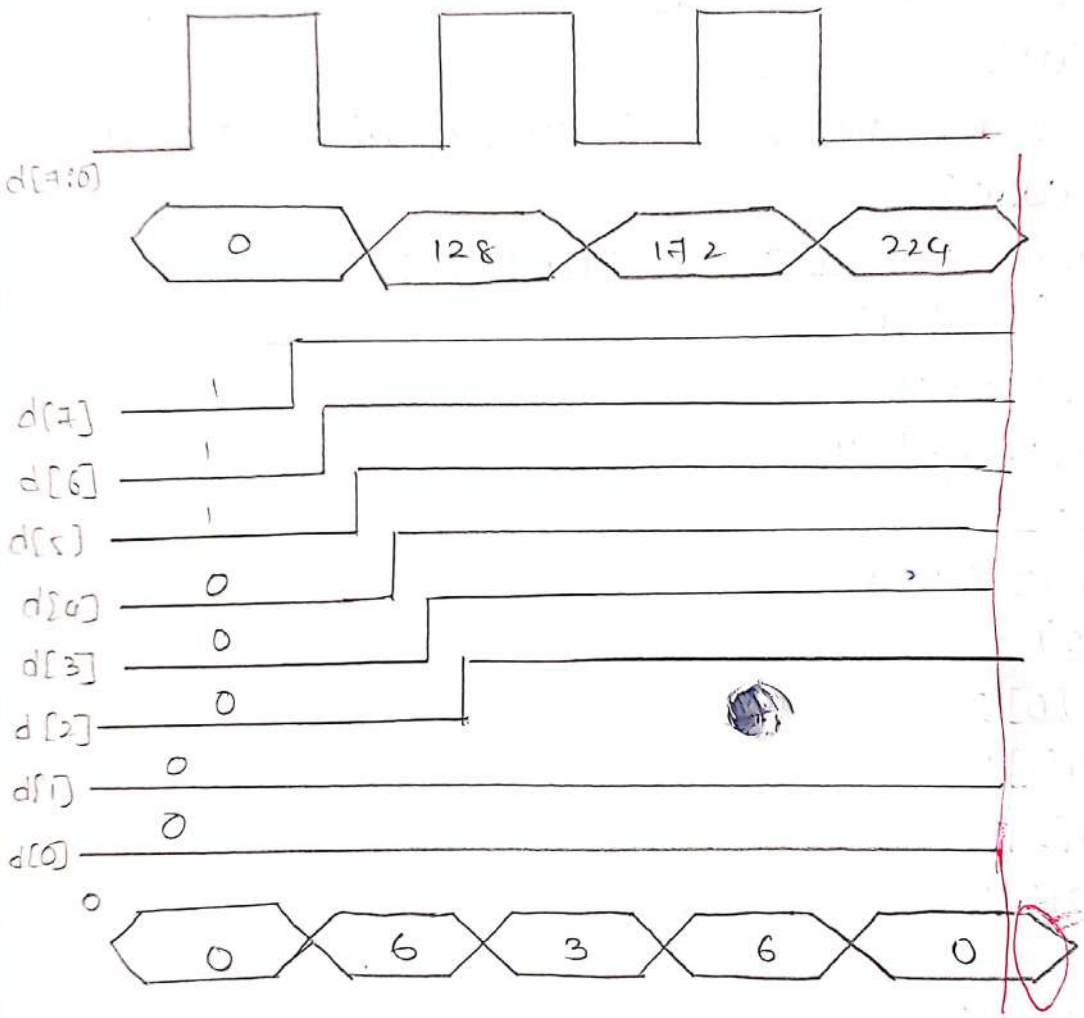
```
Module multiplex 8x1 (a,s,y);
input [7:0] a;
input [2:0] s;
input/output y;
reg y;
always @ (a,s)
case (s)
0: y = a[0];
1: y = a[1];
2: y = a[2];
3: y = a[3];
4: y = a[4];
5: y = a[5];
6: y = a[6];
7: y = a[7];
default: y = 1'bx;
end case
   end module.
```

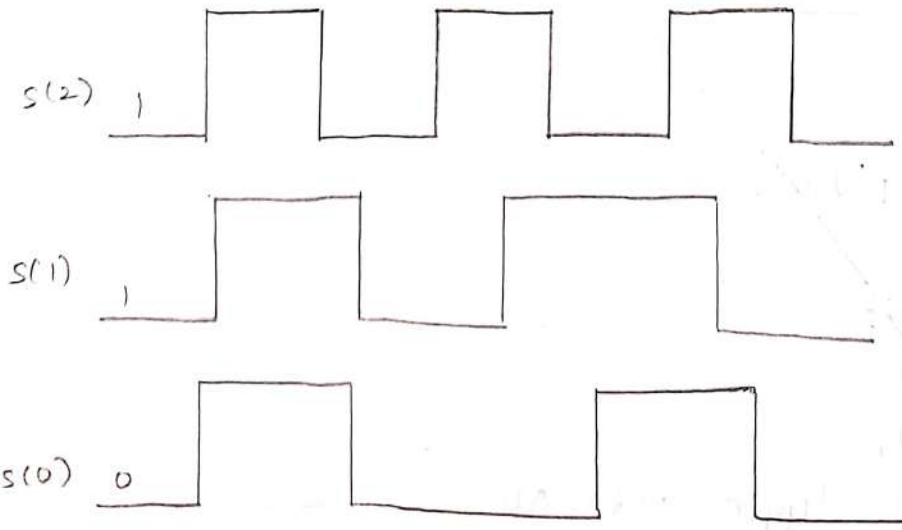
VHDL code:

```
library IEEE
USE IEEE. STD_logic_1164. All;
USE. IEEE. STD_logic_ARITH. All
USE. IEEE. STD_logic_unsigned. All
```

8)

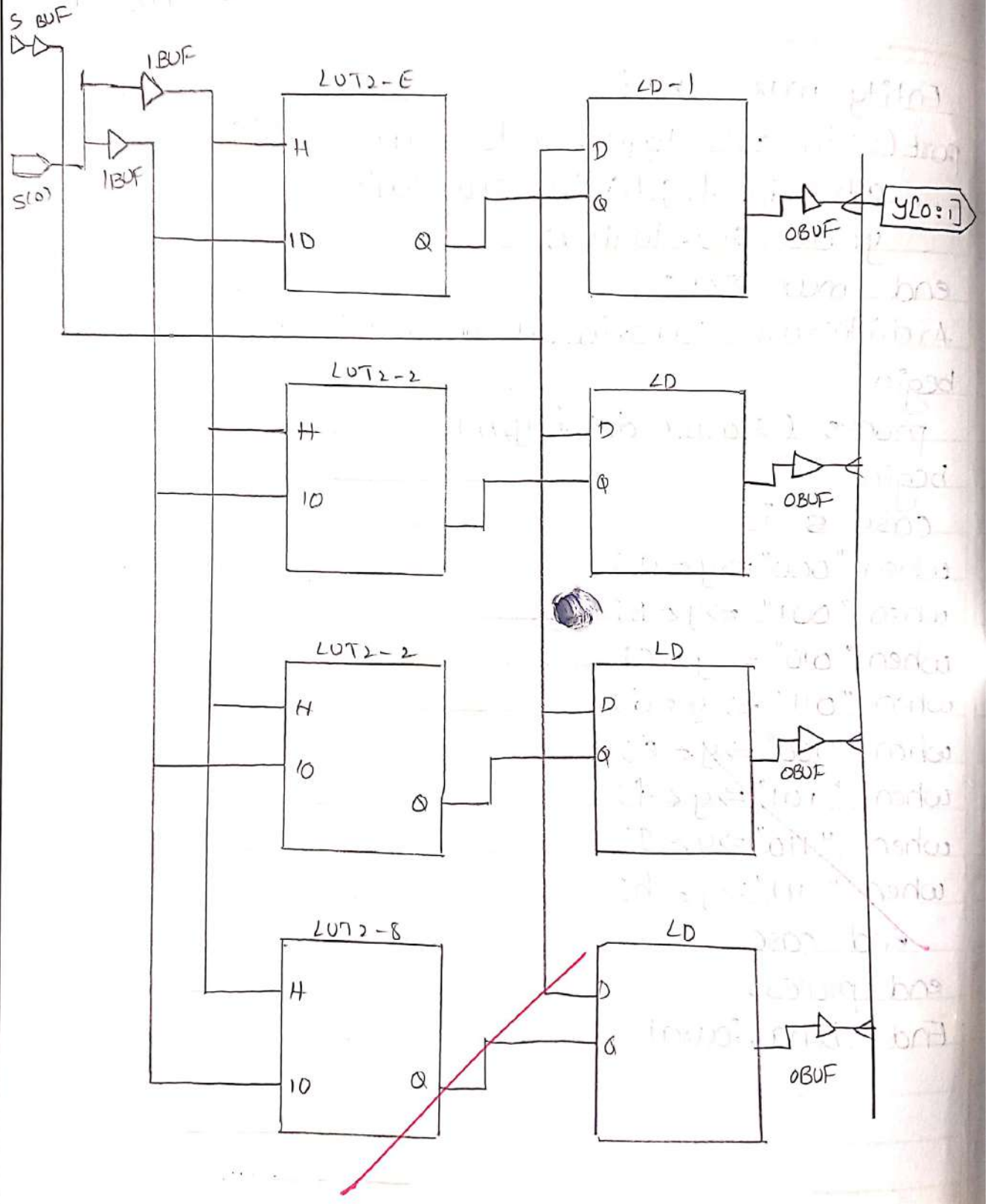
Waveforms:





```
Entity mux 8x1 is
port(s: in STD_logic_vector [2 down to 0];
     a,b,c,d,e,f,g,h: in STD_logic;
     y: out STD_logic);
end mux 8x1;
Architecture Behavioral of mux 8x1 is
begin
process (s,a,b,c,d,e,f,g,h)
begin
case s is
when "000" => y <= a;
when "001" => y <= b;
when "010" => y <= c;
when "011" => y <= d;
when "100" => y <= e;
when "101" => y <= f;
when "110" => y <= g;
when "111" => y <= h;
end case
end process
End behavioral
```

Technological schematic:



3(b) 2x4 Demultiplexer

Page No. 15
Date 9/8/19

Aim:-

To write a verilog source code for 2x4 Demultiplex 74155 and generate RTL schematic technology map, simulation and synthesis report.

Software and Hardware Required:

Hardware : Digital SC-74155

Software : xilinx 8.1

languages used:

verilog, VHDL.

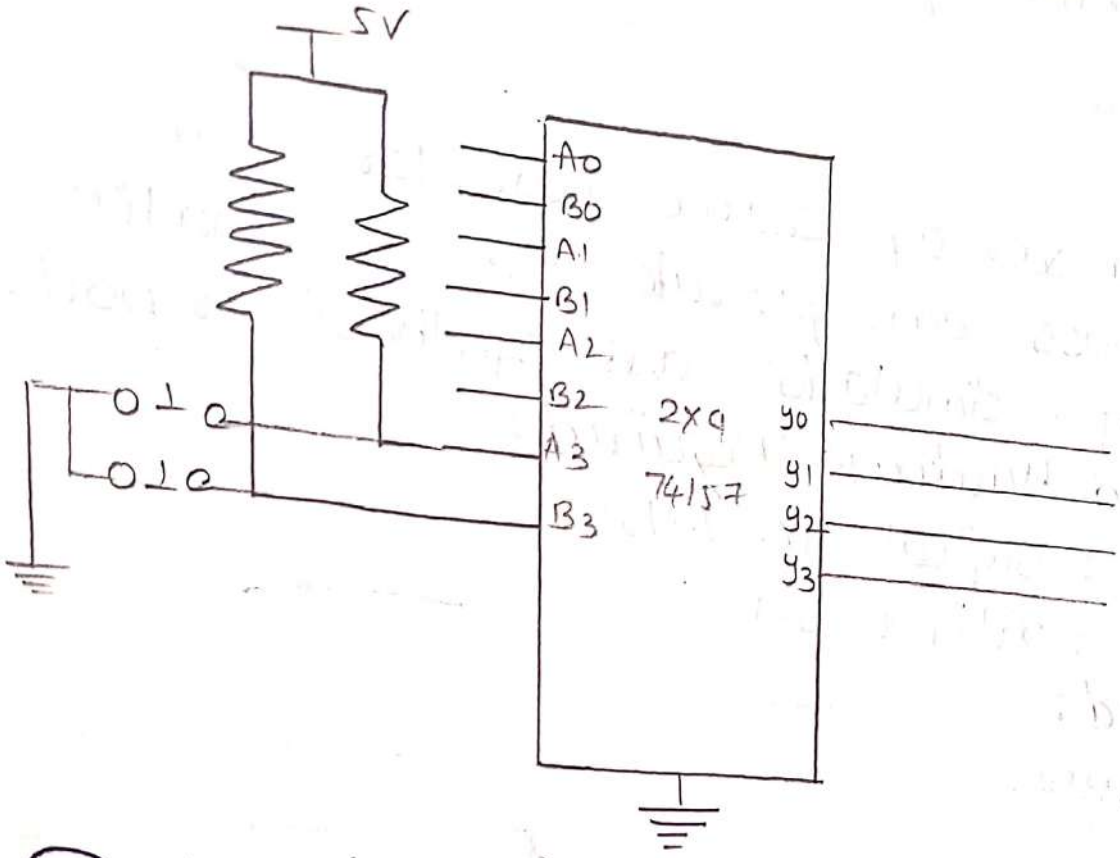
Theory:

A Demultiplexer is a device that takes a single input line and routes it to be one of several digital output lines. A demultiplexer of 2^n outputs has n select lines, which are used to select which output line to send the input. A Demultiplexer is also called as a Data Distributor.

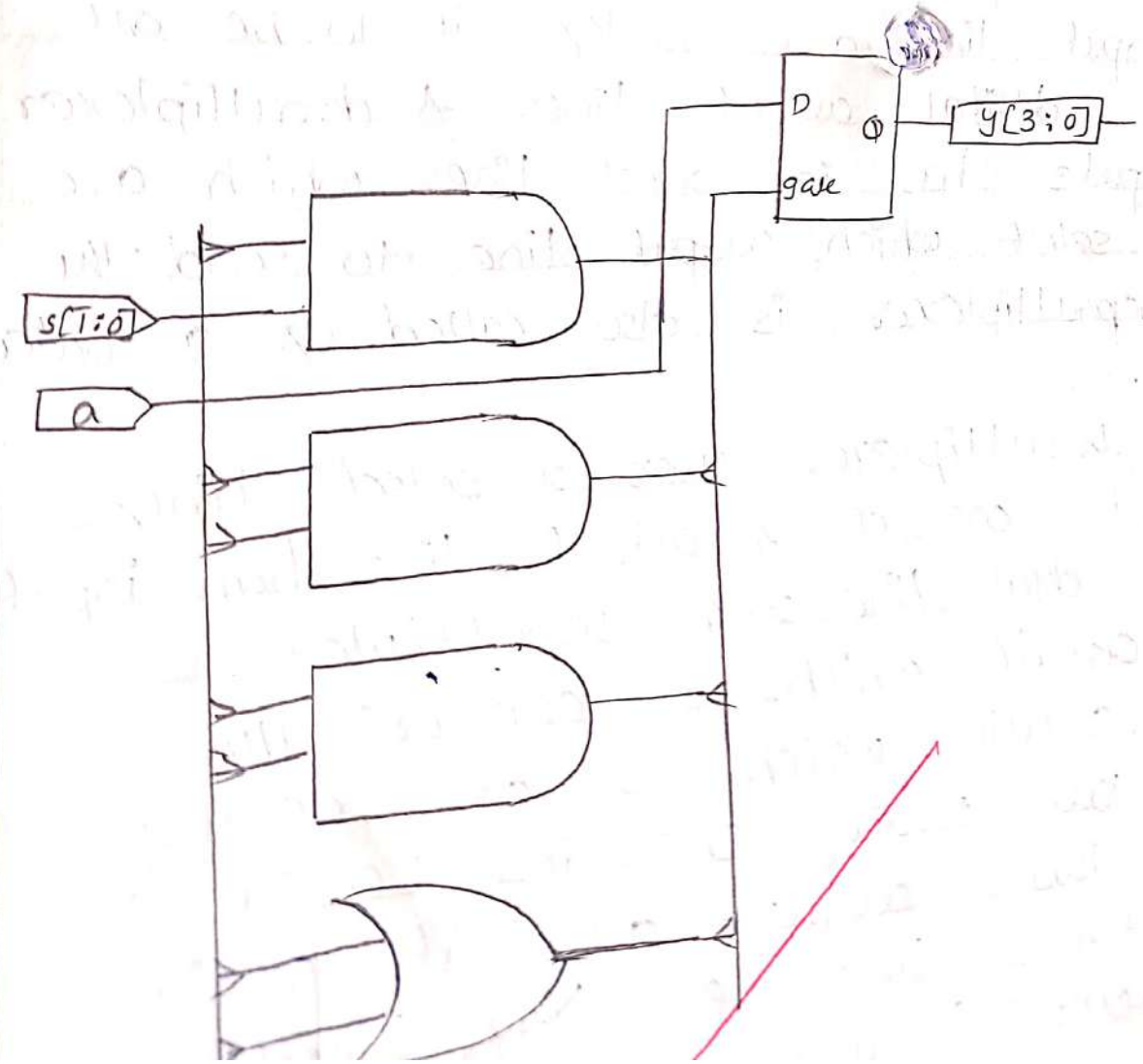
A 1x4 Demultiplexer uses 2 select lines to find out one of 4 output line from input. 74155 - the dual line 2x4 Demultiplexer integrated circuit which is commercially integrated circuit which is commercially used and low power consuming IC - It is an active low output. also it consists of an active low enabling input.

Demultiplexer is used to connect a single source to multiple destinations. The main

Hardware circuit:



RTL- schematic:



Demultiplexer is used to convert a single source to multiple destinations. The main application area of demux is communication system where multiplexers are used in reconstruction of parallel data and ALU circuits. It makes the transmission easier and also used in Arithmetic and logic unit.

Verilog source code:

```
Module demux (a,s,y);
input a;
input [1:0] s;
output [3:0] y;
reg [3:0] y;
always @(a,s)
case (s)
a: y[0] = a;
a: y[0] = a;
2: y[2] = a;
End case
   End module.
```

VHDL code:

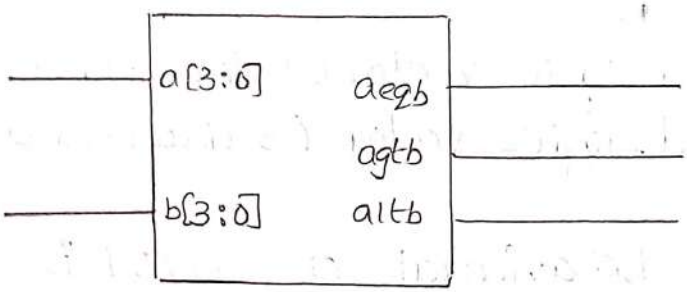
```
library IEEE;
use IEEE. STD-logic_1164. All;
use IEEE. STD-logic_ARITH. All
use IEEE. STD-logic_unsigned. All;
```

```
Entity 74157 is
port( a: in std_logic_vector(1 down to a);
      s: in std_logic_vector(3 down to a);
End 74157;
Architecture behavioural of 74157 is
begin
  process(a,s)
case 's' is
when "00" => y(0) <= a;
when "01" => y(1) <= a;
when "10" => y(2) <= a;
when others => y(3) <= a;
End case
  End process
    End behavioural;
```

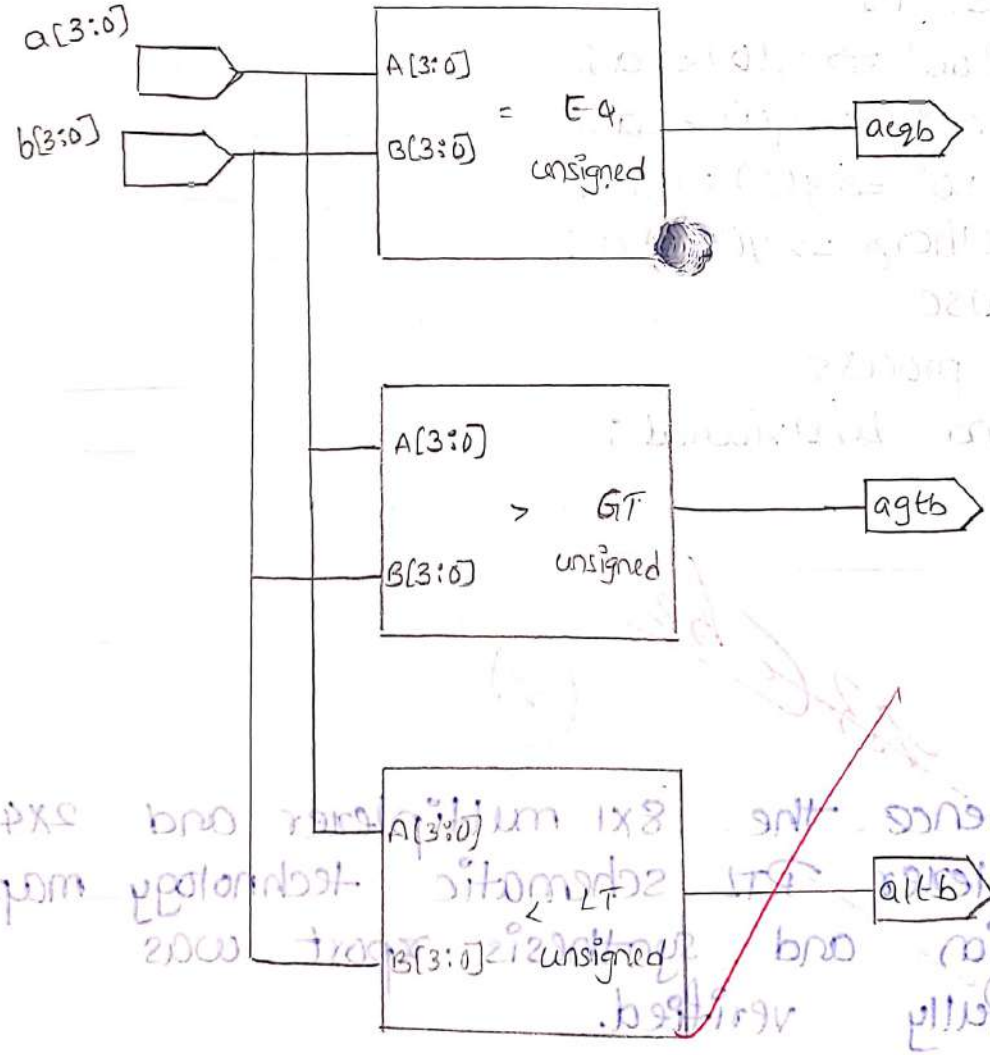
Result:

Hence the 8x1 multiplexer and 2x4 Demultiplexer RTL schematic technology map, simulation and synthesis report was successfully verified.

RTL - schematic (block diagram) :-



RTL - schematic :-



4. 4-bit comparator

Page No. 18
Date 16/8/19

Aim:

To write a VHDL source code for 4-bit comparator 74155 and generate RTL schematic technology map simulation and synthesis Report.

Software Required:

xilinx 8.1

Hardware Required:

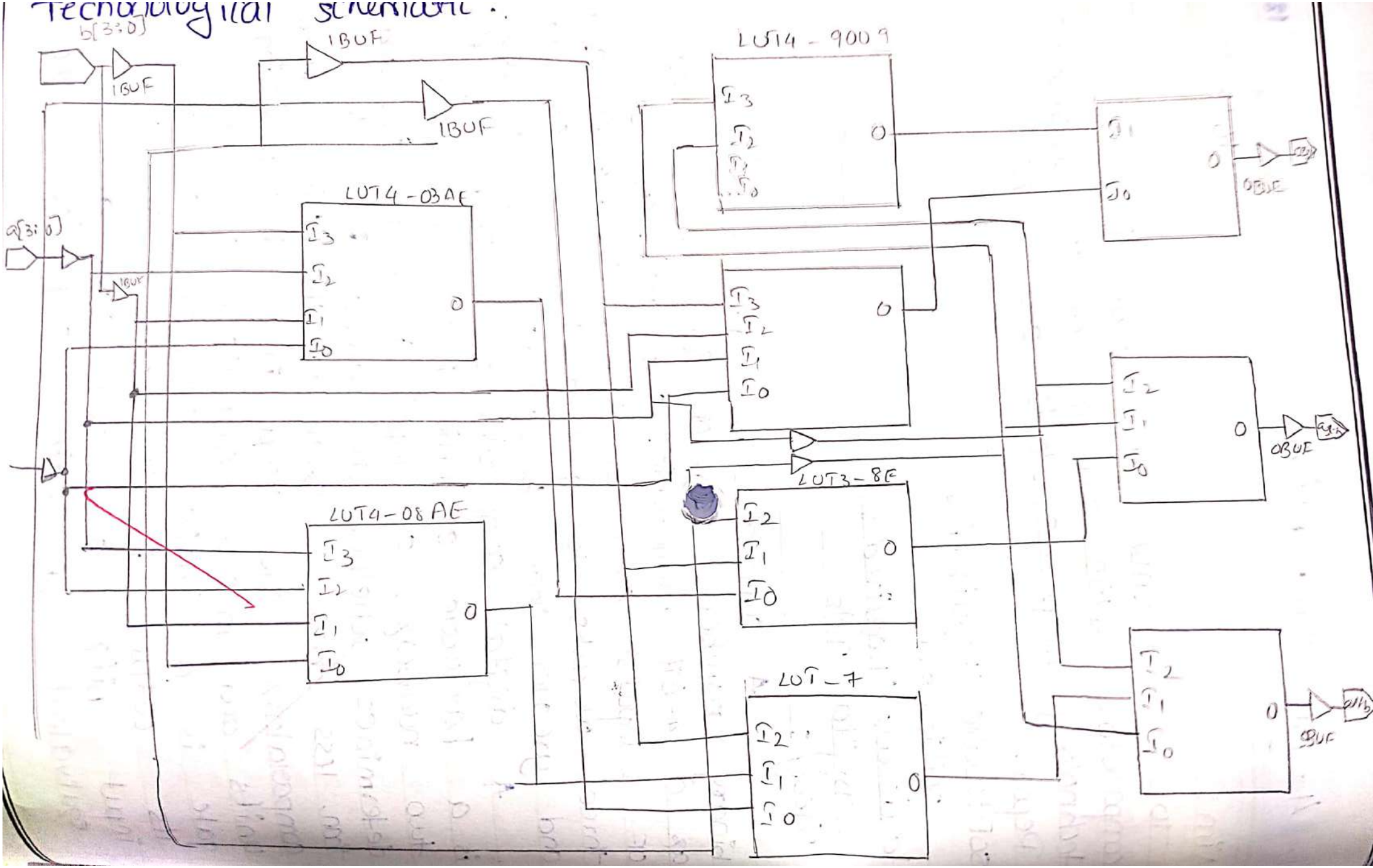
Digital Ics

Theory:

A comparator is used to compare two binary numbers each of four bits is called as a 4-bit magnitude comparator. It consists of 8 inputs each for two 4-bit numbers and three outputs to generate less than, equal to and greater than between two binary numbers.

A digital comparator or magnitude comparator is a hardware electronic device that takes two numbers as input in binary form and determines whether one number is greater than, less than or equal to the other number comparators are used in central processing units and micro controllers. An x-OR gate is a basic comparator because its output is "1" only if its two input bits are equal. The analog equivalent of digital comparator is the

Technology RTL schematic :



voltage comparator many controllers have analog comparators on some of their inputs that can be read or trigger an interrupt.

procedure:-

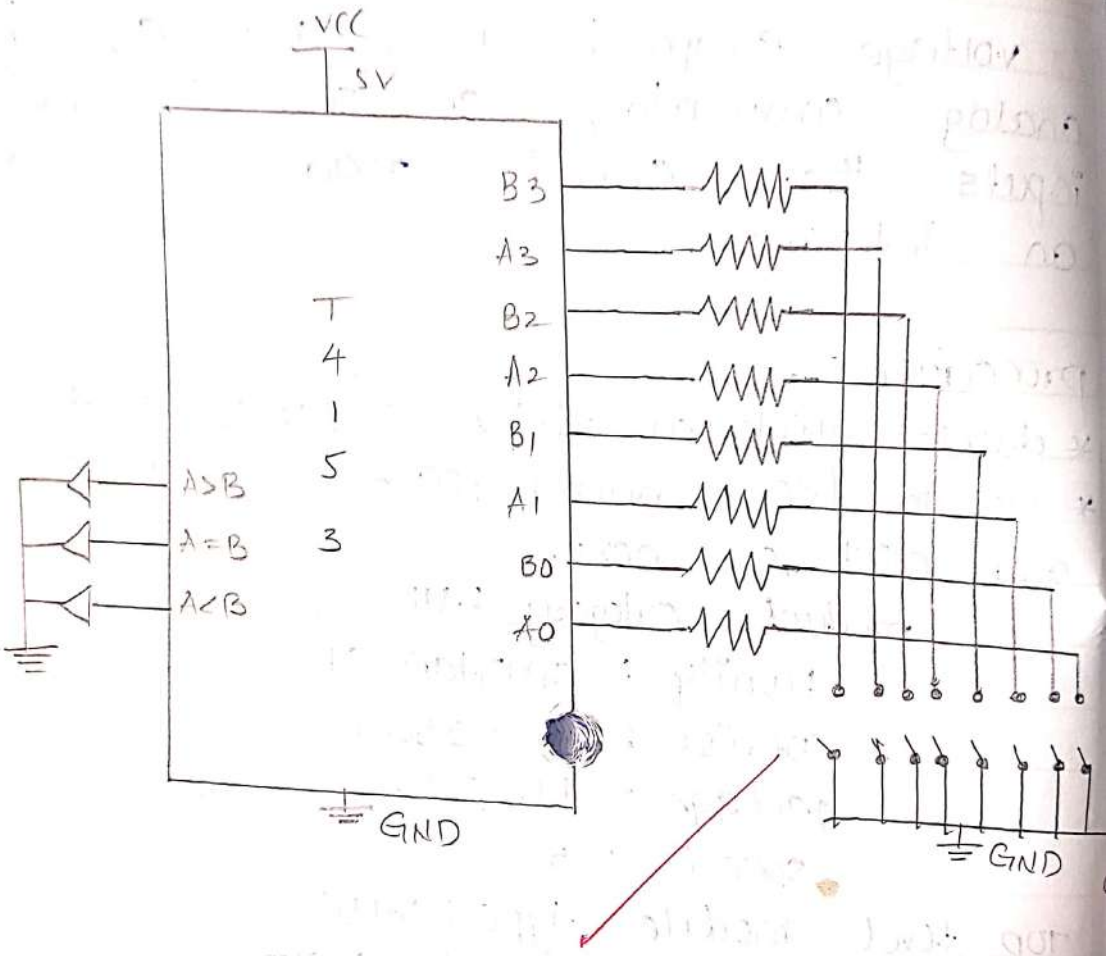
* double click on xilinx 8.1/12.3 software
* Go to file → new project → enter project name and click on next.
  - product category : All
  - Family : spartan 3E
  - Device : xc3 5500e
  - package : FG 326
  - speed : 5
- Top level module type : VHDL
- synthesis : (VHDL or verilog) XST
- simulator : ISE simulator (VHDL or verilog)

* click on next and finish
* Go to file → new file, text file → click ok.
* Enter verilog code → file save as → filename.v
* Add source file to the device.
* Right click on the parameter available at the left panel → select and source → select file name → click open → ok
* click synthesis → xst → check syntax
* one source window select behavioral.

simulation under process → select creat new folder →

P1

Hardware circuit:



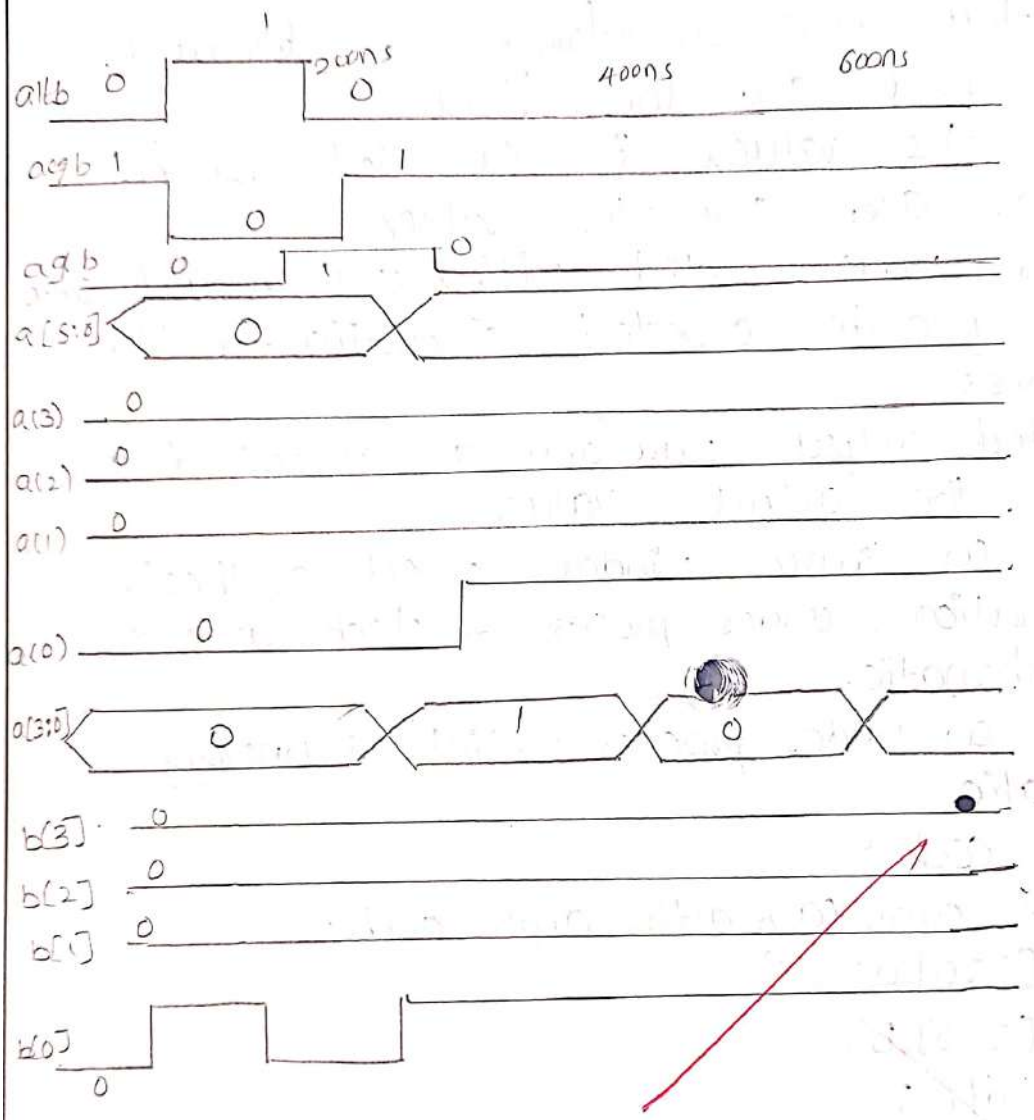
select test bench waveform → sign file name.
* click next and the finish.
* Input the values in the test bench wave-form and save the values.
* under process → click xilinx ISE modulator → click generate expected simulation → click yes → yes.
* expected output waveform as processed and observe the output values.
* click on source window select synthesis implementation under process → click on view RTL - schematic.
* click on under process → click technology schematic.

verilog code:

```
Module comp (a,b, altb, aeqb, agtb;
input [3:0] a;
input [3:0] b;
output altb;
output aeqb;
output agtb;
output altb;
assign altb = (a<b);
assign aeqb = (a==b);
assign agtb = (a>b);
```

02

waveforms:



VHDL code :-

```
library IEEE;
use IEEE.STD_logic_1164.All;
use IEEE.STD_logic_Arith.All;
use IEEE.STD_logic_unsigned.All;
  entity comparator is
port(b: in std_logic;
     clk: in std_logic;
      q: out std_logic;
    gbar: out std_logic;
end comparator;
p(a,b): in s'
Architecture arc_comp of comp is
begin
      process'(a,b)
begin
   if a>b then
   agtb <= '1';
   altb <= '0';
   aeqb <= '0';
els if a<b then
   altb <= '1';
   agtb <= '0';
   aeqb <= '0';
els if a=b then
   altb <= '0';
```

Truth table :

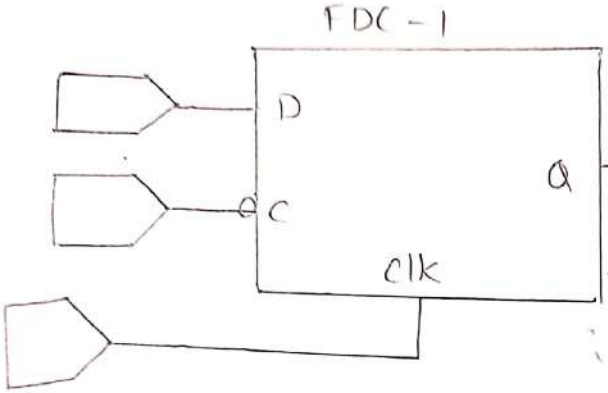
input				output		
A1	A0	B1	B0	A>B	A=B	A<B
0	0	0	0	0	1	0
0	0	0	1	0	0	1
0	0	1	0	0	0	1
0	0	1	1	0	0	1
0	1	0	0	1	0	0
0	1	0	1	0	1	0
0	1	1	0	0	0	1
0	1	1	1	0	0	1
1	0	0	0	1	0	0
1	0	0	1	1	0	0
1	0	1	0	0	1	0
1	0	1	1	0	0	1
1	1	0	0	1	0	0
1	1	0	1	1	0	0
1	1	1	0	1	0	0
1	1	1	1	1	1	0

```
agtb <= '0';
aeqb <= '1';
end if;
    end process;
end arch_comp;
```

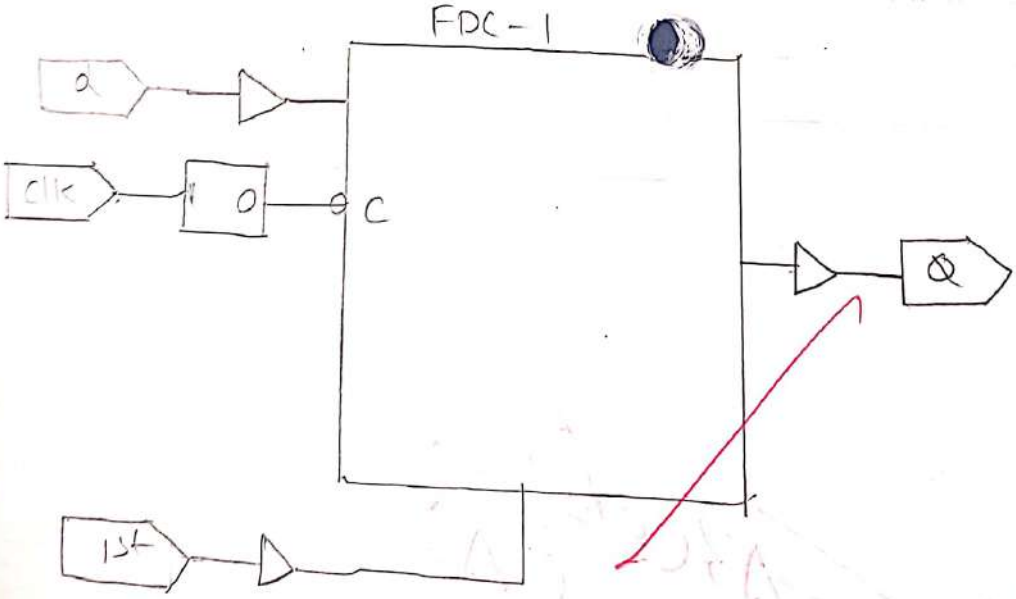
Result:

Hence the VHDL code For 4-bit comparator 74155 and generate RTL schematic technology map simulation and synthesis report was successfully verified.

RTL- schematic diagram:



Technological schematic :-



Hence the VHDL code for 4-bit comparator is simulated and synthesis report and technology schematic generated successfully verified.

5. D- Flip Flop - 7474

Page No. 23
Date 30/8/19

Aim:

To write verilog source code for a D FlipFlop- 7474 and generate RTL, schematic, technology map, simulation synthesis report.

Software Required:

Xilinx ISE 8.11

Hardware Required:

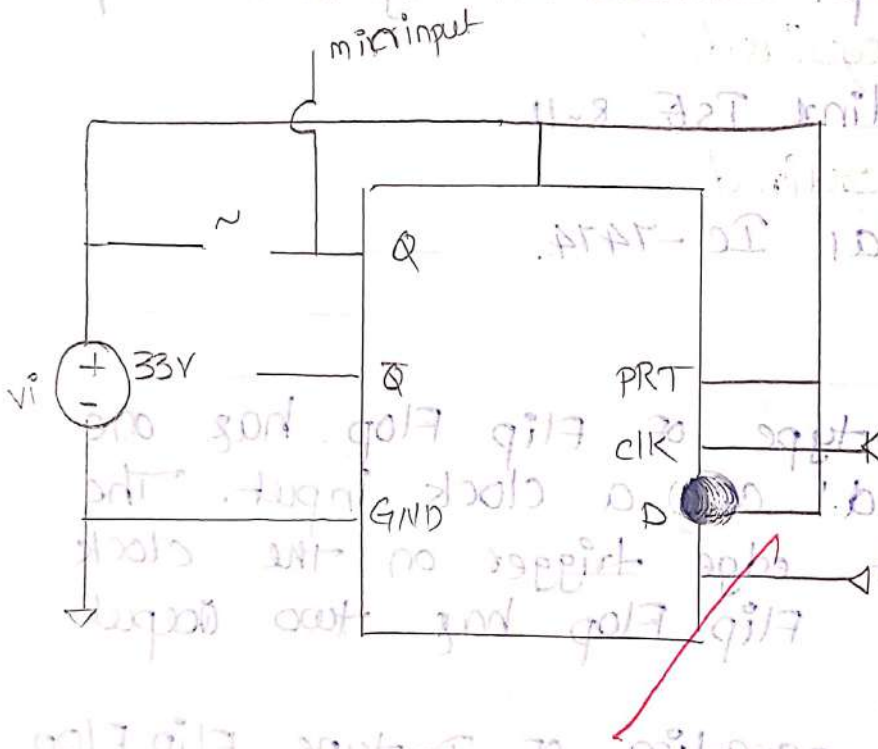
Digital IC -7474.

Theory:

The d-type of Flip Flop has one data input 'd' and a clock input. The clock circuit edge trigger on the clock input. The Flip Flop has two output Q and Q'.

The operation of D-type Flip Flop is as follows, any input operating at the input 'd' will be produced at the output Q in the time '+1' edge if in the present state we have D = anything and Q=0. D Flip Flops are used as a part of memory storage elements and data processor as well. D Flip Flop can built using NAND gate or with NOR gate. The major applications of D Flip Flop are to introduce delay in timing circuit as a buffer sampling data at specific intervals.

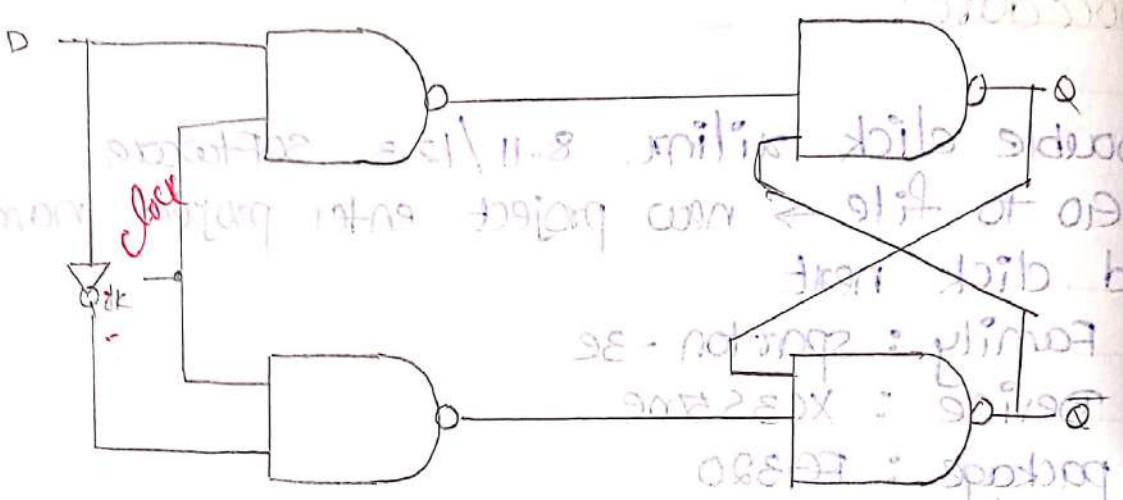
Hardware circuit:-



procedure:

* Double click xilinx 8.11/12.3 software
* Go to file → new project enter project name and click next
  Family : spartan-3e
  Device : XC3S500e
  package : FG320
  speed : 5
Top-level module type : VHDL
  synthesis tools : (VHDL/verilog)
    simulator : ISE XE-VHDL
* click next and finish.
* Go to file → new → text file → click ok.
* Enter verilog code → file save as → file name.v (For verilog) and For VHDL.
* Add source file to the device.
* Right click on the parameter available at the left panel.
* select add source → select filename.v → click open → ok.
* click synthesize → XST → check syntax
* on source window, select behavioural simulation under process → select create new source → select test bench waveform → assign file name → click next and finish

logic diagram of D-FlipFlop:



Truth table:

clk	D	Q	$\bar{Q}$
X	X	1	0
X	X	0	1
X	X	X	X
X	X	1	0
1	0	0	1
0	X	q_0	$\bar{q}_0$

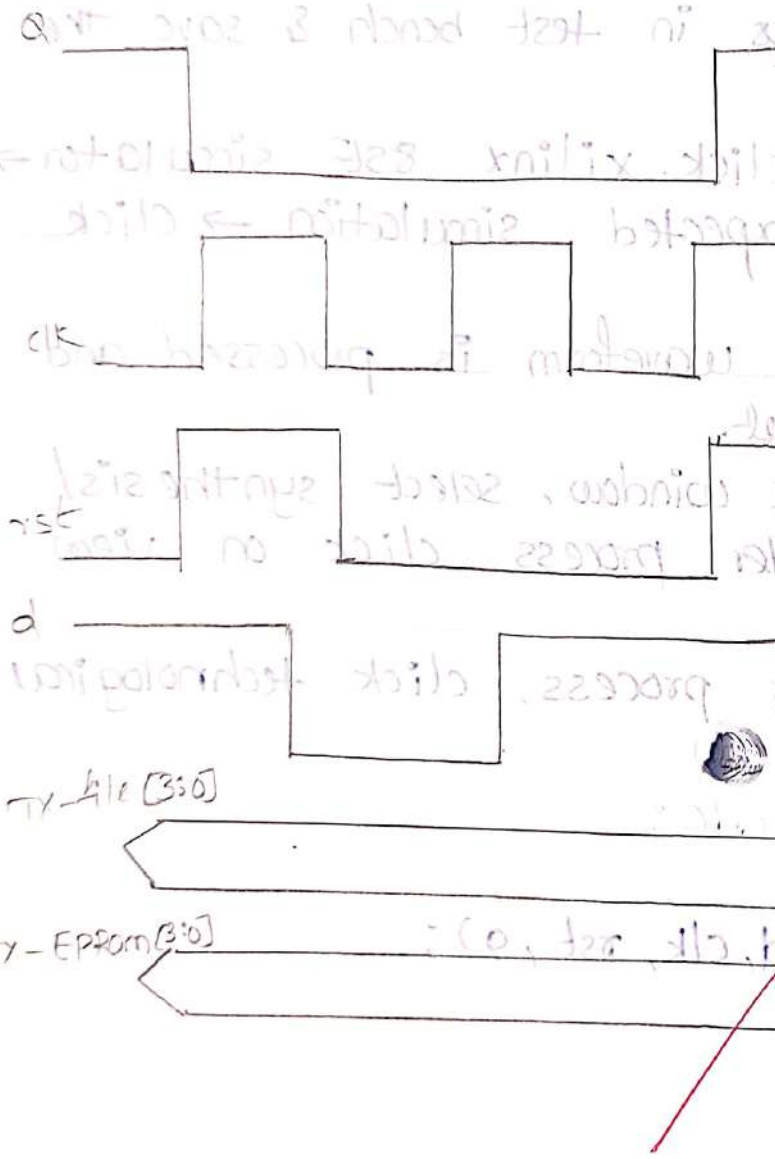
* input the values in test bench & save the values
* under process → click. xilinx ISE simulator → click generate expected simulation → click yes → yes.
* expected output waveform is processed and observe the output.
* click on source window, select synthesis/ implementation under process, click on view RTL schematic
* click on under process, click technological schematic.

Verilog source code:

```
Module diff(d, clk, rst, q);
input d;
input clk;
input rst;
output q;
reg q;
always @ (negedge clk or posedge clk)
if(rst)
   q = 1'b0;
else
   q=d;
end module;
```

25

simulation waveform:




```
Module
input d;
input clk;
input rst;
output q;
reg q;
always @ (negedge clk or posedge clk)
if(rst)
q = 1'b0;
else
q=d;
end module;
```

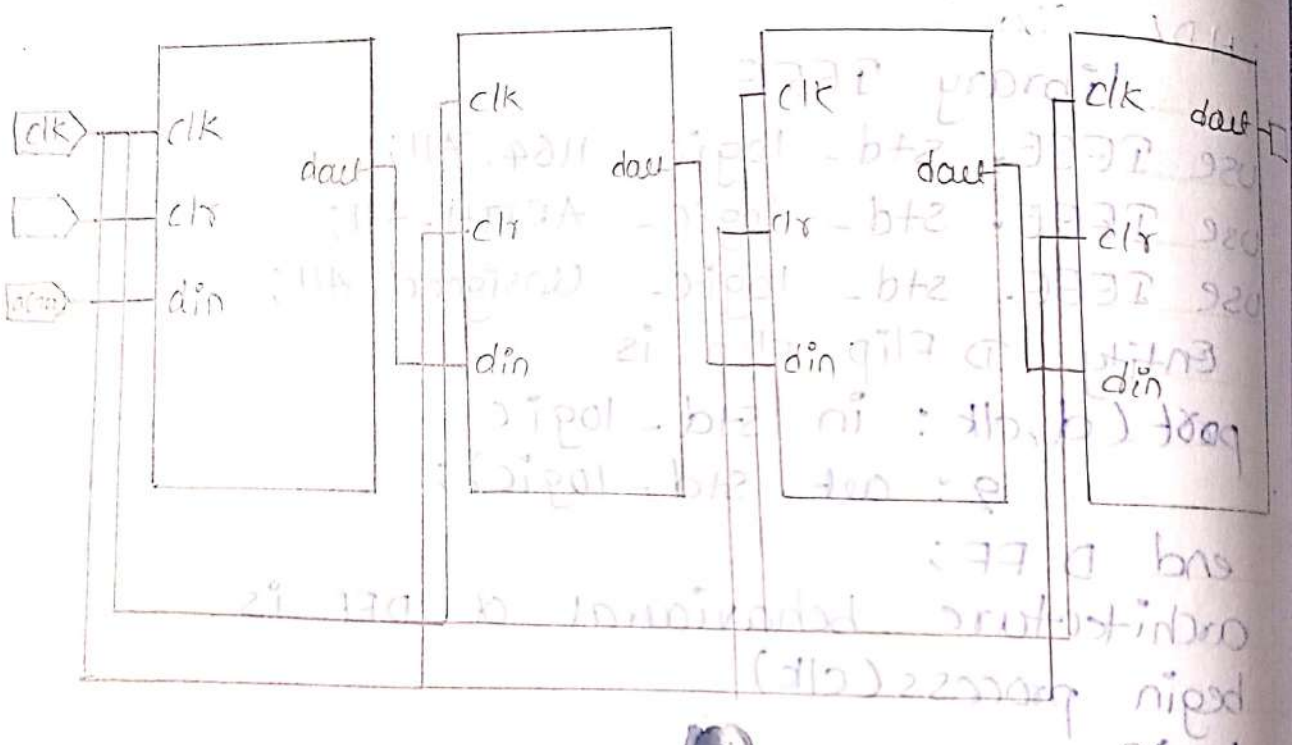
VHDL code:

```
library IEEE
use IEEE. std_logic_1164. All;
use IEEE. std_logic_ARITH. All;
use IEEE. std_logic_unsigned. All;
  Entity D Flip Flop is
port (d, clk : in std_logic
        q : out std_logic);
end D FF;
architecture behavioural of DFF is
begin process (clk)
begin
    if (clk='1' and clk 'event) then
      q <= d;
end if;
  end process;
end behavioural;
```

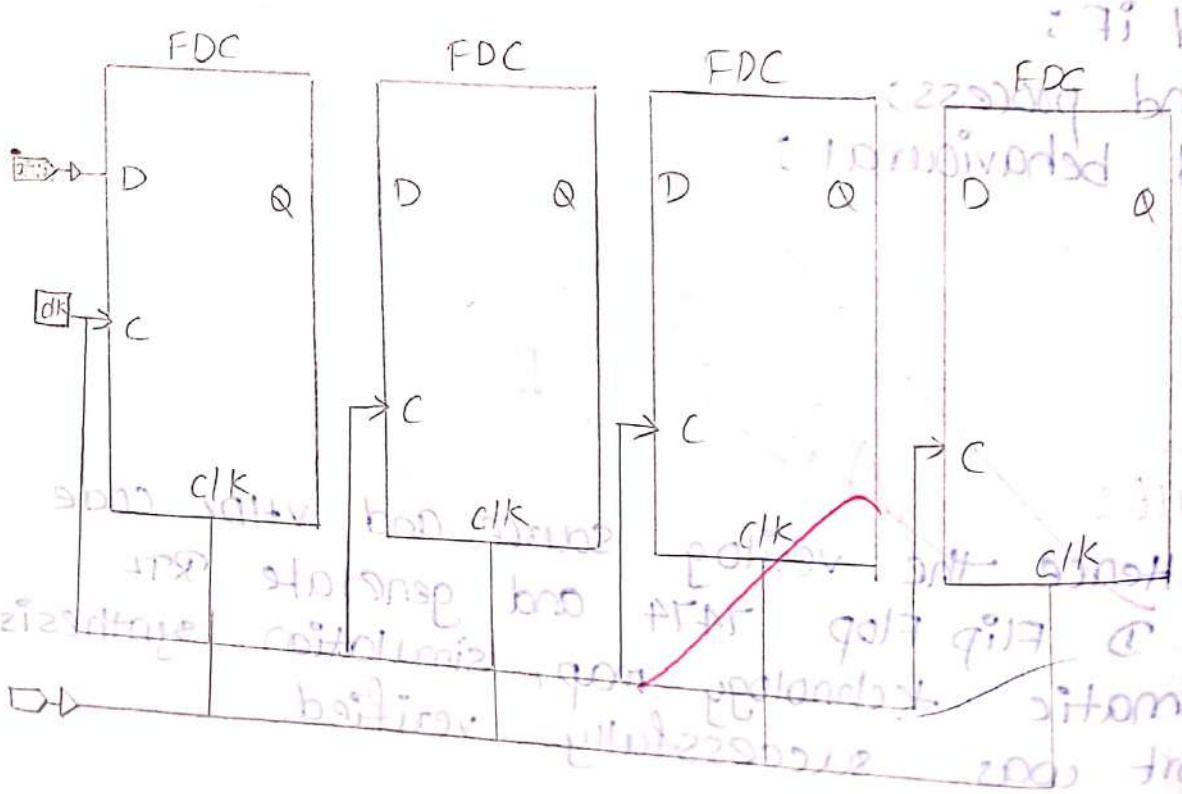
Result:

Hence the verilog source and VHDL code For D Flip Flop 7474 and generate RTL schematic technology map, simulation synthesis Report was successfully verified

RTL - schematic:



Technological schematic:



6. shift Registers

Aim:

To write a verilog source code for subject shift register 7495 and generate RTL schematic, technology map, simulation and synthesis report.

SOFtware Required:

Xilinx 8.11

Hardware Required:

Digital IC-7495

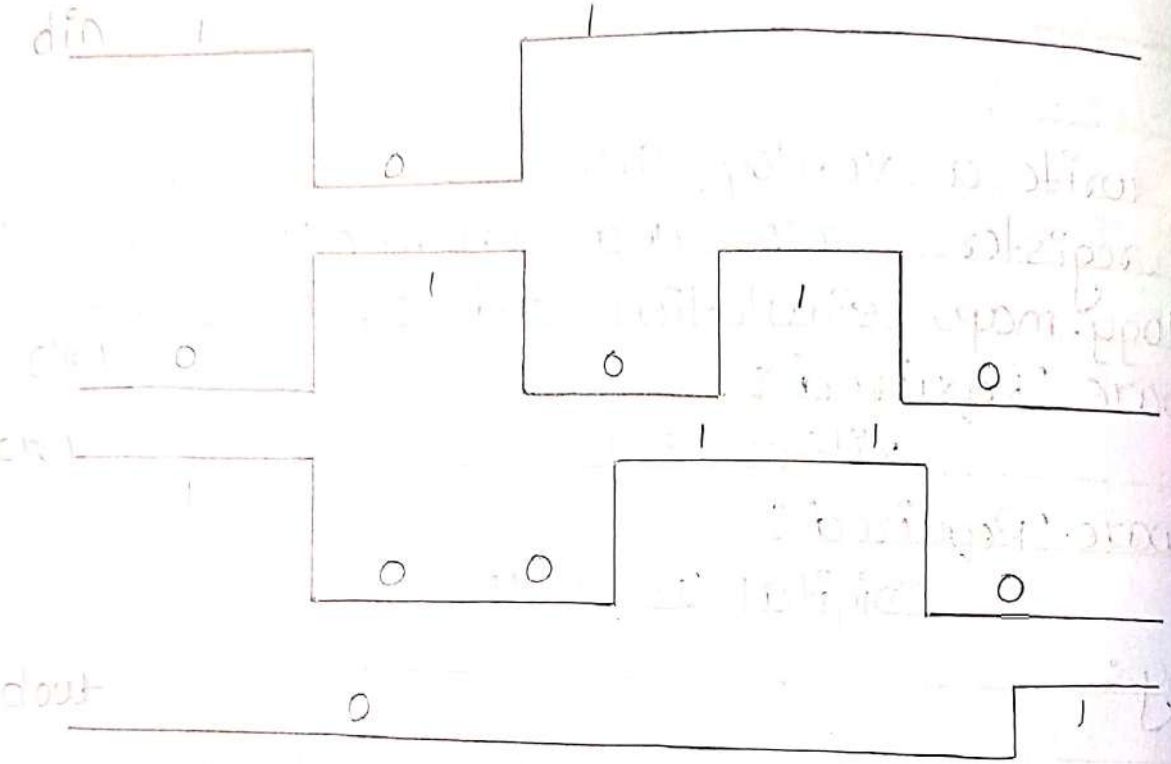
Theory:

The shift registers is cascade of FlipFlops sharing the same clock in which the output of each FlipFlop is connected to the data input of next Flip Flop in the circuit, resulting in a circuit that shifts by one position "bit Array" started in it. "shifting out" the last bit in the array at each transition of the clock input.

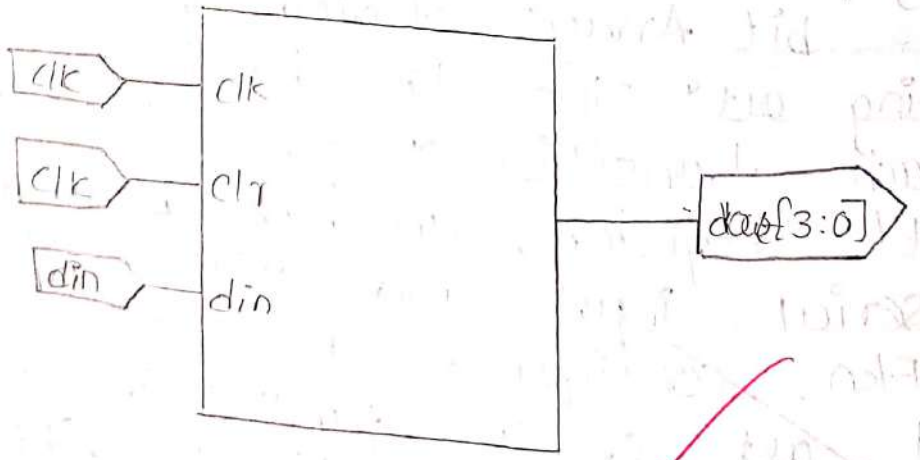
shift Register can have both parallel and serial input and outputs, there are often configured as serial in parallel out or as parallel in serial out. These are bidirectional shift register.

The data sting is presented at detain and shifted right one stage each time.

simulation waveform:



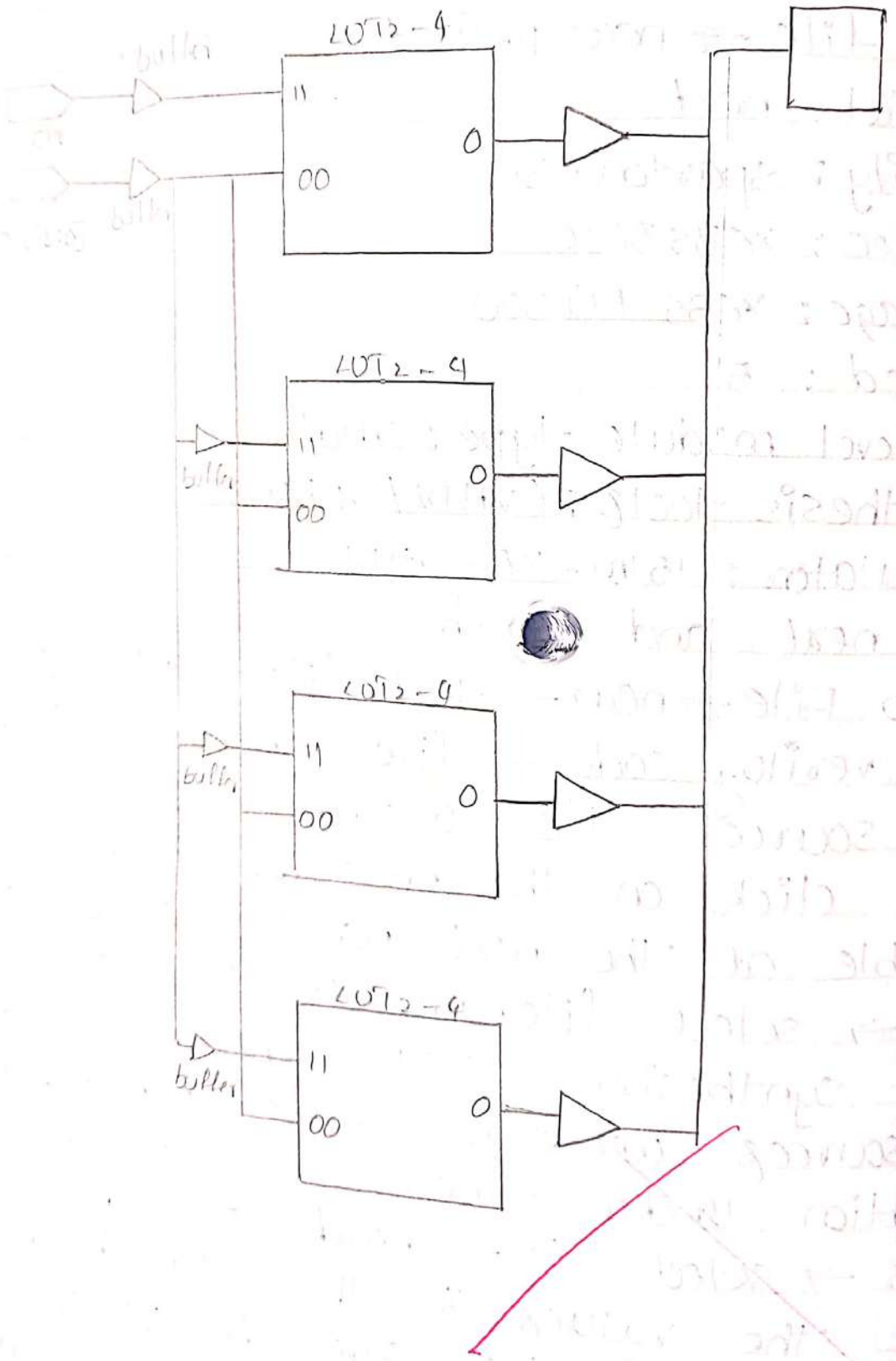
RTL-schematic:



procedure:

* Double click on xilinx 8.11/12.3 software.
* Go to file → new project → enter project name and click next
  Family: spartan-3e
  Device: XC3S500E
  package: XC3S FG320
  speed: 5
  Top level module type: vhdl
  synthesis tools: (VHDL/VERYLOG)
  simulator: ISIm-XE-VHDL.
* click next and finish.
* Go to File → new → text file → click ok.
* Enter verilog code → file save as → file name.v
* Add source file to the device.
* Right click on the processor parameter available at the next panel → select and source → select filename.v → click open → ok.
* click synthesize → XST → check syntax.
* on sources windows select behavioural simulation under process → select create new sources → select test bench waveform.
* Input the values in the test bench waveform and the save the values.
* under process → click on xilinx ISE simulator → click generated expected o/p → click yes → yes.

Techaological schematic:



* expected output wave-form is processed and observed the output values.
* click on sources window select synthesis/ implementation under process → click on new RTL schematic.
* click on under process → click on technology schematic.

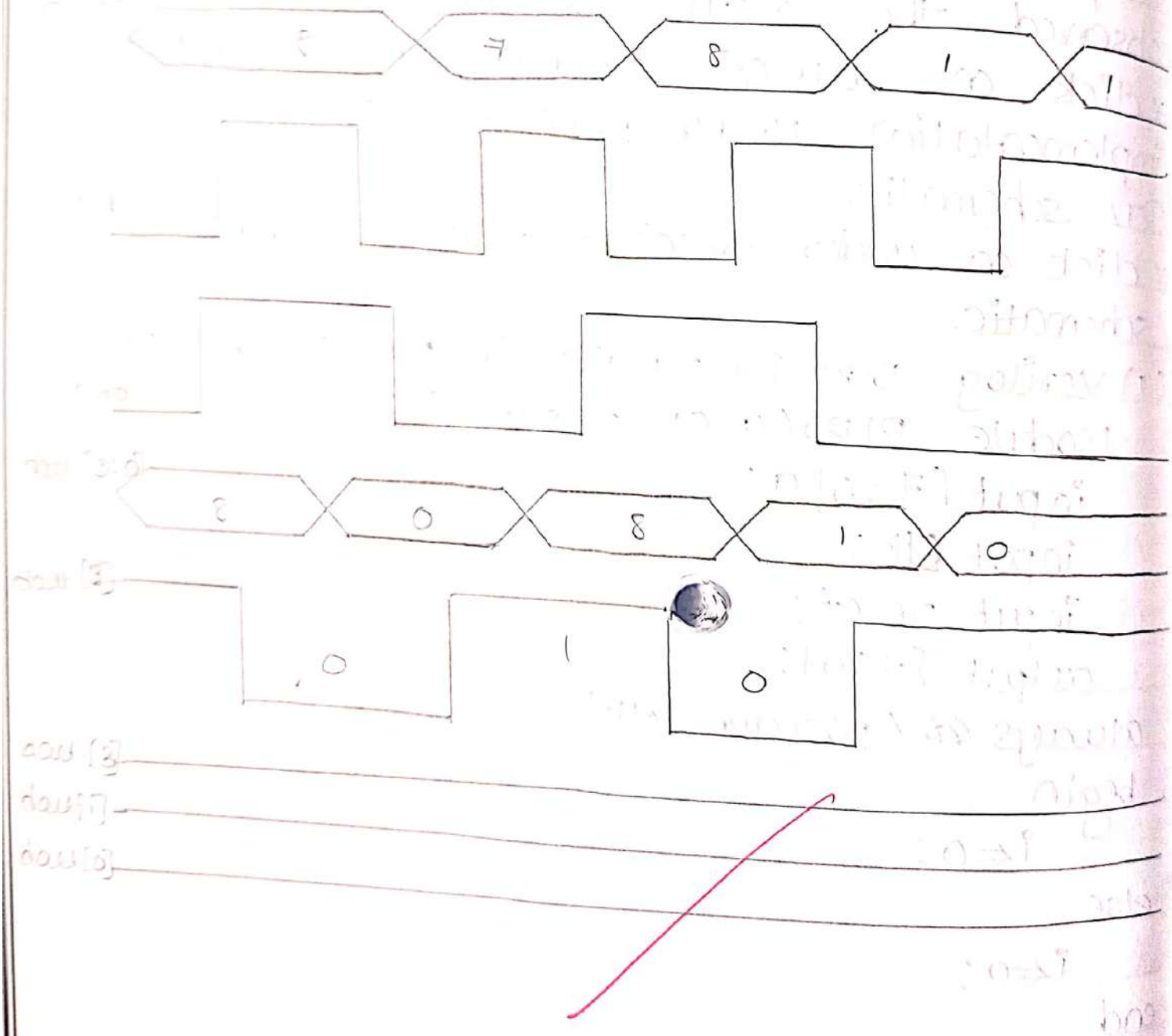
(a) Verilog code for parallel in parallel out program:

```
Module PIPO(a, clk, reset, i);
  input [7:0] a;
  input clk;
  input reset;
  output [7:0] i;
always @(posedge clk)
begin
    i<=0;
else
    i<=a;
end
  end module.
```

(b) verilog source code for serial in serial out:

```
Module lrshift( i, a, clk, rst);
 input a, clk, rst;
   output i;
 reg b, c, d, e, f, g, h, i;
always @(posedge clk)
 if(rst)
```

Simulation waveform:



```
begin
 b<=1'b0;
 c<=1'b0;
 d<=1'b0;
 e<=1'b0;
 f<=1'b0;
 g<=1'b0;
 h<=1'b0;
 i<=1'b0;
end
  else
begin
    b<=a;
    c<=b;
    d<=c;
    e<=d;
    f<=e;
    g<=f;
    h<=g;
    i<=h;
     end
end module;
```

Result:

Hence the verilog source code for shift register 7495 and generate RTL schematic, technology map, simulation and synthesis report was successfully verified.

Blinking LED With TIVA

Page No. 55
Date

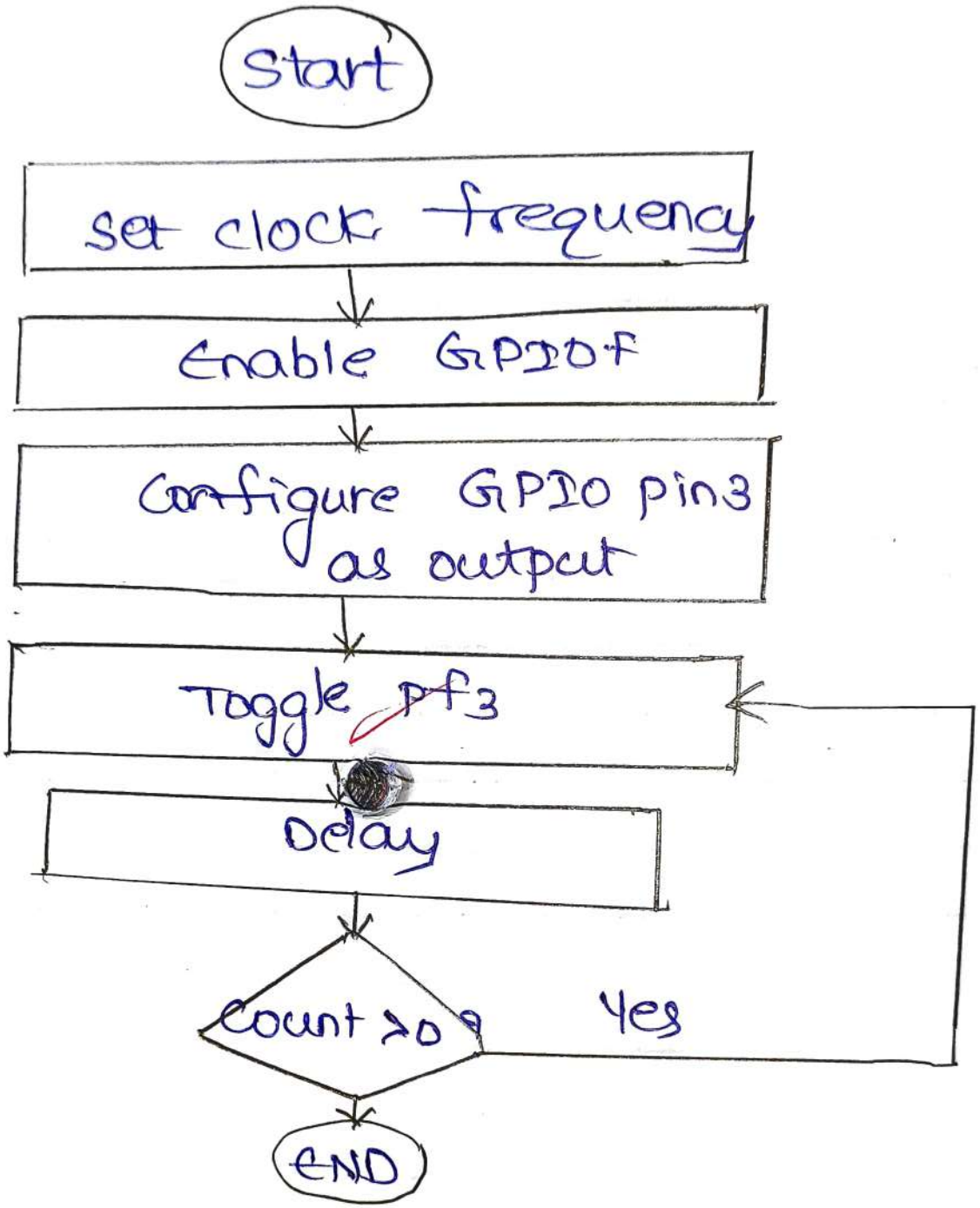
Aim:-
To blink the GREEN on-board using C language.

Software required:
Code Composer Studio.

Hardware required:
TM4C123GH6PM launched plugged into PC via USB.

Description
Tiva TM4C MCUs featured GPIO module is composed of six physical GPIO blocks, each corresponding to an individual GPIO port. (Port A, Port B, port C, port D, port E, port F)
The GPIO features are listed as the following
→ upto 43 GPIOs, depending on configuration
→ highly flexible pin muxing allows use as GPIO or one of several peripheral functions.
→ Bit masking in both read and write operations through address lines.
→ programmable control for GPIO interrupts interrupt generation masking, edge-triggered on rising, falling or both, level-sensitive on high or low values.

Flowchart:-


Start
Set clock frequency
Enable GPIOF
Configure GPIO pin3 as output
Toggle pf3
Delay
Count >0 ?
Yes
END

program:

```
#include <stdint.h>
#include <stdbool.h>
#include "inc/hw_types.h"
#include "inc/hw_memmap.h"
#include "driverlib/sysctl.h"
#include "driverlib/pin_map.h"
#include "driverlib/debug.h"
#include "driverlib/gpio.h"
int main (void)
{
SysCtlClockSet(SYSCTL_SYSDIV_5|SYSCTL_USE_
                PLL|SYSCTL_XTAL_16MHZ)
                SYSCTL_OSC_MAIN);
SYSCTL Peripheral Enable (SYSCTL_PERIPH_GPIOF);
GPIOpin Type GPIO Output (GPIO_PORTE_BASE,
                GPIO_PIN_1|GPIO_PIN_2|GPIO_PIN_3);
GPIO pin write( GPIO_PORTF_BASE: GPIO_PIN1|GPIO_PIN2
                |GPIO_PIN3, 0X08);
SysCtlDelay(20000000);
  while(1)
{
}
}
```

Result:

Hence blinking the green on-board LED (P1.23) are successfully completed using C program.

Interrupt programming with TIVA GPIO

Page No. 39
Date

Aim:
Togging of LED by configuring times interupt TIVA.

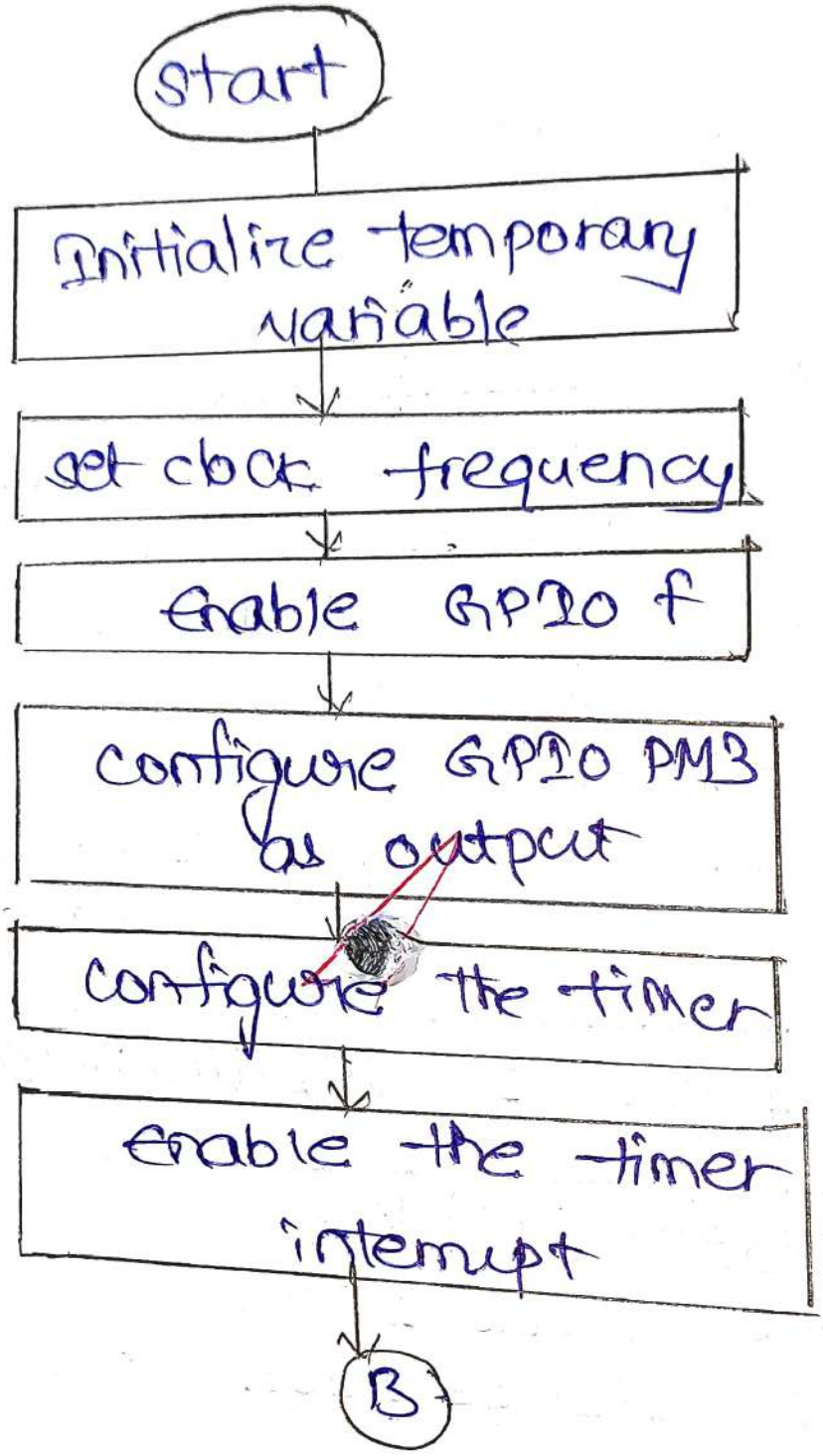
Software required:
code composer studio (ccs).

hardware required:
TM4C123GH6M launch pad plugged into PC via USB.

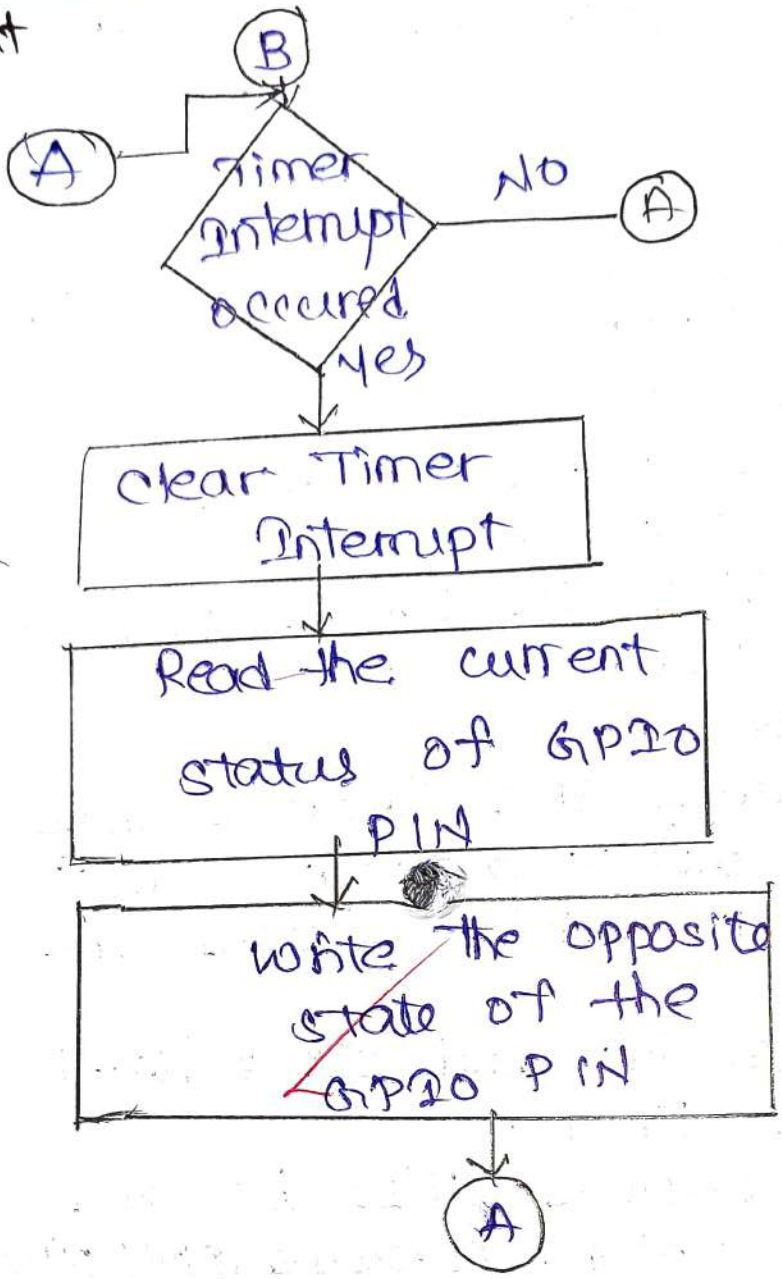
Description:
Configuring the timer to generate the interupt writer a timer interupt service. The timer module features are listed as following

→ six 16/32 bit and six 32/64-bit general purpose timers.
→ Twelve purpose 16/32 bit and twelve 32/64-bit capture/compare/PWM pins
→ Timer modes
  * One modes
  * periodic
  * Input edge count or time capture with 16-bit processor.
→ PWM generation (separated only
→ Real time clock (concatenated only
→ Count up (or) down.
→ simple PWM support for timer synchronization
→ Wsy-bars and stalling during debugging

Flowchart.


Start
Initialize temporary variable
set clock frequency
Enable GPIO f
configure GPIO PM3 as output
configure the timer
Enable the timer interrupt
B

flowchart


B
A
Timer Interupt occured
NO
A
yes
clear Timer Interupt
Read the current status of GPIO PIN
write the opposite state of the GPIO PIN
A

→ Many trigger ADC samples con DMA Transfer

Program:-

```
#include <stdio.h>
#include <stdlad.h>
#include "inc/tmc12.3ghgpm.hw"
#include "inc/hw-memap.h"
#include "inc/hw--types.h"
#include "driverlib/ syssH.h"
#include "driver lib/ interupt.h"
#include "driver lib/ gpio.h"
#include "driver lib/ timer.h"
int main (void)
{
unit 32-t ui32 period:
SYSCH clock set/ SYSCTL-SYSDIV-5/SYSCTL-USE
PLLVSYSCTI-XTAL-16MHZ /SYSCTL-OSC-MAN:
SYSCTL peripheral Enable /SYSCTL-PERIPH-GPIOE);
GPIO pin type GPIO output/ GPIO-PORTF-BASE-
GPIO-PIN-1 /GPIO-PIN-2 /GPIO-PIN-3);
SYSCTL peripheral Enable /SYSCTL-PERIPH-
                                TIMER N);
Timer configure /Timer0-Base, timer-CFG-
periodic;
Timer ui32 period = /SYSCTL clock sere) /100/2;
Time load set / TIMER0-BASS, TIMER-A,
ui32 period-1);
INT: ENABLE /INT-TIMER0A);
Timer. INT Enable /Timer0-Base/ Timer-TimA-
```

```
TIMEOUT);
INT MASTER ENABLE);
Timer enable (Timer0-Base, Timer-A);
while(1)
{
}
}
void Timer0 int Handler (void)
{
//clear the timer interupt
Timer int clear( Timer0, Base, Timer-TIMA-
  TIME OUT);
//Read the current state of the GPIO PIN
  and it
//write back the opposite state
if GPIO in Read (GPIO-PORTf-Base, GPIO PIN-2)
{
GPIO PIN write (GPIO-PORTf-Base, GPIO-PIN-2)
{
GPIO PIN write (GPIO-PORTF-Base-GPIO PIN-1)
GPIO-PIN-2 | GPIO-PIN-3, 0);
}
else
{
GPIO Pin write ( GPIO-PORTf-Base, GPIO
                                -PIN-2, 4);
}
}
```

Result:

- Hence tagging of LED by the configuring timer of interrupt TIVA programming is successfully completed.

Hibernation Mode And wake an RTC interupt.

Page No. 13
Date

Aim:-

Hibernation module and the loco power modes of the TIVA-C series device, place the device in sleep mode and wake on RTC interupt.

software required:

Code Composer studio (CCS).

Hardware required:

TM4C123GH6M Launch pad plugged into PC via USB.

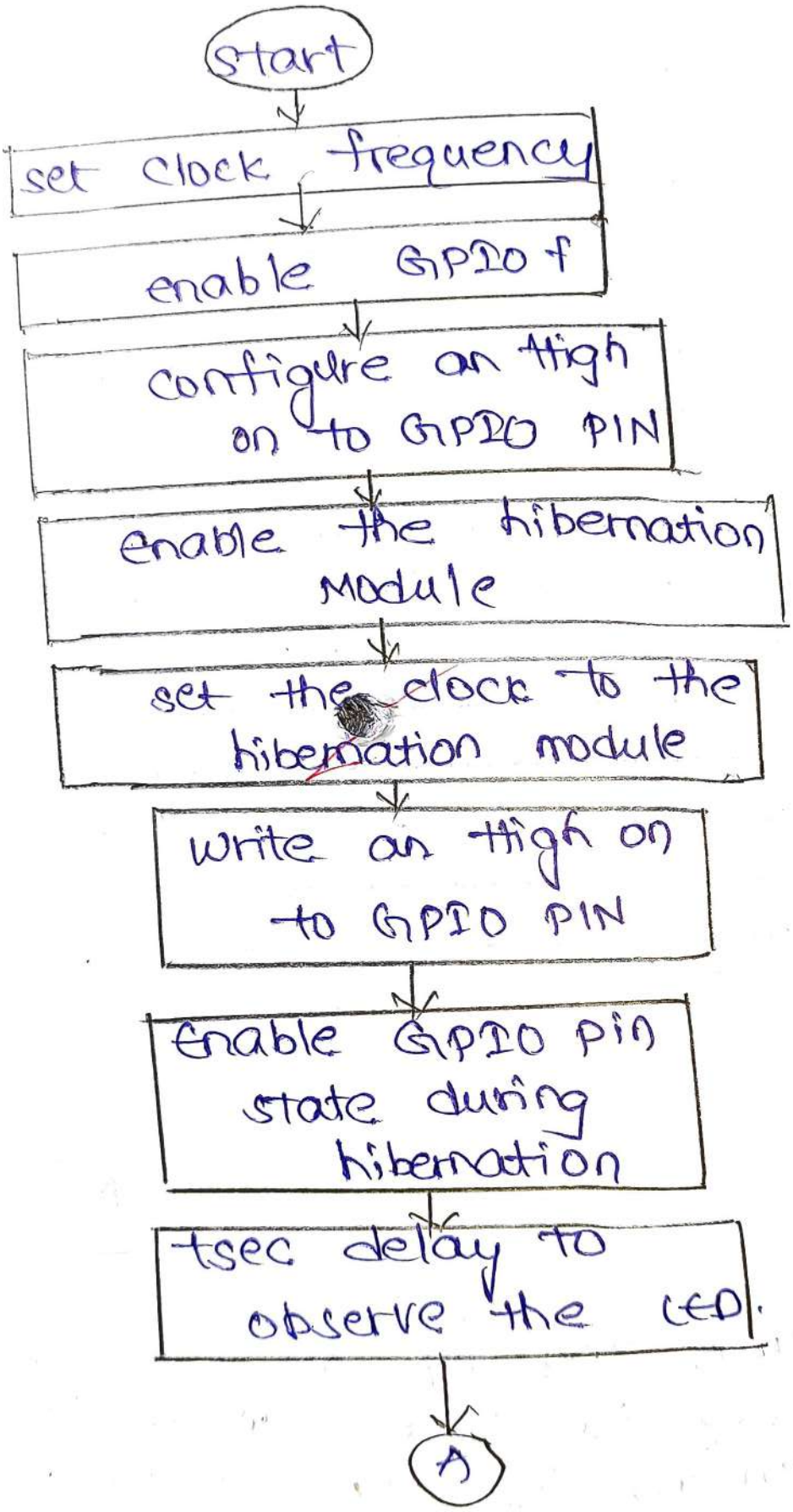
Theory:

The clock crystal (or) external oscillator can drive the hibernation module RTC to implicity and speed development in C-series device based on uc applications. The intent of this workshop is to be a place where a person with a few C and some uc experience can familarize themself with the TIVA-TM4C1236x1 series parts CCS.

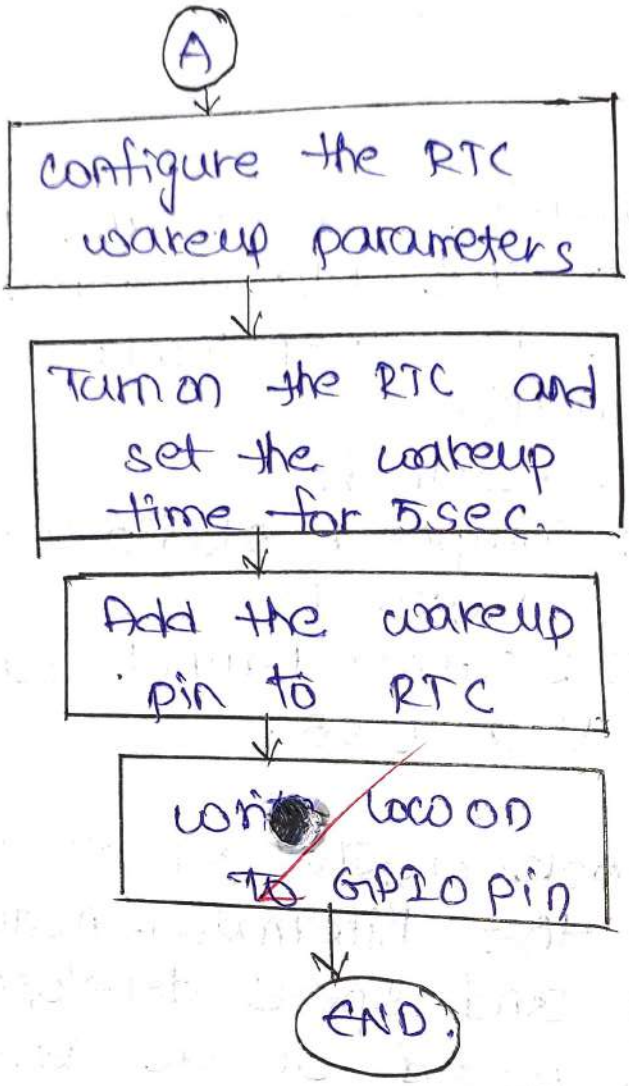
Description:

The Hibernation module manage removal of restoration of power to provide a means for reducing system power consumption when the processor and peripherals are power can be completely removed with only the hibernation module removing

flowchart:


Start
set clock frequency
enable GPIO f
configure an high on to GPIO PIN
enable the hibernation Module
set the clock to the hibernation module
write an high on to GPIO PIN
Enable GPIO pin state during hibernation
1sec delay to observe the LED.
A

flowchart


A
Configure the RTC wakeup parameters
Turn on the RTC and set the wakeup time for 5 sec.
Add the wakeup pin to RTC
write LOW to GPIO pin
END

powered power can be restored based on an external signal or act a creation time using the built in RTC.

The hibernation module can be independ supplied from an external battery coil an auxiliary power supply. The hibernation module has following features.

→ 32-bit need time seconds. counter RTC with 1/32,768 second reduction and a 15-bit subseconds counter

→ 32-bit RTC seconds match register and a 15 bit sub seconds match for timid wake up and interrupt Generation with 1/32,768 seconds resolution

→ RTC provider time for making fine Adjustments to the check or clock state of two mechanisms for power control. using discrete external regulators.

→ on chip-power control using internal switches under register control.

→ Dedicated pin for waking using an external signal

→ GPIO pin state can be retained using hibernation

→ 16 32-bit words of battery-backed memory to save state during hibernating

→ programmable interrupts for

* RTC match
* external wake
* low battery.

program:

```
#include "utils|utstdlib.h"
#include "inc|hw-types.h"
#include "inc|hw-memmap.h"
#include "driverlib|sysctl.h"
#include "driverlib|pin-mcp.h"
#include "driverlib|debug.h"
#include "driverlib|hibernate.h"
#include "driverlib|gpio.h".
int main (void)
{
SYSCTL clockset (SYSCTL-SYSDIV-5|SYSCTL-USE-PLL)
  SYSCTL-XTAL-16MHZ|SYSCTL-OSC-MAIN);
SYSCTL peripheral enable (SYSCTL-PERIPH-GPIOE);
GPIO PIN Type GPIO output (GPIO-PORTE-BASE,
                GPIO-PIN-1|GPIO-PIN-2|GPIO-PIN
GPIO pin write (GPIO-PORTF-BASE, GPIO-PIN-1|
                GPIO-PIN-2|GPIO-PIN-3, 0X0E);
SYSCTL peripheral Enable (SYSCTL-PERIOD-
                HIBERNATE);
Hibernate Enable expclk (sysctlClockGet());
Hibernate GPIO Retention Enable ();
SYSCTL Delay (6400000);
Hibernate RTC set (0);
```

```
Hibernate RTC Enable ( );
Hibernate RTC match set (0,5);
Hibernate Wake set( HIBERNATE_WAKE_PIN | HIBERNATE_WAKE_RTC);
GPIO Pin Write (GPIO_PORTF_Base, GPIO_PIN_3, 0x00);
Hibernate Request ( );
while (1)
{
}
}
```

Result:

Hence, hibernation module and the low power modes of the TIVA-C series device place the device in sleep mode and wave on RTC interrupt is successfully completed.

PWM GENERATION.

Page No. 49
Date

Aim:
To generate a pulse width modulation using PWM module on Tiva.

software required:
Code Composer studio.

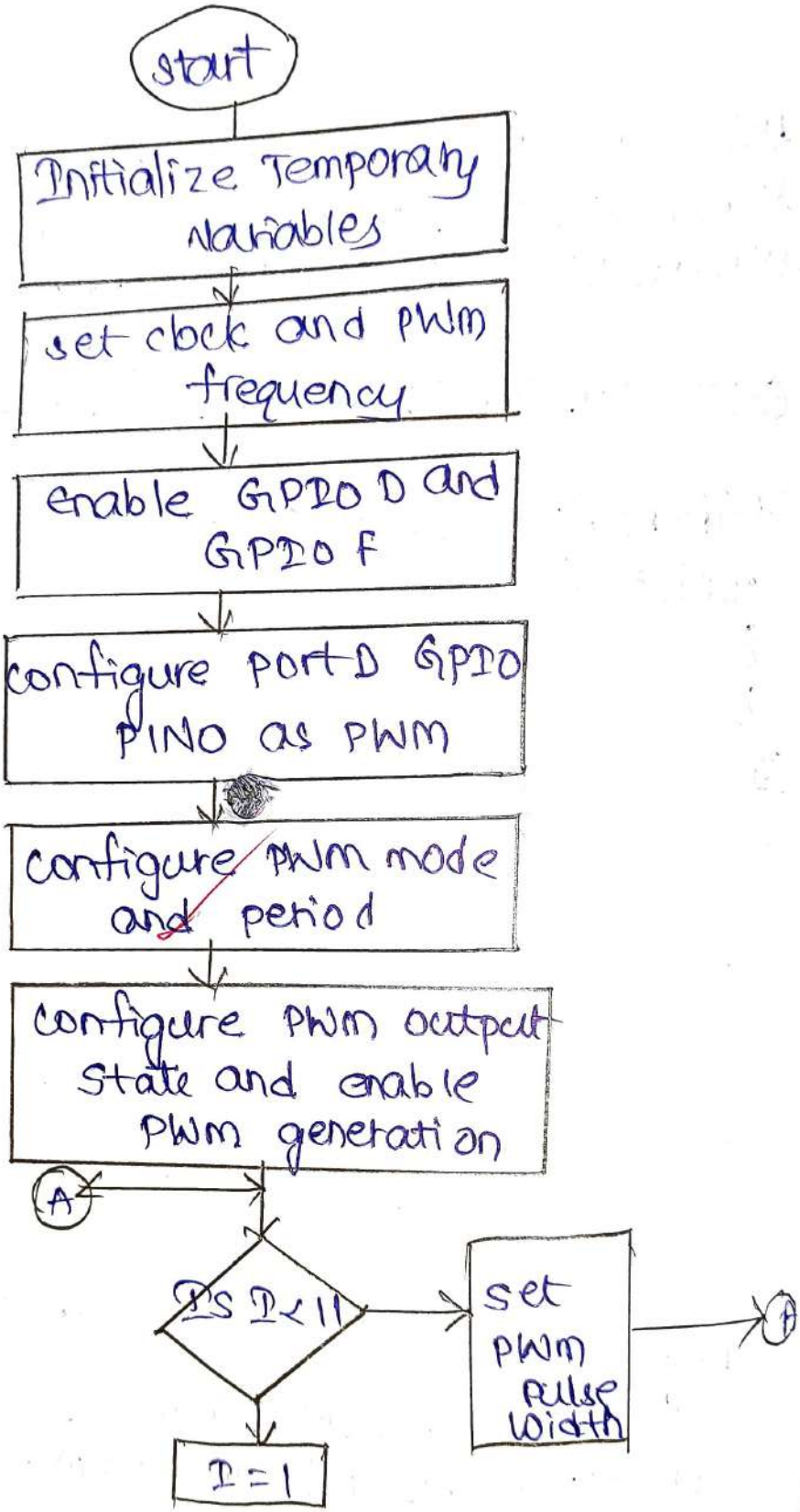
Hardware required
→ TM4C123GH6PM launchpad plugged into pc via USB
→ An oscilloscope to observe the result.

Description:
pulse width modulation is a method of digitally encoding analogy signal levels. High resolution digital counters are used to generate square wave of a given frequency and the duty cycle of that square wave is modulated to encode the analog signal.

The TM4C123GH6PM has two PWM modules each PWM module consists of:
→ four PWM generate blocks
→ A control block with determine the polarity of the sigs and which signals are passed to the pins.

each PWM generator block produces
→ Two independent output signals of the same frequency.
→ A pair of complementary signals with

flowchart:-


start
Initialize Temporary Variables
set clock and PWM frequency
Enable GPIO D and GPIO F
configure PortD GPIO PIN0 as PWM
configure PWM mode and period
configure PWM output state and enable PWM generation
A
Is I<11
set PWM pulse width
A
I=1

deadband generation.
→ eight outputs total

Program:

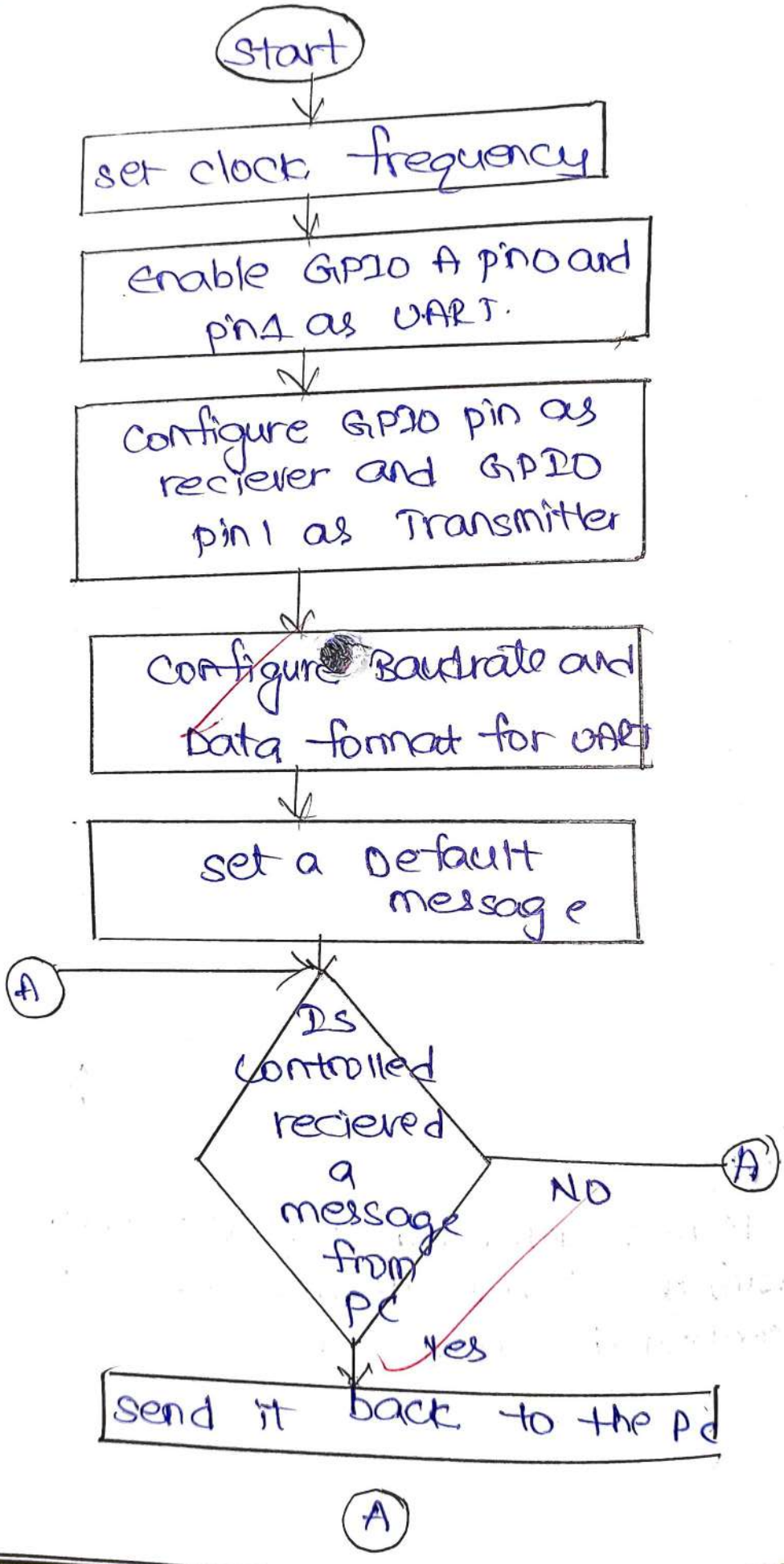
```
#include <stdint.h>
#include <stdbool.h>
#include "inc/hw_memmap.h"
#include "inc/hw_types.h"
#include "driverlib/sysctl.h"
#include "driverlib/gpio.h"
#include "driverlib/debug.h"
#include "driverlib/PWM.h"
#include "driverlib/pin_map.h"
#include "inc/hw_gpio.h"
#include "driverlib/pwm.h"
int main(void)
{
  int i=0;
volatile uint32_t ui32Adjust;
//clock SETTINGS- SET SYSTEM CLOCK TO 40MHZ
SysCtlClockSet(SYSCTL_SYSDIV_5|SYSCTL
_USE_PLL|SYSCTL_OSC_MAIN|SYSCTL_XTAL_
16MHZ);
//SET PWM CLOCK AS SYSTEM CLOCK DIVIDED
BY 64 SysCtlPWMClockSet(SYSCTL_PWMDIV_
64);
// PERIPHERAL CONFIGURATION
```

```
SysCtl peripheral enable (SYSCTL_PERIPH_PWM1);
                    // PWM PERIPHERAL ENABLE
sysctl peripheral enable (SYSCTL_PERIPH_GPIO);
                    // GPIO for PWM1
GPIO pin Type PWM (GPIO_PORTD_BASE,
                    GPIO_PIN_0);
GPIO Pin configure (GPIO_PD0_M1PWM0);
PWM Gen configure (PWM1_BASE, PWM_GEN_0,
                    PWM_GEN_MODE_DOWN);
WITH MODE of OPERATION AS COUNTING
PWM Genperiod set (PWM1_BASE, PWM_GEN_0, 100);
PWM output state (PWM1_BASE, PWM_OUT_0_BI
                    true);
PWM Gen Enable (PWM1_BASE, PWM_GEN_0);
while(1)
{
for (i=1; i<11; i++)
{
Vibadjust = i * 10;
//GPIO FOR PWM1
PWM pulse width set (PWM1_BASE, PWM_OUT_0,
                    Vib Adjust);
SysCtl Delay (4000000);
}
}
}
```

Result:

Hence generated a pulse width modulation (Pwm) using pwm module on TIVA is successfully completed.

flocochart:


Start
set clock frequency
Enable GPIO A pin0 and pin1 as UART.
Configure GPIO pin as reciever and GPIO pin1 as Transmitter
Configure Baudrate and Data format for UART
set a Default message
A
Is controlled reciever a message from PC
NO
A
Yes
send it back to the PC
A

UART - ECHO

Page No. 57
Date

Aim:

To connect the TIVA to terminal on PC and echo back the data using UART.

Software required:

→ code composer studio
→ serial terminal software.

Hardware required:

TM4C123GH6PM launch pad plugged into PC via USB.

Description:

The TM4C123GH6PM controller includes eight universal Asynchronous reciever transmitter with programmable baud-rate generator allowing speed upto 5 mbps for regular speed and 10mbps for high speed. It has separate 16x8 transmit and reciev 8-fos to reduce CPU interupt service loading programming fifo length, including 1-byte deep operation providing conventional double-buffered interface

fully programmable serial interface characteristics is given below

→ 5, 6, 7 or 8 data bits
→ even, odd, stick or no-parity bit generation/detection
→ 1 or 2 stop bit generation
→ support standard FIFO level and end-of

Transmission interupts and efficient transfers using micro Direct memory access controller which has separate channels for transmit and reciever.

program:

```
#include <stdint.h>
#include <std.bool.h>
#include "inc/hw-memmap.h"
#include "inc/hw-types.h"
#include "driverlib/gpio.h"
#include "driverlib/pin-map.h"
#include "driverlib/sysctl.h"
#include "driverlib/UART.h"
#define GPIO-PA0-U0RX 0x00000001
#define GPIO-PA1-U0TX 0x00000401
int main(void)
{
// system clock AT 40MHz
SysCtl clockset (SYSCTL-SYSDIV-5|SYSCTL-USE-PLL)
                SYSCTL-OSC-MAIN|SYSCTL-XTAL-16MHz.
// ENABLE PERIPHERAL UART0
SysCtl Peripheral enable (SYSCTL-PERIPH-UART0);
SysCtl Peripheral Enable (SYSCTL-PERIPH-GPIO);
GPIO pin Configure (GPIO-PA0-U0RX);
GPIO pin configure (GPIO-PA1-U0TX);
GPIO Pin Type UART (GPIO-PORTA BASE, GPIO-PIN-0)
                GPIO-PIN-1);
```

```
UART config set expclk (UART0_BASE, SysCtlClock
                        Get() 115200);
(UART_CONFIG_WLEN_8 | UART_CONFIG_STOP_
    ONE |
  UART_CONFIG_PAR_NONE));
UARTcharput(UART0_BASE, 'E');
UARtcharput(UART0_BASE, 'c');
UARTcharput(UART0_BASE, 'h');
UARTcharput(UART0_BASE, 'O');
UARTcharput(UART0_BASE, ' ');
UARTcharput(UART0_BASE, 'U');
UARTcharput(UART0_BASE, 't');
UARTcharput(UART0_BASE, 'P');
UARTcharput(UART0_BASE, 'U');
UARTcharput(UART0_BASE, 't');
UARTcharput(UART0_BASE, ':');
UARTcharput(UART0_BASE, ' ');
UARTcharput(UART0_BASE, '\n');
while(1)
{
if(UARTcharAvail(UART0_BASE))UARTcharput
      (UART0_BASE, UARTcharGet(UART0_BASE)
}
}
```

Result:

Here the Tiva to terminal on PC and echo back the data using a UART is successfully completed.