# DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING LAB MANUAL

# VLSI LAB

Touching new heights of excellence!

|  |  |  |
|---|---|---|
| **Semester** | **-** | **VII** |
| **Subject** | **-** | **VLSI LAB** |

# LIST OF EXPERIMENTS & SCHEDULE

| **Exp. No.** | **Title** | **Week No.** |
|---|---|---|
| **S. No** | **Experiments** | **Page. No** |
| 1 | Design of Logic gates | 1 |
| 2 | Design of Binary Adders | 2 |
| 3 | Design of Multiplexers and De-multiplexers | 3 |
| 4 | Design of Encoders and Decoders | 4 |
| 5 | Flip Flops | 5 |
| 6 | Counters | 6 |
| 7 | Bitwise Operators Using 8051 | 7 |
| 8 | Toggle a Port bit in 8051 | 8 |
| 9 | Delay Operators in 8051 | 9 |
| | | |

# Introduction to Combinational Circuit Design

## EXP:1 Design of Logic gates

### Introduction

The purpose of this experiment is to simulate the behavior of several of the basic logic gates and you will connect several logic gates together to create simple digital model.

### Software tools Requirement

Equipments:

Computer with Modelsim Software

Specifications:

HP Computer P4 Processor – 2.8 GHz, 2GB RAM, 160 GB Hard Disk

Softwares: Modelsim - 5.7c, Xilinx - 6.1i.

**Algorithm**

STEP 1: Open ModelSim XE II / Starter 5.7C

STEP 2: File -> Change directory -> D:\<register number>

STEP 3: File -> New Library -> ok

STEP 4: File -> New Source -> Verilog

STEP 5: Type the program

STEP 6: File -> Save -> <filename.v>

STEP 7: Compile the program

STEP 8: Simulate -> expand work -> select file -> ok

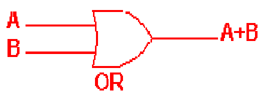
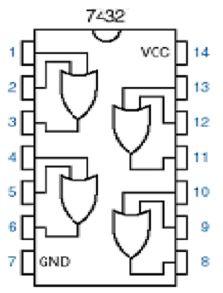
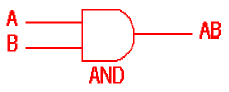
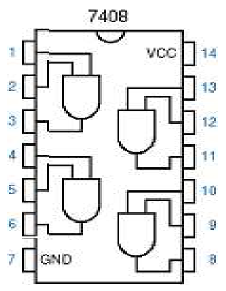
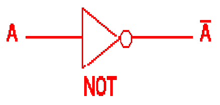
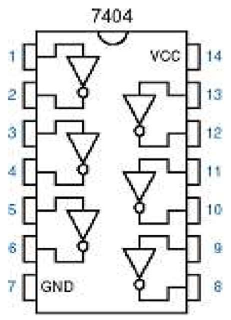
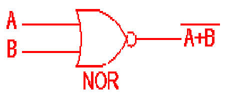
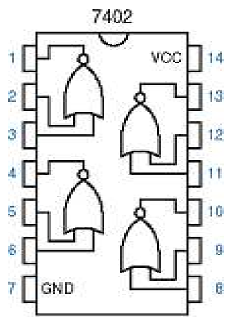
STEP 9: View -> Signals

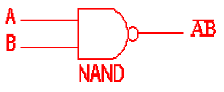
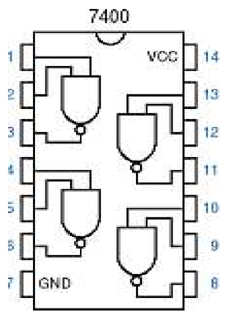
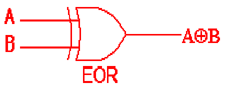
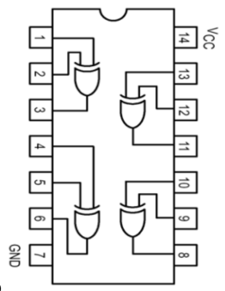
STEP 10: Select values -> Edit -> Force -> input values

STEP 11: Add -> Wave -> Selected signals -> Run

STEP 12: Change input values and run again

## Logic Gates and their Properties

| Gate | Description | Truth Table | | | Logic Symbol | Pin Diagram |
|---|---|---|---|---|---|---|
| OR | The output is active high if any one of the input is in active high state, Mathematically,<br><br>Q = A+B | A<br>0<br>0<br>1<br>1 | B<br>0<br>1<br>0<br>1 | Output Q<br>0<br>1<br>1<br>1 |  |  |
| AND | The output is active high only if both the inputs are in active high state, Mathematically,<br><br>Q = A.B | A<br>0<br>0<br>1<br>1 | B<br>0<br>1<br>0<br>1 | Output Q<br>0<br>0<br>0<br>1 |  |  |
| NOT | In this gate the output is opposite to the input state, Mathematically,<br><br>Q = A | A<br>0<br>1 | | Output Q<br>1<br>0 |  |  |
| NOR | The output is active high only if both the inputs are in active low state, Mathematically,<br><br>Q = (A+B)’ | A<br>0<br>0<br>1<br>1 | B<br>0<br>1<br>0<br>1 | Output Q<br>1<br>0<br>0<br>0 |  |  |

| | | | | | | |
|---|---|---|---|---|---|---|
| NAND | The output is active high only if any one of the input is in active low state, Mathematically,<br>Q = (A.B)’ | A<br>0<br>0<br>1<br>1 | B<br>0<br>1<br>0<br>1 | Output Q<br>1<br>1<br>1<br>0 |  |   7400<br>VCC<br>GND<br>1<br>2<br>3<br>4<br>5<br>6<br>7<br>14<br>13<br>12<br>11<br>10<br>9<br>8 |
| XOR | The output is active high only if any one of the input is in active high state, Mathematically,<br>Q = A.B’ + B.A’ | A<br>0<br>0<br>1<br>1 | B<br>0<br>1<br>0<br>1 | Output Q<br>0<br>1<br>1<br>0 |  |   VCC<br>GND<br>1<br>2<br>3<br>4<br>5<br>6<br>7<br>14<br>13<br>12<br>11<br>10<br>9<br>8 7486 |

## Pre lab Questions

1. What is truth table?

2. Which gates are called universal gates?

3. What is the difference b/w HDL and software language?

4. Define identifiers.

5. A basic 2-input logic circuit has a HIGH on one input and a LOW on the other input, and the output is HIGH. What type of logic circuit is it?

6. A logic circuit requires HIGH on all its inputs to make the output HIGH. What type of logic circuit is it?

7. Develop the truth table for a 3-input AND gate and also determine the total number of possible combinations for a 4-input AND gate.

## VERILOG Program

### a) AND Gate

| **Structural Model** | **Data Flow Model** | **BehaviouralModel** |
|---|---|---|
| moduleandstr(x,y,z);<br>inputx,y;<br>output z;<br>and g1(z,x,y);<br>endmodule | moduleanddf(x,y,z);<br>inputx,y;<br>output z;<br>assign z=(x&y);<br>endmodule | module andbeh(x,y,z);<br>input x,y;<br>output z;<br>reg z;<br>always @(x,y)<br>z=x&y;<br>endmodule |

### b) NAND Gate

| **Structural Model** | **Data Flow Model** | **BehaviouralModel** |
|---|---|---|
| modulenandstr(x,y,z);<br>inputx,y;<br>output z;<br>nand g1(z,x,y);<br>endmodule | modulenanddf(x,y,z);<br>inputx,y;<br>output z;<br>assign z= !(x&y);<br>endmodule | module nandbeh(x,y,z);<br>input x,y;<br>output z;<br>reg z;<br>always @(x,y)<br>z=!(x&y);<br>endmodule |

### c) OR Gate

| **Structural Model** | **Data Flow Model** | **BehaviouralModel** |
|---|---|---|
| module orstr(x,y,z);<br>inputx,y;<br>output z;<br>or g1(z,x,y);<br>endmodule | module ordf(x,y,z);<br>inputx,y;<br>output z;<br>assign z=(x\|y);<br>endmodule | module orbeh(x,y,z);<br>input x,y;<br>output z;<br>reg z;<br>always @(x,y)<br>z=x\|y;<br>endmodule |

### d) NOR Gate

| **Structural Model** | **Data Flow Model** | **BehaviouralModel** |
|---|---|---|
| modulenorstr(x,y,z);<br>inputx,y;<br>output z;<br>nor g1(z,x,y);<br>endmodule | modulenordf(x,y,z);<br>inputx,y;<br>output z;<br>assign z= !(x\|y);<br>endmodule | Modulenorbeh(x,y,z);<br>input x,y;<br>output z;<br>reg z;<br>always @(x,y)<br>z=!(x\|y);<br>endmodule |

**e) XOR Gate**

| **Structural Model** | **Data Flow Model** | **BehaviouralModel** |
|---|---|---|
| module xorstr(x,y,z);<br>inputx,y;<br>output z;<br>xor g1(z,x,y);<br>endmodule | module xordf(x,y,z);<br>inputx,y;<br>output z;<br>assign z=(x^y);<br>endmodule | module xorbeh(x,y,z);<br>input x,y;<br>output z;<br>reg z;<br>always @(x,y)<br>z=x^y;<br>endmodule |

**f) XNOR Gate**

| **Structural Model** | **Data Flow Model** | **BehaviouralModel** |
|---|---|---|
| modulexnorstr(x,y,z);<br>inputx,y;<br>output z;<br>xnor g1(z,x,y);<br>endmodule | modulexnordf(x,y,z);<br>inputx,y;<br>output z;<br>assign z= !(x^y);<br>endmodule | module xnorbeh(x,y,z);<br>input x,y;<br>output z;<br>reg z;<br>always @(x,y)<br>z=!(x^y);<br>endmodule |

**g) NOT Gate**

| **Structural Model** | **Data Flow Model** | **BehaviouralModel** |
|---|---|---|
| module notstr(x,z);<br>input x;<br>output z;<br>not g1(z,x);<br>endmodule | module notdf(x,z);<br>input x;<br>output z;<br>assign z= !x;<br>endmodule | module notbeh(x,z);<br>input x;<br>output z;<br>reg z;<br>always @(x)<br>z=!x;<br>endmodule |

## Post lab Questions

1. What is meant by ports?
2. Write the different types of port modes.
3. What are different types of operators?
4. What is difference b/w <= and := operators?
5. What is meant by simulation?
6. How to give the inputs in modelsim software.

## Lab Report

Each individual will be required to submit a lab report. Use the format specified in the "Lab Report Requirements” document available on the class web page. Be sure to include the following items in your lab report:

Lab cover sheet with staff verification sign.

Answer the pre-lab questions

Complete VERILOG code design for all logic gates and output signal waveforms

Answer the post-lab questions

## 1.6 Grading

| | |
|---|---|
| Pre-lab Work | 20 points |
| Lab Performance | 30 points |
| Post-lab Work | 20 points |
| Lab report | 30 points |

For the lab performance - at a minimum, demonstrate the operation of all the logic gates to your staff in-charge:

The lab report will be graded as follows (for the 30 points):

| | |
|---|---|
| VERILOG code for each logic gates | 15 points |
| Output signal waveform for all logic gates and its truth table | 15 points |

# EXP:2 Design of Binary Adders

## Introduction

The purpose of this experiment is to introduce the design of simple combinational circuits, in this case half adders, half subtractors, full adders and full subtractors.

Software tools Requirement

Equipments:

Computer with Modelsim Software

Specifications:

HP Computer P4 Processor – 2.8 GHz, 2GB RAM, 160 GB Hard Disk

Softwares: Modelsim - 5.7c, Xilinx - 6.1i.

**Algorithm**

STEP 1: Open ModelSim XE II / Starter 5.7C

STEP 2: File -> Change directory -> D:\<register number>

STEP 3: File -> New Library -> ok

STEP 4: File -> New Source -> Verilog

STEP 5: Type the program

STEP 6: File -> Save -> <filename.v>

STEP 7: Compile the program

STEP 8: Simulate -> expand work -> select file -> ok

STEP 9: View -> Signals

STEP 10: Select values -> Edit -> Force -> input values

STEP 11: Add -> Wave -> Selected signals -> Run

STEP 12: Change input values and run again

## Logic Diagram

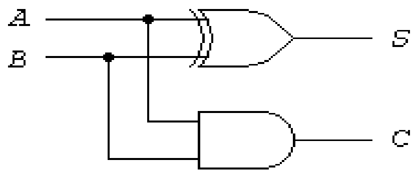


**Figure 2.2.1Half adder**

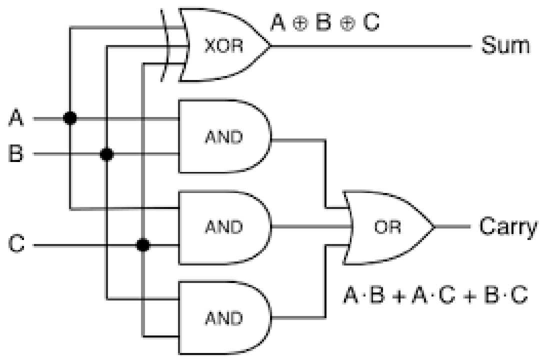


**Figure 2.2.2 Full adder**

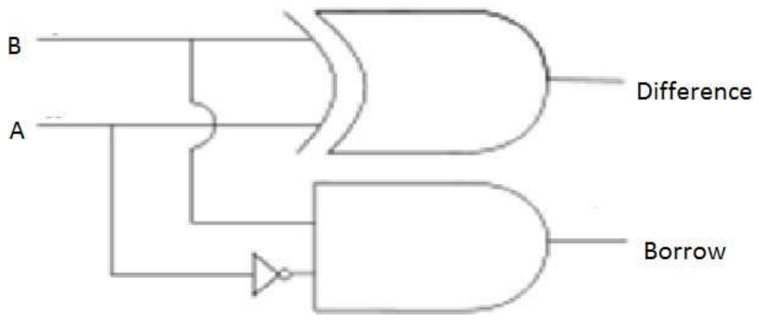


**Figure 2.2.3Halfsubtractor**

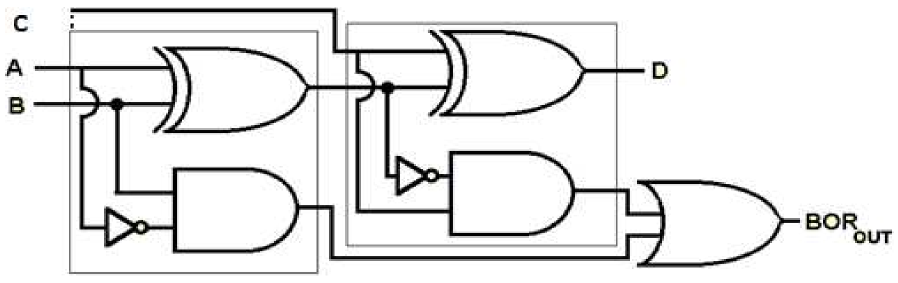


**Figure 2.2.4 Full subtractor**

**Pre lab Questions**

1. What is meant by combinational circuits?
2. Write the sum and carry expression for half and full adder.
3. Write the difference and borrow expression for half and full subtractor.
4. What is signal? How it is declared?

**VERILOG Program**

**HALF ADDER:**

| **Structural model** | **Dataflow model** | **Behaviouralmodel** |
|---|---|---|
| modulehalfaddstr(sum,carry,a,b);<br>outputsum,carry;<br>inputa,b;<br>xor(sum,a,b);<br>and(carry,a,b);<br>endmodule | modulehalfadddf(sum,carry,a,b);<br>outputsum,carry;<br>inputa,b;<br>assign sum = a ^ b;<br>assign carry=a&b;<br>endmodule | modulehalfaddbeh(sum,carry,a,b);<br>outputsum,carry;<br>inputa,b;<br>regsum,carry;<br>always @(a,b);<br>sum = a ^ b;<br>carry=a&b;<br>endmodule |

**FULL ADDER:**

| **Structural model** | **Dataflow model** | **Behaviouralmodel** |
|---|---|---|
| module fulladdstr(sum,carry,a,b,c);<br>outputsum,carry;<br>inputa,b,c;<br>xor g1(sum,a,b,c);<br>and g2(x,a,b);<br>and g3(y,b,c);<br>and g4(z,c,a);<br>or g5(carry,x,z,y);<br>endmodule | modulefulladddf(sum,carry,a,b,c);<br>outputsum,carry;<br>inputa,b,c;<br>assign sum = a ^ b^c;<br>assign carry=(a&b) \| (b&c) \| (c&a);<br>endmodule | modulefulladdbeh(sum,carry,a,b,c);<br>outputsum,carry;<br>inputa,b,c;<br>regsum,carry;<br>always @ (a,b,c)<br>sum = a ^ b^c;<br>carry=(a&b) \| (b&c) \| (c&a);<br>endmodule |

**HALF SUBTRACTOR:**

| **Structural model** | **Dataflow Model** | **BehaviouralModel** |
|---|---|---|
| modulehalfsubtstr(diff,borrow,a,b);<br>outputdiff,borrow;<br>inputa,b;<br>xor(diff,a,b);<br>and( borrow,~a,b);<br>endmodule | modulehalfsubtdf(diff,borrow,a,b);<br>outputdiff,borrow;<br>inputa,b;<br>assign diff = a ^ b;<br>assign borrow=(~a&b);<br>endmodule | modulehalfsubtbeh(diff,borrow,a,b);<br>outputdiff,borrow;<br>inputa,b;<br>regdiff,borrow;<br>always @(a,b)<br>diff = a ^ b;<br>borrow=(~a&b);<br>endmodule |

**FULL SUBTRACTOR:**

| **Structural model** | **Dataflow Model** | **BehaviouralModel** |
|---|---|---|
| module<br>fullsubtstr(diff,borrow,a,b,c)<br>;<br>outputdiff,borrow;<br>inputa,b,c;<br>wire a0,q,r,s,t;<br>not(a0,a);<br>xor(x,a,b);<br>xor(diff,x,c);<br>and(y,a0,b);<br>and(z,~x,c);<br>or(borrow,y,z);<br>endmodule | modulefullsubtdf(diff,borrow,a,b,c)<br>;<br>outputdiff,borrow;<br>inputa,b,c;<br>assign diff = a^b^c;<br>assign borrow=(~a&b)\|(~(a^b)&c);<br>endmodule | modulefullsubtbeh(diff,borrow,a,b,c)<br>;<br>outputdiff,borrow;<br>inputa,b,c;<br>outputdiff,borrow;<br>always@(a,b,)<br>diff = a^b^c;<br>borrow=(~a&b)\|(~(a^b)&c);<br>endmodule |

**Post lab Questions**

1. What are the signal assignment statements?
2. What are the concurrent statements?
3. Write short notes on : a) Process statement b) Block statement
4. Write about sequential statements.
5. What is the difference b/w high impedance state of the signal(Z) and unknown state of the signal(X).

**Lab Report**

Each individual will be required to submit a lab report. Use the format specified in the "Lab Report Requirements” document available on the class web page. Be sure to include the following items in your lab report:

Lab cover sheet with staff verification sign.

Answer the pre-lab questions

Complete VERILOG code design for all logic gates and output signal waveforms

Answer the post-lab questions

**Grading**

| | |
|---|---|
| Pre-lab Work | 20 points |
| Lab Performance | 30 points |
| Post-lab Work | 20 points |
| Lab report | 30 points |

For the lab performance - at a minimum, demonstrate the operation of all the logic gates to your staff in-charge

The lab report will be graded as follows (for the 30 points):

| | |
|---|---|
| VERILOG code for each experiments | 15 points |
| Output signal waveform for all experiments and its truth table | 15 points |

# EXP:3 Design of Multiplexers and Demultiplexers

## Introduction

The purpose of this experiment is to write and simulate a VERILOG program for Multiplexers and Demultiplexers.

## Software tools Requirement:

Equipments:

Computer with Modelsim Software

Specifications:

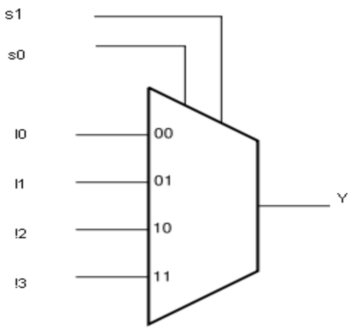
HP Computer P4 Processor – 2.8 GHz, 2GB RAM, 160 GB Hard Disk

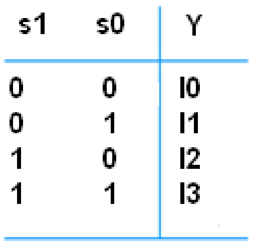
Softwares: Modelsim - 5.7c, Xilinx - 6.1i.

**Algorithm**

```
STEP 1: Open ModelSim XE II / Starter 5.7C
STEP 2: File -> Change directory -> D:\<register number>
STEP 3: File -> New Library -> ok
STEP 4: File -> New Source -> Verilog
STEP 5: Type the program
STEP 6: File -> Save -> <filename.v>
STEP 7: Compile the program
STEP 8: Simulate -> expand work -> select file -> ok
STEP 9: View -> Signals
STEP 10: Select values -> Edit -> Force -> input values
STEP 11: Add -> Wave -> Selected signals -> Run
STEP 12: Change input values and run again
```

## Logic Diagram

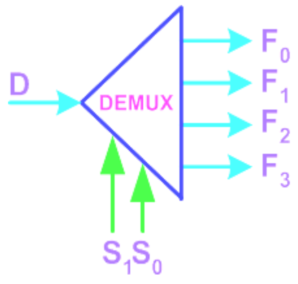


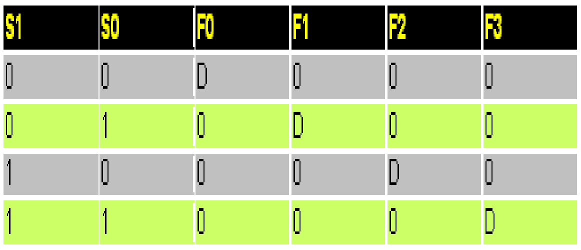


| s1 | s0 | Y |
|---|---|---|
| 0 | 0 | I0 |
| 0 | 1 | I1 |
| 1 | 0 | I2 |
| 1 | 1 | I3 |

**Function Table**

**Figure 3.2.1 4:1 Multiplexer Block diagram**





| S1 | S0 | F0 | F1 | F2 | F3 |
|---|---|---|---|---|---|
| 0 | 0 | D | 0 | 0 | 0 |
| 0 | 1 | 0 | D | 0 | 0 |
| 1 | 0 | 0 | 0 | D | 0 |
| 1 | 1 | 0 | 0 | 0 | D |

**Figure 3.2.21:4 Demux Symbol** **Function Table**

**Logic Diagram**

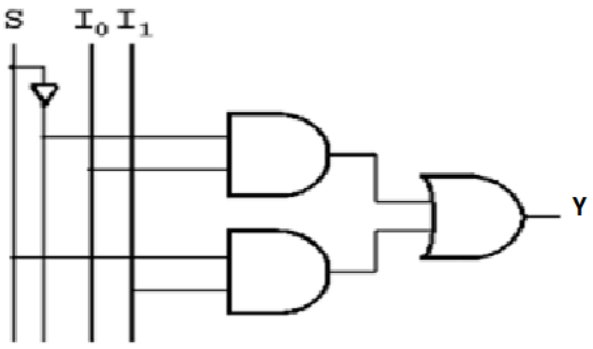


**Figure 3.2.3 2:1 Multiplexer**

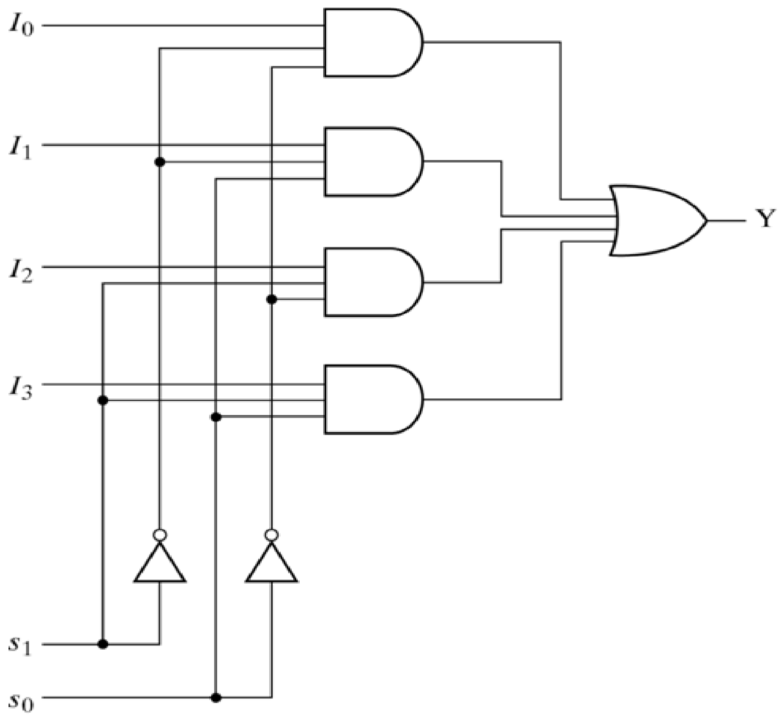


**Figure 3.2.4 4:1 Multiplexer**

**VERILOG Program**

**Multiplexers 2:1 MUX**

| **Structural Model** | **Dataflow Model** | **BehaviouralModel** |
|---|---|---|
| module mux21str(i0,i1,s,y);<br>input i0,i1,s;<br>output y;<br>wire net1,net2,net3;<br>not g1(net1,s);<br>and g2(net2,i1,s);<br>and g3(net3,i0,net1);<br>or g4(y,net3,net2);<br>endmodule | module mux21df(i0,i1,s,y);<br>input i0,i1,s;<br>output y;<br>assign y =(i0&(~s))\|(i1&s);<br>endmodule | module mux21beh(i0,i1,s,y);<br>input i0,i1,s;<br>output y;<br>reg y;<br>always@(i0,i1)<br>  begin<br>  if(s==0) y=i1;<br>  if(s==1)y=i0;<br>  end<br>endmodule |

## 4:1 MUX

| **Structural  Model** | **Dataflow Model** | **BehaviouralModel** |
|---|---|---|
| module<br>mux41str(i0,i1,i2,i3,s0,s1,y);<br>input i0,i1,i2,i3,s0,s1;<br> wire a,b,c,d;<br>output y;<br> and g1(a,i0,s0,s1);<br> and g2(b,i1,(~s0),s1);<br>and g3(c,i2,s0,(~s1));<br>and g4(d,i3,(~s0),(~s1));<br>or(y,a,b,c,d);<br>endmodule | module<br>mux41df(i0,i1,i2,i3,s0,s1,y);<br>input i0,i1,i2,i3,s0,s1;<br>output y;<br>assign<br>y=((i0&(~(s0))&(~(s1)))\|<br>(i1&(~(s0))&s1)\|<br>\|(i2&s0&(~(s1)))\|<br>(i3&s0&s1);<br>endmodule | module mux41beh(in,s,y );<br>output y ;<br>input [3:0] in ;<br>input [1:0] s ;<br>reg y;<br>always @ (in,s)<br>begin<br>if (s[0]==0&s[1]==0)<br>y = in[3];<br>else if (s[0]==0&s[1]==1)<br>y = in[2];<br>else if (s[0]l==1&s[1]==0)<br>y = in[1];<br>else<br>y = in[0];<br>end<br>endmodule |

**Logic Diagram**

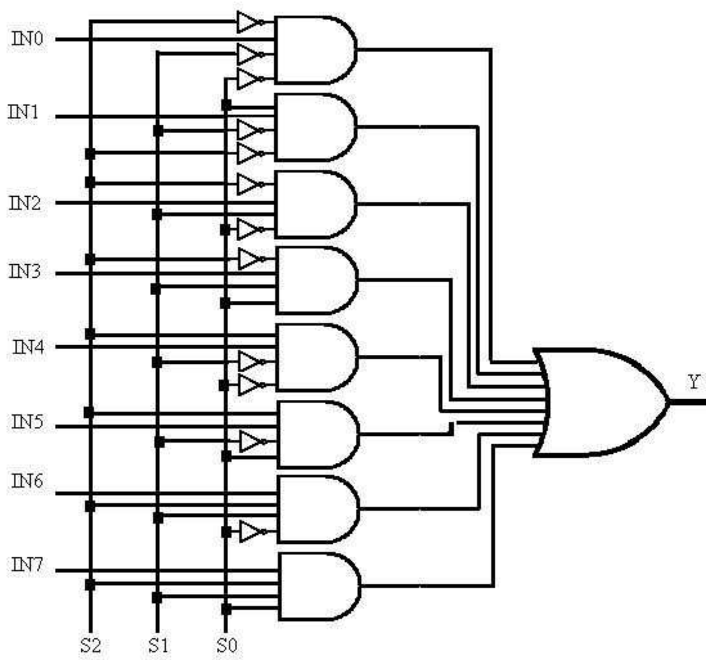


**Figure 3.2.5 8:1 Multiplexer**

**VERILOG Program**

**8:1 MUX**

| **Structural Model** | **Dataflow Model** | **BehaviouralModel** |
|---|---|---|
| modulemux81str(i0,i1,i2,i3,i4,i5,i6,i7,s0,s1,s2,y);<br>input i0,i1,i2,i3,i4,i5,i6,i7,s0,s1,s2;<br>wire a,b,c,d,e,f,g,h;<br>output y;<br>and g1(a,i7,s0,s1,s2);<br>and g2(b,i6,(~s0),s1,s2);<br>and g3(c,i5,s0,(~s1),s2);<br>and g4(d,i4,(~s0),(~s1),s2);<br>and g5(e,i3,s0,s1,(~s2));<br>and g6(f,i2,(~s0),s1,(~s2));<br>and g7(g,i1,s0,(~s1),(s2));<br>and g8(h,i0,(~s0),(~s1),(~s2));<br>or(y,a,b,c,d,e,f,g,h);<br>endmodule | modulemux81df(y,i,s);<br>output y;<br>input [7:0] i;<br>input [2:0] s;<br>wire se1;<br>assign<br>se1=(s[2]*4)\|(s[1]*2)\|(s[0]);<br>assign y=i[se1];<br>endmodule | modulemux81beh(s,i0,i1,i2,i3,i4,i5,i6,i7,y);<br>input [2:0] s;<br>input i0,i1,i2,i3,i4,i5,i6,i7;<br>regy;<br>always@(i0,i1,i2,i3,i4,i5,i6,i7,s) begin<br>case(s) begin<br>3'd0:MUX_OUT=i0;<br>3'd1:MUX_OUT=i1;<br>3'd2:MUX_OUT=i2;<br>3'd3:MUX_OUT=i3;<br>3'd4:MUX_OUT=i4;<br>3'd5:MUX_OUT=i5;<br>3'd6:MUX_OUT=i6;<br>3'd7:MUX_OUT=i7;<br>endcase<br>end<br>endmodule |

**Logic Diagram**

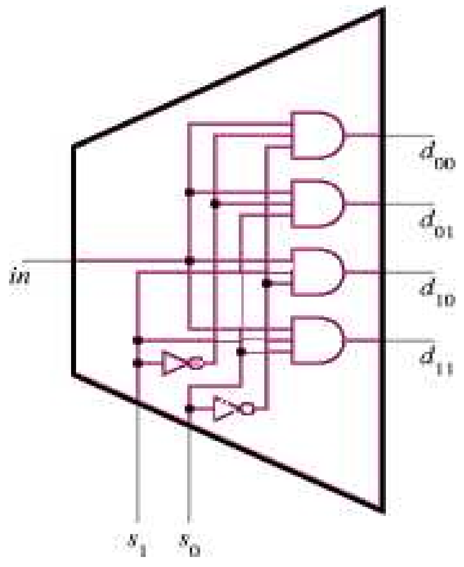


**Figure 3.2.6 1:4 Demultiplexer**

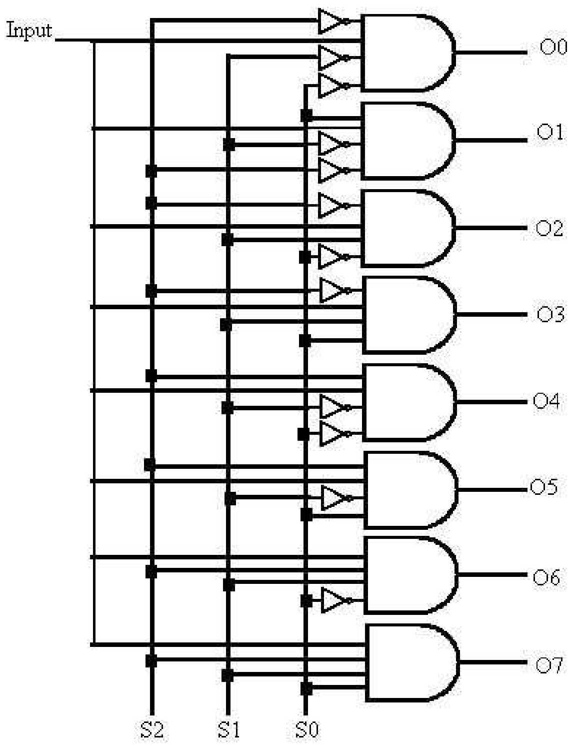


**Figure 3.2.7 1:8 Demultiplexer**

**VERILOG Program**

**1:4 DEMUX**

| **Structural Model** | **Dataflow Model** | **BehaviouralModel** |
|---|---|---|
| module demux14str(in,d0,d1,d2,d3,s0,s1);<br>output d0,d1,d2,d3;<br>input in,s0,s1;<br>and g1(d0,in,s0,s1);<br>and g2(d1,in,(~s0),s1);<br>and g3(d2,in,s0,(~s1));<br>and g4(d3,in,(~s0),(~s1));<br>endmodule | module demux14df( in,d0,d1,d2,d3,s0,s1);<br>output d0,d1,d2,3;<br>input in,s0,s1;<br>assign s0 = in & (~s0) & (~s1);<br>assign d1= in & (~s0) & s1;<br>assign d2= in & s0 & (~s1);<br>assign d3= in & s0 & s1;<br>endmodule | module demux14beh( din,sel,dout );<br>output [3:0] dout ;<br>reg [3:0] dout ;<br>input din ;<br>wire din ;<br>input [1:0] sel ;<br>wire [1:0] sel ;<br>always @ (din or sel) begin<br>case (sel)<br>0 : dout = {din,3'b000};<br>1 : dout = {1'b0,din,2'b00};<br>2 : dout = {2'b00,din,1'b0};<br>default : dout = {3'b000,din};<br>endcase<br>end<br>endmodule |

**1:8 DEMUX**

| **Structural  Model** | **Dataflow Model** | **BehaviouralModel** |
|---|---|---|
| module | module | module |
| demux18str(in,s0,s1,s2,d0,d1 | demux18df(in,s0,s1,s2,i0,d1,d2,d3,d | demux18beh(i, sel, |
| ,d2,d3,d4,d5,d6,d7); | 4,d5,d6,d7); | y); |
| input in,s0,s1,s2; | input in,s0,s1,s2; | input i; |
| output | output d0,d1,d2,d3,d4,d5,d6,d7; | input [2:0] sel; |
| d0,d1,d2,d3,d4,d5,d6,d7; | assign d0 = in & s0 & s1 & s2; | output [7 :0] y ; |
| and g1(d0,in,s0,s1,s2); | assign d1 = in & (~s0) & s1 & s2; | reg [7:0] y; |
| and g2(d1,in,(~s0),s1,s2); | assign d2 = in & s0 & (~s1) & s2; | always@(i,sel) |
| and g3(d2,in,s0,(~s1),s2); | assign d3 = in & (~s0) &( ~s1) & s2; | begin |
| and g4(d3,in,(~s0),(~s1),s2); | assign d4 = in & s0 & s1 & (~s2); | y=8'd0; |
| and g5(d4,in,s0,s1,(~s2)); | assign d5 = in & (~s0) & s1 & (~s2); | case(sel) |
| and g6(d5,in,(~s0),s1,(~s2)); | assign d6 = in & s0 & (~s1) & (~s2); | 3'd0:y[0]=i; |
| and g7(d6,in,s0,(~s1),(~s2)); | assign d7 = in & (~s0) & (~s1) & | 3'd1:y[1]=i; |
| and | (~s2); | 3'd2:y[2]=i; |
| g8(d7,in,(~s0),(~s1),(~s2)); | endmodule | 3'd3:y[3]=i; |
| endmodule | | 3'd4:y[4]=i; |
| | | 3'd5:y[5]=i; |
| | | 3'd6:y[6]=i; |
| | | default:y[7]=i; |
| | | endcase |
| | | end |
| | | endmodule |

## Post Lab questions

1. Implement the function f(A,B,C)=Σm(0,1,3,5,7) by using Mux.
2. Write the VERILOG code for the above design
3. Write the VERILOG code for full subtractor usingDemux.

## Lab Report

Each individual will be required to submit a lab report. Use the format specified in the "Lab Report Requirements" document available on the class web page. Be sure to include the following items in your lab report:

Lab cover sheet with staff verification sign.

Answer the pre-lab questions

Complete VERILOG code design for all logic gates and output signal waveforms

Answer the post-lab questions

## Grading

| | |
|---|---|
| Pre-lab Work | 20 points |
| Lab Performance | 30 points |
| Post-lab Work | 20 points |
| Lab report | 30 points |

For the lab performance - at a minimum, demonstrate the operation of all the logic gates to your staff in-charge

The lab report will be graded as follows (for the 30 points):

| | |
|---|---|
| VERILOG code for each experiments | 15 points |
| Output signal waveform for all experiments and its truth table | 15 points |

# EXP:4 Design of Encoders and Decoders

## Introduction

The purpose of this experiment is to introduce you to the basics of Encoders and Decoders. In this lab, you have to implement Priority Encoder and the Boolean function using Decoders.

Software tools Requirement

Equipments:

Computer with Modelsim Software

Specifications:

HP Computer P4 Processor – 2.8 GHz, 2GB RAM, 160 GB Hard Disk

Softwares: Modelsim - 5.7c, Xilinx - 6.1i.

**Algorithm**

STEP 1: Open ModelSim XE II / Starter 5.7C

STEP 2: File -> Change directory -> D:\<register number>

STEP 3: File -> New Library -> ok

STEP 4: File -> New Source -> Verilog

STEP 5: Type the program

STEP 6: File -> Save -> <filename.v>

STEP 7: Compile the program

STEP 8: Simulate -> expand work -> select file -> ok

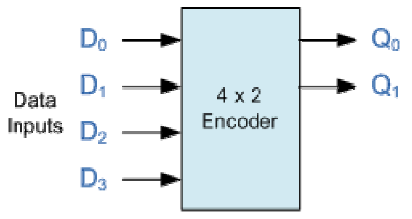
STEP 9: View -> Signals

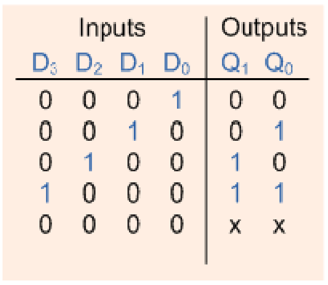
STEP 10: Select values -> Edit -> Force -> input values

STEP 11: Add -> Wave -> Selected signals -> Run

STEP 12: Change input values and run again

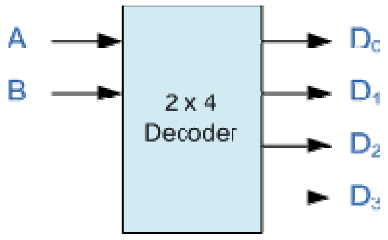
## Logic Diagram

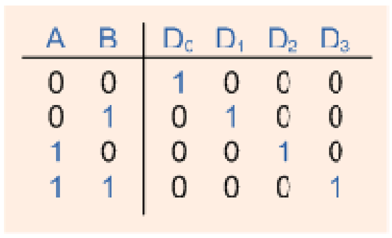




| Inputs | | | | Outputs | |
|---|---|---|---|---|---|
| $D_3$ | $D_2$ | $D_1$ | $D_0$ | $Q_1$ | $Q_0$ |
| 0 | 0 | 0 | 1 | 0 | 0 |
| 0 | 0 | 1 | 0 | 0 | 1 |
| 0 | 1 | 0 | 0 | 1 | 0 |
| 1 | 0 | 0 | 0 | 1 | 1 |
| 0 | 0 | 0 | 0 | x | x |

**Figure 4.1.1 4-to-2 bit Encoder**





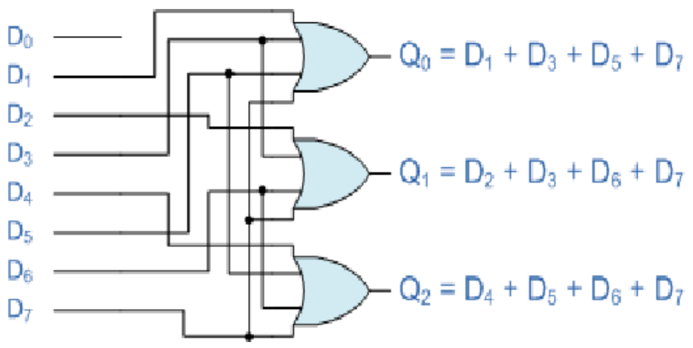
| A | B | $D_0$ | $D_1$ | $D_2$ | $D_3$ |
|---|---|---|---|---|---|
| 0 | 0 | 1 | 0 | 0 | 0 |
| 0 | 1 | 0 | 1 | 0 | 0 |
| 1 | 0 | 0 | 0 | 1 | 0 |
| 1 | 1 | 0 | 0 | 0 | 1 |

**Figure 4.2.2 2-to-4 Binary Decoders**

**Prelab Questions**

1. What is difference b/w encoder and data selector.
2. What is the difference b/w decoder and data distributor.
3. Give the applications of encoder and decoder.
4. Write short notes on " with – select" statement.

**Logic Diagram**



$D_0$
$D_1$
$D_2$
$D_3$
$D_4$
$D_5$
$D_6$
$D_7$
$Q_0 = D_1 + D_3 + D_5 + D_7$
$Q_1 = D_2 + D_3 + D_6 + D_7$
$Q_2 = D_4 + D_5 + D_6 + D_7$

**Figure 4.2.4 8:3 Encoder**

**VERILOG Program**

**8:3 Encoder**

| **Structural Model** | **Data Flow Model** | **BehaviouralModel** |
|---|---|---|
| Module<br>enc83str(d0,d1,d2,d3,d4,d5,d6,d7,q0,q1,q2);<br>Input d0,d1,d2,d3,d4,d5,d6,d7;<br>Output q0,q1,q2;<br>Or g1(q0,d1,d3,d5,d7);<br>Or g2(q1,d2,d3,d6,d7);<br>Or g3(q2,d4,d5,d6,d7);<br>Endmodule | Module<br>enc83df(d0,d1,d2,d3,d4,d5,d6,d7,q0,q1,q2);<br>Input d0,d1,d2,d3,d4,d5,d6,d7;<br>Output q0,q1,q2;<br>Assign q0=d1\|d3\|d5\|d7;<br>Assign q1=d2\|d3\|d6\|d7;<br>Assign q2=d4\|d5\|d6\|d7;<br>Endmodule | module enc83beh<br>(din,a,b,c);<br>input [0:7]din;<br>outputa,b,c;<br>rega,b,c;<br>always@(din)<br>case(din)<br>8'b10000000:begin<br>a=1'b0;b=1'b0,c=1'b0;end<br>8'b01000000:begin<br>a=1'b0;b=1'b0;c=1'b1;end<br>8'b00100000:begin<br>a=1'b0;b=1'b1;c=1'b0;end<br>8'b00010000:begin<br>a=1'b0;b=1'b1;c=1'b1;end<br>8'b10001000:begin<br>a=1'b1;b=1'b0,c=1'b0;end<br>8'b10000100:begin<br>a=1'b1;b=1'b0,c=1'b1;end<br>8'b10000010:begin<br>a=1'b1;b=1'b1,c=1'b0;end<br>8'b10000001:begin<br>a=1'b1;b=1'b1,c=1'b1;end<br>endcase<br>endmodule |

**Logic Diagram**

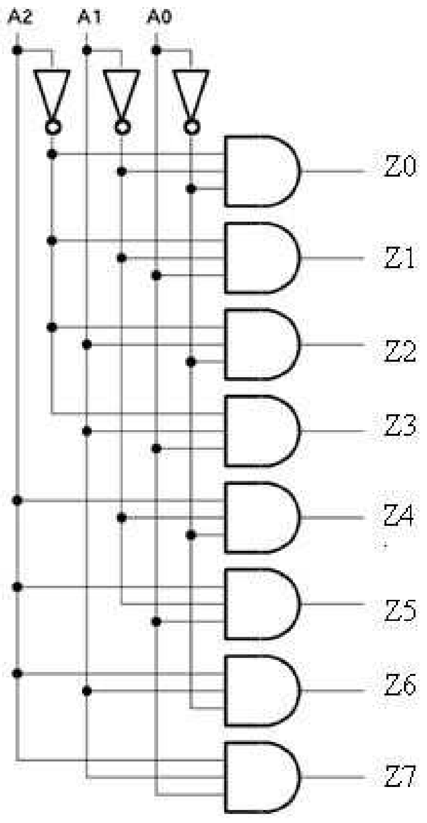


**Figure 4.2.5 3:8 Decoder**

**VERILOG Program**

**3:8 Decoder**

| **Structural Model** | **Data Flow Model** | **BehaviouralModel** |
|---|---|---|
| module decoder38str(z0,z1,z2,z3,z4,z5,z6,z7,a0,a1,a2 ); <br> output z0,z1,z2,z3,z4,z5,z6,z7; <br> input a0,a1,a2; <br> not (s0,a0); <br> not (s1,a1); <br> not (s2,a2); <br> and (z0,s0,s1,s2); <br> and (z1,a0,s1,s2); <br> and (z2,s0,a1,s2); | module decoder38df(z,a0,a1,a2) ; <br> output [7:0] z; <br> input a0,a1,a2; <br> assign z[0] = ~a0 & ~a1 & ~a2; <br> assign z[1] = ~a0& ~a1& a2; <br> assign z[2] = ~a0& a1& ~a2; | module decoder38beh(sel,out1 ); <br><br> input [2:0] sel; <br> outputreg [7:0] out1; <br><br> always @(sel,out1) <br> case (sel) <br> 3'b000 : out1 = 8'b00000001; |

|  |  |  |
|---|---|---|
| and z3,a0,a1,s2);<br>and (z4,s0,s1,a2);<br>and (z5,a0,s1,a2);<br>and (z6,s0,a1,a2);<br>and (z7,a0,a1,a2);<br>endmodule | assign z[3] = ~a0& a1& a2;<br>assign z[4] = a0& ~a1& ~a2;<br>assign z[5] = a0& ~a1& a2;<br>assign z[6] = a0& a1& ~a2;<br>assign z[7] = a0& a1& a2;<br>endmodule | 3’b001 : out1 = 8’b00000010;<br>3’b010 : out1 = 8’b00000100;<br>3’b011 : out1 = 8’b00001000;<br>3’b100 : out1 = 8’b00010000;<br>3’b101 : out1 = 8’b00100000;<br>3’b110 : out1 = 8’b01000000;<br>default : out1 = 8’b10000000;<br>endcase<br><br>endmodule |

**Post Lab questions**

1. Implement full adder by using suitable decoder.
2. Write the VERILOG code for the above design
3. Write the VERILOG code for 3 bit Gray to binary code converter.
4. Write short notes on “test bench” with examples.

**Lab Report**

Each individual will be required to submit a lab report. Use the format specified in the "Lab Report Requirements"document available on the class web page. Be sure to include the following items in your lab report:

Lab cover sheet with staff verification sign.

Answer the pre-lab questions

Complete VERILOG code design for all logic gates and output signal waveforms

Answer the post-lab questions

## Grading

| | |
|---|---|
| Pre-lab Work | 20 points |
| Lab Performance | 30 points |
| Post-lab Work | 20 points |
| Lab report | 30 points |

For the lab performance - at a minimum, demonstrate the operation of all the logic gates to your staff in-charge

The lab report will be graded as follows (for the 30 points):

| | |
|---|---|
| VERILOG code for each experiments | 15 points |
| Output signal waveform for all experiments and its truth table | 15 point |

# EXP 5: Flip Flops

## Introduction

The purpose of this experiment is to introduce you to the basics of flip-flops. In this lab, you will test the behavior of several flip-flops and you will connect several logic gates together to create simple sequential circuits.

**Software tools Requirement**

Equipments:

Computer with Modelsim Software

Specifications:

HP Computer P4 Processor – 2.8 GHz, 2GB RAM, 160 GB Hard Disk

Softwares: Modelsim - 5.7c, Xilinx - 6.1i.

**Algorithm**

STEP 1: Open ModelSim XE II / Starter 5.7C

STEP 2: File -> Change directory -> D:\<register number>

STEP 3: File -> New Library -> ok

STEP 4: File -> New Source -> Verilog

STEP 5: Type the program

STEP 6: File -> Save -> <filename.v>

STEP 7: Compile the program

STEP 8: Simulate -> expand work -> select file -> ok

STEP 9: View -> Signals

STEP 10: Select values -> Edit -> Force -> input values

STEP 11: Add -> Wave -> Selected signals -> Run

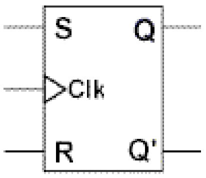
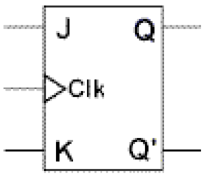
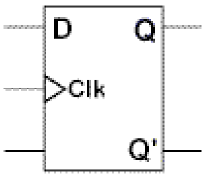
STEP 12: Change input values and run again

## 5.2 Flip-Flops Logic diagram and their properties

Flip-flops are synchronous bitable devices. The term synchronous means the output changes state only when the clock input is triggered. That is, changes in the output occur in synchronization with the clock.

A flip-flop circuit has two outputs, one for the normal value and one for the complement value of the stored bit. Since memory elements in sequential circuits are usually flip-flops, it is worth summarizing the behavior of various flip-flop types before proceeding further.

All flip-flops can be divided into four basic types: SR, JK, D and T. They differ in the number of inputs and in the response invoked by different value of input signals. The four types of flip-flops are defined in the Table 5.1. Each of these flip-flops can be uniquely described by its graphical symbol, its characteristic table, its characteristic equation or excitation table. All flip-flops have output signals Q and Q'.

| Flip-Flop Name | Flip-Flop Symbol | Characteristic Table | Characteristic Equation | Excitation Table |
|---|---|---|---|---|
| SR |  | S \| R \| Q(next)<br>0 \| 0 \| Q<br>0 \| 1 \| 0<br>1 \| 0 \| 1<br>1 \| 1 \| ? | Q(next) = S + R’Q<br>SR = 0 | Q \| Q(next) \| S \| R<br>0 \| 0 \| 0 \| X<br>0 \| 1 \| 1 \| 0<br>1 \| 0 \| 0 \| 1<br>1 \| 1 \| X \| 0 |
| JK |  | J \| K \| Q(next)<br>0 \| 0 \| Q<br>0 \| 1 \| 0<br>1 \| 0 \| 1<br>1 \| 1 \| Q’ | Q(next) = JQ’ + K’Q | Q \| Q(next) \| J \| K<br>0 \| 0 \| 0 \| X<br>0 \| 1 \| 1 \| X<br>1 \| 0 \| X \| 1<br>1 \| 1 \| X \| 0 |
| D |  | D \| Q(next)<br>0 \| 0<br>1 \| 1 | Q(next) = D | Q \| Q(next) \| D<br>0 \| 0 \| 0<br>0 \| 1 \| 1<br>1 \| 0 \| 0<br>1 \| 1 \| 1 |

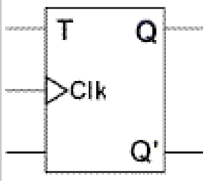
| | | | | |
|---|---|---|---|---|
| T |  | T \| Q(next)<br>0 \| Q<br>1 \| Q’ | $Q(next) = TQ' + T'Q$ | Q \| Q(next) \| T<br>0 \| 0 \| 0<br>0 \| 1 \| 1<br>1 \| 0 \| 1<br>1 \| 1 \| 0 |

**Table 5.2 Flip-flops and their properties**

## 5.2 Logic Diagram

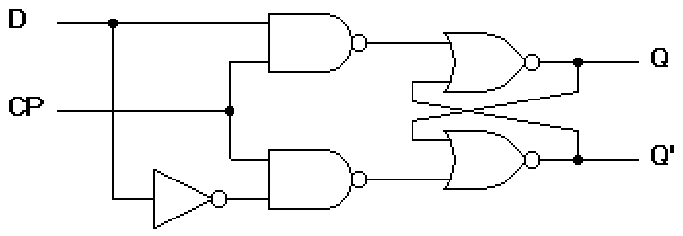


**Figure 5.2.1 D- Flip Flop**

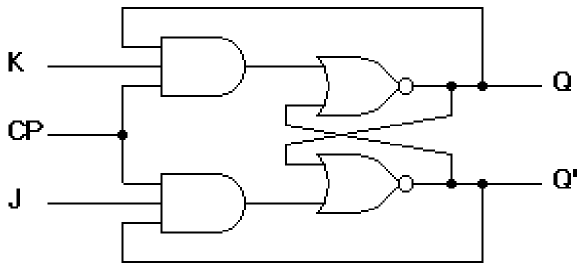


**Figure 5.2 .2 JK Flip Flop**

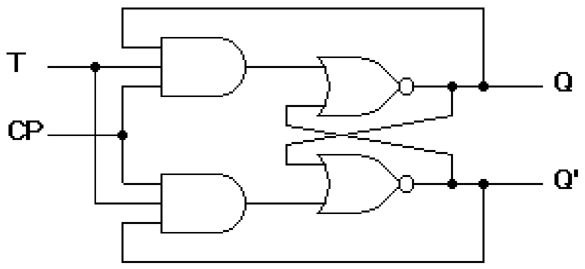


**Figure 5.2.3 T Flip Flop**

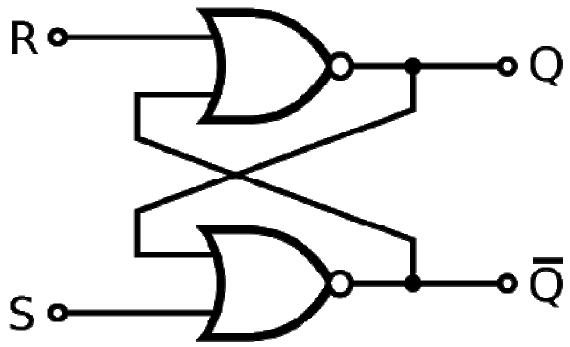


**Figure 5.2.4 T Flip Flop**

## 5. 3 Pre-lab Questions

1. Describe the main difference between a gated S-R latch and an edge-triggered S-R flip-flop.
2. How does a JK flip-flop differ from an SR flip-flop in its basic operation?
3. Describe the basic difference between pulse-triggered and edge-triggered flip-flops.
4. What is use of characteristic and excitation table?
5. What are synchronous and asynchronous circuits?
6. How many flip flops due you require storing the data 1101?

**S-R Flip Flop**

*Dataflow Modelling*

```
modulesr_df (s, r, q, q_n);
input s, r;
output q, q_n;
assignq_n = ~(s | q);
assign q = ~(r | q_n);
endmodule
```

*Structural Modelling*

```
module sr_st(s,r,q,q_n);
input s, r;
output q, q_n;
or g1(q_n,~s,~q);
or g2(q,~r,~q_n);
endmodule
```

*Behavioral Modelling*

```
module sr_beh(s,r,q,q_n);
input s, r;
output q, q_n;
regq, q_n;
always@(s,r)
begin
q,n = ~(s|q);
assign q = ~(r | q_n);
endmodule
```

## T Flip Flop

*Behavioral Modelling*

```
module t_beh(q,q1,t,c);
output q,q1;
inputt,c;
reg q,q1;
initial
begin
q=1'b1;
q1=1'b0;
end
always @ (c)
begin
        if(c)
        begin
        if (t==1'b0) begin q=q; q1=q1; end
        else begin q=~q; q1=~q1; end
        end
end
endmodule
```

*Structural Modelling*

```
        module t_st(q,q1,t,c);
output q,q1;
input t,c;
wire w1,w2;
assign w1=t&c&q;
assign w2=t&c&q1;
        assign q=~(w1|q1);
assign q1=~(w2|q);
endmodule
```

*Dataflow Modelling*

```
        module t_df(q,q,1,t,c);
output q,q1;
input t,c;
and g1(w1,t,c,q);
and g2(w2,t,c,q1);
nor g3(q,w1,q1);
        nor g4(q1,w2,q);
endmodule
```

**D Flip Flop**

| *Behavioral Modelling* | *Dataflow Modelling* | *Structural Modelling* |
|---|---|---|
| Module dff_async_reset( data, clk,<br>reset ,q );<br>input data, clk, reset ;<br>output q;<br>reg q;<br>always @ ( posedgeclk or negedge<br>reset)<br>if (~reset) begin<br>q <= 1'b0;<br>end<br>else begin<br>q <= data;<br>end<br>endmodule | module dff_df(d,c,q,q1);<br>input d,c;<br>output q,q1;<br>assign w1=d&c;<br>assign w2=~d&c;<br>q=~(w1\|q1);<br>q1=~(w2\|q);<br>endmodule | module dff_df(d,c,q,q1);<br>input d,c;<br>output q,q1;<br>and g1(w1,d,c);<br>and g2(w2,~d,c);<br>nor g3(q,w1,q1);<br>nor g4(q1,w2,q);<br>endmodule |

**JK Flip Flop**

| *Behavioral Modelling* | *Dataflow Modelling* | *Structural Modelling* |
|---|---|---|
| module jk(q,q1,j,k,c);<br>output q,q1;<br>input j,k,c;<br>reg q,q1;<br>initial begin q=1'b0; q1=1'b1; end<br>always @ (posedge c)<br>begin<br>case({j,k})<br>{1'b0,1'b0}:begin<br>q=q; q1=q1; end<br>{1'b0,1'b1}: begin<br>q=1'b0; q1=1'b1; end<br>{1'b1,1'b0}:begin<br>q=1'b1; q1=1'b0; end<br>{1'b1,1'b1}: begin<br>q=~q; q1=~q1; end<br>endcase<br>end<br>endmodule | module jkflip_df (j,k,q,qn);<br>input j,k,q;<br>output qn;<br>wire w1,w2;<br>assign w1=~q;<br>assign w2=~k;<br>assign qn=(j & w1 \| w2 & q);<br>endmodule | module jkflip_st(j,k,q,qn);<br>input j,k,q;<br>output qn;<br>and g1(w1,j,~q);<br>and g2(w2,~k,q);<br>or g3(qn,w1,w2);<br>endmodule |

## Post lab

1. Discuss the application of flip-flops in data storage.
2. Draw the logic diagram of Master Slave JK flip-flop.
3. A flip-flop is presently in the RESET state and must go to the SET state on the next clock pulse. What must J and K be?
4. What do you know about clk and clk event in VERILOG?
5. Convert the following.
    a. JK to T f/f
    b. SR to D
6. Write the VERILOG code for question no 5.

## Lab Report

Each individual will be required to submit a lab report. Use the format specified in the "Lab Report Requirements" document available on the class web page. Be sure to include the following items in your lab report:

Lab cover sheet with staff verification for circuit diagram

Answer the pre-lab questions

Complete paper design for all three designs including K-maps and minimized equations and the truth table for each of the output signals.

Answer the post-lab questions

## Grading

| | |
|---|---|
| Pre-lab Work | 20 points |
| Lab Performance | 30 points |
| Post-lab Work | 20 points |
| Lab report | 30 points |

For the lab performance - at a minimum, demonstrate the operation of all the circuits to your staff in-charge

The lab report will be graded as follows (for the 30 points):

| | |
|---|---|
| VERILOG code for each experiments | 15 points |

Output signal waveform for all experiments and its truth table 15 points

# EXP 6: COUNTERS

## Introduction

The purpose of this experiment is to introduce the design of Synchronous Counters. The student should also be able to design n-bit up/down Counter.

## Software tools Requirement

Equipments:

Computer with Modelsim Software

Specifications:

HP Computer P4 Processor – 2.8 GHz, 2GB RAM, 160 GB Hard Disk

Softwares: Modelsim - 5.7c, Xilinx - 6.1i.

**Algorithm**

STEP 1: Open ModelSim XE II / Starter 5.7C

STEP 2: File -> Change directory -> D:\<register number>

STEP 3: File -> New Library -> ok

STEP 4: File -> New Source -> Verilog

STEP 5: Type the program

STEP 6: File -> Save -> <filename.v>

STEP 7: Compile the program

STEP 8: Simulate -> expand work -> select file -> ok

STEP 9: View -> Signals

STEP 10: Select values -> Edit -> Force -> input values

STEP 11: Add -> Wave -> Selected signals -> Run

STEP 12: Change input values and run again

## Logic Diagram

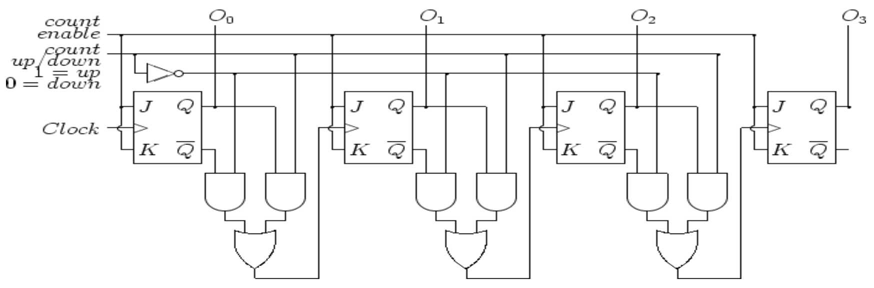


**Figure 6.3.1 Updown Counter**

## PreLab questions

1. How does synchronous counter differ from asynchronous counter?
2. How many flip-flops do you require to design Mod-6 counter.
3. What are the different types of counters?
4. What are the different types of shift registers?
5. How many f/fs are needed for n-bit counter?
6. What is meant by universal shift register?

## VERILOG Program

## Up Down Counter

```
moduleupdown(out,clk,reset,updown);
output [3:0]out;
inputclk,reset,updown;
reg [3:0]out;
always @(posedgeclk)
if(reset) begin
out<= 4'b0;
end else if(updown) begin
out<=out+1;
end else begin
out<=out-1;
end
```

```
endmodule
```

## Post Lab questions

1. Write the use of enable and reset signal.
2. What is the function of generic statement?
3. Design mod-6 counter using d flf and write the VERILOG code.

## Lab Report

Each individual will be required to submit a lab report. Use the format specified in the Lab Report Requirements document available on the class web page. Be sure to include the following items in your lab report:

Lab cover sheet with staff verification for circuit diagram

Answer the pre-lab questions

Complete paper design for all three designs including K-maps and minimized equations and the truth table for each of the output signals.

Answer the post-lab questions

## Grading

| | |
|---|---|
| Pre-lab Work | 20 points |
| Lab Performance | 30 points |
| Post-lab Work | 20 points |
| Lab report | 30 points |

For the lab performance - at a minimum, demonstrate the operation of all the circuits to your staff in-charge

The lab report will be graded as follows (for the 30 points):

| | |
|---|---|
| VERILOG code for each experiments | 15 points |
| Output signal waveform for all experiments and its truth table | 15 points |

# EXP 7: BITWISE OPERATORS USING 8051

## Introduction

The purpose of this experiment is to implement bitwise operators using 8051. The student should also be able to implement Logical Operations in 8051.

## Software tools Requirement

Equipments:

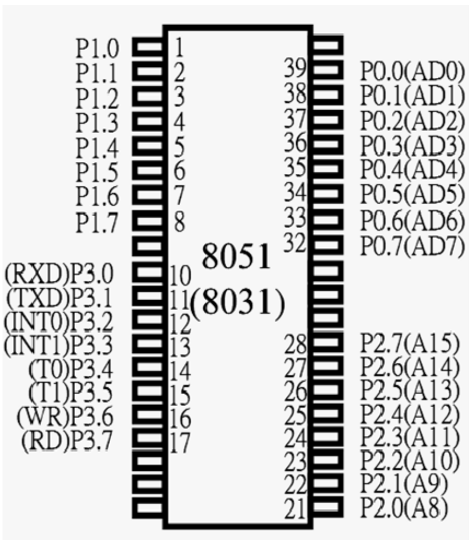
Computer with Keil µversion II Software

Specifications:

HP Computer P4 Processor – 2.8 GHz, 2GB RAM, 160 GB Hard Disk

Softwares: Keil µversion II

## Pin Description of 8051

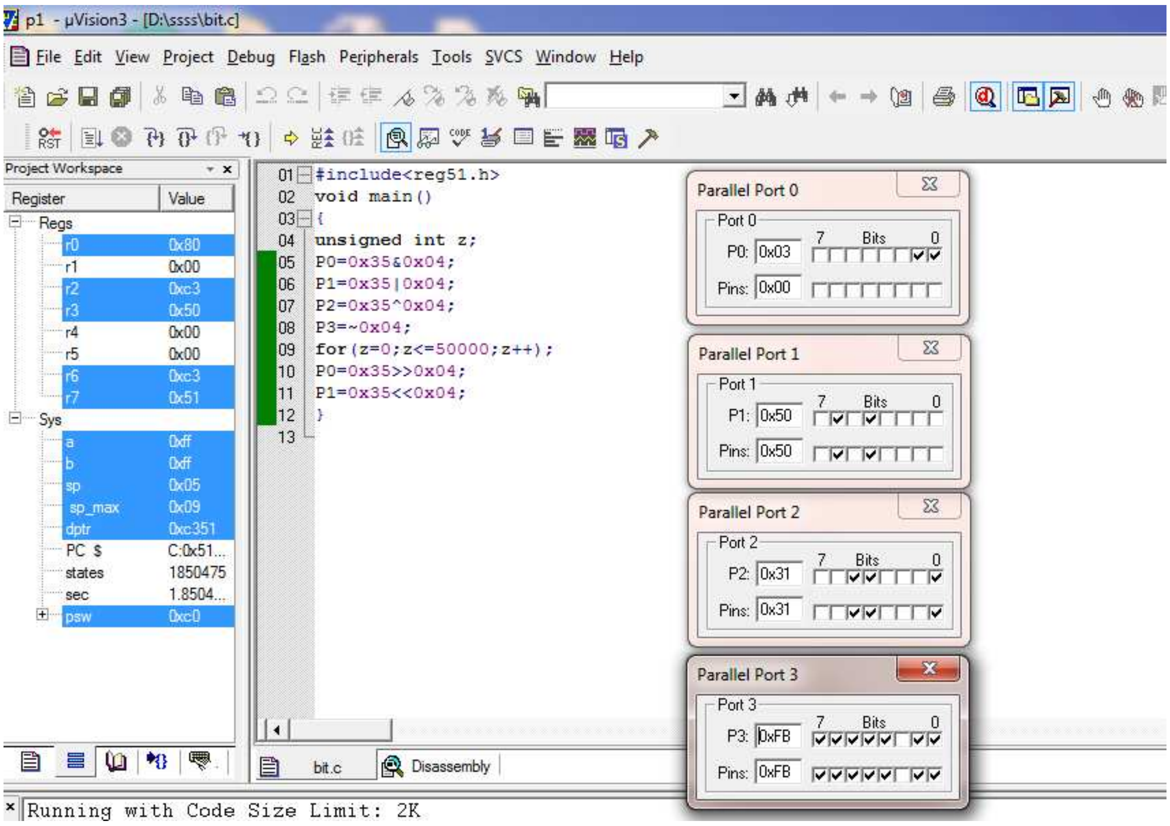


## Pre lab questions

1. Write an 8051 C program to toggle bits of P1 continuously forever with some delay.
2. What is the use of Watchdog timer?
3. What is sbit, sbyte?
4. What is DPTR?
5. What is Power ON Reset?

## Embedded C Program

```
#include<reg51.h>
void main()
{
unsigned int z;
P0=0x35&0x04;
P1=0x35|0x04;
P2=0x35^0x04;
P3=~0x04;
for(z=0;z<=50000;z++);
P0=0x35>>0x04;
P1=0x35<<0x04;
}
```


p1 - µVision3 - [D:\ssss\bit.c]
File Edit View Project Debug Flash Peripherals Tools SVCS Window Help
Project Workspace
bit.c
Disassembly

| Register | Value |
|---|---|
| Regs | |
| r0 | 0x80 |
| r1 | 0x00 |
| r2 | 0xc3 |
| r3 | 0x50 |
| r4 | 0x00 |
| r5 | 0x00 |
| r6 | 0xc3 |
| r7 | 0x51 |
| Sys | |
| a | 0xff |
| b | 0xff |
| sp | 0x05 |
| sp_max | 0x09 |
| dptr | 0xc351 |
| PC $ | C:0x51... |
| states | 1850475 |
| sec | 1.8504... |
| psw | 0xc0 |

```
#include<reg51.h>
void main()
{
unsigned int z;
P0=0x35&0x04;
P1=0x35|0x04;
P2=0x35^0x04;
P3=~0x04;
for(z=0;z<=50000;z++);
P0=0x35>>0x04;
P1=0x35<<0x04;
}
```

Parallel Port 0
Port 0
P0: 0x03 — Bits 7…0: ☐☐☐☐☐☐✓✓
Pins: 0x00 — ☐☐☐☐☐☐☐☐

Parallel Port 1
Port 1
P1: 0x50 — Bits 7…0: ☐✓☐✓☐☐☐☐
Pins: 0x50 — ☐✓☐✓☐☐☐☐

Parallel Port 2
Port 2
P2: 0x31 — Bits 7…0: ☐☐✓✓☐☐☐✓
Pins: 0x31 — ☐☐✓✓☐☐☐✓

Parallel Port 3
Port 3
P3: 0xFB — Bits 7…0: ✓✓✓✓✓☐✓✓
Pins: 0xFB — ✓✓✓✓✓☐✓✓

```
Running with Code Size Limit: 2K
```

## Post lab:

1. Design a Calculator using 8051.
2. Write the Embedded C Program for the above.
3. Write the Embedded C Program for Bit Operations.

## Lab Report

Each individual will be required to submit a lab report. Use the format specified in the "Lab Report Requirements" document available on the class web page. Be sure to include the following items in your lab report:

Lab cover sheet with staff verification for circuit diagram

Answer the pre-lab questions

Complete paper design for all three designs including K-maps and minimized equations and the truth table for each of the output signals.

Answer the post-lab questions

## Grading

| | |
|---|---|
| Pre-lab Work | 20 points |
| Lab Performance | 30 points |
| Post-lab Work | 20 points |
| Lab report | 30 points |

For the lab performance - at a minimum, demonstrate the operation of all the circuits to your staff in-charge

The lab report will be graded as follows (for the 30 points):

| | |
|---|---|
| Embedded C code for each experiments | 15 points |
| Output signal for all experiments and its model calculation | 15 points |

# EXP 8: TOGGLE A PORT BIT IN 8051

## Introduction

The purpose of this experiment is to Toggle a Port bit in 8051. The student should also be able to control Port Pin in 8051.

## Software tools Requirement

Equipments:

Computer with Keil µversion II Software

Specifications:

HP Computer P4 Processor – 2.8 GHz, 2GB RAM, 160 GB Hard Disk

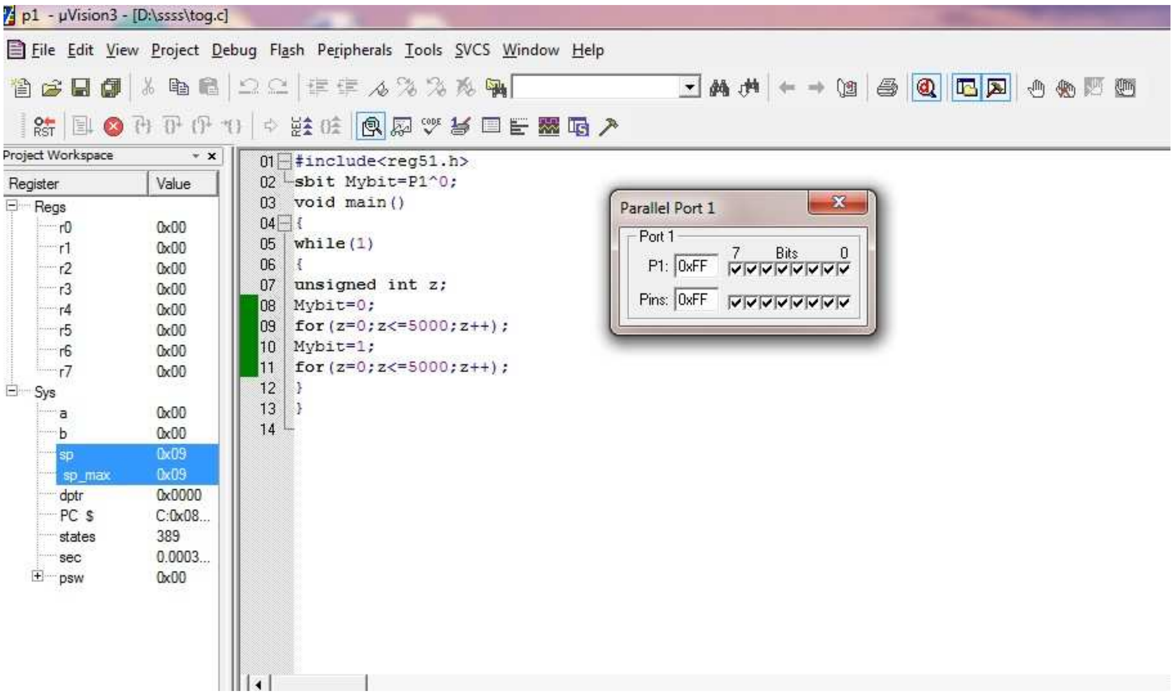
Softwares: Keil µversion II

## Pre lab questions

1. Write an 8051 C program to toggle bits of P1 continuously forever with some delay.
2. What is the use of Watchdog timer?
3. What is sbit, sbyte?
4. What is DPTR?
5. What is Power ON Reset?

## Embedded C Program

```
#include<reg51.h>
sbit Mybit=P1^0;
void main()
{
while(1)
{
unsigned int z;
Mybit=0;
for(z=0;z<=5000;z++);
Mybit=1;
for(z=0;z<=5000;z++);
}
}
```


p1 - µVision3 - [D:\ssss\tog.c]
File Edit View Project Debug Flash Peripherals Tools SVCS Window Help
Project Workspace
Parallel Port 1
Port 1
7 Bits 0

```
#include<reg51.h>
sbit Mybit=P1^0;
void main()
{
while(1)
{
unsigned int z;
Mybit=0;
for(z=0;z<=5000;z++);
Mybit=1;
for(z=0;z<=5000;z++);
}
}
```

| Register | Value |
|---|---|
| Regs | |
| r0 | 0x00 |
| r1 | 0x00 |
| r2 | 0x00 |
| r3 | 0x00 |
| r4 | 0x00 |
| r5 | 0x00 |
| r6 | 0x00 |
| r7 | 0x00 |
| Sys | |
| a | 0x00 |
| b | 0x00 |
| sp | 0x09 |
| sp_max | 0x09 |
| dptr | 0x0000 |
| PC $ | C:0x08... |
| states | 389 |
| sec | 0.0003... |
| psw | 0x00 |

P1: 0xFF ✓✓✓✓✓✓✓✓
Pins: 0xFF ✓✓✓✓✓✓✓✓

## Post lab:

1. A door sensor is connected to the P1.1 pin, and a buzzer is connected to P1.7. Write an 8051 C program to monitor the door sensor, and when it opens, sound the buzzer. You can sound the buzzer by sending a square wave of a few hundred Hz.
2. Write an 8051 C program to get the status of bit P1.0, save it, and send it to P2.7 continuously.

## Lab Report

Each individual will be required to submit a lab report. Use the format specified in the "Lab Report Requirements" document available on the class web page. Be sure to include the following items in your lab report:

Lab cover sheet with staff verification for circuit diagram

Answer the pre-lab questions

Complete paper design for all three designs including K-maps and minimized equations and the truth table for each of the output signals.

Answer the post-lab questions

## Grading

| | |
|---|---|
| Pre-lab Work | 20 points |
| Lab Performance | 30 points |
| Post-lab Work | 20 points |
| Lab report | 30 points |

For the lab performance - at a minimum, demonstrate the operation of all the circuits to your staff in-charge

The lab report will be graded as follows (for the 30 points):

| | |
|---|---|
| Embedded C code for each experiments | 15 points |
| Output signal for all experiments and its model calculation | 15 points |

# EXP 9: DELAY OPERATORS IN 8051

## Introduction

The purpose of this experiment is to introduce delay operators 8051. The student should also be able to write ISR for various Interrupts in 8051.

## Software tools Requirement

Equipments:

Computer with Keil µversion II Software

Specifications:

HP Computer P4 Processor – 2.8 GHz, 2GB RAM, 160 GB Hard Disk

Softwares: Keil µversion II

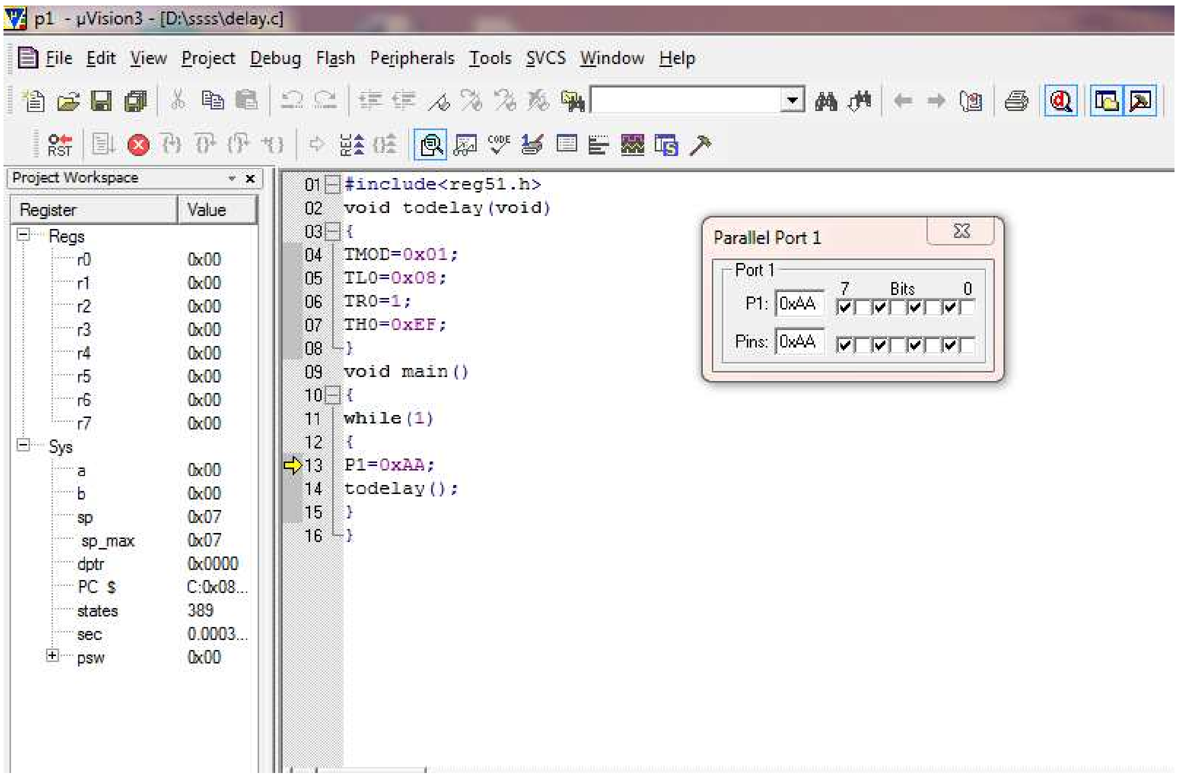
## Pre lab questions

1. Write an 8051 C program to gets a single bit of data from P1.7 and sends it to P1.0.
2. What is ISR?
3. Name the two ways to access Interrupts?
4. What is Power ON Reset?

## Embedded C Program

```
#include<reg51.h>
void todelay(void)
{
TMOD=0x01;
TL0=0x08;
TR0=1;
TH0=0xEF;
}
void main()
{
```

```
while(1)
{
P1=0xAA;
todelay();
} }
```


p1 - µVision3 - [D:\ssss\delay.c]
File Edit View Project Debug Flash Peripherals Tools SVCS Window Help
Project Workspace

| Register | Value |
|---|---|
| Regs | |
| r0 | 0x00 |
| r1 | 0x00 |
| r2 | 0x00 |
| r3 | 0x00 |
| r4 | 0x00 |
| r5 | 0x00 |
| r6 | 0x00 |
| r7 | 0x00 |
| Sys | |
| a | 0x00 |
| b | 0x00 |
| sp | 0x07 |
| sp_max | 0x07 |
| dptr | 0x0000 |
| PC $ | C:0x08... |
| states | 389 |
| sec | 0.0003... |
| psw | 0x00 |

```
#include<reg51.h>
void todelay(void)
{
TMOD=0x01;
TL0=0x08;
TR0=1;
TH0=0xEF;
}
void main()
{
while(1)
{
P1=0xAA;
todelay();
}
}
```

Parallel Port 1
Port 1
P1: 0xAA  7 Bits 0
Pins: 0xAA

## Postlab:

1. A door sensor is connected to the P1.1 pin, and a buzzer is connected to P1.7. Write an 8051 C program to monitor the door sensor, and when it opens, sound the buzzer. You can sound the buzzer by sending a square wave of a few hundred Hz.
2. Write an 8051 C program to get the status of bit P1.0, save it, and send it to P2.7 continuously.

## Lab Report

Each individual will be required to submit a lab report. Use the format specified in the "Lab

Report Requirements" document available on the class web page. Be sure to include the following items in your lab report:

Lab cover sheet with staff verification for circuit diagram

Answer the pre-lab questions

Complete paper design for all three designs including K-maps and minimized equations and the truth table for each of the output signals.

Answer the post-lab questions

## Grading

| | |
|---|---|
| Pre-lab Work | 20 points |
| Lab Performance | 30 points |
| Post-lab Work | 20 points |
| Lab report | 30 points |

For the lab performance - at a minimum, demonstrate the operation of all the circuits to your staff in-charge

The lab report will be graded as follows (for the 30 points):

| | |
|---|---|
| Embedded C code for each experiments | 15 points |
| Output signal for all experiments and its model calculation | 15 points |