

Chapter 2 Gate Level Modeling

2.1 Gate type and syntax

2.1.1 And / or gate

and a1(out, in1,in2) ;

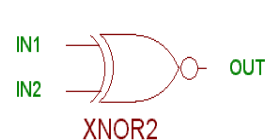
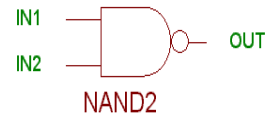
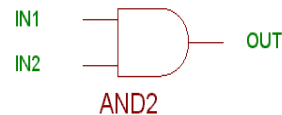
nand a2(out, in1,in2) ;

or a3(out, in1,in2) ;

nor a4(out, in1,in2) ;

xor x1(out,in1,in2) ;

xnor x2(out,in1,in2) ;



And / or gates have one scalar output and multiple scalar inputs.
Examples of and/or gate instantiations are shown below.

More than two inputs

No instance name

nand g1(out,in1,in2,in3) ;

and (out,in1,in2) ;

2.1.2 Buf/not gate

buf b1(out, in) ;

not n1(out, in) ;



Buf/not gates have one scalar input and one or more scalar outputs. Examples of buf /not gates instantiations are shown below.

More than two outputs

buf b1_2(out1,out2, in) ;
not g2 (out , in) ;

2.1.3 Bufif /notif gates

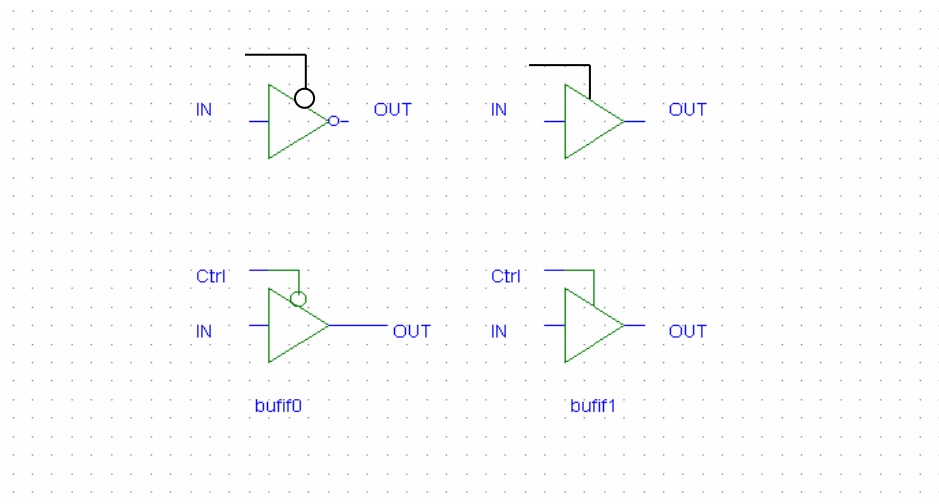
bufif1 b0(out,in,ctrl) ;

bufif0 b1(out,in,ctrl) ;

notif1 b2(out,in,ctrl) ;

notif0 b3(out,in,ctrl) ;

Gates with an additional control signal on buf and not gates also available. These gate propagate only if their control is asserted.



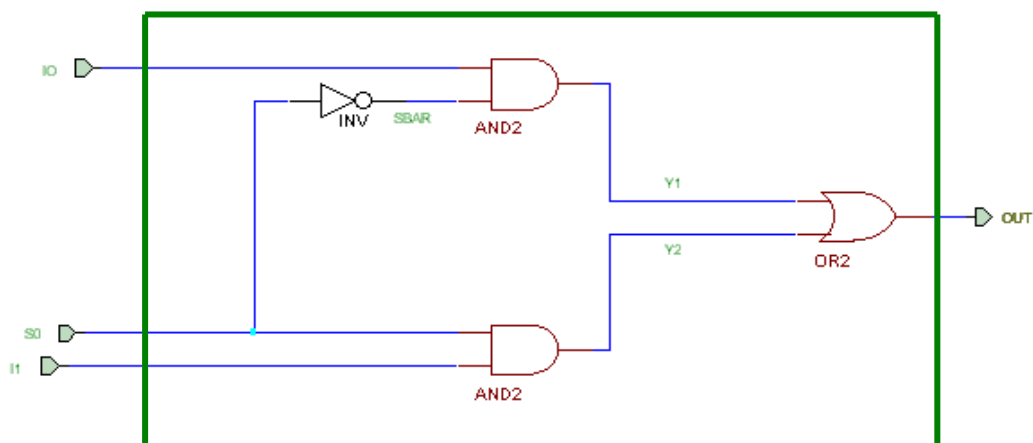
2.2 Basic Design Example

There are five examples to describe the gate level design. Each example will contain (1) logic circuit (2) verilog code (3) test stimulus code (4)simulation result (5)simulation waveform

2.2.1 2-to-1 Multiplexer Design

We will design 2-1 Multiplexer with one select signal. S0 is a selected signal wire. The I/O diagram and truth table for the Multiplexer are shown below.

s0	out
0	i0
1	i1



```
module mux2 (out, i0, i1, s0)
    .....
    .....
endmodule
```

1. Gate level
2. Dataflow level
3. Behavioral level
(Algorithm level)

Verilog Code

Declaration
I/O port

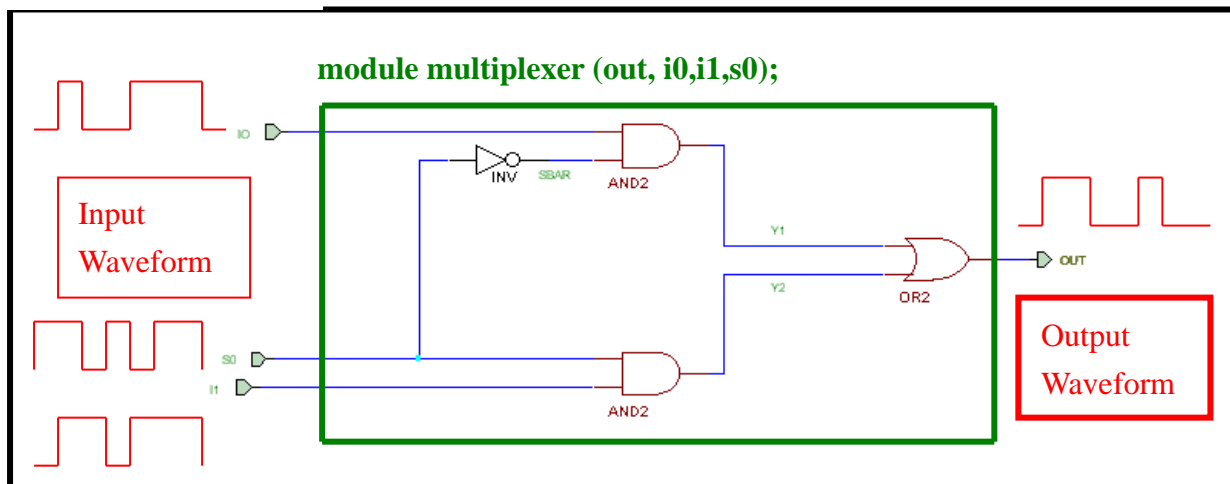
Description with
gate level

```
module mux2(out,i0,i1,s0);  
  output out;  
  input i0,i1,s0;  
  //internal wires  
  wire sbar,y1,y2;  
  
  not g1(sbar,s0);  
  and g2(y1,i0,sbar);  
  and g3(y2,i1,s0);  
  or g4(out,y1,y2);  
endmodule
```

Comment
注解

Test Stimulus Code

module stimulus;



The 2-1 Multiplexer can be tested with the stimulus .The system task \$monitor could also be used to display the signals when they change values.

Call mux2
module

Shown the
result

Input test value after
50 ns

```
module test_mux2;
reg ii0,ii1, ss0;
wire outt1, outt2;
mux2 mymux(outt,ii0,ii1,ss0);
outt2=outt1;
initial
$monitor($time\b,"s0=%b,i0=%b,i1=%b,out=%b\n",ss0,ii0,ii1,
outt);
initial
begin
    ss0=0; ii0=0; ii1=0;
    #50 ss0=0; ii0=0; ii1=1;
    #50 ss0=1; ii0=0; ii1=1;
    #50 ss0=0; ii0=1; ii1=0;
    #50 ss0=1; ii0=1; ii1=0;
end
```

Simulation Result

The output of the simulation is shown below.

使用\$monitor 之電腦螢幕

Simulation result change
to 50n sec

```
control .sav=3
control .savcell=0
control .disk=1000M
Ready: File .sav=mux2_1
Ready: input d:\verilog\gate\mux2_1.v
Reading "d:\verilog\gate\mux2_1.v"
Ready: sim to 0
  Highest level modules (that have been auto-instantiated):
    (text_mux2 text_mux2
  7 total devices.
  Linking ...
  8 nets total: 16 saved and 0 monitored.
  67 registers total: 67 saved.
  Done.
  LINIT begins
                                0s0=0,i0=0,i1=0,out=0

  LINIT done

  0 State changes on observable nets.

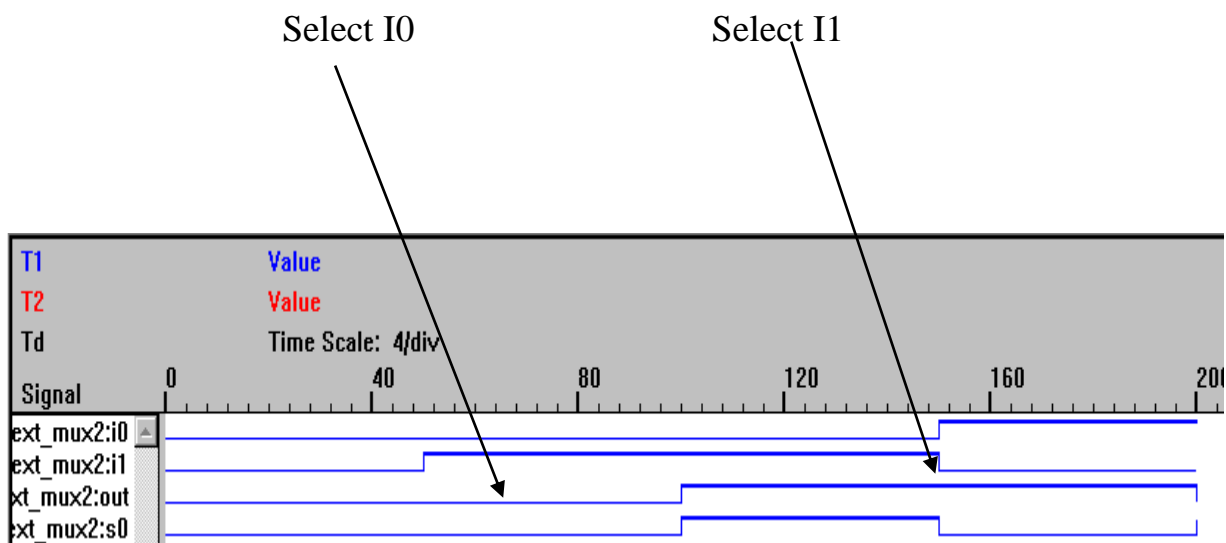
  Simulation stopped at the end of time 0.
Ready: sim
                                50s0=0,i0=0,i1=1,out=0
                                100s0=1,i0=0,i1=1,out=1
                                150s0=0,i0=1,i1=0,out=1
                                200s0=1,i0=1,i1=0,out=0

  23 State changes on observable nets.

  Simulation stopped at the end of time 200.
Ready: |
```

Simulation Waveform (使用 simulator 軟體工具之電腦螢)

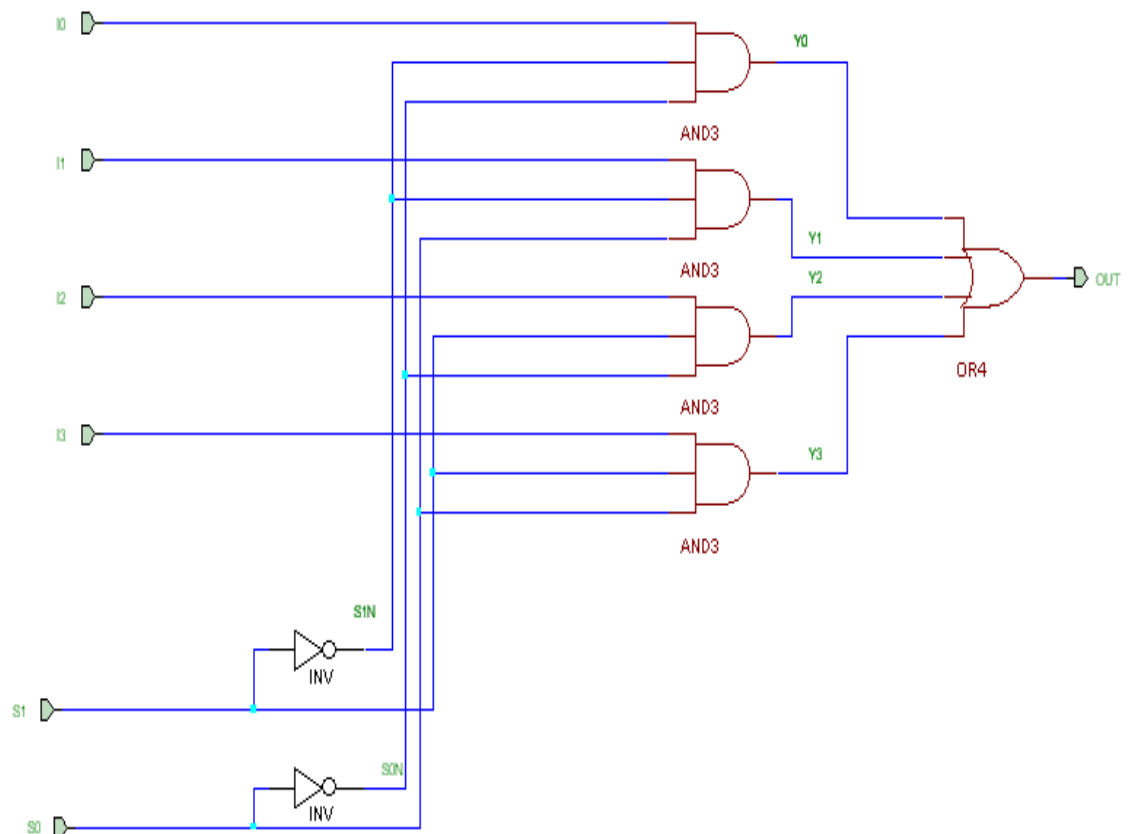
The waveforms from the simulation are shown below.



2.2.2 4-to-1 Multiplexer Design

For 4-1 Multiplexer with 2- select signals, we will assume for the signals s1 and s2 to select data .The logic diagram and truth table for the multiplexer are shown below.

s1	s0	out
0	0	I0
0	1	I1
1	0	I2
1	1	I3



Verilog Code

The Verilog description for the multiplexer is shown below. Two intermediate nets s0n and s1n are created.

Create s1n and s0n

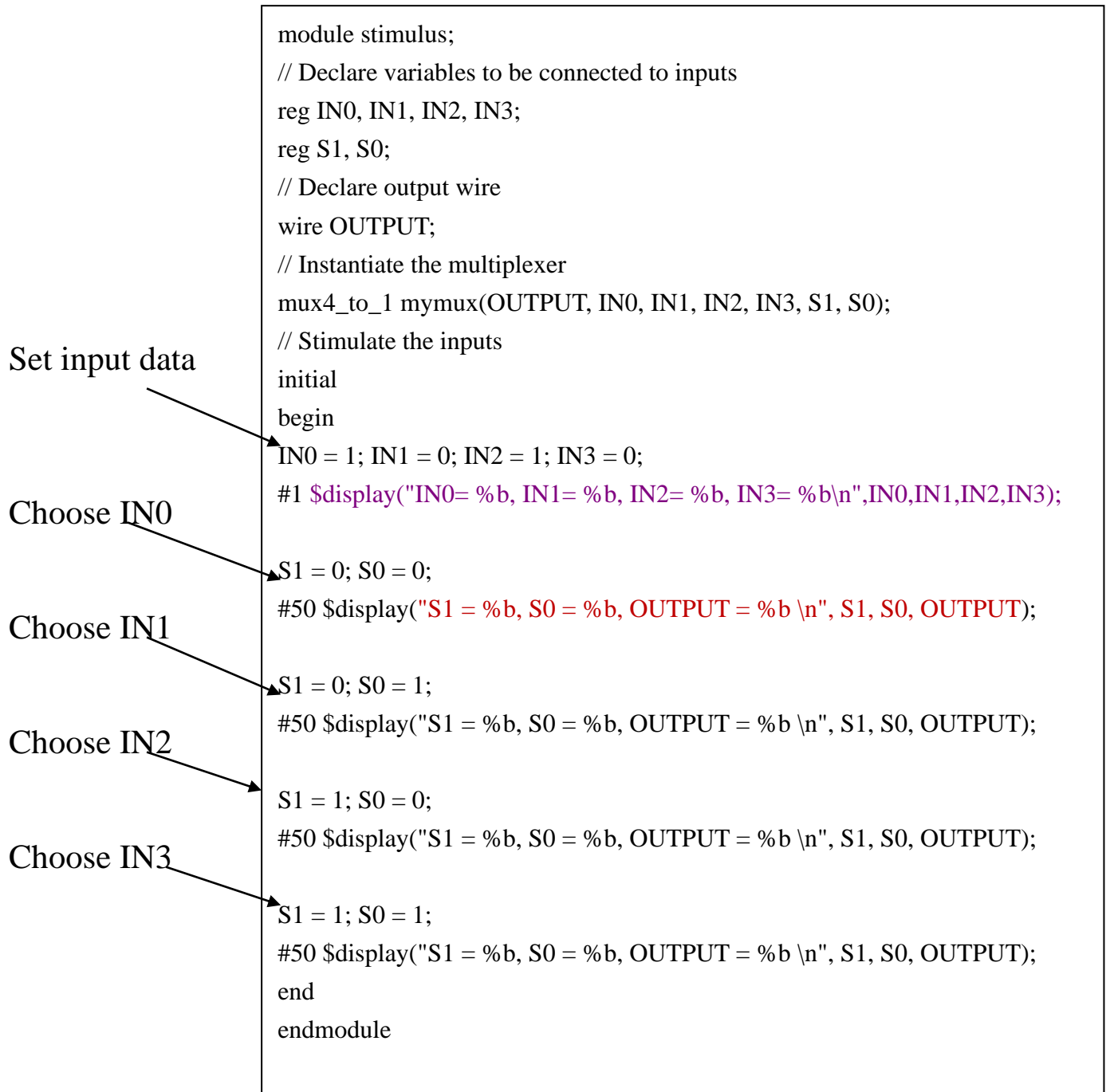
3-input and gate
instantiation

4-input or gate
instantiation

```
module mux4_to_1 (out, i0, i1, i2, i3, s1, s0);  
  // Port declarations from the I/O diagram \  
  output out;  
  input i0, i1, i2, i3;  
  input s1, s0;  
  // Internal wire declarations  
  wire s1n, s0n;  
  wire y0, y1, y2, y3;  
  // Gate instantiations  
  not (s1n, s1);  
  not (s0n, s0);  
  and (y0, i0, s1n, s0n);  
  and (y1, i1, s1n, s0);  
  and (y2, i2, s1, s0n);  
  and (y3, i3, s1, s0);  
  or (out, y0, y1, y2, y3);  
endmodule
```


Test Stimulus Code

The 4-1 multiplexer can be tested with the stimulus is shown below . The system task \$display could be also used to display signal.



Simulation Result

The output of the simulation is shown below. Each combination of the select signals is tested.

Select data with s1 and s0 signals

```
Ready:~sim to 0      - -      -
Highest level modules (that have been auto-instantiated):
(stimulus stimulus
10 total devices.
Linking ...
14 nets total: 22 saved and 0 monitored.
70 registers total: 70 saved.
Done.
LIMIT begins
LIMIT done

0 State changes on observable nets.

Simulation stopped at the end of time 0.
Ready: sim
IN0= 1, IN1= 0, IN2= 1, IN3= 0

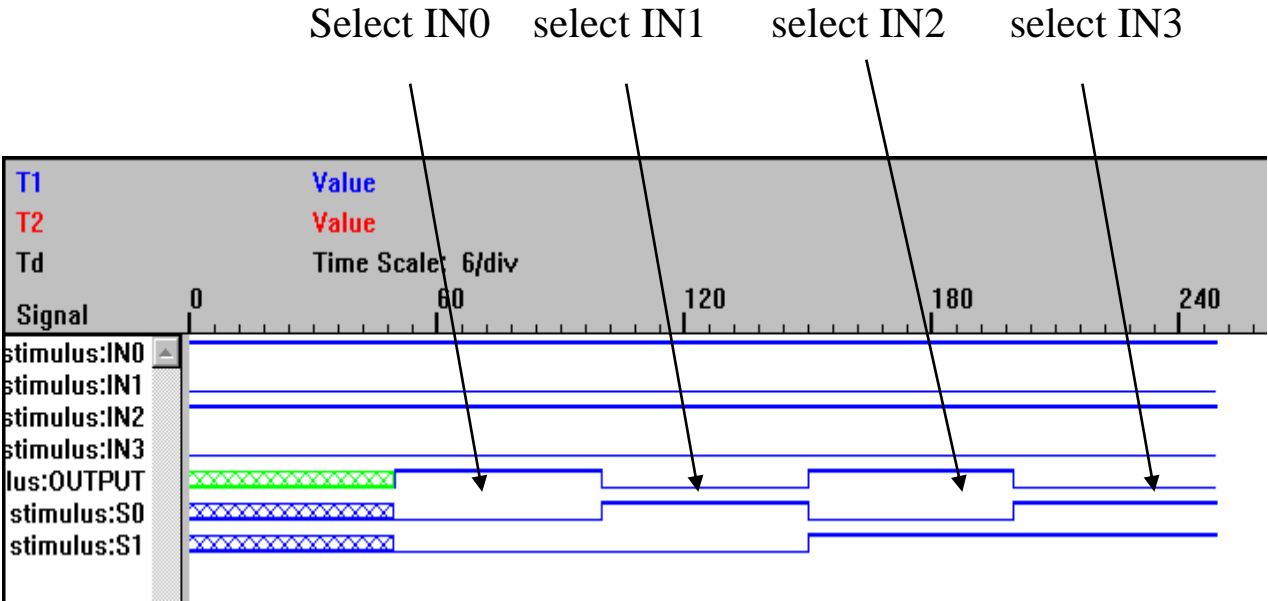
S1 = 0, S0 = 0, OUTPUT = 1
S1 = 0, S0 = 1, OUTPUT = 0
S1 = 1, S0 = 0, OUTPUT = 1
S1 = 1, S0 = 1, OUTPUT = 0

27 State changes on observable nets.

Simulation stopped at the end of time 4.
Ready: |
```

Simulation Waveform

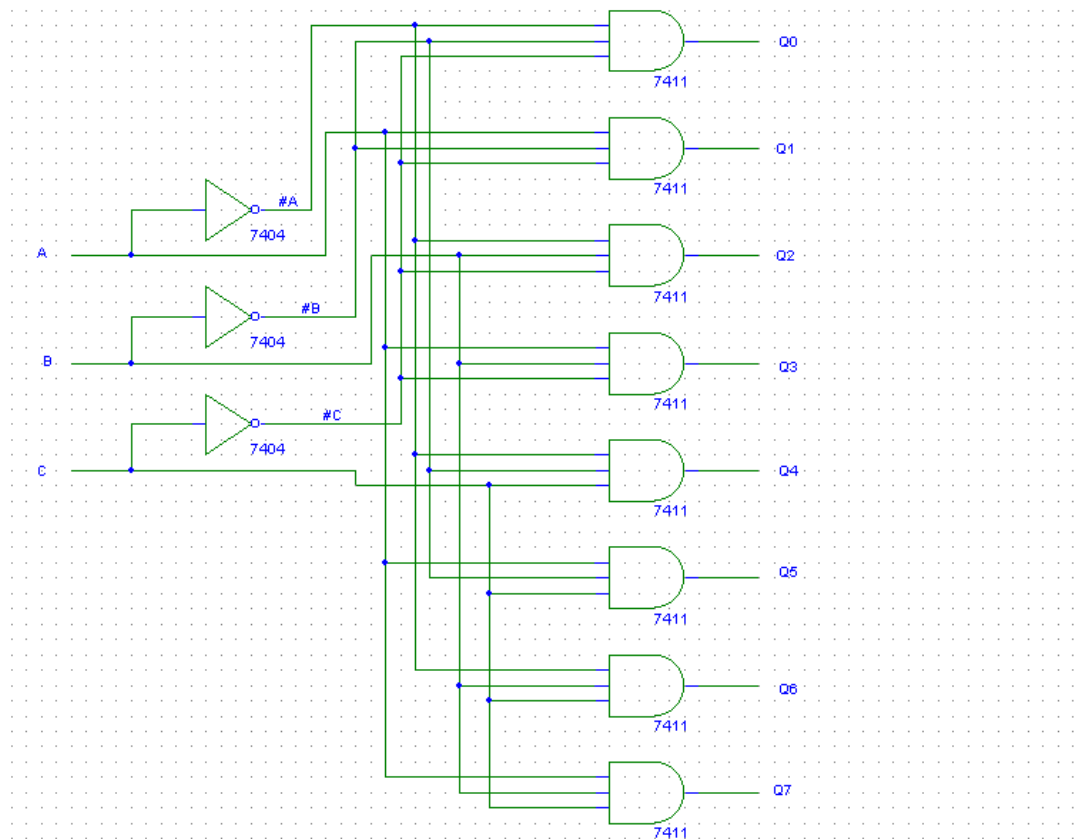
Simulation result is shown below.



2.2.3 3-to-8 Decoder Design

In the example, we design 3 to 8 decoder. We use primitive logic gates, There are 3 inputs and 8 outputs. The truth table is shown below.

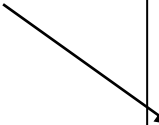
C	B	A	Q0	Q1	Q2	Q3	Q4	Q5	Q6	Q7
0	0	0	1	0	0	0	0	0	0	0
0	0	1	0	1	0	0	0	0	0	0
0	1	0	0	0	1	0	0	0	0	0
0	1	1	0	0	0	1	0	0	0	0
1	0	0	0	0	0	0	1	0	0	0
1	0	1	0	0	0	0	0	1	0	0
1	1	0	0	0	0	0	0	0	1	0
1	1	1	0	0	0	0	0	0	0	1



Verilog Code

The logic diagram for the 3 to 8 decoder is converted to a Verilog is shown below.

Description with
gate level



```
module decoder(Q0,Q1,Q2,Q3,Q4,Q5,Q6,Q7,A,B,C);
output Q0,Q1,Q2,Q3,Q4,Q5,Q6,Q7;
input A,B,C;

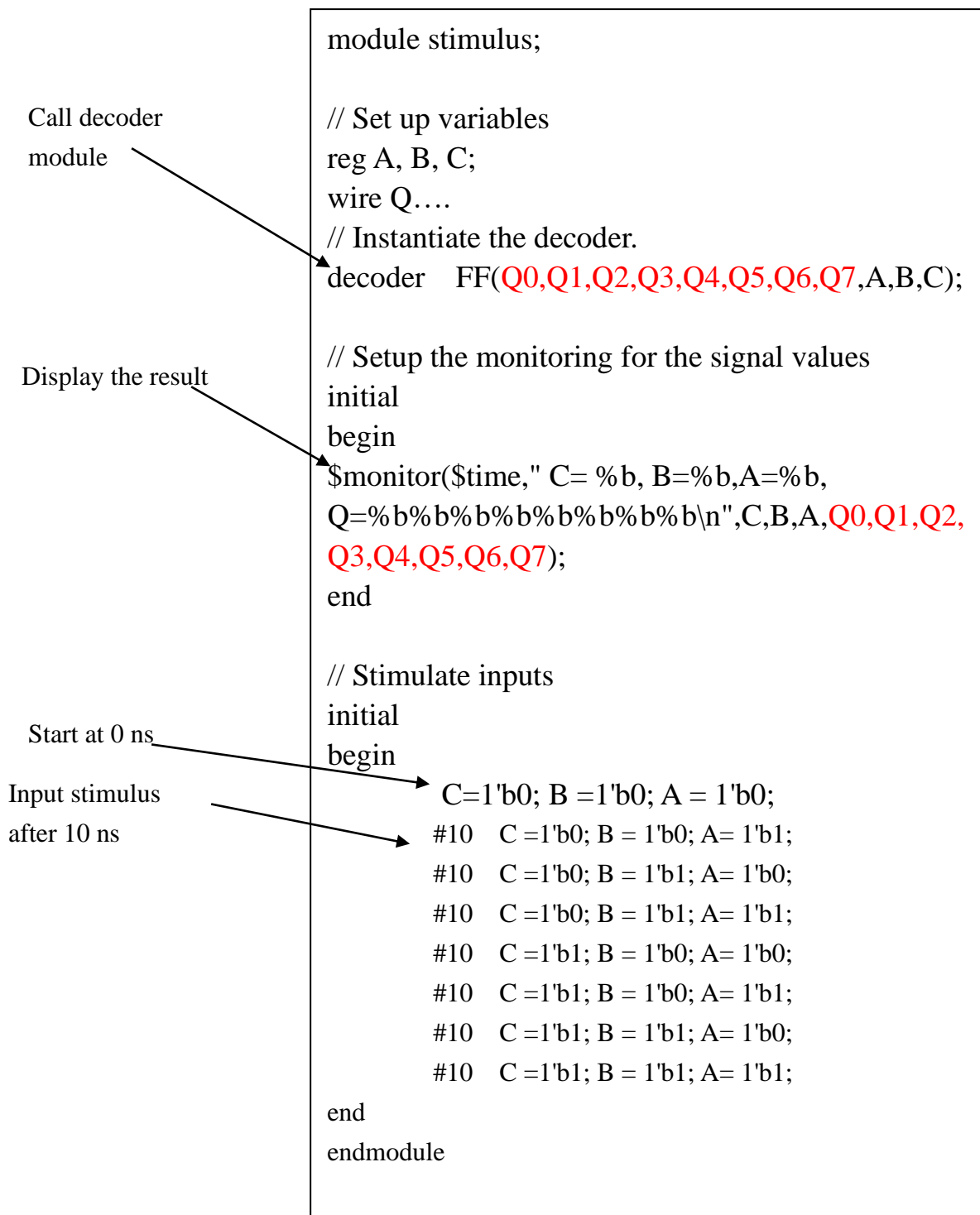
wire s0,s1,s2;

not (s0,A);
not (s1,B);
not (s2,C);
and (Q0,s0,s1,s2);
and (Q1,A,s1,s2);
and (Q2,s0,B,s2);
and (Q3,A,B,s2);
and (Q4,s0,s1,C);
and (Q5,A,s1,C);
and (Q6,s0,B,C);
and (Q7,A,B,C);

endmodule
```

Test Stimulus Code

The design must be checked by applying stimulus is shown below. The module simulates the 3-to-8 decoder by applying a few input combinations and monitors the result.



Simulation Result

The output of the simulation is shown below.

Initial value

```
ready: sim to 0
Highest level modules (that have been auto-instantiated):
(stimulus stimulus
14 total devices.
Linking ...
15 nets total: 23 saved and 0 monitored.
67 registers total: 67 saved.
Done.
LINIT Begins
0 C= 0, B=0,A=0, Q=10000000

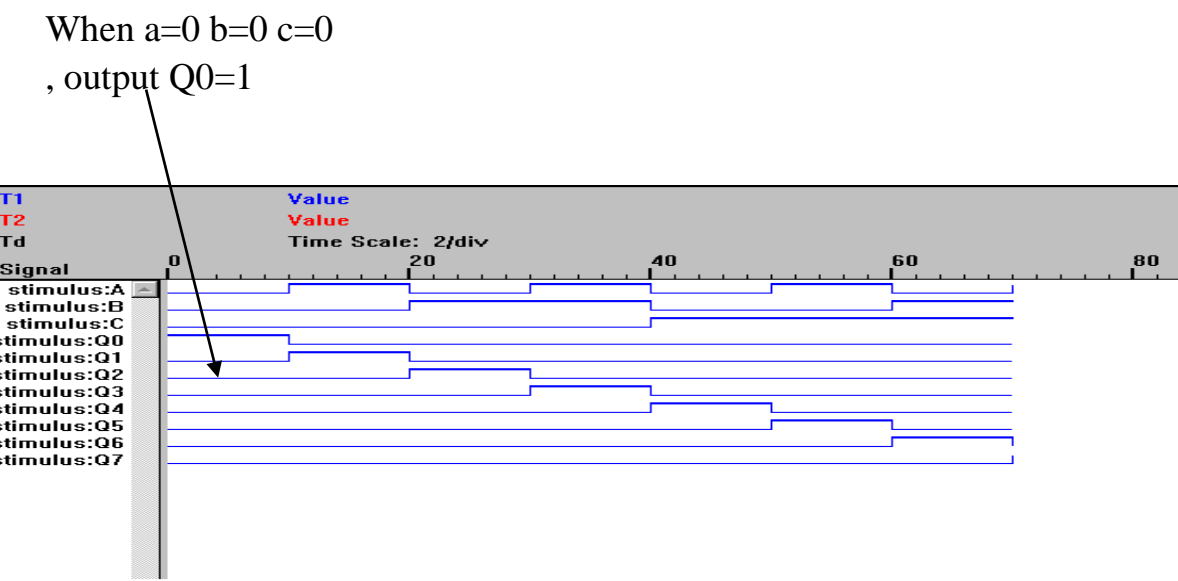
LINIT done

Simulation result
0 State changes on observable nets.
Simulation stopped at the end of time 0.
ready: sim
10 C= 0, B=0,A=1, Q=01000000
20 C= 0, B=1,A=0, Q=00100000
30 C= 0, B=1,A=1, Q=00010000
40 C= 1, B=0,A=0, Q=00001000
50 C= 1, B=0,A=1, Q=00000100
60 C= 1, B=1,A=0, Q=00000010
70 C= 1, B=1,A=1, Q=00000001

47 State changes on observable nets.
Simulation stopped at the end of time 70.
ready:
```

Simulation Waveform

According to truth table, the output result is shown below.



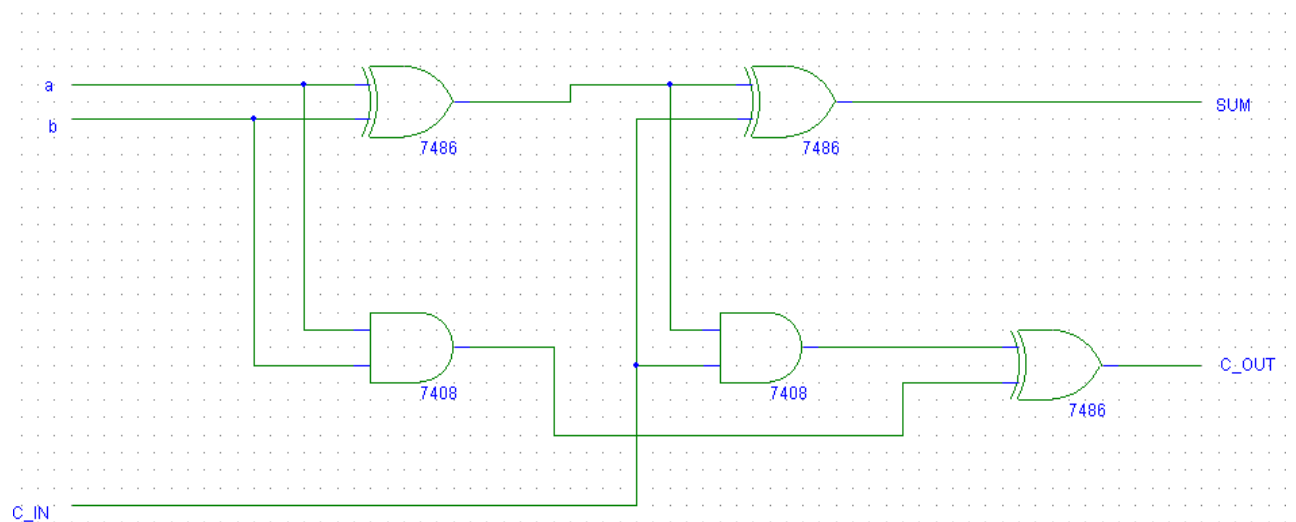
2.2.4 1-bit Full Adder Design

We will implement a ripple carry adder. The basic block building is a 1-bit full adder. The mathematical equation for a 1-bit full adder is shown below.

$$\text{sum} = (a \oplus b \oplus \text{cin})$$

$$\text{cout} = (a \times b) + \text{cin} \times (a \oplus b)$$

The logic diagram for a 1-bit full adder is shown below.



Verilog Code

This logic diagram for the 1-bit full adder is converted to a Verilog description is show below.

I/O port declaration

Internal nets

```
//define a 1bit fulladder
module fulladd(sum, c_out, a, b, c_in);
    output sum, c_out;
    input a, b, c_in;

    wire s1, c1, c2;

    xor  (s1, a, b);
    and  (c1, a, b);
    xor  (sum, s1, c_in);
    and  (c2, s1, c_in);
    xor  (c_out, c2, c1); // or(c_out, c2, c1)

endmodule
```

Test Stimulus Code

The module stimulus simulation the 1-bit full adder and display the result.

Call full adder
module

```
module stimulus;
    //declare variables to be connected
    reg A, B;
    reg C_IN;
    wire SUM;
    wire C_OUT;
    // Instantiate the 1-bit full adder. call it FA1
    fulladd FA1(SUM, C_OUT, A, B, C_IN);
endmodule
```


Setup the monitoring
for signal values

Simulation input

```
// Setup the monitoring for the signal values
initial
begin
$monitor($time," A= %b, B=%b, C_IN= %b,
C_OUT=%b,SUM= %b\n",A,B,C_IN,C_OUT,SUM);
end
initial
begin
A = 1'b0; B = 1'b0; C_IN = 1'b0;
#50 A = 1'b0; B = 1'b0; C_IN=1'b1;
#50 A = 1'b0; B = 1'b1; C_IN=1'b0;
#50 A = 1'b0; B = 1'b1; C_IN=1'b1;
#50 A = 1'b1; B = 1'b0; C_IN=1'b0;
#50 A = 1'b1; B = 1'b0; C_IN=1'b1;
#50 A = 1'b1; B = 1'b1; C_IN=1'b1;
end
endmodule
```

Simulation Result

The output of 1-bit full adder is shown below.

```
Reading "d:\verilog\gate\Fulladder1.v"
Ready: sim to 0
Highest level modules (that have been auto-instantiated):
(stimulus stimulus
8 total devices.
Linking ...
9 nets total: 17 saved and 0 monitored.
67 registers total: 67 saved.
Done.
LINIT begins
0 A= 0, B=0, C_IN= 0, C_OUT= 0,SUM= 0

LINIT done

0 State changes on observable nets.

Simulation stopped at the end of time 0.
Ready: sim
50 A= 0, B=0, C_IN= 1, C_OUT= 0,SUM= 1
100 A= 0, B=1, C_IN= 0, C_OUT= 0,SUM= 1
150 A= 0, B=1, C_IN= 1, C_OUT= 1,SUM= 0
200 A= 1, B=0, C_IN= 0, C_OUT= 0,SUM= 1
250 A= 1, B=0, C_IN= 1, C_OUT= 1,SUM= 0
300 A= 1, B=1, C_IN= 1, C_OUT= 1,SUM= 1

35 State changes on observable nets.

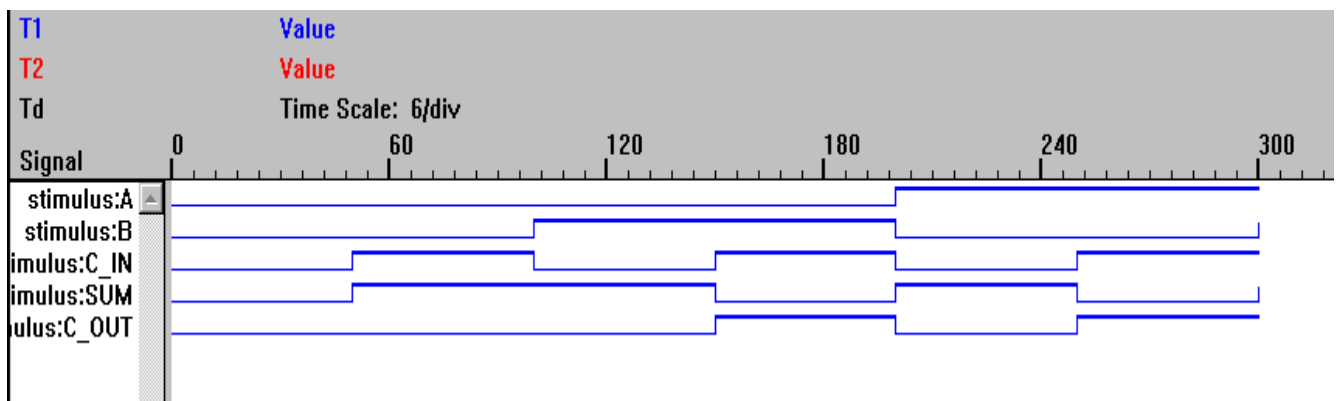
Simulation stopped at the end of time 300.
Ready: |
```

Simulation Waveform

According to mathematical equation for 1-bit full adder, We can get waveform are shown below.

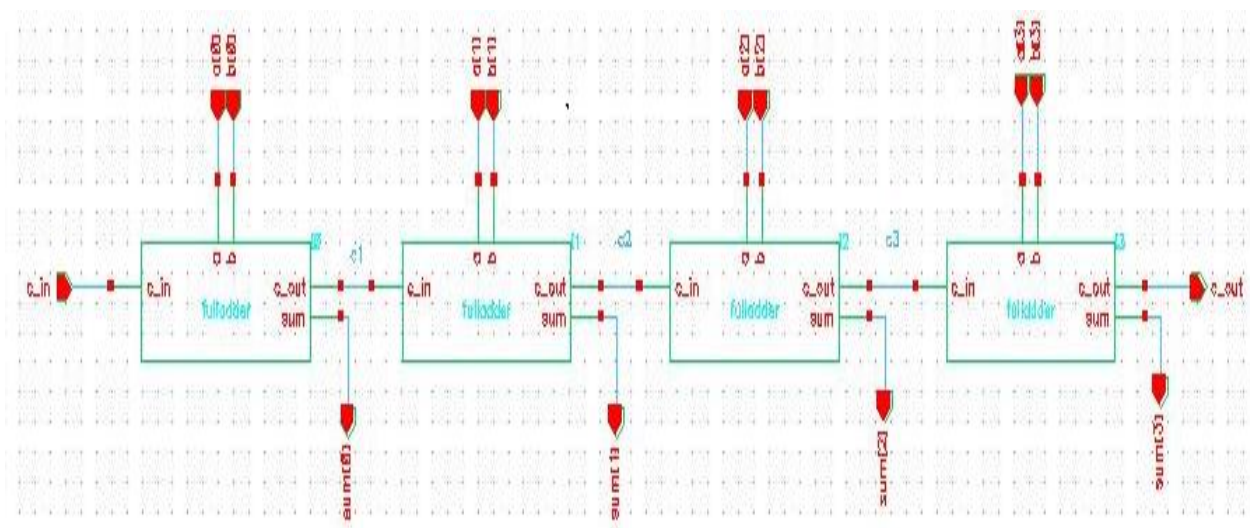
$$\text{sum} = (a \oplus b \oplus \text{cin})$$

$$\text{cout} = (a \times b) + \text{cin} \times (a \oplus b)$$



2.2.5 4-bits Full Adder Design

The 4-bit full adder can be constructed from four 1-bit full adder. The logic diagram is shown below. we again use gate level describe to a 4-bit full adder.



Verilog Code

fa0 fa1 fa2 and fa3 are instances of the module fulladd(1 bits full adder)

Call fulladd module

```
//define a 4-bit full adder
module fulladd4(sum, c_out, a, b, c_in);
//i/o port declaration
output [3:0] sum;
output c_out;
input [3:0] a, b;
input c_in;
//internal net
wire c1, c2, c3;

fulladd fa0(sum[0], c1, a[0], b[0], c_in);
fulladd fa1(sum[1], c2, a[1], b[1], c1);
fulladd fa2(sum[2], c3, a[2], b[2], c2);
fulladd fa3(sum[3], c_out, a[3], b[3], c3);

endmodule
```

Test Stimulus Code

Call fulladder4 module



```
//define the stimulus module
module stimulus;
//declare variables to be connected
reg [3:0] A, B;
reg C_IN;
wire [3:0] SUM;
wire C_OUT;
fulladd4 FA1_4(SUM, C_OUT, A, B, C_IN);
// Setup the monitoring for the signal values
initial
begin
$monitor($time," A= %d, B=%d, C_IN= %d, C_OUT=%d,
SUM= %d\n", A, B, C_IN, C_OUT, SUM);
end
initial
begin
A = 4'h0; B = 4'h0; C_IN = 1'h0;
#50 A = 4'h3; B = 4'h4; C_IN=1'h0;
#50 A = 4'h2; B = 4'h5; C_IN=1'h0;
#50 A = 4'h9; B = 4'h9; C_IN=1'h0;
#50 A = 4'ha; B = 4'hf; C_IN=1'h0;
#50 A = 4'h1; B = 4'h0; C_IN=1'h1;
#50 A = 4'h1; B = 4'h1; C_IN=1'h1;
end
endmodule
```

Simulation Result

In this example, if sum more than 1111 , c_out =1,otherwise c_out=0;

Output after 50n sec

```
Ready: file .sav=flladd4
Ready: input d:\verilog\gate\Fulladd4.v
Reading "d:\verilog\gate\Fulladd4.v"
Ready: sim to 0
  Highest level modules (that have been auto-instantiated):
    (stimulus stimulus
  27 total devices.
  Linking ...
  30 nets total: 38 saved and 0 monitored.
  73 registers total: 73 saved.
  Done.
  LINIT begins
    0 A= 0000, B=0000, C_IN= 0, C_OUT= 0,SUM= 0000

  LINIT done

  0 State changes on observable nets.

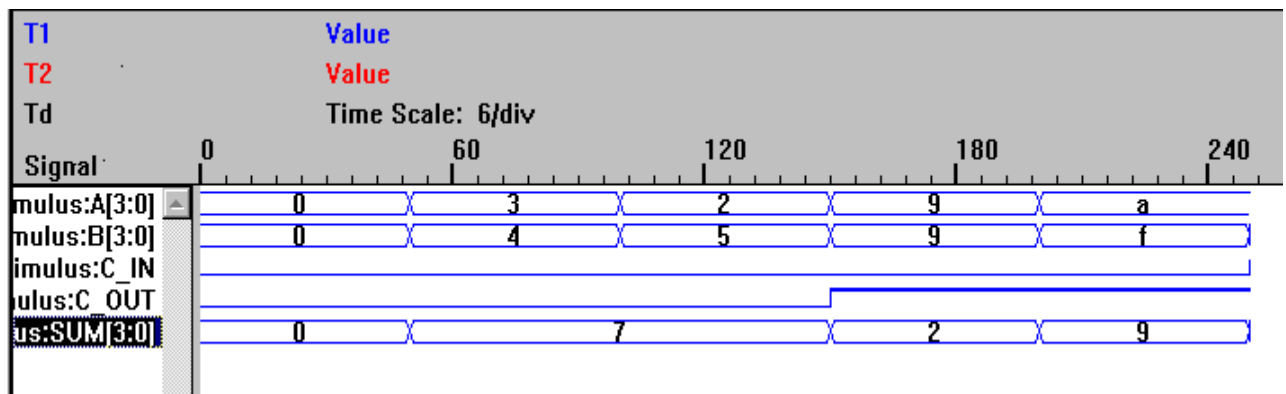
  Simulation stopped at the end of time 0.
Ready: sim
    50 A= 0100, B=0100, C_IN= 1, C_OUT= 0,SUM= 1001
    100 A= 0010, B=0101, C_IN= 0, C_OUT= 0,SUM= 0111
    150 A= 0001, B=0010, C_IN= 1, C_OUT= 0,SUM= 0100
    200 A= 1111, B=0100, C_IN= 0, C_OUT= 1,SUM= 0011
    250 A= 1000, B=1000, C_IN= 1, C_OUT= 1,SUM= 0001

  105 State changes on observable nets.

  Simulation stopped at the end of time 250.
Ready: |
```

Simulation Waveform

Simulation waveform checks 4-bit full adder correctly.



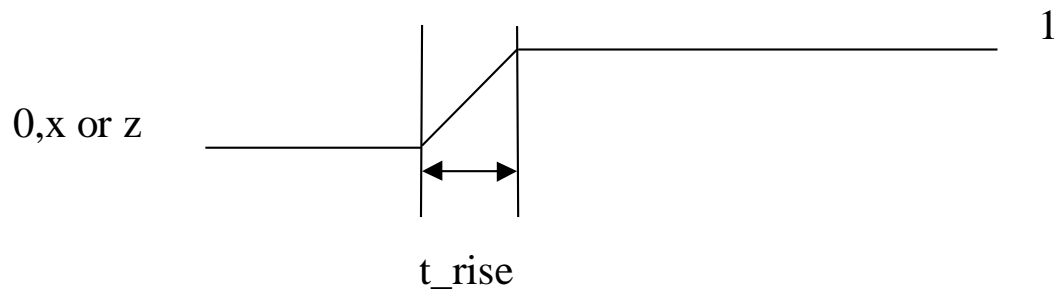
2.3 Gate Delay

2.3.1 Rise, Fall, and Turn-off Delays

There are three types of the delay from the inputs to the output of a primitive gate.

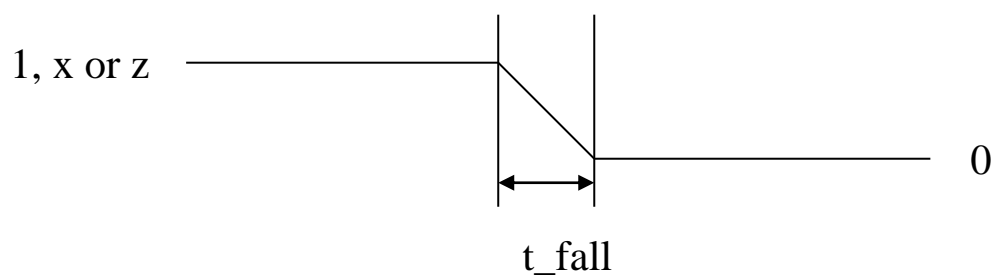
Rise time

The rise delay is associated with a gate output transition to a 1 from another value.



Fall time

The fall delay is associated with a gate output transition to a 0 another value.

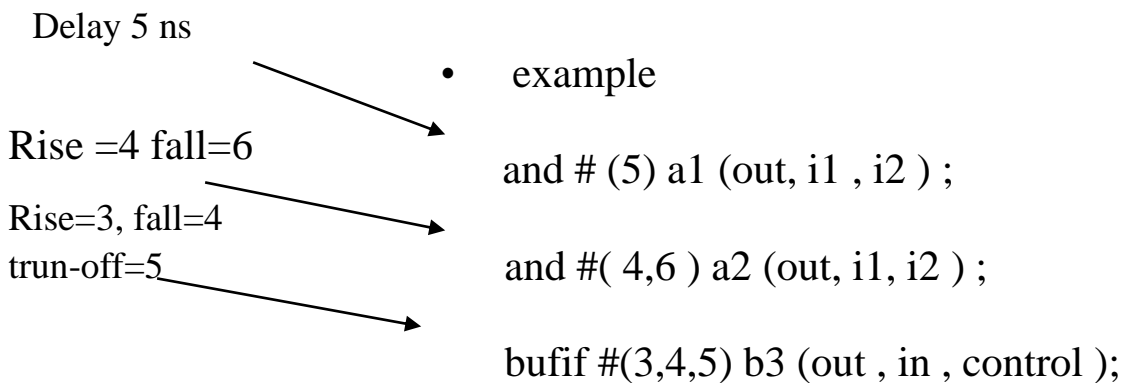


Turn-off delay

The turn-off delay is associated with a gate output transition to the high impedance value from another value.

Syntax list

- `and (delay time) a1(out,i1,i2) ;`
- `and (rise_val , fall_val) a2(out,i1,i2) ;`
- `bufif0 (rise_val , fall_val ,trunoff_val) a3(out,i1,i2) ;`



2.3.1 Specifying Time Units

``timescale <time_unit> / <time_precision>`

Table 6.7 Arguments for `timescale compiler directive

Unit of Measurement	Abbreviation
seconds	s
milliseconds	ms
microseconds	us
nanoseconds	ns
picoseconds	ps
femtoseconds	fs

Ex. **``timescale 10ns/0.1ns`**

Table 6.8 Time delay / precision specifications

Unit / precision	Delay specification	Time delayed	Comments
10ns / 1ns	#7	70ns	The delay is 7*time_unit,or 70ns
10ns / 1ns	#7.748	77ns	7.748 is rounded to one decimal place(due to the difference between 10ns and 1ns)and multiplied by the time_unit
10ns / 0.1ns	#7.748	77.5ns	7.748 is rounded to two decimal places and multiplied by the time_unit
10ns / 1ns	#7.5	75ns	7.5 is rounded to one decimal place and multiplied by 10
10ns / 10ns	#7.5	80ns	7.5 is rounded to the nearest integer(no decimal places)and multiplied by 10

2.3.2 Min / Typical /Max Values

Min value

The min vale is the minimum delay value that the designer expects the gate to have.

Typical value

The type value is the typical delay value that the designer expects the gate to have.

Max value

The max value is the maximum delay value that the designer expects the gate to have.


```

//min delay=4
//type delay=5
//max delay=6
and #(4:5:6) a1(out, i1, i2) ;

//min delay, rise=3 , fall =5
//type delay, rise=4 , fall =6
//max delay, rise=5 , fall =7
and #(3:4:5, 5:6:7) a2(out, i1, i2 ) ;

//min delay, rise=2 , fall =3 , turn-off =4
//type delay, rise=3 , fall =4 , trun-off=5
//max delay, rise=4 , fall =5 , trun-off=6

and #(2:3:4, 3:4:5, 4:5:6) a3(out, i1, i2 ) ;

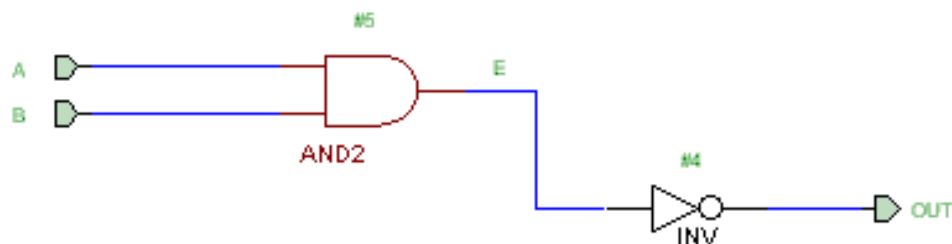
```

Delay Example

Let us consider a simple example to the use of gate delays model timing in the logic circuits.

A sample module is shown in the following logic equation.

Out = a b + c



Verilog Code

The net e is internal wire, this is output of and gate with delay 7n sec.

The net out is output of not gate with delay 3n sec.

Delay of the 5
on gate a1

Delay of 4 on
gate o1

Test Stimulus

```
// Define a simple combination module called D
module D (out, a, b);
// I/O port declarations
output out;
input a,b;
// Internal nets
wire e;
// Instantiate primitive gates to build the circuit
and #(7) a1(e, a, b);
not #(3) o1(out, e );
endmodule
```

Code

Simulate the
input

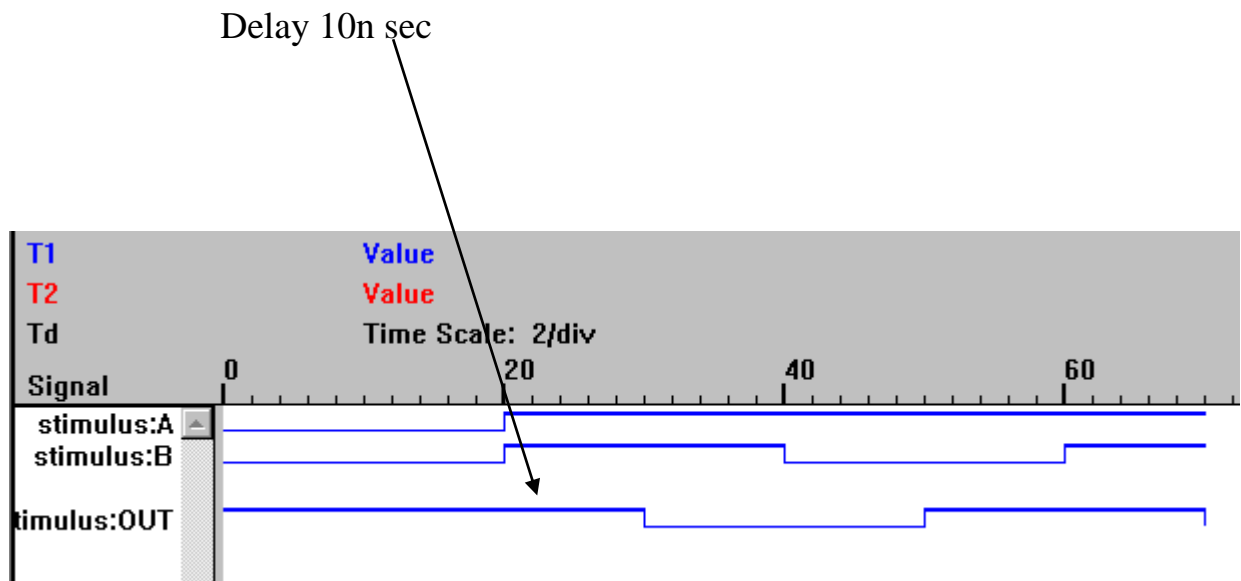
```
module stimulus;
reg A, B, C;
wire e;
wire OUT;

D d1( OUT, A, B, C);

initial
$monitor($time, " A= %b, B=%b, OUT= %b\n",A, B, OUT);
initial
begin
A= 1'b0; B= 1'b0;
#20 A= 1'b1; B= 1'b1;
#20 A= 1'b1; B= 1'b0;
#20 A= 1'b1; B= 1'b1;
end
endmodule
```

Simulation Waveform

The waveforms from the simulation are shown below to illustrate the effect of specifying delays on gates. Because delay time is 7n sec from input to the net, and delay time is 3n sec from the net to output, so delay time is 10n sec from input to output.



Chapter 5 Exercises: 2, 3, 4, 5

Homework-1 ALU (Arithmetic Logic Unit) 算術邏輯單元

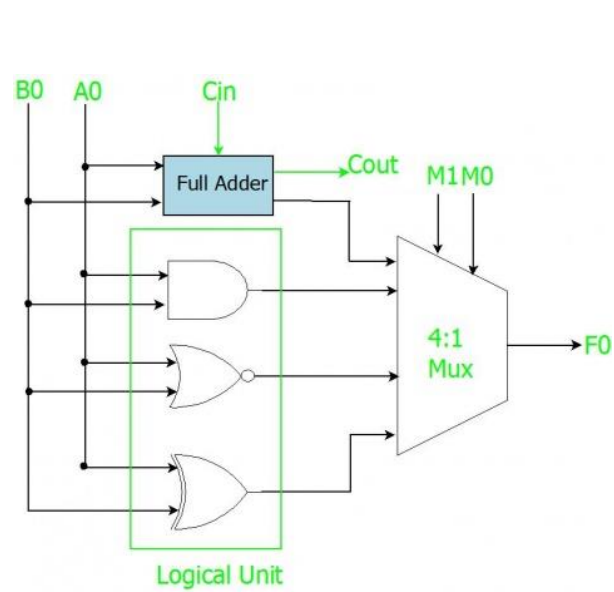
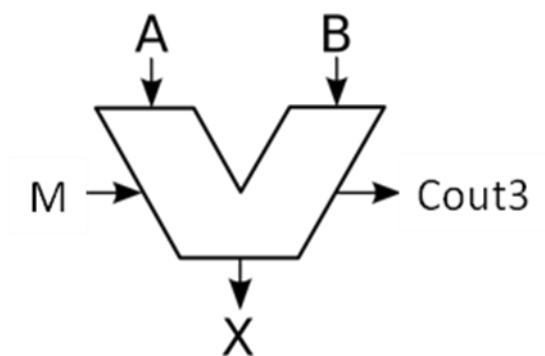
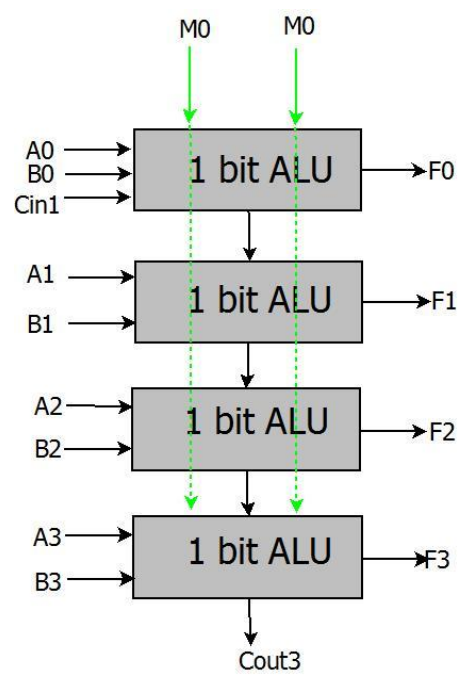


Figure: 1 bit ALU



Homework-2

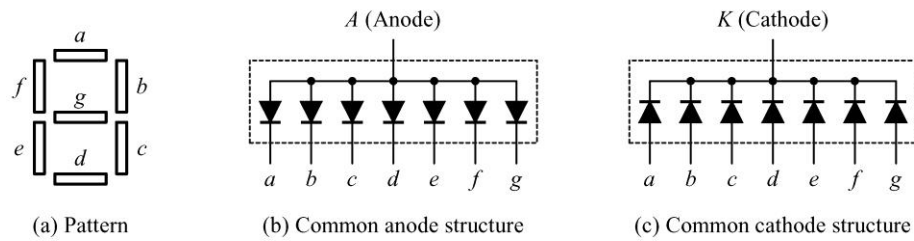
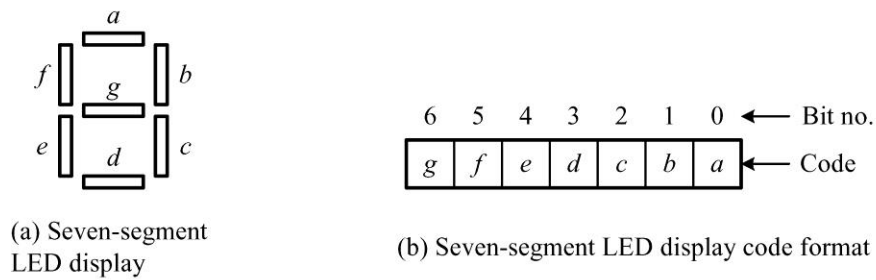


Figure 8.16: The structures of seven-segment LEDs.



Digit	<i>g</i>	<i>f</i>	<i>e</i>	<i>d</i>	<i>c</i>	<i>b</i>	<i>a</i>	Code	Digit	<i>g</i>	<i>f</i>	<i>e</i>	<i>d</i>	<i>c</i>	<i>b</i>	<i>a</i>	Code
0	1	0	0	0	0	0	0	40	8	0	0	0	0	0	0	0	00
1	1	1	1	1	0	0	1	79	9	0	0	1	0	0	0	0	10
2	0	1	0	0	1	0	0	24	<i>A (a)</i>	0	0	0	1	0	0	0	08
3	0	1	1	0	0	0	0	30	<i>B (b)</i>	0	0	0	0	0	1	1	03
4	0	0	1	1	0	0	1	19	<i>C (c)</i>	1	0	0	0	1	1	0	46
5	0	0	1	0	0	1	0	12	<i>D (d)</i>	0	1	0	0	0	0	1	21
6	0	0	0	0	0	1	0	02	<i>E (e)</i>	0	0	0	0	1	1	0	06
7	1	1	1	1	0	0	0	78	<i>F (f)</i>	0	0	0	1	1	1	0	0E

(c) Seven-segment LED display code

Figure 8.18: The relationship between digit and common-anode seven-segment LED display code.

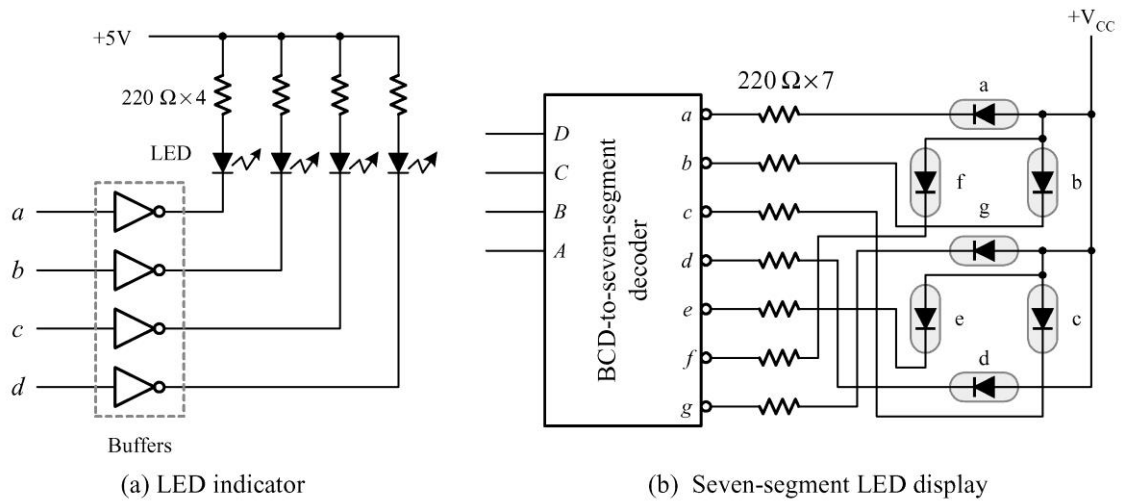


Figure 8.17: Examples of LED indicators and a seven-segment LED display circuit.

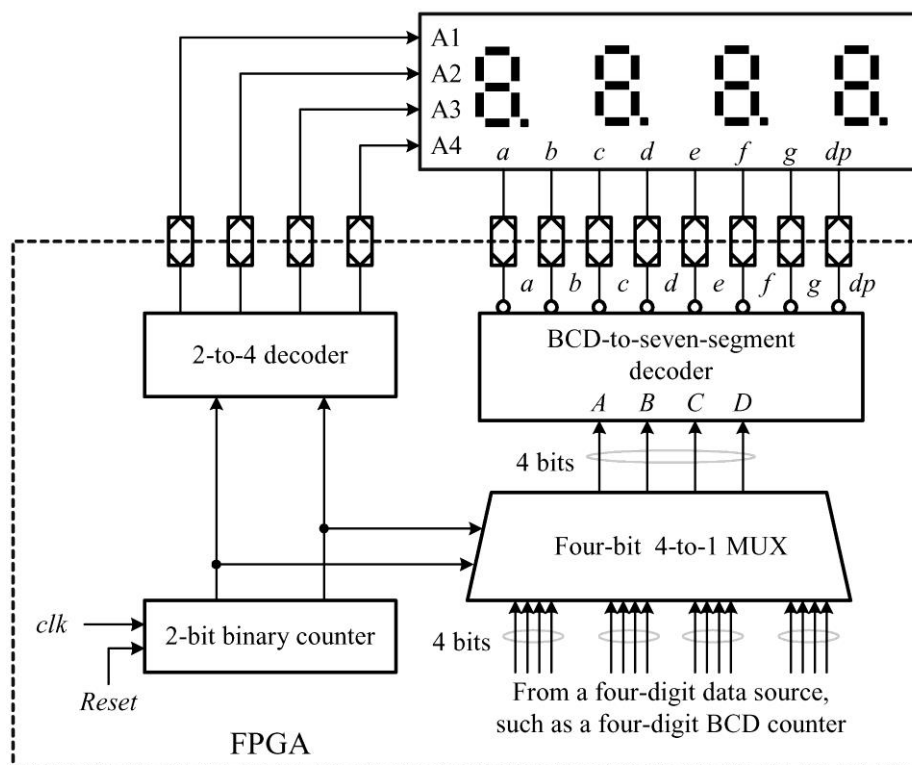
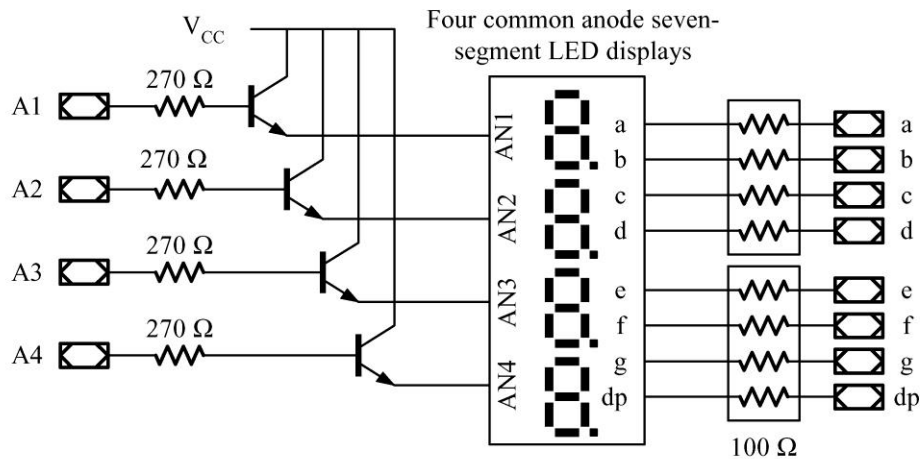
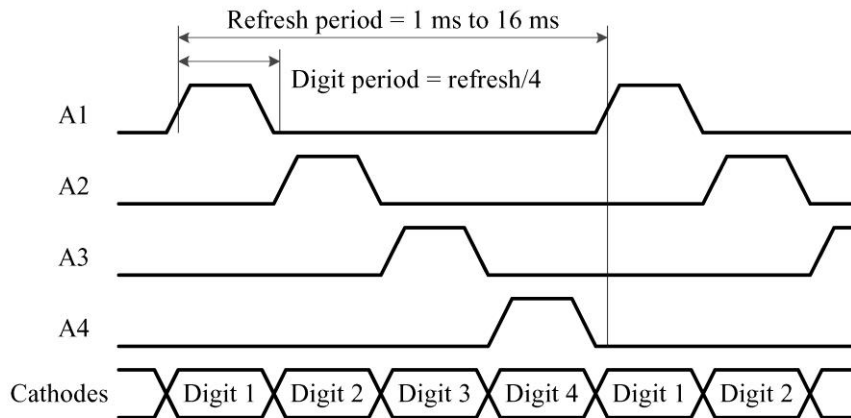


Figure 8.20: The complete logic circuit of a four-digit multiplexing-driven seven-segment LED display.



(a) A four-seven-segment LED display using multiplexing technique



(b) Timing diagram for the four-seven-segment LED display shown in (a)

Figure 8.19: The logic circuit and timing diagram of a four-digit multiplexing-driven seven-segment LED display.

1. Consider a common-cathode seven-segment LED display:

- Define the seven-segment LED display code.
- Write a binary-to-seven-segment decoder module.
- Write a 2-to-4 Decoder module.
- Write a 4-bit 4-to-1 multiplexer module.

Truth Table for BCD-to-7 segment converter

A	B	C	D	a	b	c	d	e	f	g	dp
0	0	0	0	0	0	0	0	0	0	1	1
				0	1	0	0	0	0	0	0
				0	0	1	0	0	0	0	0
				0	0	0	1	0	0	0	0
				0	0	0	0	1	0	0	0
				0	0	0	0	0	1	0	0
				0	0	0	0	0	0	1	0
1	1	1	1	0	0	0	0	0	0	0	1