$y \backslash x_1x_2$	00	01	11	10
a	$\textcircled{a}$, 0	$\textcircled{a}$, 0	$\textcircled{a}$, 0	b, -
b	$\textcircled{b}$, 1	a, -	$\textcircled{b}$, 1	$\textcircled{b}$, 1

(d) When the input is 01, the output is 0. When the input is 10, the output is 1. Whenever the input assumes one of the other two combinations, the output retains its previous value.

9.4

$$Y_1 = x_1x_2 + x_1y'_2 + x'_2y_1$$

$y_1y_2 \backslash x_1x_2$	00	01	11	10
00	$\textcircled{00}$	01	11	10
01	00	01	11	$\textcircled{01}$
11	$\textcircled{11}$	01	$\textcircled{11}$	10
10	11	01	11	$\textcircled{10}$

Transition Table

$$Y_2 = x_2 + x_1y'_1y' + x'_1y_1$$

x_1x_2

$y_1y_2 \backslash x_1x_2$	00	01	11	10
00	0	1	1	0
01	0	1	1	0
11	1	1	1	1
10	1	1	1	1

Output Map

$$z = x_2 + y_1$$

y_1y_2	00	01	11	10
a = 00	$\textcircled{a}$, 0	b, 1	c, 1	d, 0
b = 01	a, 0	$\textcircled{b}$, 1	c, 1	$\textcircled{b}$, 0
c = 11	$\textcircled{c}$, 1	b, 1	$\textcircled{c}$, 1	d, 1
d = 10	c, 1	b, 1	c, 1	$\textcircled{d}$, 1

Flow Table

9.5

y_1y_2 \ x_1x_2	00	01	11	10
$a = 00$	00	11	01	00
$c = 01$	00	11	01	01
$b = 11$	00	11	11	01
10	00	11	x	x

Noncritical races

y_1y_2 \ x_1x_2	00	01	11	10
00	0	0	1	1
01	0	11	1	0
11	0	0	11	0
10	0	0	x	x

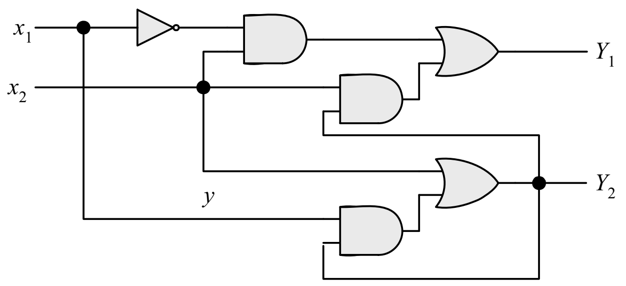
Output map: $z = x_1x_2y'_1 + x_1y'_2$

y_1y_2 \ x_1x_2	00	01	11	10
00		1		
01		1		
11		1	1	
10		1	x	x

$Y_1 = x'_1x_2 + x_2y_2$

y_1y_2 \ x_1x_2	00	01	11	10
00		1	1	
01		1	1	1
11		1	1	1
10		1	x	x

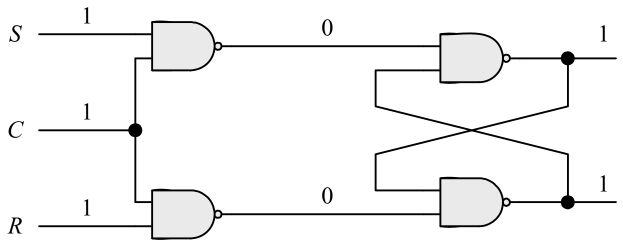
$Y_2 = x_2 + x_1y_2$



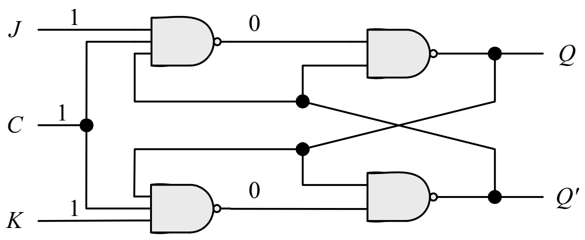
9.6

```
From total            To total
state  y1 y2 x1 x2    state  y1 y2 x1 x2
 0 0 0 1  ───────→  1 1 1 1
          ───────→  1 0 1 1                      ┐ Critical race
          ───────→  0 1 1 1        1 0 1 1       ┘
 0 0 0 1  ───────→  1 0 1 0 ────→ 1 1 0 0 ────→ 0 1 0 0   Cycle
 0 1 0 0  ───────→  1 0 1 0
          ───────→  0 0 1 0 ────→ 1 0 1 0   Critical race
          ───────→  1 1 1 0
 1 1 1 1  ───────→  0 0 0 1
          ───────→  0 1 0 1 ────→ 0 0 0 1   Noncritical race
          ───────→  1 0 0 1 ────→ 0 0 0 1
 1 0 1 0  ───────→  1 1 0 0 ────→ 0 1 0 0   Cycle
          ───────→
```

9.7 When all the inputs are equal to 1, both outputs are equal to 1. (See Fig. 9.11). Going from $SR = 11$ to $SR = 00$ when $C = 1$ produces an unstable output.



9.8 When $C = 1$ and $J = K = 1$,the outputs complement and again repeat complementing as long as C -= 1.



9.9 **(a)**

$$S_1 = x'_1 \qquad R_1 = y'_2$$
$$S_2 = x'_1 y'_1 \qquad R_2 = x'_2$$

$$Y_1 = S'_1 + R_1 y_1 = x_1 + y_1 y'_2$$
$$Y_2 = S'_2 + R_2 y_2 = x_1 + y_1 + x'_2 y_2$$

$$Q = Y_2$$

(b)

y_1y_2 \ x_1x_2	00	01	11	10
00	$\textcircled{00}$	$\textcircled{00}$	11	11
01	$\textcircled{01}$	00	11	11
11	01	01	$\textcircled{11}$	$\textcircled{11}$
10	00	11	11	11

Transition table

y_1y_2 \ x_1x_2	00	01	11	10
00	0	0	1	1
01	1	0	1	1
11	1	1	1	1
10	1	1	1	1

Output map

9.10

$$Y = x_1x'_2 + (x_1 + x'_2)y = S + R'y$$

y \ x_1x_2	00	01	11	10
0	0	0	0	0
1	1	1	1	1

Transition map

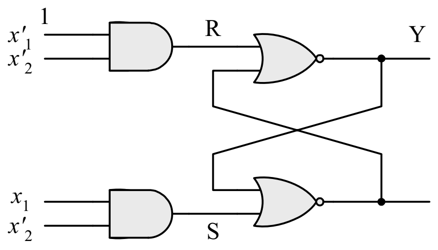
y	Y	S	R
0	0	0	x
0	1	1	0
1	0	0	1
1	1	x	0

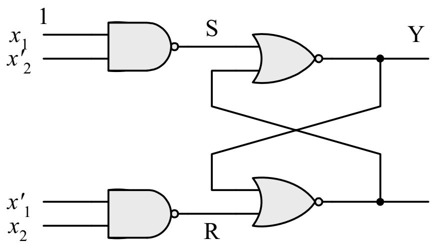
y \ x_1x_2	00	01	11	10
0	0	0	0	0
1	1	1	1	1

$S = x_1x'_2$

y \ x_1x_2	00	01	11	10
0	0	0	0	0
1	1	1	1	1

$R = x'_1x_2$





9.11 Use solution of Prob. 9.4 transition table.

y_1y_2 \ x_1x_2	00	01	11	10
00	0	0	1	1
01	0	0	1	0
11	x	0	x	x
10	x	0	x	x

$S_1 = x_1x_2 + x_2y'_2$

y_1y_2 \ x_1x_2	00	01	11	10
00	x	x	0	0
01	x	x	0	x
11	0	1	0	0
10	0	1	0	0

$R_1 = x'_1x_2$

y_1y_2 \ x_1x_2	00	01	11	10
00	0	1	1	0
01	0	x	x	x
11	x	x	x	0
10	1	1	1	0

$S_2 = x_2 + x'_1y_1$

y_1y_2 \ x_1x_2	00	01	11	10
00	x	0	0	x
01	1	0	0	0
11	0	0	0	1
10	0	0	0	x

$R_2 = x'_1x'_2y'_1 + x_1x'_2y_1$

The circuit may be drawn as in Fig. 9.38(b) or Fig. 9.46. It will require 6 NAND gates for the inputs to the latches, 4 NAND gates for the two latches, and one NAND gate for the out put $z = x_2 + y_1 = (x'_2y'_1)'$.

9.12

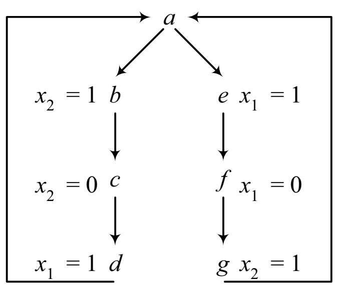
	x_1x_2 = 00	01	11	10	
a	ⓐ,00	b,-	-,-	c,-	Initial state
b	a,-	ⓑ,00	d,-	-,-	No change of output
c	a,-	-,-	g,-	ⓒ,00	No change of output
d	-,-	f,-	ⓓ,10	e,-	x_1x_2 from 01 to 11 [Condition (c)]
e	a,-	-,-	g,-	ⓔ,10	No change of output
f	a,-	ⓕ,10	g,-	-,-	No change of output
g	-,-	j,-	ⓖ,01	h,-	x_1x_2 from 10 to 11 [Condition (b)]
h	a,-	-,-	g,-	ⓗ,01	No change of output
j	a,-	ⓙ,01	d,-	-,-	No change of output

9.13

State	Switch input x_1x_2	Output light (red = 1) z	Comments
a	0 0	0	No train in intersection
b	0 1	1	Train turns on x_2 after state a
c	0 0	1	Train between two switches
d	1 0	1	Train turns on x_1 after state c
e	1 0	1	Train trains on x_1 after state a
f	0 0	1	Train between two switches
g	0 1	1	Train turns on x_2 after state f

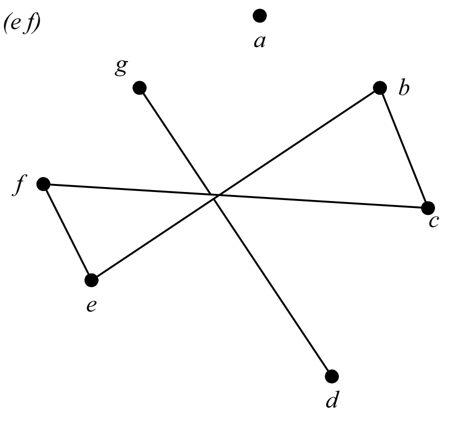
$z = x_2 + y_1$

y_1y_2	00	01	11	10
a	a, 0	b, -	-, -	e, -
b	c, -	b, 1	-, -	-, -
c	c, 1	-, -	-, -	d, -
d	a, -	-, -	-, -	d, 1
e	f, -	-, -	-, -	e, 1
f	f, 1	g, -	-, -	-, -
g	a, -	d, 1	-, -	-, -



b	*a, c x*					
c	*x*	√				
d	*d, e x*	*a, c x*	*a, c x*			
e	*a. f x*	*c, f* √	*c, f d, e x*	*a, f x*		
f	*x*	*c, f b g x*	√	*a, f x*	√	
g	*b g x*	*a, c x*	*a, c x*	√	*a, f x*	*x*
	a	*b*	*c*	*d*	*e*	*f*

(b, c) (b, e) c, f) (d, g) (e f)



There are two possibilities for a four-row flow table:
(a) *(b, c) (d, g) (e, f)* or *(a) (b, e) (c, f) (d, g)*. Continue with the solution of Prob. 9.25.

9.14 **(b)**

$y \backslash x_1x_2$	00	01	11	10
a = 0	0	0	1	0
b = 1	0	1	1	1

$Y = yx_1 + yx_2 + x_1x_2$

$y \backslash x_1x_2$	00	01	11	10
0	0	1	x	1
1	0	0	0	0

$z = y'(x_1 + x_2)$

9.15

	00	01	11	10
00	0	x	x	x
01	0	1	x	x
11	0	x	1	0
10	x	1	1	1

(a)

	00	01	11	10
00	0	0	x	0
01	0	0	1	1
11	x	1	1	1
10	0	1	1	0

(b)

9.16

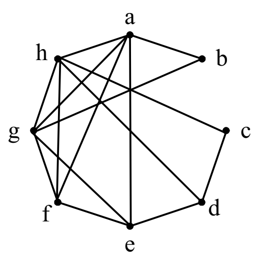
b	*a, c x* *b, d x*			
c	*b, d x*	*a, c x*		
d	*x*	*x*	*x*	
e	*x*	*x*	*x*	*a, e x*
	a	*b*	*c*	*d*

9.17

b	*b, c x* *d, f x*						
c	*b, e* ✓	*d, f x* *c, e x*					
d	*x*	*x*	*x*				
e	*b, c x* *d, f x*	✓	*d, f x*	*x*			
f	*x*	*x*	*x*	*x*	*x*		
g	*x*	*x*	*x*	*x*	*x*	*x*	
h	*x*	*x*	*x*	✓	*x*	*x*	*x*
	a	*b*	*c*	*d*	*e*	*f*	*g*

(a, c) (b, e) (d, h) (f) (g)
See solution to Prob. 6.14 for the reduced state table.

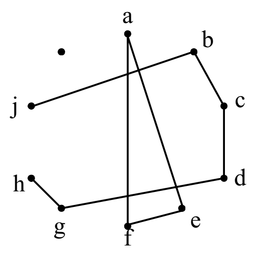
9.18 (a)



1: (a, b) (a, e) (a, f) (a, g) (a, h)
(b, g) (b, h) (c, d) (c, h)
(d, e) (d, h) (e, f) (e, g) (e, h)
(f, g) (f, h) (g, h)

2, 3: (a, b) (c, d) e, f, g, h)

(b)



1: (a, e) (a, f) (b, c) (b, j) (c, d)
(d, g) (e, f) (g, h)

2: (a, e, f) (b, c) (b, j) (c, d)
(d, g) (g, h) (k)

3: (a, e, f) (b, j) (c, d) (g, h) (k)

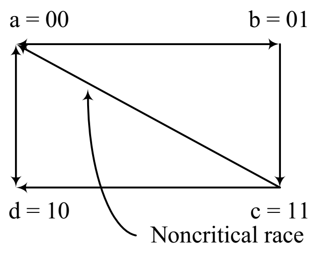
9.19

$y_1y_2 \backslash x_1x_2$	00	01	11	10
a = 00	00	00	01	10
b = 01	00	01	01	11
c = 11	00	xx	10	11
d = 10	00	00	10	10

Transition table

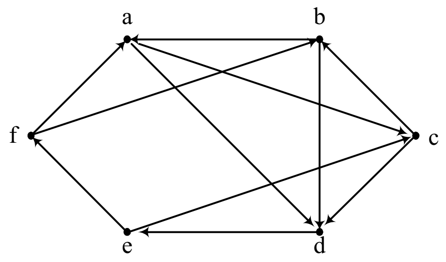
$y_1y_2 \backslash x_1x_2$	00	01	11	10
00	0	1	x	x
01	0	0	0	0
11	0	x	x	0
10	x	1	1	1

Output map = z



$S_1 = x_1x'_2$
$R_1 = x'_1$
$S_2 = x_1x_2y'_1$
$R_2 = x'_1x'_2 + x_2y_1$
$z = x_2y'_2 + x_1y'_2$

9.20



Transition diagram

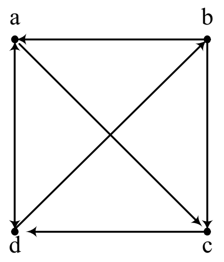
	00	01	11	10
0	a	g	b	f
1	c	h	d	e

Binary assignment
(Add states g and h)

	00	01	11	10
000 = a	$\textcircled{a}$	$\cancel{d}$ g	$\textcircled{a}$	c
011 = b	$\cancel{a}$ g	$\textcircled{b}$	$\textcircled{b}$	d
100 = c	$\cancel{d}$ h	$\textcircled{c}$	$\cancel{b}$ h	$\textcircled{c}$
111 = d	$\textcircled{d}$	$\textcircled{d}$	e	$\textcircled{d}$
110 = e	f	c	$\textcircled{e}$	c
010 = f	$\textcircled{f}$	b	a	$\textcircled{f}$
001 = g	a	h	b	-
101 = h	d	d	g	-

Modified flow table

9.21



Transition diagram

	00	01	11	10
0	a_1	b_1	c_1	d_1
1	c_2	d_2	a_2	b_2

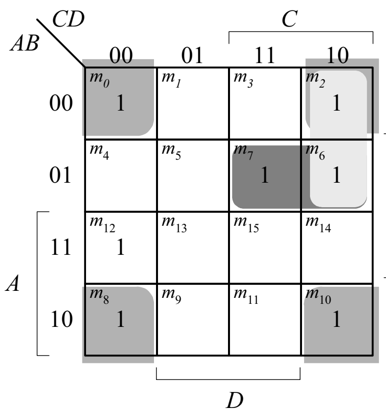
Binary assignment

	00	01	11	10
000 = a_1	$\textcircled{a_1}$	c_2	$\textcircled{a_1}$	d_1
011 = a_2	$\textcircled{a_2}$	c_1	$\textcircled{a_2}$	d_2
100 = b_1	a_1	$\textcircled{b_1}$	c_1	$\textcircled{b_1}$
111 = b_2	a_2	$\textcircled{b_2}$	c_2	$\textcircled{b_2}$
110 = c_1	$\textcircled{c_1}$	$\textcircled{c_1}$	$\textcircled{c_1}$	d_1
010 = c_2	$\textcircled{c_2}$	$\textcircled{c_2}$	$\textcircled{c_2}$	d_2
001 = d_1	$\textcircled{d_1}$	b_2	a_1	$\textcircled{d_1}$
101 = d_2	$\textcircled{d_2}$	b_1	a_2	$\textcircled{d_2}$

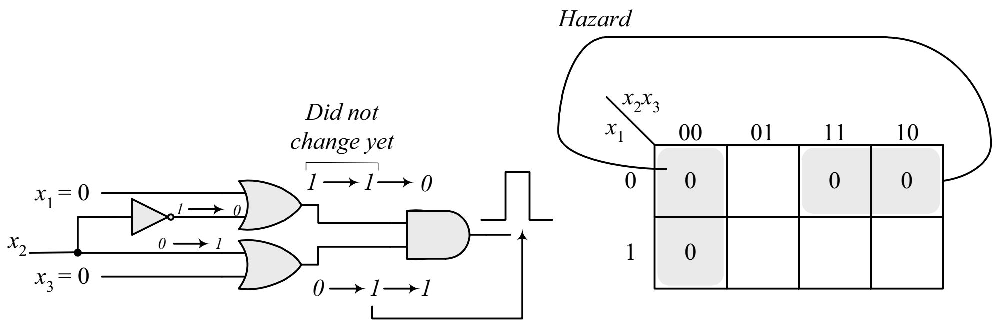
Flow table

9.22 $F(A, B, C D) = \Sigma(0, 2, 6, 7, 8, 10, 12)$

$F = A'D' + AC'D' + A'BC + A'CD'$ (Note: the term A'CD' is redundant)



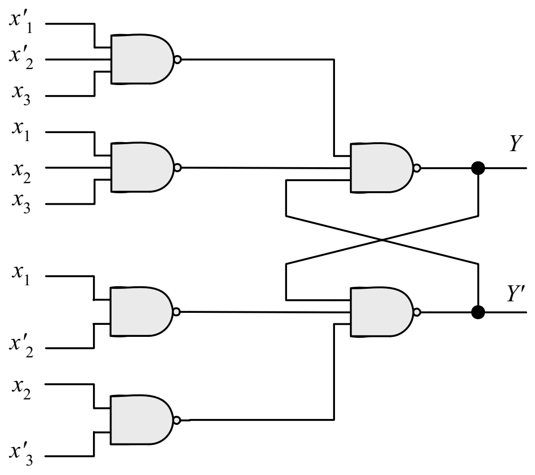
9.23



$Y = (x_1 + x_2')(x_2 + x_3)(x_1 + x_3)$

Add a third term

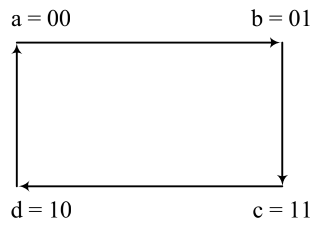
9.24

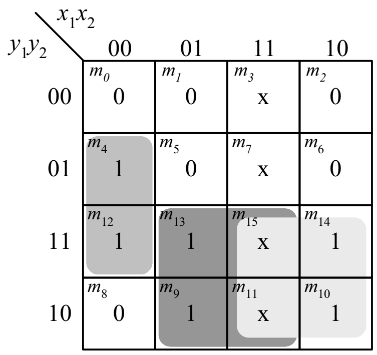


9.25 Continued from solution to Prob. 9.13. The four row table obtained from the compatibles (a) (b, c) (d, g) (e, f) requires three state variables because there is a critical race. Use (a) (b, e) (c, f) (d, g).

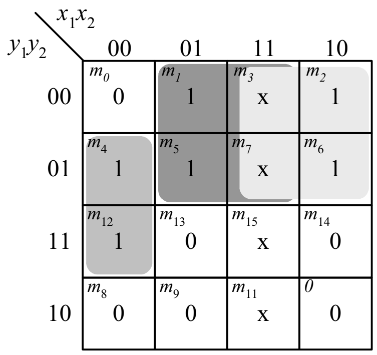
	00	01	11	10
00 = a	a, 0	b, -	-, -	b, -
01 = b	c, -	b, 1	-, -	b, 1
11 = c	c, 1	d, -	-, -	d, -
10 = d	a, -	d, 1	-, -	d, 1

Transition diagram

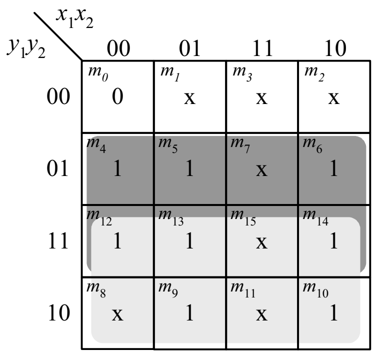




$Y_1 = x_1y_1 + x_2y_1 + x'_1x'_2y_2$



$Y_2 = x_1y'_1 + x_2y'_1 + x'_1x'_2y_2$



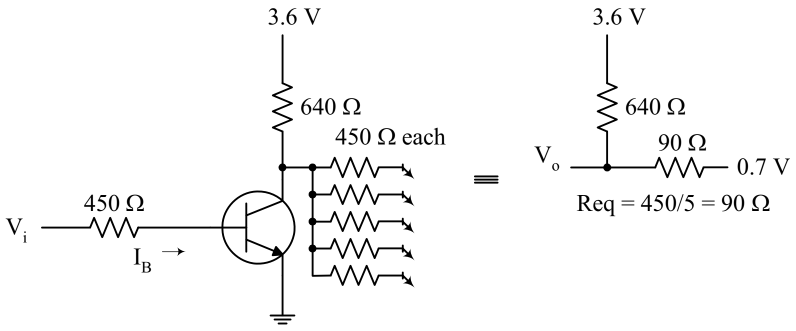
Output map

$z = y_1 + y_2$

CHAPTER 10

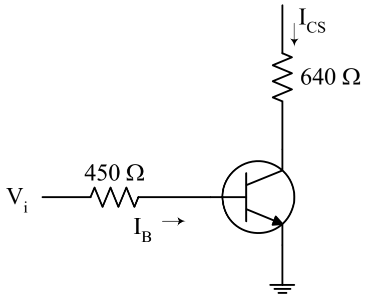
10.1 Fan-out = $I_{OL}/I_{IL} = 20/2 = 10$; $I_{OH}/I_{IH} = 1/0.050 = 20$
Power dissipation 5 x (10 + 20)/2 = 75 mW for 4 gates, 75/4 = 18.75 mW per gate
Propagation delay = 3 ns Noise margin = 2.7 -2 = 0.7 or 0.8 =0.5 = 0.3 V

10.2



(a) Vo = 90/(640 + 90) x 3.6 + 0.7 x 640/(640 + 90) = 0.444 + 0.614 = 1.058 V

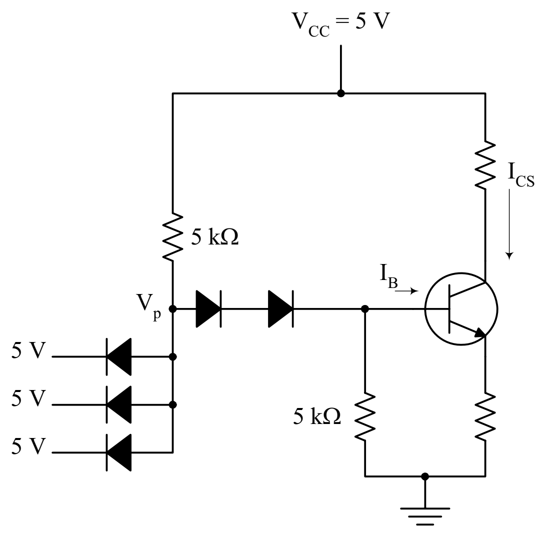
(b)



$I_{CS}/h_{FE} = (3.6 – 0.2)/640/20 = 0.266$ mA
For saturation: $I_B = (V_i – 0.7)/450 \geq 0.266$ mA
$V_i \geq 0.266 \times 10^{-3} \times 450 + 0.7 = 0.82$ V

(c) 1.058 – 0.82 = 0.238 V

10.3



Vp = 0.7 + 0.7 + 0.7 = 2.1 V (Input diodes are off)

I_B = (5 – 2.1)/5000 – 0.7/5000 = 0.58 – 0.14 = 0.44 mA
I_{CS} = (5 – 0.2)/2000 = 2.4 mA

The transistor is saturated because I_b = 0.44 mA $\geq$ I_{CS}/h_{FE} = 2.4/20 mA = 0.12 mA

10.4 **(a)** I_R = (5 – 0.2). 2K = 2.4 mA

(b) I_L = (5 – 0.9)/ rK = 0.82 mA

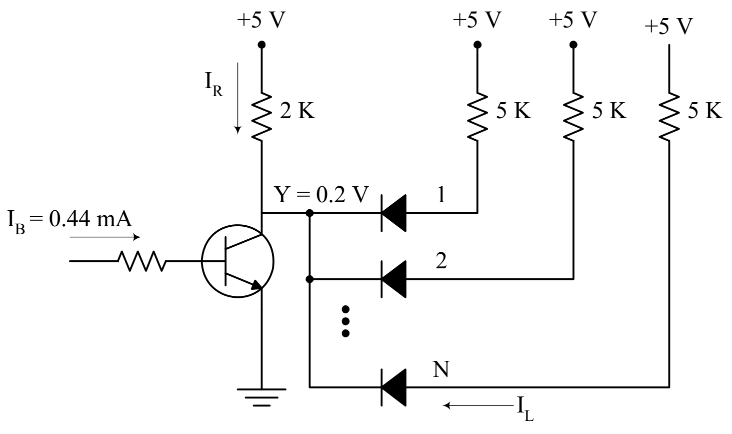
(c) I_{CS} = I_R + N I_L = 2.4 + 0.82*N*

(d) I_B h_{FE} = 0.44 x 20 = 8.8 mA

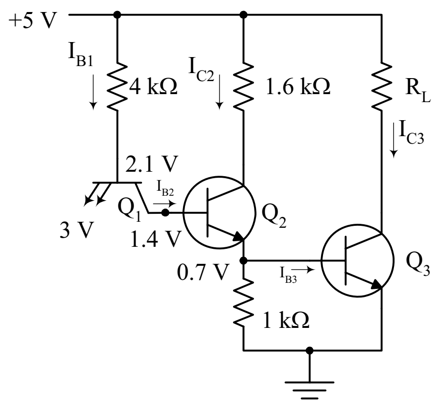
I_{CS} = 2.4 + 0.82 N $\leq$ I_B h_{FE} = 8.8

N $\leq$ (8.9 – 2.4)/0.82 = 7.8

(e) Fanout = 7



10.5



(a) $V_B(Q_1) = 2.1$ V $V_C(Q_1) = 1.4$ V
$V_B(Q_2) = 1.4$ V $V_C(Q_2) = 0.9$ V
$V_B(Q_3) = 0.7$ V $V_C(Q_3) = 0.2$ V
$V_E(Q_1) = 3.0$ V $V_E(Q_2) = 0.7$ V

(b) $I_{B1} = I_{B2} = (5 – 2.1)/4000 = 0.725$ mA
$I_{C2} = (5 – 0.9)/1600 = 2.56$ mA
h_{FE} (Q2) $\geq 2.56/0.726 = 3{,}53$

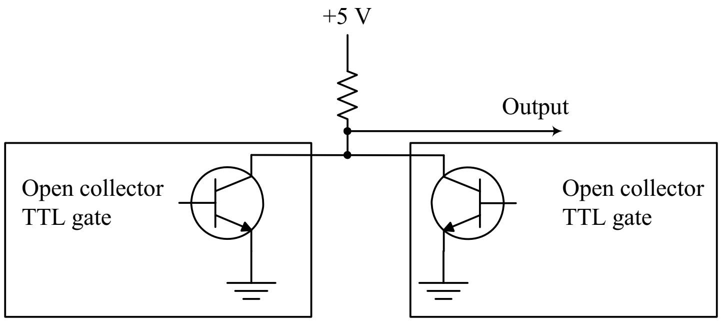
(c) $I_{B3} = I_{B2} + I_{C2} – 0.7/1000 = 0.725 + 2.56 – 0.7 = 2.585$ mA2.585 mA

(d) $I_{C3} < I_{B3}\ h_{FE} = 2.585 \times 6.18 = 16$ mA16 mA

(e) $(5 – 0.2)/R_L < 16$ mA

$R_L > 4.8 / 16 \times 10^{-3} = 4800/16 = 300$

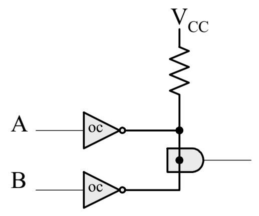
10.6 **(a)**



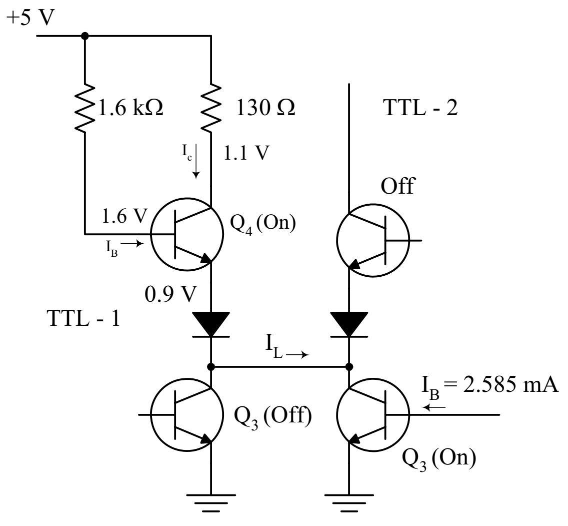
If one or more transistor are ON, the output is at a low level. The output is high if and only if all transistors are off.

Q1	Q2	Output	AND
ON	ON	LOW	= 0
ON	OFF	LOW	= 0
OFF	ON	LOW	=0
OFF	OFF	HIGH	= 1

(b) A' B' = (A + B)'



10.7



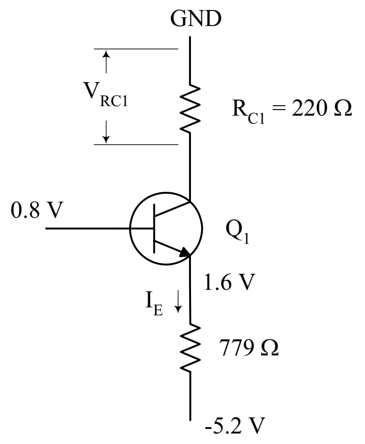
See Prob. 10.5

$I_B(Q4) = (5 - 1.6)/1600 = 2.123$ mA
$I_C(Q4) = (5 - 1.1)/130 = 30$ mA
$I_L = I_B + I_C = 2.125 + 30 = 32.125$ mA
Note: 32 mA of current flows between the two circuits provided Q4 (of TTL -1) and Q3 (of TTL – 2) are both saturated.

10.8

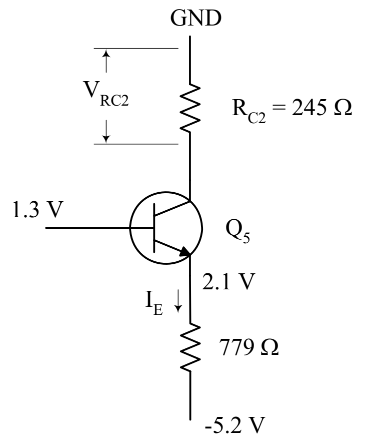
	(a) C=L A=L	(b) C=L A=H	(c) C=L A=H
Q1 (base-emitter)	ON	OFF	ON
Q1(base-collector)	OFF	ON	OFF
Q2	OFF	ON	OFF
Q3 Totem-pole output	OFF	ON	OFF
Q4 Totem-pole output	ON	OFF	OFF
Q5	ON	OFF	OFF
Q6 (base-emittter)	ON	ON	OFF
Q6 (base-collector)	OFF	OFF	ON
Q7	OFF	OFF	ON
Q8	OFF	OFF	ON
State of output	HIGH	LOW	HIGH IMPEDANCE

10.9 **(a)** When one or more inputs are HIGH, Q5 is OFF and the input transistor(s) is ON.



I_E = (5.2 – 1.6)/779 = 4.62 mA
V_{RC1} = 4.62 x 10^{-3} x 220 – 1.02 V

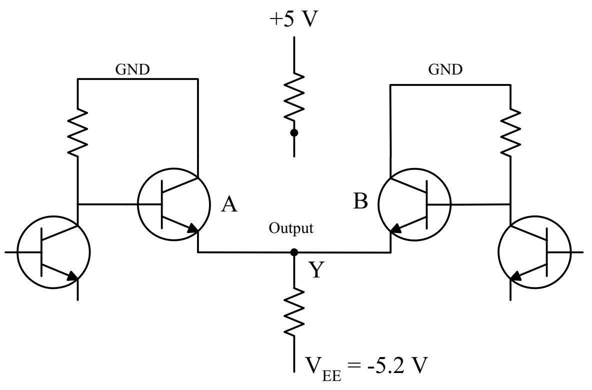
(b) When all inputs are low, Q5 conducts and the input transistors are OFF.



I_E = (5.2 – 2.1)/779 = 4 mA
V_{RC2} = 4 x 10^{-3} x 245 = 0.98 V

10.10 Noise margin = 0.6 – 0.3 = 0.3 V

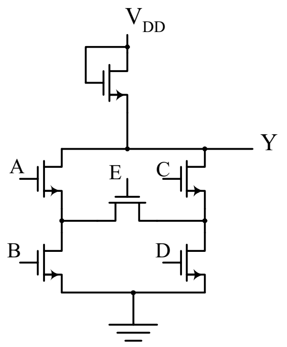
10.11



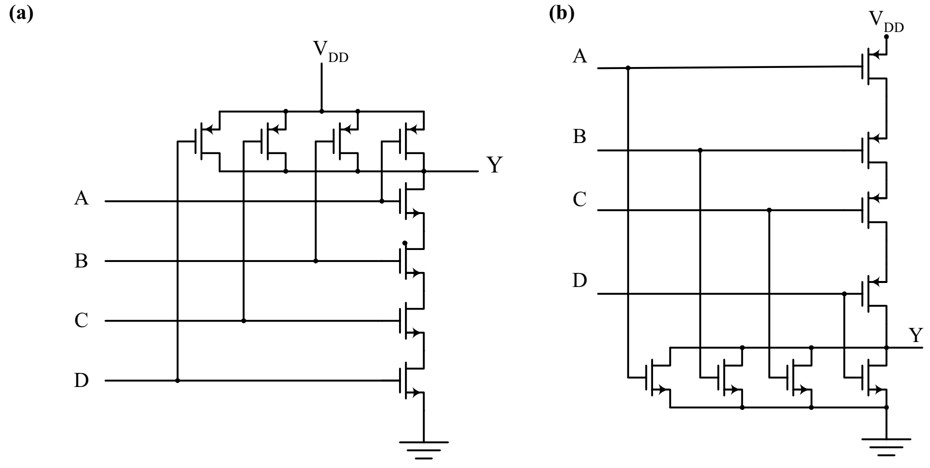
A	B	Y	OR
-1 V	-1 V	-1.8 V (L)	= 0
-1 V	GND	-0.8 V (H)	= 1
GND	-1 V	-0.8 V (H)	=1
GND	GND	-0.8 V (H)	= 1

-1 V = 0, GND = 1

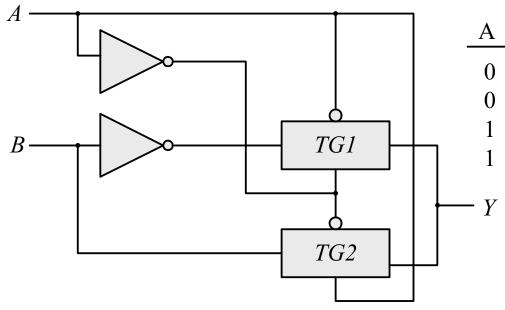
10.12



10.13



10.14

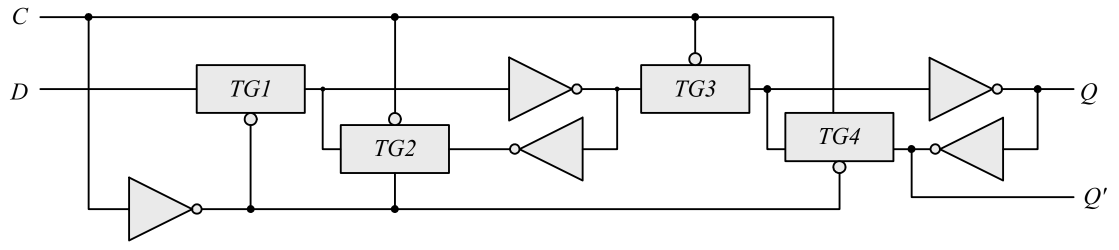


A	B	TG1	TG2	Y	
0	0	Close	Open	1	= B'
0	1	Close	Open	0	
1	0	Open	Close	0	= B
1	1	Open	Close	1	

10.15

Add another level to the diagram of Fig. 10.26 and a third selection input, S_2. The first level is controlled by S_0 and has 8 transmission gates between the data inputs and the four lines labeled I_0 ... I_3 in Fig. 10.26. The other two levels are as in Fig. 10.26 except that the second level is controlled by S_1 and the third level is controlled by S_2.

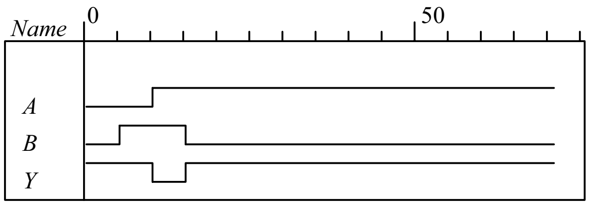
10.16



10.17

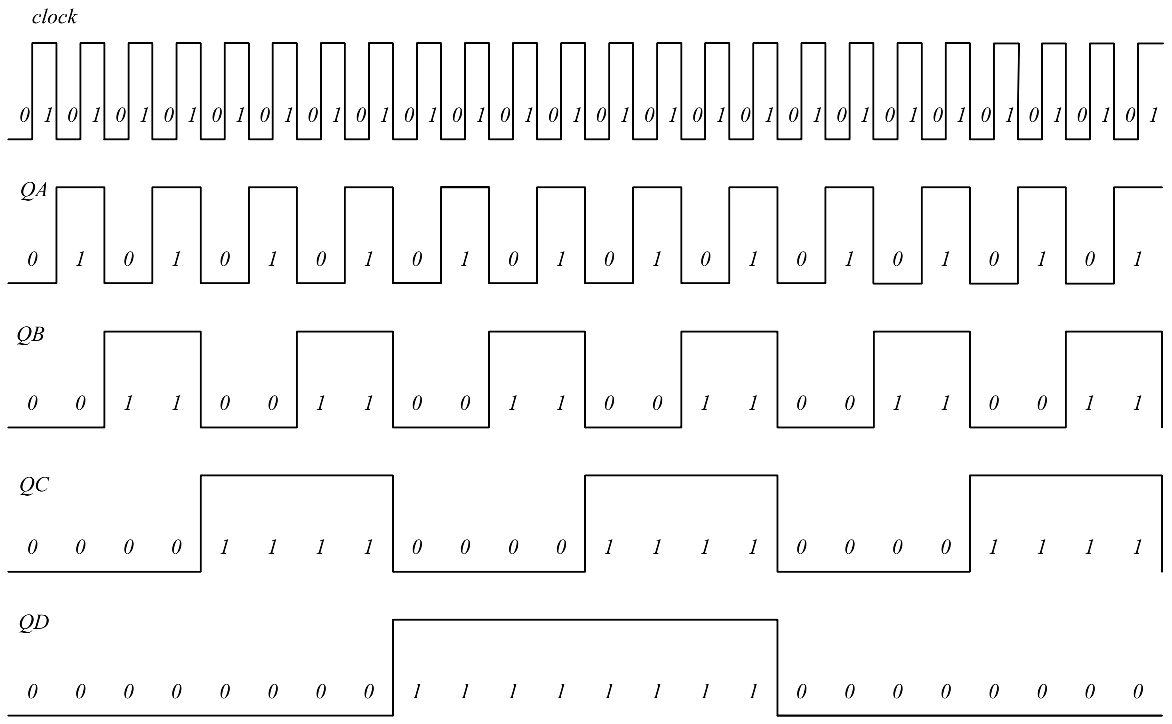
```
// Test bench for NAND2

module t_NAND2;
  reg    A, B;
  wire  Y;

  NAND2 M0 (Y, A, B);          // Instantiate NAND2

  initial #100 $finish;
  initial fork
    A = 0;
    B = 0;
    #5 B = 1;
    #10 A = 1;
    #15 B = 0;
  join

  initial $monitor ($time, ,"A = %b  B = %b  Y = %b", A, B, Y);
endmodule

//CMOS 2–input NAND Fig. 10.22(b)

module NAND2 (Y, A, B);
  input A, B;
  output Y;
  supply1 PWR;
  supply0 GRD;
  wire W1;                    // terminal between two nmos
  pmos (Y, PWR, A);       // source connected to Vdd
  pmos (Y, PWR, B);       // parallel connection
  nmos (Y, W1, A);         // serial connection
  nmos (W1, GRD, B);    // source connected to ground
endmodule
```

```
 0 A = 0  B = 0  Y = 1
 5 A = 0  B = 1  Y = 1
10 A = 1  B = 1  Y = 0
15 A = 1  B = 0  Y = 1
```



CHAPTER 11

11.1 Oscilloscope display:



BCD count: Oscilloscope displays from 0000 to 1001

Output pattern:

QA = alternate 1's and 0s
QB = Two 1's, two 0's, two 1's, four 0's
QC = Four 1's, six 0's
QD = Two 1's, eight 0's.

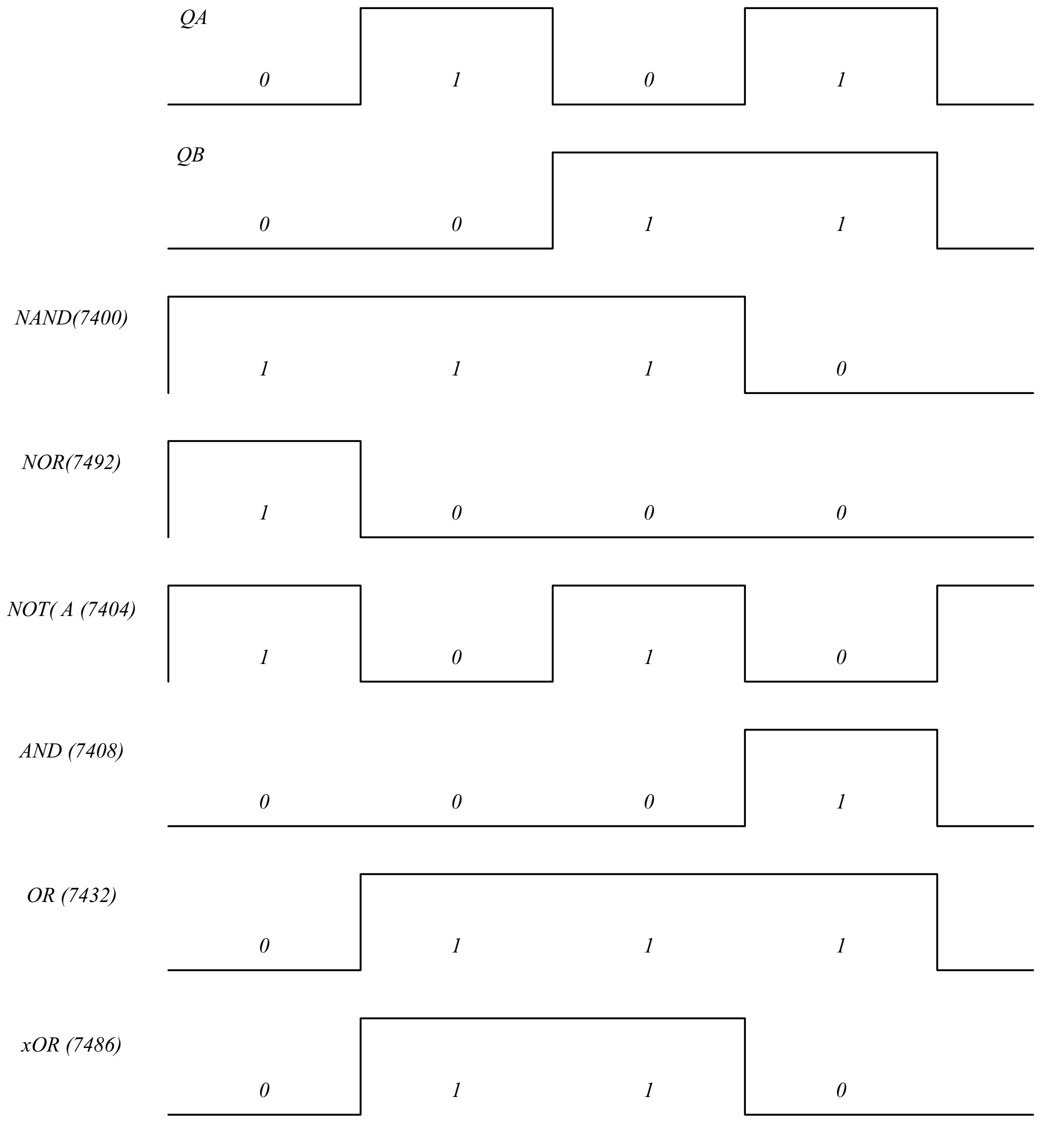
Other counts:

(a) 0101 must reset at 0110 – connect QB to R1, QC to R2
(b) 0111 must reset at 1000 – connect QD to both R1 and R2
(c) 1011 must reset at 1100 – connect QC to R1, QD to R2

11.2 Truth table:

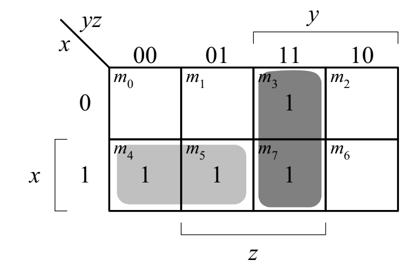
Inputs							
A	B	NAND	NOR	NOT(A)	AND	OR	XOR
0	0	1	0	1	0	0	0
0	1	1	1	0	0	1	1
1	0	1	1	1	0	1	1
1	1	0	1	0	1	1	0

Waveforms:

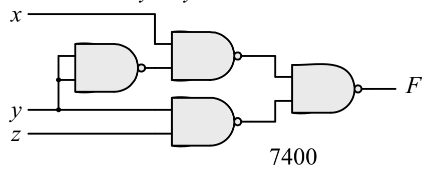


11.3

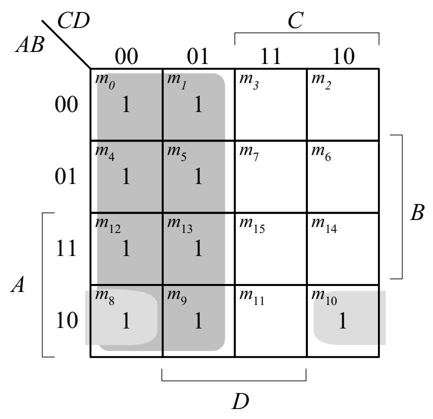
Logic Diagram



$F = xy' + yz$

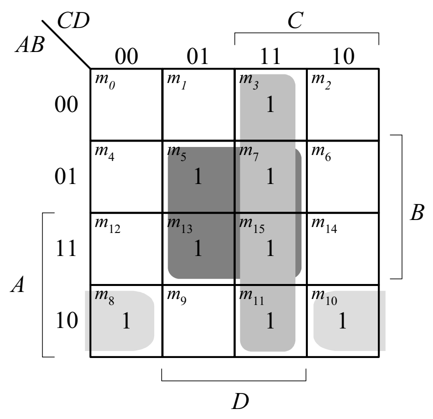


Boolean Functions:



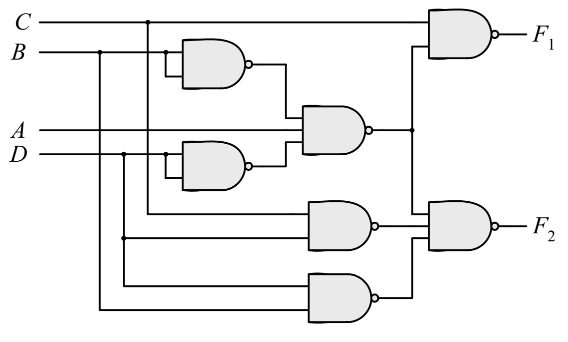
$F_1 = C' + AB'D'$

Boolean Functions:

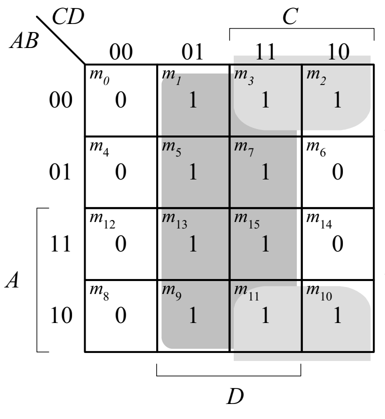


$F_2 = BD + CD + AB'D'$

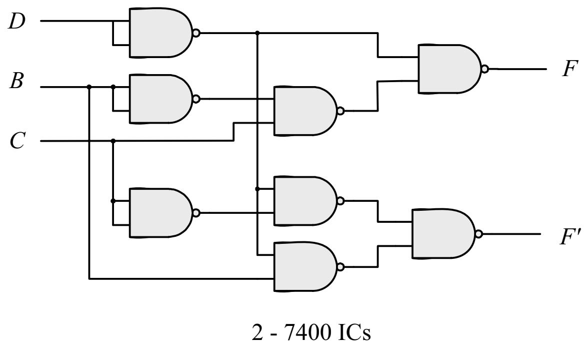
2 ICs: 7400, 7410



Complement:

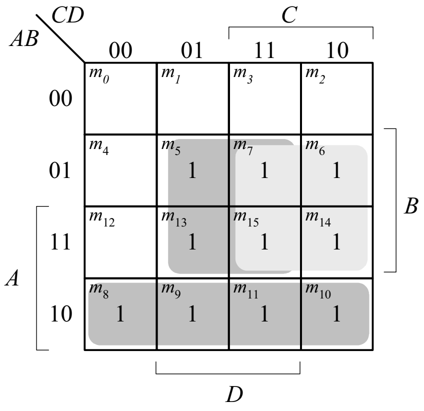


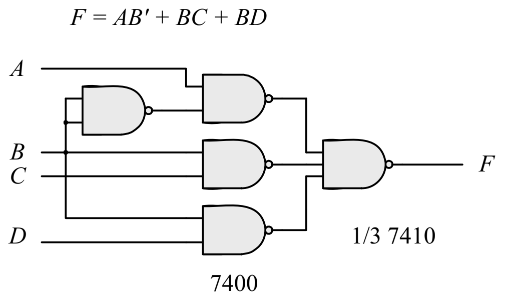
$F = D + B'C$
$F' = C'D' + BD'$



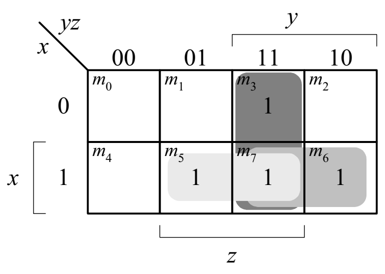
11.4

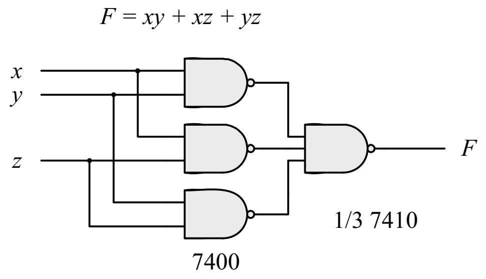
Design Example:

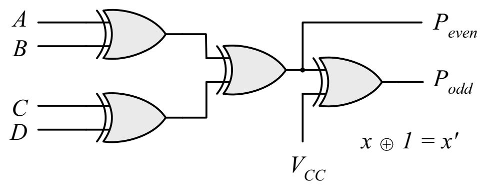




Majority Logic





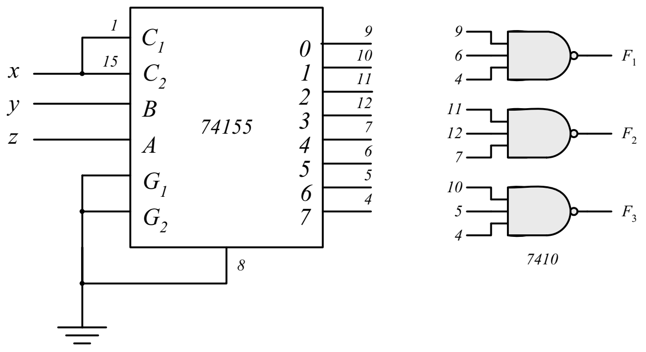


Decoder Implementation

$F_1 = xz + x'y'z' = \Sigma(0, 5, 7)$

$F_2 = x'y + xy'z' = \Sigma(2, 3, 4)$

$F_3 = xy + x'y'z = \Sigma(1, 6, 7)$

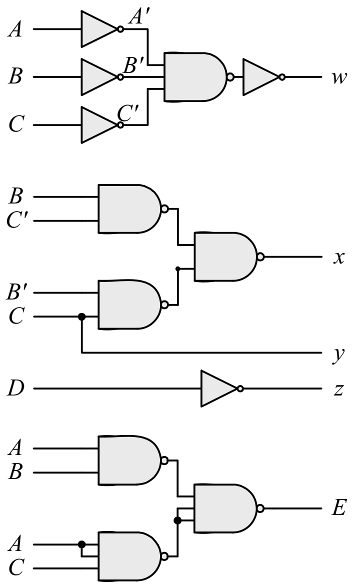


11.5 Gray code to Binary – See solution to Prob. 4.7.

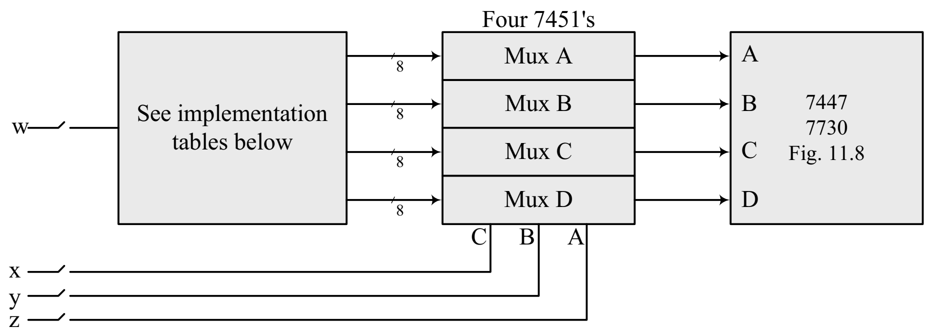
9's complementer – See solution to Prob. 4.18.

w = A'B'C'
x = BC' + B'C
y = C
z = D'
E = AB + AC

3 ICs: 7400, 7404, 7410



11.6



$A = \sum(0, 2, 3, 6, 7, 8, 9, 12, 13)$
$B = \sum(0, 2, 3, 4, 512, 13, 14)$
$C = \sum(0, 1, 3, 5, 6, 9, 10, 13, 14)$
$D = \sum(0, 7, 11)$

Mux A

D0	*D1*	*D2*	*D3*	*D4*	*D5*	*D6*	*D7*
↓	↓	↓	↓	↓	↓	↓	↓
1	*w*	*w'*	*w'*	*w*	*w*	*w'*	*w'*

Mux B

D0	*D1*	*D2*	*D3*	*D4*	*D5*	*D6*	*D7*
↓	↓	↓	↓	↓	↓	↓	↓
w'	*0*	*w'*	*w'*	*1*	*1*	*w*	*0*

Mux C

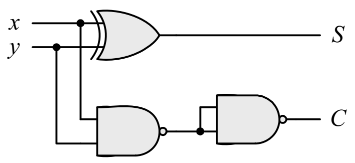
D0	*D1*	*D2*	*D3*	*D4*	*D5*	*D6*	*D7*
↓	↓	↓	↓	↓	↓	↓	↓
w'	*1*	*w*	*w'*	*0*	*1*	*1*	*0*

Mux D

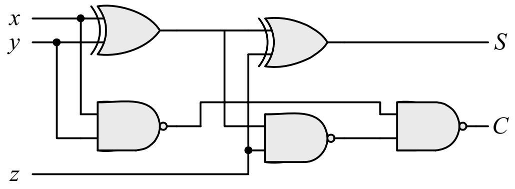
D0	*D1*	*D2*	*D3*	*D4*	*D5*	*D6*	*D7*
↓	↓	↓	↓	↓	↓	↓	↓
w'	*0*	*0*	*w*	*0*	*0*	*0*	*w'*

11.7

Half - Adder

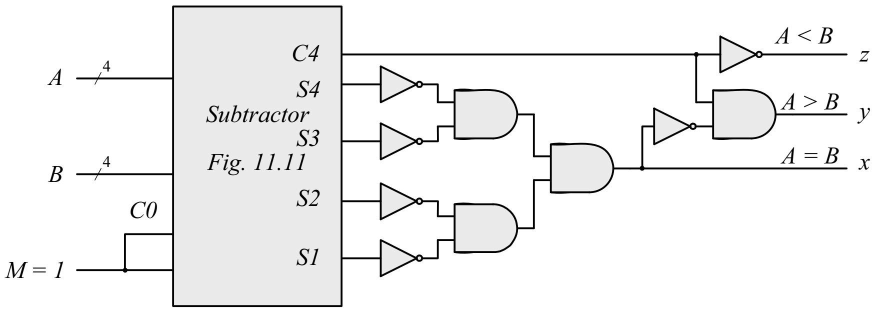


Full- Adder



Parallel adder - See circuit of Fig. 11.10.
Adder-subtractor – See circuit of Fig. 11.11.

	Inputs				Outputs		
Operation	*M*	*A*	*B*	C_0	*S*	C_4	
9 + 5 = 14	0	1001	0101	0	1110	0	sum < 15
9 + 9 = 19 = 16 + 2	0	1001	1001	0	0010	1	sum > 15
9 + 15 = 24 = 16 + 8	0	1001	1111	0	1000	1	sum > 15
9 - 5 = 4	1	1001	0101	1	0100	1	A > B
9 - 9 = 0	1	1001	1001	1	0000	1	A = B
9 - 15 = -6	1	1001	1111	1	1010	0	A < B

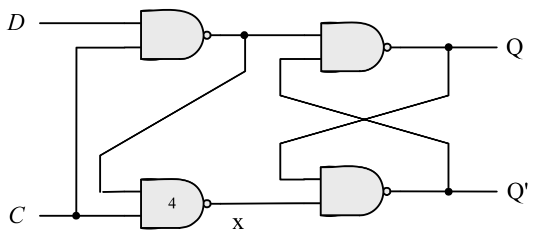


11.8 **SR Latch:** See Fig. 5.4.

D Latch:

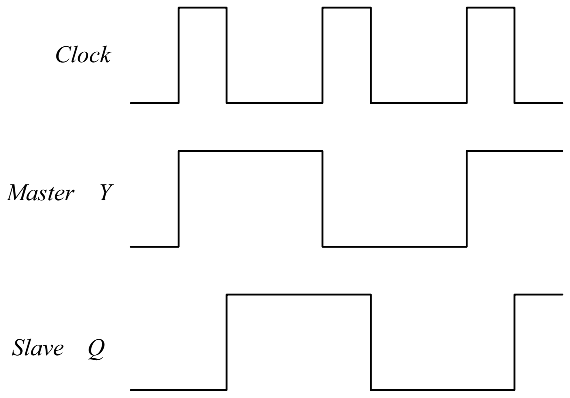
Let *CP* = *C*, *x* = output of gate 4.

x = [(DC)'C]' = (D'C)'

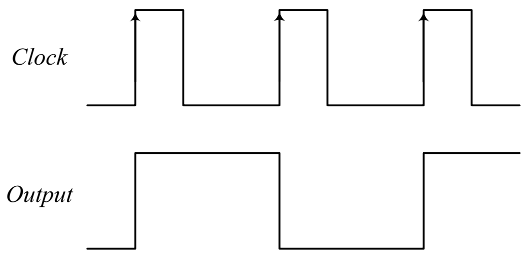


Master-Slave D Flip-Flop: The circuit is as in Fig. 5.9.

The oscilloscope display:



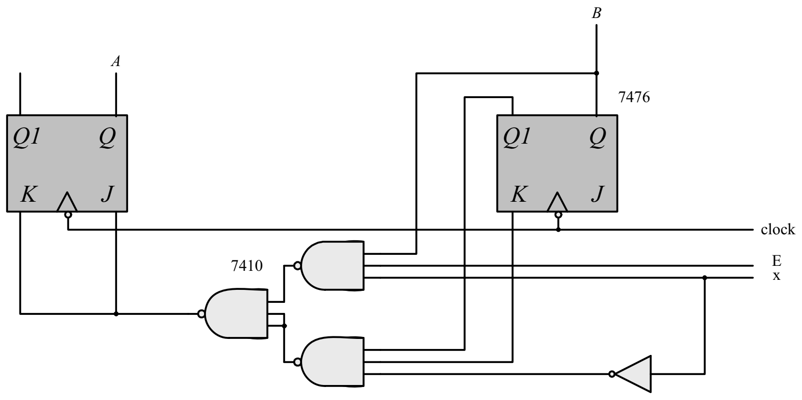
Edge-Triggered D Flip-Flop: Circuit is shown in Fig. 5.10.



IC Flip-Flops:

Connect all inputs to toggle switches, the clock to a pulser, and the outputs to indicator lamps.

11.9 Up-Down Counter with Enable:

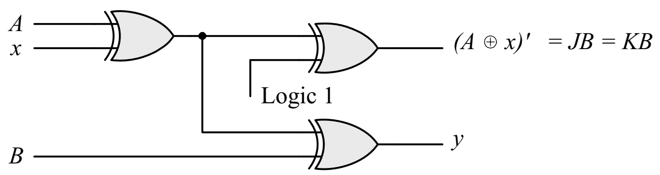


$J_B = K_B = E$ (Complement B when E = 1)

$J_A = K_A = E\,(Bx + B'x')$
Complement A when E = 1 and:
B = 1 when x = 1 (Count up)
B = 0 when x = 0 (Count down)

State Diagram:

$J_A = B$ $\quad$ $J_B = Ax + A'x' = (A \oplus x)'$ $\quad$ $Y = A \oplus B \oplus x$
$K_A = B'$ $\quad$ $K_B = Ax + A'x' = (A \oplus x)'$



Design of Counter: ABCD

$J_A = K_A = B(CD)$	$0000 \rightarrow 0101 \rightarrow 0110$
$J_B = K_B = CD$	$1000 \rightarrow 1001 \rightarrow 1010$
$J_C = D$	$K_C = AD$
$J_D = K_D = 1$	

11.10 Ripple counter: See Fig. 6.8
Down counter: Either take outputs from Q' outputs or connect complement Q' to next clock input.

Synchronous counter: See Fig. 6.12.

BCD counter: See solution to Prob. 6.19.
Unused states:

$$\begin{tikzcd}
10 \arrow[r] & 11 \arrow[r] & 6 \\
12 \arrow[r] & 13 \arrow[r] & 4 \\
14 \arrow[r] & 15 \arrow[r] & 2
\end{tikzcd}$$

Binary counter wth parallel load:

Connect QA and QD through a NAND gate to the load. See Fig. 6.15.

11.11 Ring counter:

See Fig. 6.17(a).
States of register:

QA	*QB*	*QC*	*QD*
1	0	0	0
0	1	0	0
0	0	1	0
0	0	0	1

Switch-tail ring counter: See Fig. 6.18(a). Connect (QD)' at pin 12 to the serial input at pin 4. State sequence as in Fig. 6.18(b).

Feedback shift register: Serial input = QC ⊕ QD (Use 7486).

Sequence of states:

QA	*QB*	*QC*	*QD*								
1	0	0	0	0	1	1	0	1	1	1	0
0	1	0	0	1	0	1	1	1	1	1	1
0	0	1	0	0	1	0	1	0	1	1	1
1	0	0	1	1	0	1	0	0	0	1	1
1	1	0	0	1	1	0	1	0	0	0	1

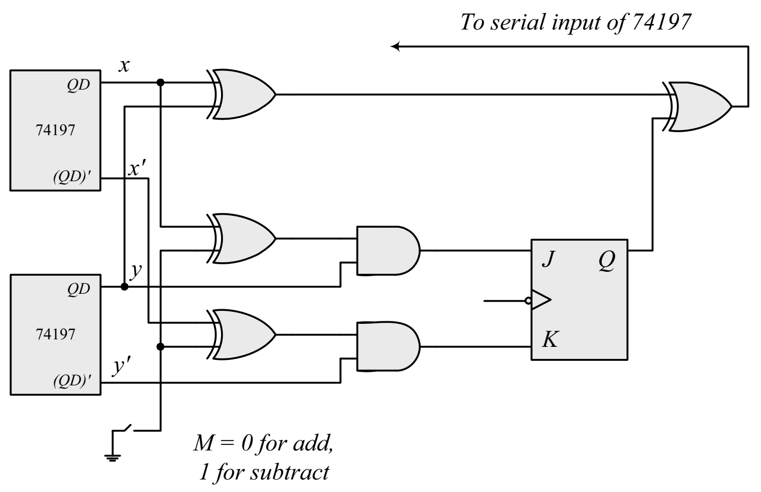
Bidirectional shift register with parallel load:

Function table:

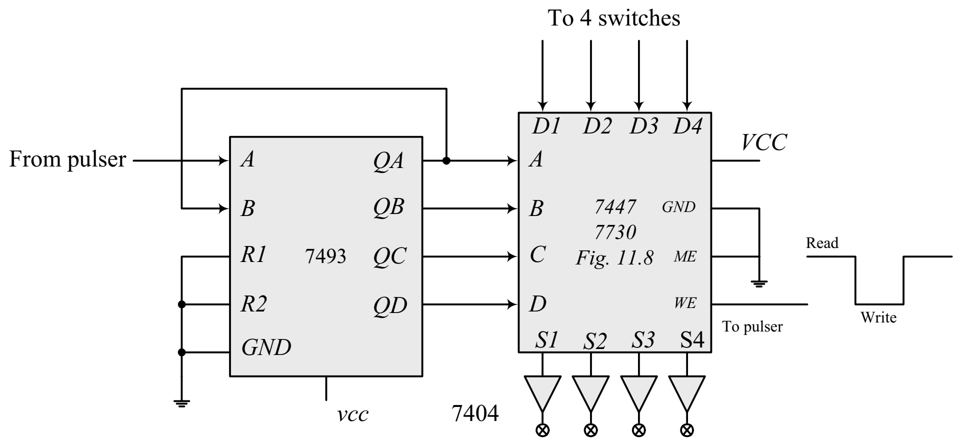
74195			74157		
Clear	Clock	SH/LD	STROBE	SELECT	Function
0	x	x	x	x	Async clear
1	↑	1	x	x	Shift right (*QA* → *QB*)
1	↑	0	0	1	Shift left (Select *B*)*
1	↑	0	0	0	Parallel Load (Select *A*)
1	↑	0	1	x	Synchronous clear

* B inputs come from QA-QD shifted by one position.

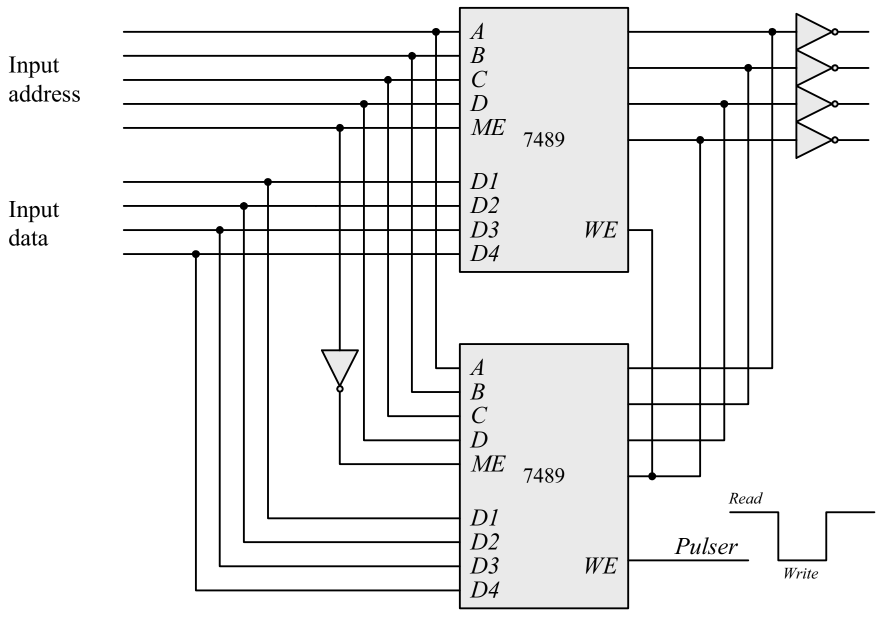
11.12



11.13 Testing the RAM:



Memory Expansion:



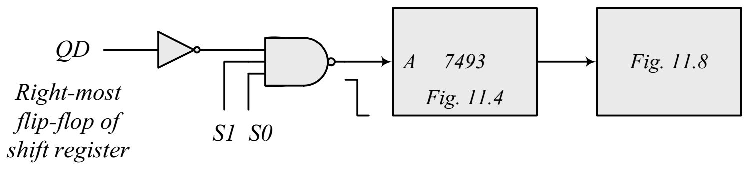
11.14 Circuit Analysis – Answers to questions:

1) Resets to 0 the two 74194 ICs, the two D flip-flops, and the start SR latch. This makes *S1S0* = 11 (parallel load).
2) The start switch sets the SR latch to 1. The clock pulses load 0000_0001 into the 8-bit register. If the start switch stays on, the register never clears to all 0s when *S1S0* = 11 (right-most *QD* stays on).
3) Pressing the pulser makes *S1S0* = 10 and the light shifts left. When *QC* becomes 1, the start SR latch is cleared to 0. When *QA* of the left 74194 becomes 1, it changes *S1* to 0 (through the *PR* input) and *S0* to 1 (through the *CLR* input. with *S1S0* = 01, the single light shifts right.
4) If the pulser is pressed while the light is moving to the left or the right, *S1S0* becomes 11 and all 0s are loaded into the register in parallel. The light goes off.
5) When the right-most *QD* becomes a 1, *S1S0* changes from 01 (shift right) to 11 (parallel load). If the pulser is pressed before the next clock pulse, *S1S0* goes to 10 (shift left). If not pressed, an all 0s value is loaded into the register in parallel. (Provided the start switch is in the logic 1 position.)

Lamp Ping-Pong

Add a left pulser. Three wire changes to the D flip-flop on the left:
1) Connect the clock input of the flip-flop to the pulser.
2) Connect the D input to the *QA* of the left 74197
3) Connect the input of the inverter (that goes to *PR*) to ground.

Counting the Losses

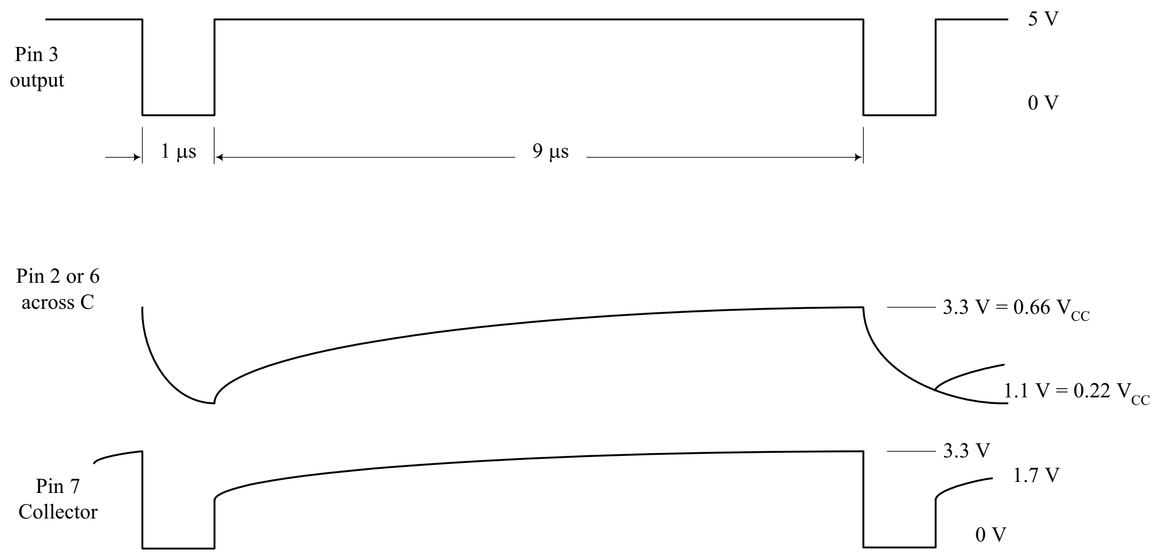


11.15 Clock Pulse Generator

$t_L = 0.693\ R_BC = 10^{-6}$
$R_B = 10^{-6} / (0.693 \text{ x } 0.001 \text{ x } 10^{-6}) = 10^3 / 0.693 = 1.44\ K\Omega$ (Use $R_B = 1.5\ K\Omega$)

$t_H/t_L = 0.693\ (R_A + R_B)C / (0.693\ R_B\ C) = (R_A + R_B) / R_B = 9/ 1 = 9$
$9\ R_B = R_A + R_B \quad R_A = 8\ R_B = 8 \text{ x } 1.5\ K\Omega = 12\ K\Omega$

Oscilloscope Waveforms (Actual results may be off by $\pm$ 20 %.)

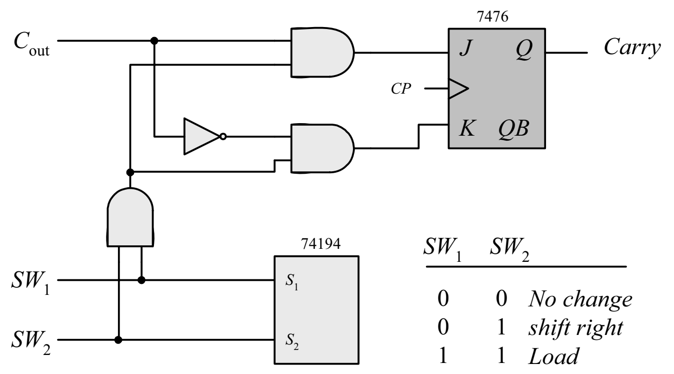


Variable Frequency Pulse Generator:

20 KHz: $10^{-3} / 20 = 0.05 \text{ x } 10^{-3} = 50\ \mu s$
100 KHz: $10^{-3} / 100 = 10^{-5} = 10\ \mu s$

$t_H = 49\ \mu s$: $(R_A + R_P + R_B) / R_B = 49/ 1 = 49$
$R_P = 48\ R_B - R_A = 48 \text{ x } 1.5 - 11 = 60\ K\Omega$

11.15 Control of Register



SW_1	SW_2	
0	0	*No change*
0	1	*shift right*
1	1	*Load*

Checking the Circuit:

	Carry	Register
Initial	0	0 0 0 0
+ 0110	0	0110
+ 1110	1	0100
+ 1101	1	0001
+ 0101	0	0110
+ 0011	0	1001

Circuit Operation:

Address	Carry	RAM	
0	0	0110	RAM Value
1	0	0110	RAM + Register
2	0	0011	Shfit Register
3		1110	RAM Value
4	1	0001	RAM + Register
5	1	1000	SHIFT
6		1101	RAM Value
7	1	0101	RAM + Register
8	1	1010	SHIFT
9		0101	RAM Value
10	0	1111	RAM + Register
11	0	0111	SHIFT
12		0011	RAM Value
13	0	1010	RAM + REgiser
14	0	0101	SHIFT

11.17 Multiplication Example)11 x 15 = 165)
Multiplicand B = 1111

		C	*A*	*Q*	*P*
Initial:	T_1= 1	0	0000	1011	0000
T_2 = 1	Add B; P <= P+1		1111		
		0	1111	1011	0001
T_3 = 1	Shift CAQ	0	0111	1101	0001
T_2 = 1	Add B; P <= P+1		1111		
		1	0110	1101	0010
T_3 = 1	Shift CAQ	0	1011	0110	0010
T_2 = 1	P <= P+1	0	1011	0110	0011
T_3 = 1	Shift CAQ	0	0101	1011	0011
T_2 = 1	Add B; P <= P+1		1111		
		1	0100	1011	0100
T_3 = 1	Shift CAQ	0	1010	0101	0100
T_0 = 1	(Because P_C = 1)		1010	0101	= Product

Data Processor Design

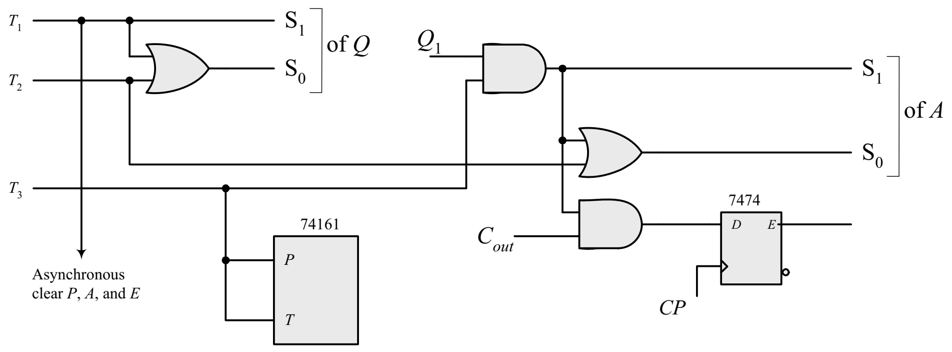
Load Q	Load A	Shift AQ	Register Q	Register A
T_1	T_2Q_1	T_3	S_1 S_0	S_1 S_0
0	0	0	0 0	0 0
1	0	0	1 1	0 0
0	1	0	0 0	1 1
0	0	1	0 1	0 1

$S_1(Q) = T_1$

$S_4(Q) = T_1 + T_3$

$S_1(A) = T_2Q_1$

$S_0(A) = T_2Q_1 + T_3$

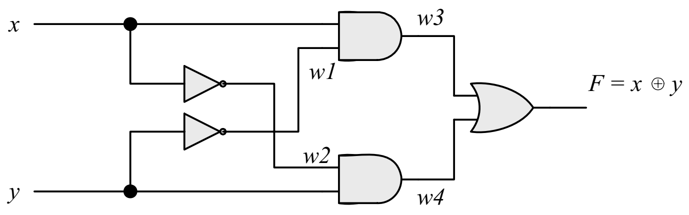


Design of Control: See Section 8.8.

SOLUTIONS FOR SECTION 11.20

Supplement to Experiment 2:

(a)

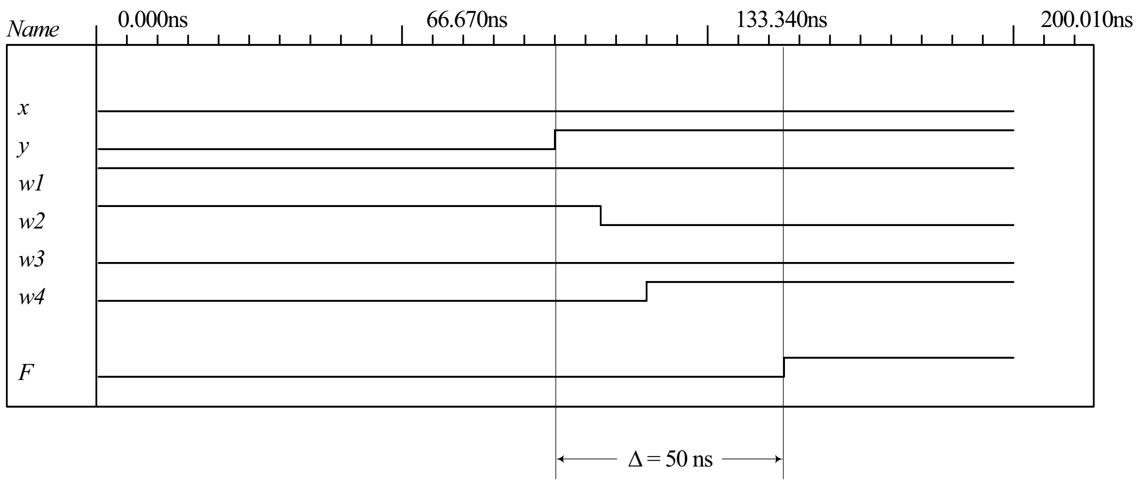


Initially, with *xy* = 00, *w1* = w2 = 1, *w3* = *w4* = 0 and *F* = 0. *w1* should change to 0 10ns after *xy* changes to 01. *w4* should change to 1 20 ns after xy changes to 01. *F* should change from 0 to 1 30 ns after *w4* changes from 0 to 1, i.e., 50 ns after *xy* changes from 00 to 01. w3 should remain unchanged because x = 0 for the entire simulation.

(b)

```
`timescale 1ns/1ps

module Prob_3_33 (output F, input x, y);
wire w1, w2, w3, w4;

and #20 (w3, x, w1);
not #10 (w1, x);
and #20 (w4, y, w1);
not #10 (w2, y);
or  #30 (F, w3, w4);

endmodule

module t_Prob_3_33 ();
  reg x, y;
  wire F;

  Prob_3_33 M0 (F, x, y);

  initial #200 $finish;
  initial fork
    x = 0;
    y = 0;
    #100 y = 1;
  join
endmodule
```

(c) To simulate the circuit, it is assumed that the inputs xy = 00 have been applied sufficiently long for the circuit to be stable before xy = 01 is applied. The testbench sets xy = 00 at t = 0 ns, and xy = 1 at t = 100 ns. The simulator assumes that xy = 00 has been applied long enough for the circuit to be in a stable state at t = 0 ns, and shows F = 0 as the value of the output at t = 0. The waveforms show the response to xy = 01 applied at t = 100 ns.



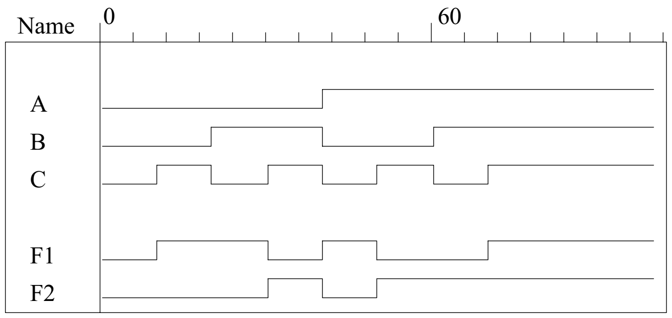
Supplement to Experiment 4:

(a)

```
// Gate-level description of circuit in Fig. 4-2
module Circuit_of_Fig_4_2 (
  output   F1, F2,
  input     A, B, C);
  wire  T1, T2, T3, F2_not, E1, E2, E3;
  orG1 (T1, A, B, C);
  and   G2 (T2, A, B, C);
  and   G3 (E1, A, B);
  and   G4 (E2, A, C);
  and   G5 (E3, B, C);
  orG6 (F2, E1, E2, E3);
  not   G7 (F2_not, F2);
  and   G8 (T3, T1, F2_not);
  orG9 (F1, T2, T3);
endmodule

module t_Circuit_of_Fig_4_2;
  reg   [2: 0] D;
  wire  F1, F2;
  parameter stop_time = 100;

  Circuit_of_Fig_4_2 M1 (F1, F2, D[2], D[1], D[0]);

  initial # stop_time $finish;
  initial begin                    // Stimulus generator
    D = 3'b000;
   repeat (7)
    #10 D = D + 1'b1;
  end

  initial begin
   $display ("A     B     C     F1   F2");
   $monitor ("%b     %b     %b     %b    %b", D[2], D[1], D[0], F1, F2);
  end
endmodule
```

```
/*
A   B   C   F1  F2
0   0   0   0   0
0   0   1   1   0
0   1   0   1   0
0   1   1   0   1
1   0   0   1   0
1   0   1   0   1
1   1   0   0   1
1   1   1   1   1

*/
```

The simulation results demonstrate the behavior of a full adder, with F1 = sum, and F2 – carry.

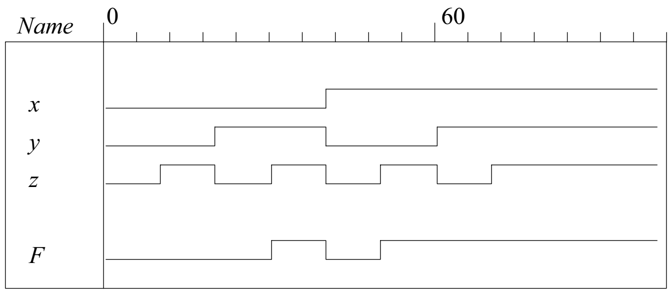


(b)

```
 // 3-INPUT MAJORITY DETECTOR CIRCUIT.
// Circuit implements F = xy + xz +yz.
module Majority_Detector (output F, input x, y, z);
  wire wl, w2, w3;
  nand nl(wl, x, y),
           n2(w2, x, z),
           n3(w3, y, z),
           n4(F, wl, w2, w3) ;
endmodule

 // Test bench
 //Treating inputs to majority detector as a vector, reg [2:0]D; //D[2] = x, D[l] = y, D[0] = z. wire F;
module t_Majority_Detector ();
   wire F;
  reg [2: 0] D;
  wire x = D[2];
  wire y = D[1];
  wire z = D[0];

 Majority_Detector M0 (F, x, y, z);

  initial #100 $finish;
  initial $monitor ($time,, "xyz = %b F = %b", D, F);
 initial begin
   D = 0;
   repeat (7)
     #10 D = D + 1;
 end
endmodule
```

Simulation results:

```
  0 xyz = 000 F = 0
 10 xyz = 001 F = 0
 20 xyz = 010 F = 0
 30 xyz = 011 F = 1
 40 xyz = 100 F = 0
 50 xyz = 101 F = 1
 60 xyz = 110 F = 1
 70 xyz = 111 F = 1
```



Supplement to Experiment 5: See the solution to Prob. 4.42.

Supplement to Experiment 7:

(a)

```
//BEHAVIORAL DESCRIPTION OF 7483 4-BIT ADDER,
module Adder_7483 (
  output S4, S3, S2, S1, C4,
  input A4, A3, A2, A1, B4, B3, B2, B1, C0, VCC, GND
);
// Note: connect VCC and GND to supply1 and supply0 in the test bench
  wire [4: 1] sum;
  wire [4: 1] A = {A4, A3, A2, A1};
  wire [4: 1] B = {B4, B3, B2, B1};
  assign S4 = sum[4];
  assign S3 = sum[3];
  assign S2 = sum[2];
  assign S1 = sum[1];

  assign {C4, sum} = A + B + C0;
endmodule
```

```
module t_Adder_7483 ();
  wire S4, S3, S2, S1, C4;
  wire A4, A3, A2, A1, B4, B3, B2, B1;
  reg C0;
  supply1 VCC;
  supply0 GND;
  reg [4:1] A, B;
  assign A4 = A[4];
  assign A3 = A[3];
  assign A2 = A[2];
  assign A1 = A[1];
  assign B4 = B[4];
  assign B3 = B[3];
  assign B2 = B[2];
  assign B1 = B[1];

  Adder_7483 M0 (S4, S3, S2, S1, C4, A4, A3, A2, A1, B4, B3, B2, B1, C0, VCC, GND);

  initial #2600 $finish;
  initial begin
    A = 0; B = 0; C0 = 0;
    repeat (256) #5 {A, B} = {A, B} + 1;
    A = 0; B = 0; C0 = 1;
    repeat (256) #5 {A, B} = {A, B} + 1;
  end
endmodule
```

(b)

```
module Supp_11_17b (output [4: 1] S, output carry, input [4: 1] A, B, input M, VCC, GND);
  wire B4, B3, B2, B1;
  xor (B4, M, B[4]);
  xor (B3, M, B[3]);
  xor (B2, M, B[2]);
  xor (B1, M, B[1]);
  Adder_7483 M0 (S[4], S[3], S[2], S[1], carry, A[4], A[3], A[2], A[1], B4, B3, B2, B1, M, VCC, GND);
endmodule

module Adder_7483 (
  output S4, S3, S2, S1, C4,
  input A4, A3, A2, A1, B4, B3, B2, B1, C0, VCC, GND
);
// Note: connect VCC and GND to supply1 and supply0 in the test bench
  wire [4: 1] sum;
  wire [4: 1] A = {A4, A3, A2, A1};
  wire [4: 1] B = {B4, B3, B2, B1};
  assign S4 = sum[4];
  assign S3 = sum[3];
  assign S2 = sum[2];
  assign S1 = sum[1];
  assign {C4, sum} = A + B + C0;
endmodule
```

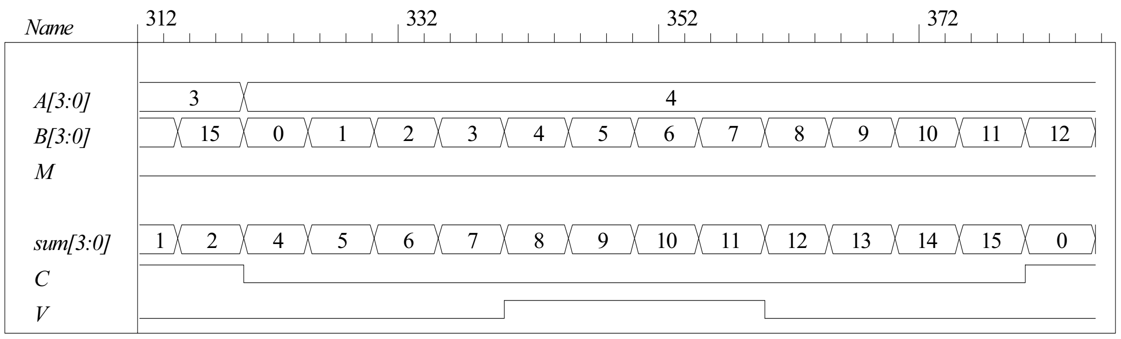
```
module t_Supp_11_17b ();
  wire [4: 1] S;
  wire carry;
  reg C0;
  reg [4: 1] A, B;
  reg M;
  supply1 VCC;
  supply0 GND;

  Supp_11_17b M0 (S, carry, A, B, M, VCC, GND);
  initial #2600 $finish;
  initial begin
    A = 0; B = 0; M = 0;
    repeat (256) #5 {A, B} = {A, B} + 1;
    A = 0; B = 0; M = 1;
    repeat (256) #5 {A, B} = {A, B} + 1;
  end
endmodule
```

(c), (d)

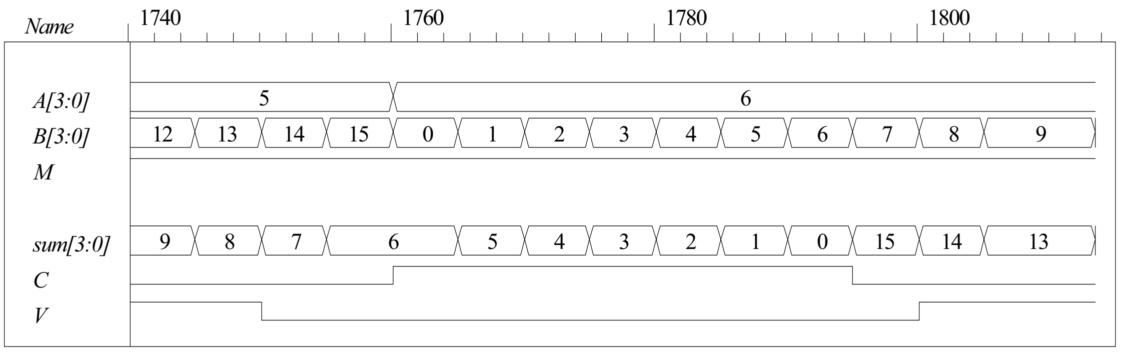
```
module supp_11_7c (output S3, S2, S1, S0, C, V, input A3, A2, A1, A0, B3, B2, B1, B0, M);
  wire [3: 0] Sum, B;
  assign S3 = Sum[3];
  assign S2 = Sum[2];
  assign S1 = Sum[1];
  assign S0 = Sum[0];
  wire [3:0]  A = {A3, A2, A1, A0};
  xor(B[3], B3, M);
  xor(B[2], B2, M);
  xor(B[1], B1, M);
  xor(B[0], B0, M);
  xor (V, C, C3);
  ripple_carry_4_bit_adder M0 (Sum, C, C3, A, B, M);
endmodule

module t_supp_11_7c ();
  wire S3, S2, S1, S0, C, V;
  reg A3, A2, A1, A0, B3, B2, B1, B0, M;
  wire [3: 0] sum = {S3, S2, S1, S0};
  wire [3: 0] A = {A3, A2, A1, A0};
  wire [3: 0] B = {B3, B2, B1, B0};

  supp_11_7c M0 (S3, S2, S1, S0, C, V, A3, A2, A1, A0, B3, B2, B1, B0, M);

  initial #2600 $finish;
  initial begin
    {A3, A2, A1, A0, B3, B2, B1, B0} = 0; M = 0;
    repeat (256) #5 {A3, A2, A1, A0, B3, B2, B1, B0} = {A3, A2, A1, A0, B3, B2, B1, B0} + 1;
    {A3, A2, A1, A0, B3, B2, B1, B0} = 0; M = 1;
    repeat (256) #5 {A3, A2, A1, A0, B3, B2, B1, B0} = {A3, A2, A1, A0, B3, B2, B1, B0} + 1;
  end
endmodule
```

```
module half_adder (output S, C, input x, y);         // Verilog 2001, 2005 syntax
 // Instantiate primitive gates
  xor (S, x, y);
  and (C, x, y);
endmodule

module full_adder (output S, C, input x, y, z);
  wire  S1, C1, C2;

// Instantiate half adders
  half_adder HA1 (S1, C1, x, y);
  half_adder HA2 (S, C2, S1, z);
  or G1 (C, C2, C1);
endmodule

// Modify for C3 output
module ripple_carry_4_bit_adder ( output [3: 0] Sum, output C4, C3, input [3:0]  A, B, input C0);
  wire       C1, C2;     // Intermediate carries
// Instantiate chain of full adders
  full_adder     FA0 (Sum[0], C1, A[0], B[0], C0),
                 FA1 (Sum[1], C2, A[1], B[1], C1),
                 FA2 (Sum[2], C3, A[2], B[2], C2),
                 FA3 (Sum[3], C4, A[3], B[3], C3);
endmodule
```

Addition:



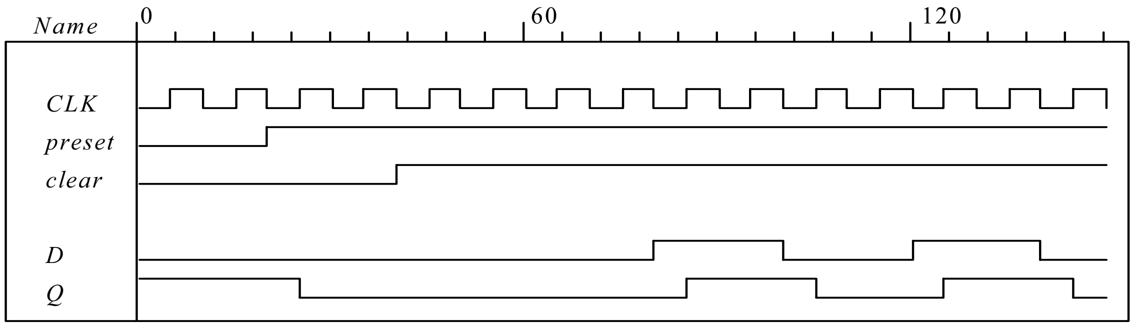
Subtraction:



Supplement to Experiment 8:

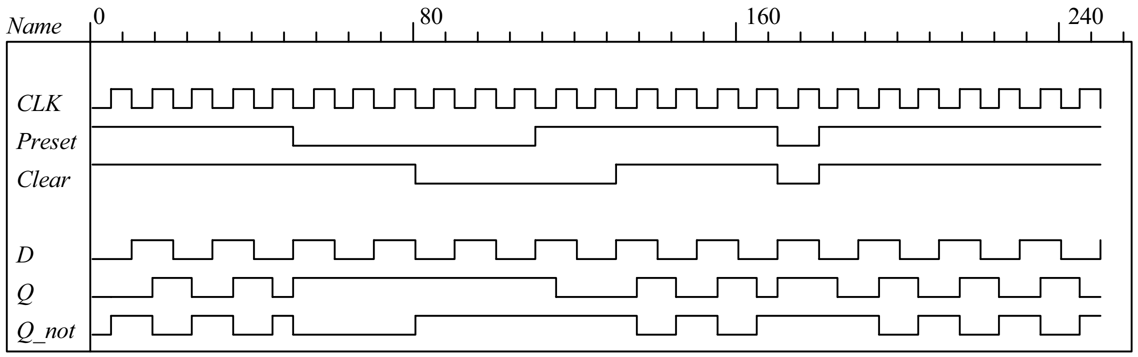
(a)

```
module Flip_flop_7474 (output reg Q, input D, CLK, preset, clear);
  always @ (posedge CLK, negedge preset , negedge clear)
  if (!preset)                 Q <= 1'b1;
  else if (!clear)             Q <= 1'b0;
  else                         Q <= D;
endmodule

module t_Flip_flop_7474 ();
  wire   Q;
  reg    D, CLK, preset, clear;

  Flip_flop_7474 M0 (Q, D, CLK, preset, clear);

  initial #150 $finish;
  initial begin CLK = 0; forever #5 CLK = ~CLK; end

  initial fork
    preset = 0; clear = 0;
    #20 preset = 1;
    #40 clear = 1;
  join

  initial begin D = 0; #60 forever #20 D = ~D; end
endmodule
```



(b)

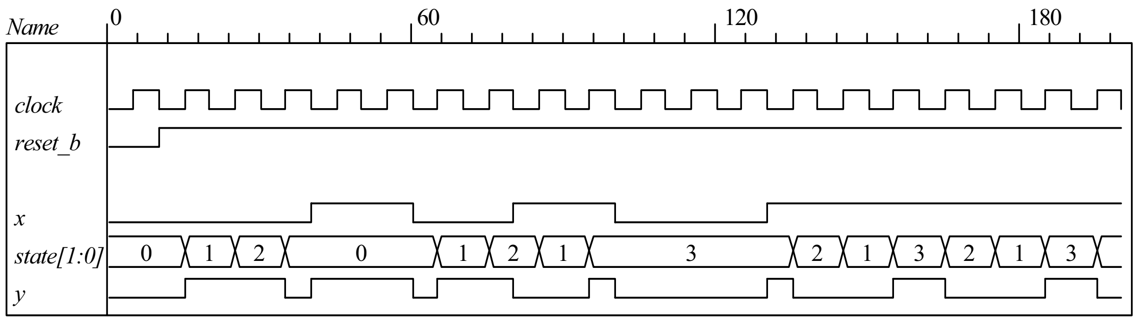
```
//Solution to supplement Experiment 8(b)
//Behavioral description of a 7474 D flip-flop with Q_not
module Flip_Flop_7474_with_Q_not (output reg Q, Q_not, input D, CLK, Preset, Clear);

  always @ (posedge CLK, negedge Preset, negedge Clear)
  /* case ({Preset, Clear})
     2'b00:  begin Q <= 1; Q_not <= 1; end
     2'b01:  begin Q <= 1; Q_not <= 0; end
     2'b10:  begin Q <= 0; Q_not <= 1; end
     2'b11:  begin Q <= D; Q_not <= ~D; end

     // NOTE: Q_not <= ~Q will produce a pipeline effect and delay Q_not by one clock
    endcase*/
if (Preset == 0) begin  Q <= 1; if (Clear == 0) Q_not <= 1; else Q_not <= 0; end
else if (Clear == 0) begin Q <= 0; Q_not <= 1; end
else begin Q <= D; Q_not <= ~D; end
endmodule

 // Note: this model will not work if Preset and Clear are // both brought low and then high again.
 // A case statement for both Q and Q_not is also OK.
module t_Flip_Flop_7474_with_Q_not ();
 wire   Q, Q_not;
 reg    D, CLK, Preset, Clear;

 Flip_Flop_7474_with_Q_not M0 (Q, Q_not, D, CLK, Preset, Clear);

 initial #250 $finish;
 initial begin CLK = 0; forever #5 CLK = ~CLK; end

 initial fork
   Preset = 1; Clear = 1;
   #50 Preset = 0;
   #80 Clear =
```



Supplement to Experiment #9:

(a)

```
module Figure_11_9a (output reg y, input x, clock, reset_b);
  reg [1: 0] state, next_state;
  parameter S0 = 2'b00, S1 = 2'b01, S2 = 2'b10, S3 = 2'b11;
  always @ (posedge clock, negedge reset_b) if (reset_b == 0) state <= S0; else state <= next_state;
  always @ (state, x) begin
    y = 0;
    case (state)
      S0:if (x) begin next_state = S0; y = 1; end else begin next_state = S1; y = 0; end
      S1:if (x) begin next_state = S3; y = 0; end else begin next_state = S2; y = 1; end
      S2:if (x) begin next_state = S1; y = 0; end else begin next_state = S0; y = 1; end
      S3:if (x) begin next_state = S2; y = 1; end else begin next_state = S3; y = 0; end
    endcase
  end
endmodule

module t_Figure_11_9a ();
  wire y;
  reg x, clock, reset_b;

  Figure_11_9a M0 (y, x, clock, reset_b);

  initial #200 $finish;
  initial begin clock = 0; forever #5 clock = ~clock; end
  initial fork
    reset_b = 0;
    x = 0;              // S0. S1, S2 after release of reset_b
    #10 reset_b = 1;
    #40 x = 1;          // Stay in S0
    #60 x= 0;           // S1, S2
    #80 x = 1;          //  s1, S3,
    #100  x = 0;// S3
    #130 x = 1;         // S2, S1, S3 cycle
  join
endmodule
```

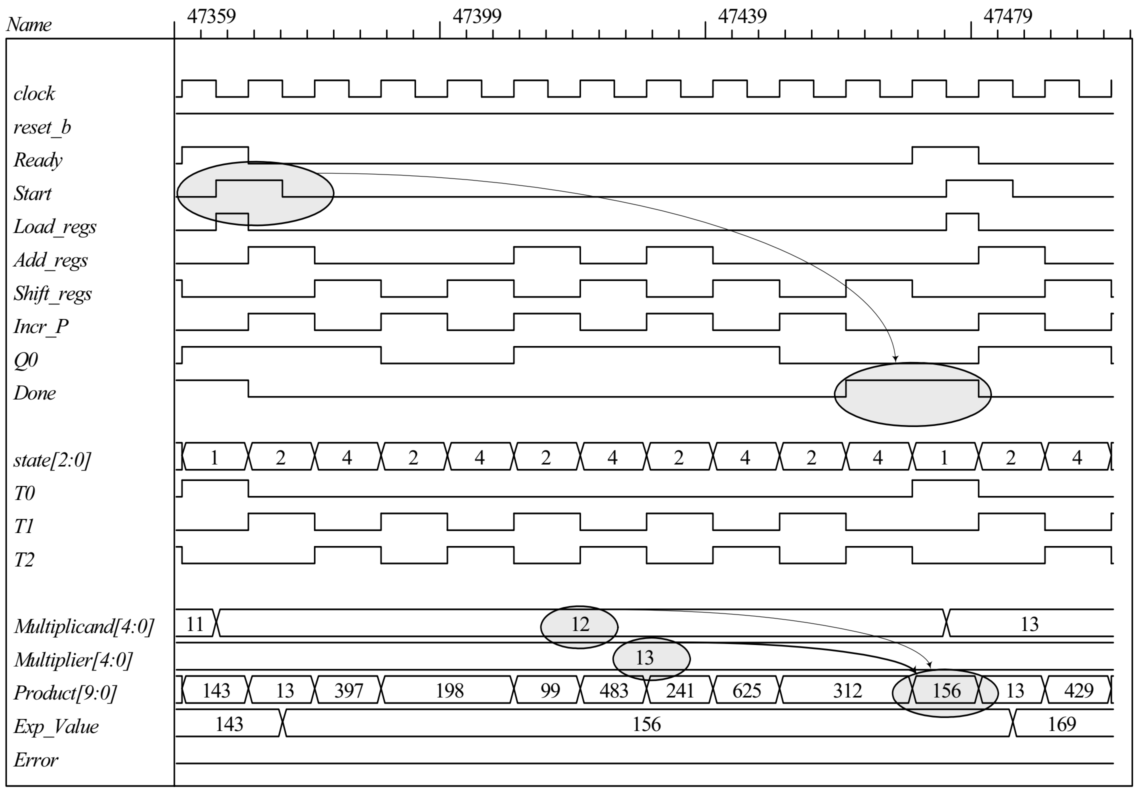


(b) The solution depends on the particular design.

(c, d)
Note: The HDL description of the state diagram produces outputs *T0*, *T1*, and *T2*. Additional logic must form the signals that control the datapath unit (*Load_regs*, *Incr_P*, *Add_regs*, and *Shift_regs)*. An alternative controller that generates the control signals, rather than the states, as the outputs is given below too. It produces identical simulation results.

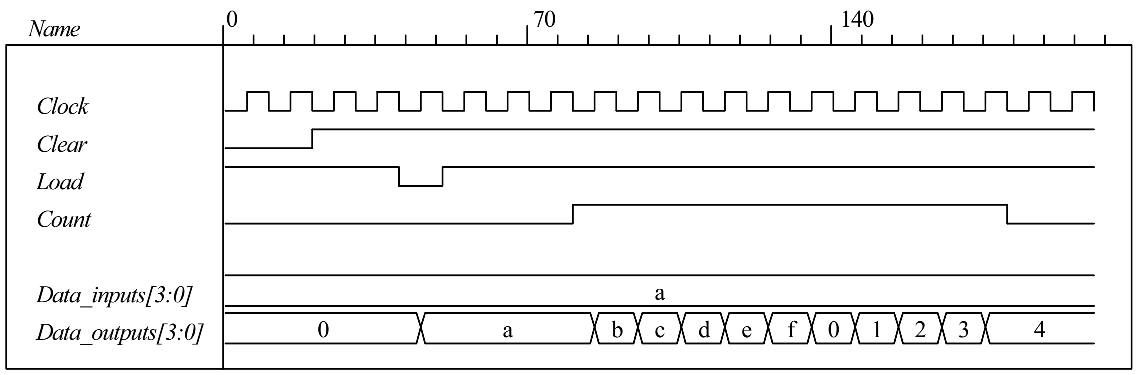
```
module Supp_11_9cd # (parameter dp_width = 5)
(
  output   [2*dp_width - 1: 0]   Product,
  output                         Ready,
  input    [dp_width - 1: 0]     Multiplicand, Multiplier,
  input                          Start, clock, reset_b
);
  wire Load_regs, Incr_P, Add_regs, Shift_regs, Done, Q0;

  Controller M0 (
    Ready, Load_regs, Incr_P, Add_regs, Shift_regs, Start, Done, Q0,
    clock, reset_b
);

Datapath M1(Product, Q0, Done, Multiplicand, Multiplier,
  Start, Load_regs, Incr_P, Add_regs, Shift_regs, clock, reset_b);
endmodule

/* // This alternative controller directly  produces the signals needed to control the datapath.
module Controller (
  output Ready,
  output reg Load_regs, Incr_P, Add_regs, Shift_regs,
  input Start, Done, Q0, clock, reset_b
);

  parameter   S_idle =        3'b001,          // one-hot code
              S_add =    3'b010,
              S_shift =  3'b100;
  reg    [2: 0]        state, next_state;       // sized for one-hot
  assign         Ready = (state == S_idle);

  always @ (posedge clock, negedge reset_b)
    if (~reset_b) state <= S_idle; else state <= next_state;

  always @ (state, Start, Q0, Done) begin
    next_state = S_idle;
    Load_regs = 0;
    Incr_P = 0;
    Add_regs = 0;
    Shift_regs = 0;
    case (state)
      S_idle:    if (Start) begin next_state = S_add; Load_regs = 1; end
      S_add:     begin next_state = S_shift; Incr_P = 1; if (Q0) Add_regs = 1; end
      S_shift:   begin Shift_regs = 1;
                   if (Done) next_state = S_idle;
                   else next_state = S_add;
                 end
      default:   next_state = S_idle;
    endcase
  end
endmodule
*/


// This controller has an embedded unit to generate T0, T1, and T2 and additional logic to form // // the
signals needed to control the datapath.

module Controller (
  output Ready, Load_regs, Incr_P, Add_regs, Shift_regs,
  input Start, Done, Q0, clock, reset_b
);

  State_Generator M0 (T0, T1, T2, Start, Done, Q0, clock, reset_b);
  assign Ready = T0;
  assign Load_regs = T0 && Start;
  assign Incr_P = T1;
  assign Add_regs = T1 && Q0;
  assign Shift_regs = T2;
endmodule

module State_Generator (output T0,T1, T2, input Start, Done, Q0, clock, reset_b);
  parameter        S_idle =   3'b001,          // one-hot code
                   S_add =    3'b010,
                   S_shift =  3'b100;
  reg    [2: 0]    state, next_state;       // sized for one-hot
  assign    T0 = (state == S_idle);
  assign    T1 = (state == S_add);
  assign    T2 = (state == S_shift);

  always @ (posedge clock, negedge reset_b)
    if (~reset_b) state <= S_idle; else state <= next_state;
```

```
 always @ (state, Start, Q0, Done) begin
  next_state = S_idle;
  case (state)
   S_idle:     if (Start) next_state = S_add;
   S_add:      next_state = S_shift;
   S_shift:    if (Done) next_state = S_idle; else next_state = S_add;
   default:    next_state = S_idle;
  endcase
 end
endmodule

module Datapath #(parameter dp_width = 5, BC_size = 3) (
 output [2*dp_width - 1: 0] Product, output Q0, output Done,
 input [dp_width - 1: 0] Multiplicand, Multiplier,
 input Start, Load_regs, Incr_P, Add_regs, Shift_regs, clock, reset_b
);
// Default configuration: 5-bit datapath
 reg   [dp_width - 1: 0]     A, B, Q;               // Sized for datapath
 reg                         C;
 reg   [BC_size - 1: 0]      P;             // Bit counter

 assign Q0 = Q[0];
 assign   Done = (P == dp_width );                  // Multiplier is exhausted
 assign Product = {C, A, Q};
 always @ (posedge clock, negedge reset_b)
  if (reset_b == 0) begin             // Added to this solution, but
   P <= 0;                            // not really necessary since Load_regs
   B <= 0;                            // initializes the datapath
   C <= 0;
   A <= 0;
   Q <= 0;
  end
  else begin
  if (Load_regs) begin
   P <= 0;
   A <= 0;
   C <= 0;
   B <= Multiplicand;
   Q <= Multiplier;
  end
  if (Add_regs) {C, A} <= A + B;
  if (Shift_regs) {C, A, Q} <= {C, A, Q} >> 1;
  if (Incr_P) P <= P+1 ;
 end
endmodule

module t_Supp_11_9cd;
 parameter                   dp_width = 5;          // Width of datapath
 wire  [2 * dp_width - 1: 0] Product;
 wire                        Ready;
 reg   [dp_width - 1: 0]     Multiplicand, Multiplier;
 reg                         Start, clock, reset_b;
 integer                     Exp_Value;
 reg                         Error;

 Supp_11_9cd M0(Product, Ready, Multiplicand, Multiplier, Start, clock, reset_b);

 initial #115000 $finish;
 initial begin clock = 0; #5 forever #5 clock = ~clock; end
initial fork
  reset_b = 1;
  #2 reset_b = 0;
  #3 reset_b = 1;
 join
 always @ (negedge Start) begin
  Exp_Value = Multiplier * Multiplicand;
  //Exp_Value = Multiplier * Multiplicand +1;    // Inject error to confirm detection
 end
 always @ (posedge Ready) begin
  # 1 Error <= (Exp_Value ^ Product) ;
 end

 initial begin
  #5 Multiplicand = 0;
  Multiplier = 0;

  repeat (32) #10 begin
     Start = 1;
     #10 Start = 0;
     repeat (32) begin
       Start = 1;
       #10 Start = 0;
       #100 Multiplicand = Multiplicand + 1;
     end
    Multiplier = Multiplier + 1;
  end
 end
endmodule
```



Supplement to Experiment #10:

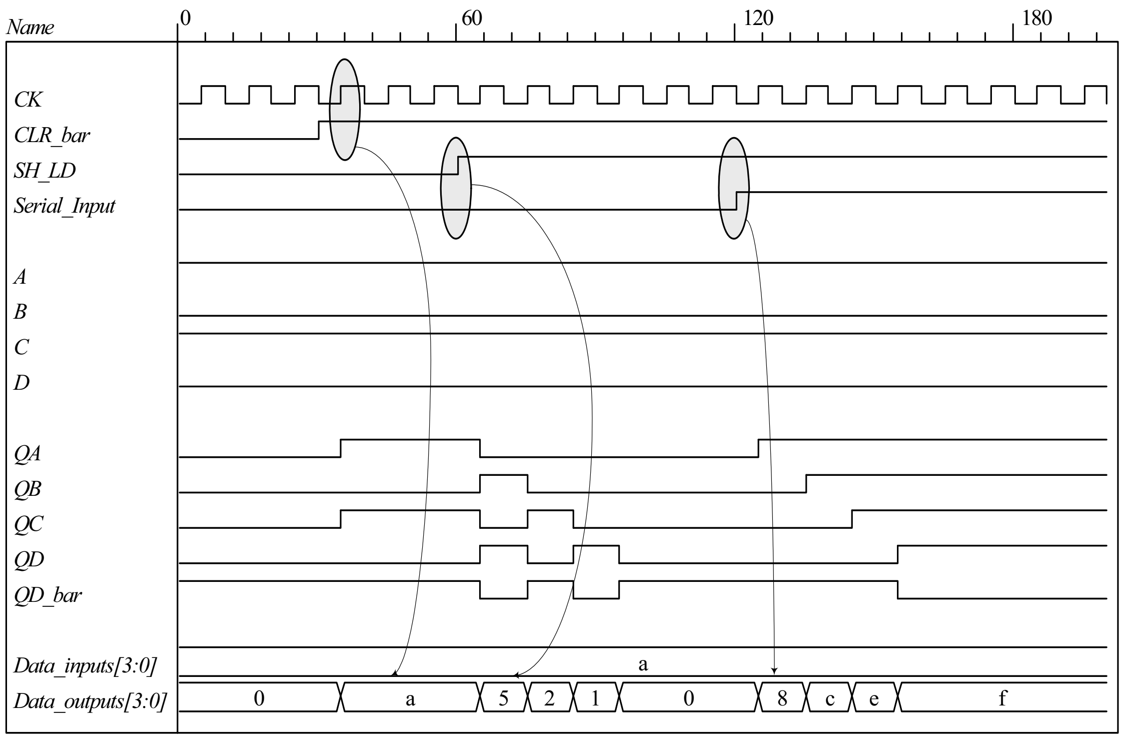
```
module Counter_74161 (
  output      QD, QC, QB, QA,      // Data output
  output      COUT,                // Output carry
  input       D, C, B, A,          // Data input
  input       P, T,                // Active high to count
              L,                   // Active low to load
              CK,                  // Positive edge sensitive
              CLR                  // Active low to clear
);

  reg [3: 0] A_count;
  assign QD = A_count[3];
  assign QC = A_count[2];
  assign QB = A_count[1];
  assign QA = A_count[0];

  assign COUT = ((P == 1) && (T == 1) && (L == 1) && (A_count == 4'b1111));

  always @ (posedge CK, negedge CLR)
    if (CLR == 0)          A_count <= 4'b0000;
    else if (L == 0)       A_count <= {D, C, B, A};
    else if ((P == 1) && (T == 1))  A_count <= A_count + 1'b1;
    else A_count <= A_count; // redundant statement
endmodule

module t_Counter_74161 ();
  wire        QD, QC, QB, QA;
  wire [3: 0] Data_outputs = {QD, QC, QB, QA};
  wire        Carry_out;         // Output carry
  reg [3:0]   Data_inputs;       // Data input
  reg         Count,             // Active high to count
              Load,              // Active low to load
              Clock,             // Positive edge sensitive
              Clear;             // Active low to clear


Counter_74161 M0 (QD, QC, QB, QA, Carry_out,
  Data_inputs[3], Data_inputs[2], Data_inputs[1], Data_inputs[0], Count, Count, Load, Clock, Clear);

  initial #200 $finish;
  initial begin Clock = 0; forever #5 Clock = ~Clock; end

  initial fork
    Clear = 0;
    Load = 1;
    Count = 0;
    #20 Clear = 1;
    #40 Load = 0;
    #50 Load = 1;
    #80 Count = 1;
    #180 Count = 0;
    Data_inputs = 4'ha;        // 10
  join
endmodule
```



Supplement to Experiment #11.

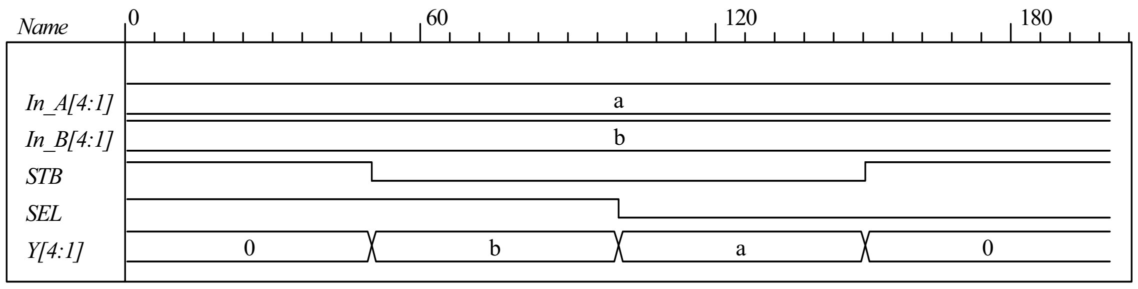
(a)

```
// Note: J and K_bar are assumed to be connected together.
module SReg_74195 (
  output reg QA, QB, QC, QD,
  output QD_bar,
  input A, B, C, D, SH_LD, J, K_bar, CLR_bar, CK
);
  assign QD_bar = ~QD;

  always @ (posedge CK, negedge CLR_bar)
    if (!CLR_bar) {QA, QB, QC, QD} <= 4'b0;
    else if (!SH_LD) {QA, QB, QC, QD} <= {A, B, C, D};
    else case ({J, K_bar})
      2'b00: {QA, QB, QC, QD} <= {1'b0, QA, QB, QC};
      2'b11: {QA, QB, QC, QD} <= {1'b1, QA, QB, QC};
      2'b01: {QA, QB, QC, QD} <= {QA, QA, QB, QC};   // unused
      2'b10: {QA, QB, QC, QD} <= {~QA, QA, QB, QC}; // unused
    endcase
endmodule

module t_SReg_74195 ();
  wire QA, QB, QC, QD;
  wire QD_bar;
  reg A, B, C, D, SH_LD, CLR_bar, CK;
  reg   Serial_Input;
  wire J = Serial_Input;
  wire K_bar = Serial_Input;
  wire [3: 0] Data_inputs = {A, B, C, D};
  wire [3: 0] Data_outputs = {QA, QB, QC, QD};

  SReg_74195 M0 (QA, QB, QC, QD, QD_bar, A, B, C, D, SH_LD, J, K_bar, CLR_bar, CK);

  initial #200 $finish;
  initial begin CK = 0; forever #5 CK = ~CK; end
  initial fork
    {A, B, C, D} = 4'ha;
    CLR_bar = 0;
    Serial_Input = 0;
    SH_LD = 0;
    #30 CLR_bar = 1;
    #60 SH_LD = 1;
    #120 Serial_Input = 1;
  join
endmodule
```


Name
0
60
120
180
CK
CLR_bar
SH_LD
Serial_Input
A
B
C
D
QA
QB
QC
QD
QD_bar
Data_inputs[3:0]
a
Data_outputs[3:0]
0
a
5
2
1
0
8
c
e
f

(b)

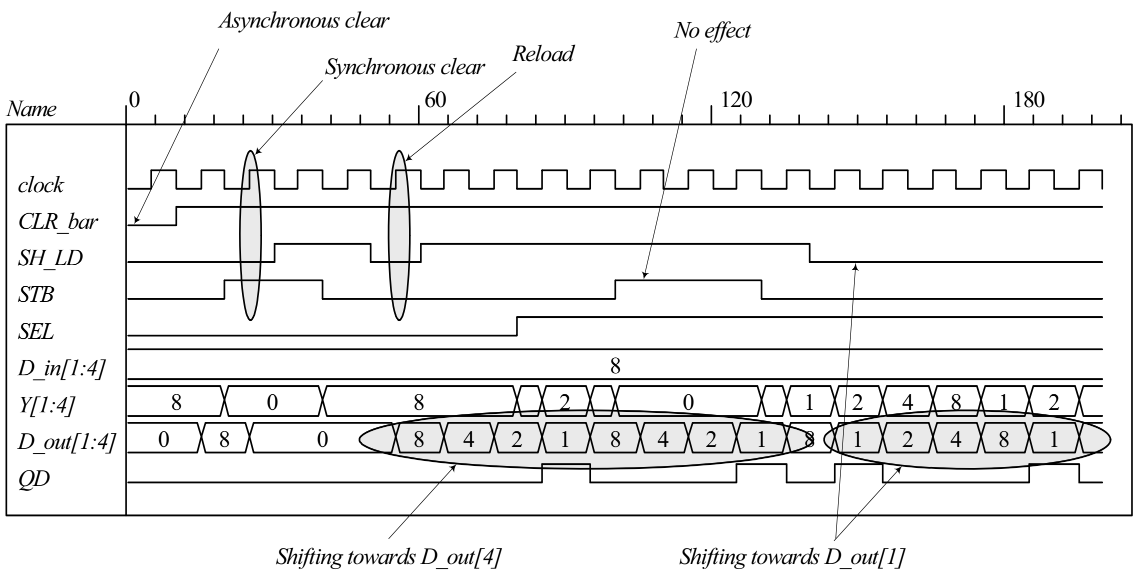
```
module Mux_74157 (
  output reg Y1, Y2, Y3, Y4,
  input A1, A2, A3, A4, B1, B2, B3, B4, SEL, STB
);
  wire [4: 1] In_A = {A1, A2, A3, A4};
  wire [4: 1] In_B = {B1, B2, B3, B4};


  always @ (In_A, In_B, SEL, STB)
    if (STB) {Y1, Y2, Y3, Y4} = 4'b0;
    else if (SEL) {Y1, Y2, Y3, Y4} = In_B;
    else {Y1, Y2, Y3, Y4} = In_A;
endmodule


module t_Mux_74157 ();
  wire Y1, Y2, Y3, Y4;
  reg A1, A2, A3, A4, B1, B2, B3, B4, SEL, STB;
  wire [4: 1] In_A = {A1, A2, A3, A4};
  wire [4: 1] In_B = {B1, B2, B3, B4};
  wire [4: 1] Y = {Y1, Y2, Y3, Y4};

  Mux_74157 M0 (Y1, Y2, Y3, Y4, A1, A2, A3, A4, B1, B2, B3, B4, SEL, STB);

  initial #200 $finish;
  initial fork
    {A1, A2, A3, A4} = 4'ha;
    {B1, B2, B3, B4} = 4'hb;
    STB = 1;
    SEL = 1;
    #50 STB = 0;
    #100 SEL = 0;
    #150 STB = 1;
  join
endmodule
```



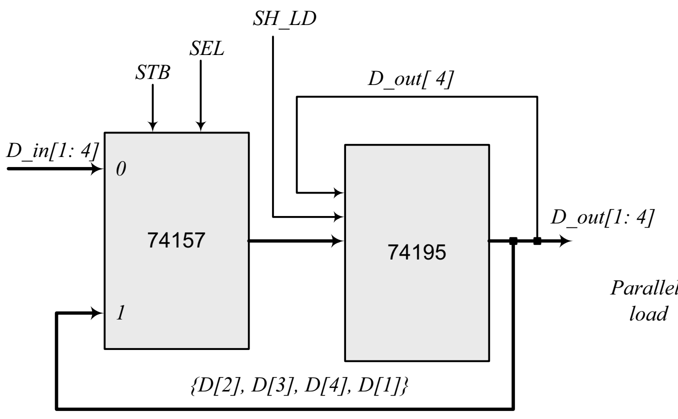
(c)

```
module Bi_Dir_Shift_Reg (output [1: 4] D_out, input [1: 4] D_in, input SEL, STB, SH_LD, clock,
CLR_bar);
  wire          QD_bar;
  wire [1: 4]   Y;
  SReg_74195 M0 (D_out[1], D_out[2], D_out[3], D_out[4], QD_bar, Y[1], Y[2], Y[3], Y[4],
    SH_LD, D_out[4], D_out[4], CLR_bar, clock
  );
```

```
 Mux_74157 M1 (Y[1], Y[2], Y[3], Y[4], D_in[1], D_in[2], D_in[3], D_in[4],
   D_out[2], D_out[3], D_out[4], D_out[1], SEL, STB
 );
endmodule

module SReg_74195 (
 output reg   QA, QB, QC, QD,
 output       QD_bar,
 input        A, B, C, D, SH_LD, J, K_bar, CLR_bar, CK
);
 assign       QD_bar = ~QD;
 always @ (posedge CK, negedge CLR_bar)
   if (!CLR_bar) {QA, QB, QC, QD} <= 4'b0;
   else if (!SH_LD) {QA, QB, QC, QD} <= {A, B, C, D};
   else case ({J, K_bar})
     2'b00:  {QA, QB, QC, QD} <= {1'b0, QA, QB, QC};
     2'b11:  {QA, QB, QC, QD} <= {1'b1, QA, QB, QC};
     2'b01:  {QA, QB, QC, QD} <= {QA, QA, QB, QC};   // unused
     2'b10:  {QA, QB, QC, QD} <= {~QA, QA, QB, QC}; // unused
   endcase
endmodule

module Mux_74157 (
 output reg Y1, Y2, Y3, Y4,
 input A1, A2, A3, A4, B1, B2, B3, B4, SEL, STB
);
 wire [4: 1] In_A = {A1, A2, A3, A4};
 wire [4: 1] In_B = {B1, B2, B3, B4};

 always @ (In_A, In_B, SEL, STB)
   if (STB) {Y1, Y2, Y3, Y4} = 4'b0;
   else if (SEL) {Y1, Y2, Y3, Y4} = In_B;      // SEL = 1
   else {Y1, Y2, Y3, Y4} = In_A;              // SEL = 0
endmodule

module t_Bi_Dir_Shift_Reg ();
 wire [1: 4] D_out;
 reg [1: 4] D_in;
 reg SEL, STB, SH_LD, clock, CLR_bar;
 Bi_Dir_Shift_Reg M0 (D_out, D_in, SEL, STB, SH_LD, clock, CLR_bar);
 initial #200 $finish;
 initial begin clock = 0; forever #5 clock = ~clock; end
 initial fork
   D_in = 4'h8;          // Data for walking 1 to right
   CLR_bar = 0;
   STB = 0;
   SEL = 0;              // Selects D_in
   SH_LD = 0;            // load D_in
   #10 CLR_bar = 1;
   #20 STB = 1;
   #40 STB = 0;
   #30 SH_LD = 1;
   #50 SH_LD = 0;     // Interrupt count to load
   #60 SH_LD = 1;
   #80 SEL = 1;
   #100 STB = 1;
   #130 STB = 0;
   #140 SH_LD = 0;
   //#150 SH_LD = 1;
 join
endmodule
```



The behavioral model is listed below. The two models have matching simulation results.



Note: CLR_b provides active-low asynchronous clear of D_out , overriding the functionality shown in the table below.

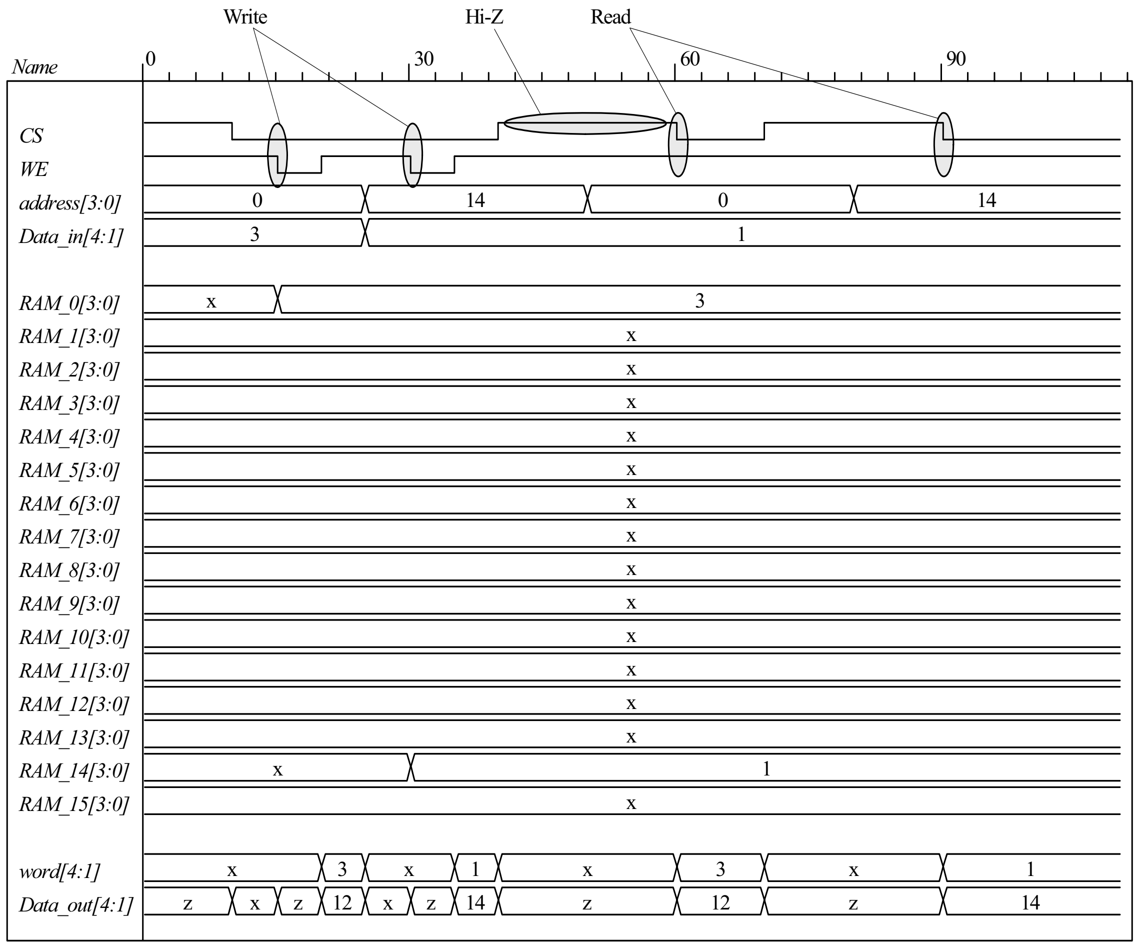
	SH_LD	*STB*	*SEL*	
Parallel load →	*0*	*0*	*0*	*D_out <= D_in*
	0	*0*	*1*	*Shift_D_out towards D[1] (left)*
	0	*1*	*x*	*Synchronous clear: D_out <= 4'b0*
	1	*x*	*x*	*Shift towards D_out[4] (right)*

```
module Bi_Dir_Shift_Reg_beh (output reg [1: 4] D_out, input [1: 4] D_in, input SEL, STB, SH_LD, clock,
CLR_bar);
  always @ (posedge clock, negedge CLR_bar)
    if (!CLR_bar) D_out <= 4'b0;
    else if (SH_LD ) D_out <= {D_out[4], D_out[1], D_out[2], D_out[3]};
   else if (!STB) D_out <= SEL ? {D_out[2: 4], D_out[1]}: D_in;
   else D_out <= 4'b0;
 endmodule

module t_Bi_Dir_Shift_Reg_beh ();
  wire [1: 4] D_out;
  reg [1: 4] D_in;
  reg SEL, STB, SH_LD, clock, CLR_bar;

  Bi_Dir_Shift_Reg_beh M0 (D_out, D_in, SEL, STB, SH_LD, clock, CLR_bar);

  initial #200 $finish;
  initial begin clock = 0; forever #5 clock = ~clock; end
  initial fork
    D_in = 4'h8;           // Data for walking 1 to right
    CLR_bar = 0;
    STB = 0;
    SEL = 0;          // Selects D_in
    SH_LD = 0;      // load D_in
    #10 CLR_bar = 1;
    #20 STB = 1;
    #40 STB = 0;
    #30 SH_LD = 1;
    #50 SH_LD = 0;     // Interrupt count to load
    #60 SH_LD = 1;
    #80 SEL = 1;
    #100 STB = 1;
    #130 STB = 0;
    #140 SH_LD = 0;
    //#150 SH_LD = 1;
  join
endmodule
```

Supplement to Experiment #13.

```
module RAM_74189 (output S4, S3, S2, S1, input D4, D3, D2, D1, A3, A2, A1, A0, CS, WE);
// Note: active-low CS and WE
  wire [3: 0]                address = {A3, A2, A1, A0};
  reg [3: 0]                 RAM [0: 15];             // 16 x 4 memory
  wire [4: 1]                Data_in = { D4, D3, D2, D1}; // Input word
  tri [4: 1]                 Data;                 // Output data word, three-state output
  assign S1 = Data[1];               // Output bits
  assign S2 = Data[2];
  assign S3 = Data[3];
  assign S4 = Data[4];

  always @ (Data_in, address, CS, WE) if (~CS && ~WE) RAM[address] = Data_in;
  assign Data = (~CS && WE) ? ~RAM[address] : 4'bz;
endmodule

module t_RAM_74189 ();
  reg [4: 1] Data_in;
  reg [3: 0] address;
  reg CS, WE;
  wire S1, S2, S3, S4;
  wire D1, D2, D3, D4;
  wire A0, A1, A2, A3;
  wire [4: 1] Data_out = {S4, S3, S2, S1};
  assign  D1 = Data_in [1];
  assign  D2 = Data_in [2];
  assign  D3 = Data_in [3];
  assign  D4 = Data_in [4];
  assign A0 = address[0];
  assign A1 = address[1];
  assign A2 = address[2];
  assign A3 = address[3];

  wire [3: 0] RAM_0 = M0.RAM[0];
  wire [3: 0] RAM_1 = M0.RAM[1];
  wire [3: 0] RAM_2 = M0.RAM[2];
  wire [3: 0] RAM_3 = M0.RAM[3];
  wire [3: 0] RAM_4 = M0.RAM[4];
```

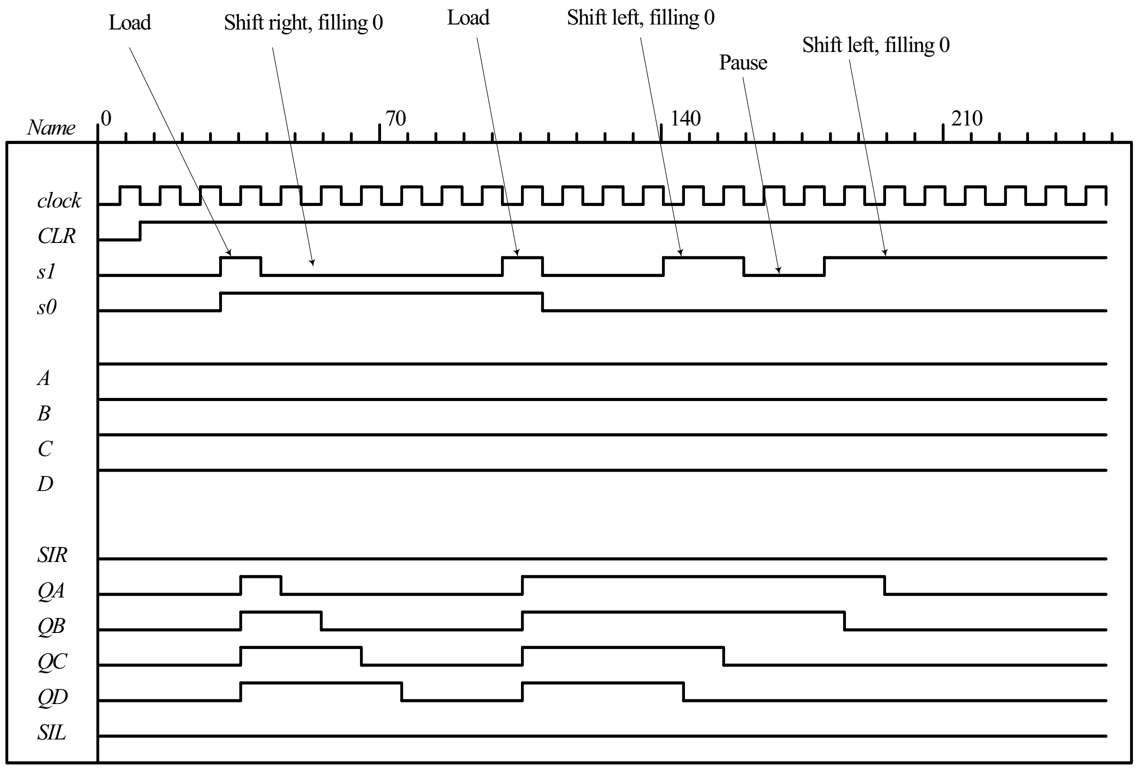
```
 wire [3: 0] RAM_5 = M0.RAM[5];
 wire [3: 0] RAM_6 = M0.RAM[6];
 wire [3: 0] RAM_7 = M0.RAM[7];
 wire [3: 0] RAM_8 = M0.RAM[8];
 wire [3: 0] RAM_9 = M0.RAM[9];
 wire [3: 0] RAM_10 = M0.RAM[10];
 wire [3: 0] RAM_11 = M0.RAM[11];
 wire [3: 0] RAM_12= M0.RAM[12];
 wire [3: 0] RAM_13 = M0.RAM[13];
 wire [3: 0] RAM_14 = M0.RAM[14];
 wire [3: 0] RAM_15 = M0.RAM[15];
 wire [4: 1] word = ~Data_out;

 RAM_74189 M0 (S4, S3, S2, S1, D4, D3, D2, D1, A3, A2, A1, A0, CS, WE);

 initial #110 $finish;
 initial fork
  WE = 1;
  CS = 1;
  address = 0;
  Data_in = 3;
  #10 CS = 0;
  #15 WE = 0;
  #20 WE = 1;
  #25 address = 14;
  #25 Data_in = 1;
  #30 WE = 0;
  #35 WE = 1;
  #40 CS = 1;
  #50 address = 0;
  #60 CS = 0;
  #70 CS = 1;
  #80 address = 14;
  #90 CS = 0;
 join
endmodule
```



Note: Data_out is the complement of the stored value

Supplement to Experiment #14.

```
module Bi_Dir_Shift_Reg_74194 (
  output reg   QA, QB, QC, QD,
  input        A, B, C, D, SIR, SIL, s1, s0, CK, CLR
);
  always @ (posedge CK, negedge CLR)
    if (!CLR) {QA, QB, QC, QD} <= 4'b0;
    else case ({s1, s0})
      2'b00:  {QA, QB, QC, QD} <= {QA, QB, QC, QD};
      2'b01:  {QA, QB, QC, QD} <= {SIR, QA, QB, QC};
      2'b10:  {QA, QB, QC, QD} <= {QB, QC, QD, SIL};
      2'b11:  {QA, QB, QC, QD} <= {A, B, C, D};
    endcase
endmodule

module t_Bi_Dir_Shift_Reg_74194 ();
  wire QA, QB, QC, QD;
  reg A, B, C, D, SIR, SIL, s1, s0, clock, CLR;

  Bi_Dir_Shift_Reg_74194 M0 (QA, QB, QC, QD, A, B, C, D, SIR, SIL, s1, s0, clock, CLR);

  initial #250 $finish;
  initial begin clock = 0; forever #5 clock = ~clock; end
  initial fork
    CLR = 0;
    {A, B, C, D} = 4'hf;
    s1 = 0;
    s0 = 0;
    SIL = 0;
    SIR = 0;
    #10 CLR = 1;
    #30 begin s1 = 1; s0 = 1; end      // load
    #40   s1 = 0;     // shift right
    #100   s1 = 1;   // load
    #110 begin s1 = 0; s0 = 0; end
    #140 s1 = 1;     // shift left
    #160 s1 = 0;     // pause
    #180 s1 = 1;     // resume
   join
endmodule
```



Supplement to Experiment #16.

The HDL behavioral descriptions of the components in the block diagram of Fig. 11.23 are described in the solutions of previous experiments, along with their test benches and simulations results: 74189 is described in Experiment 13(a); 74157 in Experiment 11(b); 74161 in Experiment 10; 7483 in Experiment 7(a); 74194 in Experiment 14; and 7474 in Experiment 8(a). The structural description of the parallel adder instantiates these components to show how they are interconnected (see the solution to the supplement for Experiment 17 for a similar procedure). A test bench and simulation results for the integrated unit are given below.

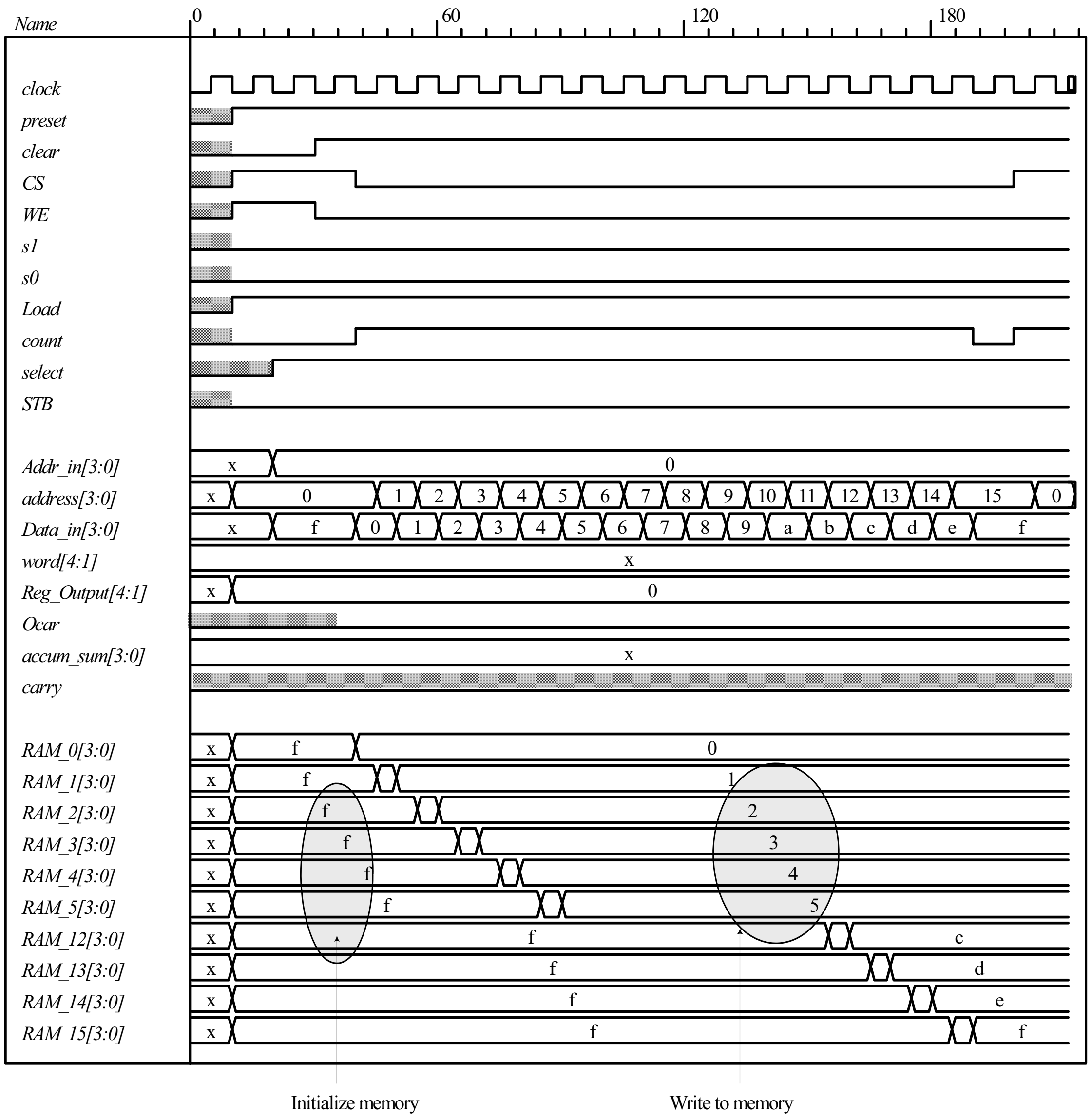
```
// LOAD condition for 74194: s1 = 1, s0 = 1
// SHIFT condition: s1 = 0, s0 = 1
// NO CHANGE condition: s1 = 0, s0 = 0

module Supp_11_16 (
  output [3: 0] accum_sum,
  output carry,
  input [3: 0] Data_in, Addr_in,
  input SIR, SIL, CS, WE, s1, s0, count, Load, select, STB, clock, preset, clear, VCC, GND
);
  wire B4 = Data_in[3];    // Data world to memory
  wire B3 = Data_in[2];
  wire B2 = Data_in[1];
  wire B1 = Data_in[0];
  wire S4, S3, S2, S1;
  wire D4, D3, D2, D1;
  wire S4b = ~S4; // Inverters
  wire S3b = ~S3;
  wire S2b = ~S2;
  wire S1b = ~S1;
  wire D = Addr_in[3];// For parallel load of address counter
  wire C = Addr_in[2];
  wire B = Addr_in[1];
  wire A = Addr_in[0];
  wire Ocar, Y1, Y2, Y3, Y4, QA, QB, QC, QD, A3, A2, A1, A0;
  assign accum_sum = {D4, D3, D2, D1};

  Flip_flop_7474   M0 (Ocar, carry, clock, preset, clear);
  Adder_7483  M1 (D4, D3, D2, D1, carry, S4b, S3b, S2b, S1b, QD, QC, QB, QA, Ocar, VCC, GND);
  Mux_74157   M2 (Y4, Y3, Y2, Y1, QD, QC, QB, QA, B4, B3, B2, B1, select, STB);
  Counter_74161  M3 (A3, A2, A1, A0, COUT, D, C, B, A, count, count, Load, clock, clear);
  RAM_74189 M4 (S4, S3, S2, S1, Y4, Y3, Y2, Y1, A3, A2, A1, A0, CS, WE);
  Reg_74194   M5 (QD, QC, QB, QA, D4, D3, D2, D1, Ocar, SIL, s1, s0, clock, clear);
endmodule

module t_Supp_11_16 ();
  wire [3: 0] sum;
  wire carry;
  reg [3: 0] Data_in, Addr_in;
  reg SIR, SIL, CS, WE, s1, s0, count, Load, select, STB, clock, preset, clear;
  supply1 VCC;
  supply0 GND;
  wire [3: 0] RAM_0 = M0.M4.RAM[0];
  wire [3: 0] RAM_1 = M0.M4.RAM[1];
  wire [3: 0] RAM_2 = M0.M4.RAM[2];
  wire [3: 0] RAM_3 = M0.M4.RAM[3];
  wire [3: 0] RAM_4 = M0.M4.RAM[4];
  wire [3: 0] RAM_5 = M0.M4.RAM[5];
  wire [3: 0] RAM_6 = M0.M4.RAM[6];
```

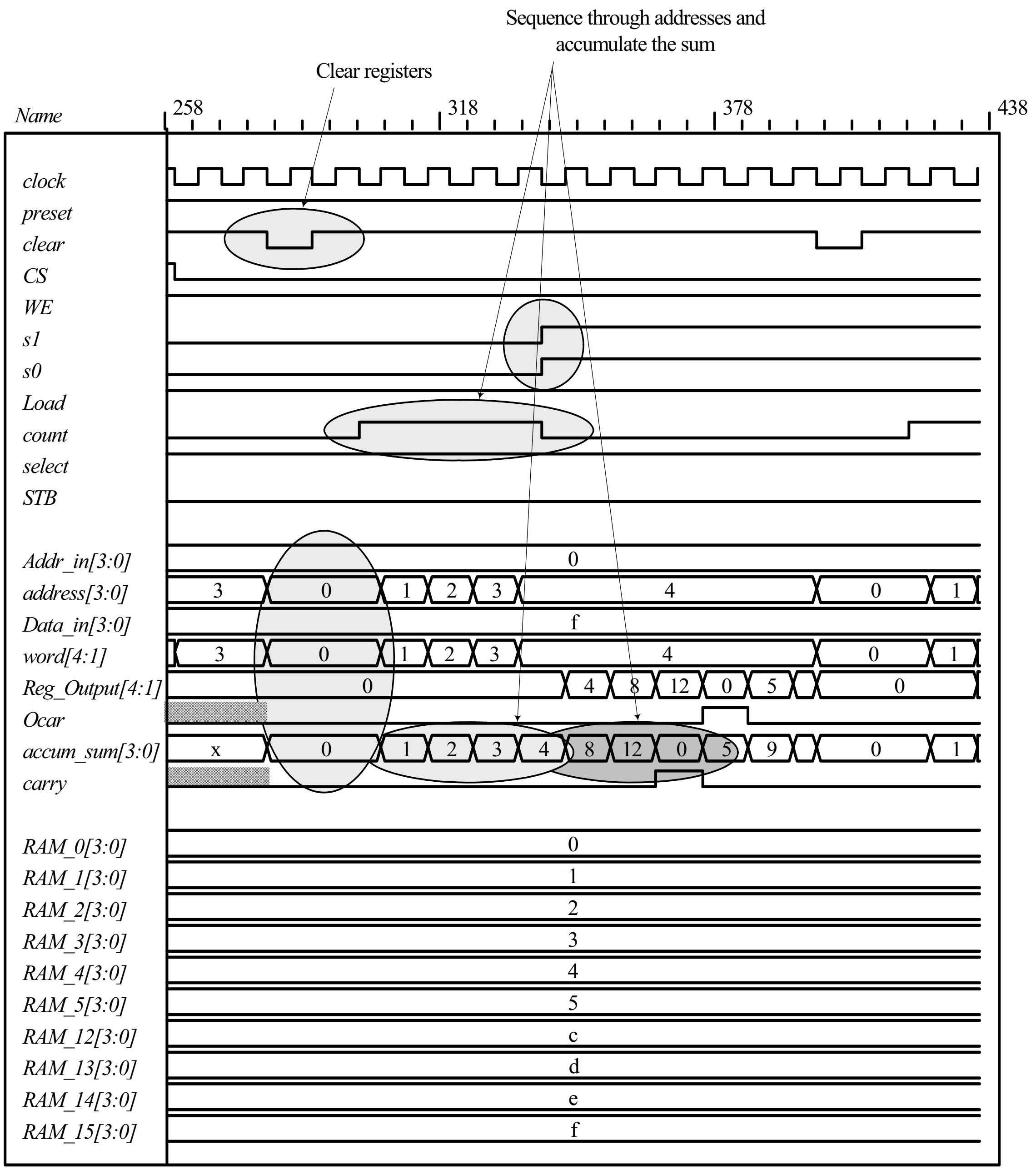
```
 wire [3: 0] RAM_7 = M0.M4.RAM[7];
 wire [3: 0] RAM_8 = M0.M4.RAM[8];
 wire [3: 0] RAM_9 = M0.M4.RAM[9];
 wire [3: 0] RAM_10 = M0.M4.RAM[10];
 wire [3: 0] RAM_11 = M0.M4.RAM[11];
 wire [3: 0] RAM_12= M0.M4.RAM[12];
 wire [3: 0] RAM_13 = M0.M4.RAM[13];
 wire [3: 0] RAM_14 = M0.M4.RAM[14];
 wire [3: 0] RAM_15 = M0.M4.RAM[15];

 wire [4: 1] word = {M0.S4b, M0.S3b,M0.S2b, M0.S1b};
wire [4: 1] mux_out = { M0.Y4, M0.Y3, M0.Y2, M0.Y1};
wire [4: 1] Reg_Output = {M0.QD, M0.QC, M0.QB, M0.QA};

Supp_11_16 M0 (sum, carry, Data_in, Addr_in, SIR, SIL, CS, WE, s1, s0, count, Load,
 select, STB,   clock, preset, clear, VCC, GND);

 integer k;
 initial #600 $finish;
 initial begin clock = 0; forever #5 clock = ~clock; end
 initial fork
   #10 begin preset = 1; clear = 0; s1 = 0; s0 = 0; Load = 1; count = 0; CS = 1; WE = 1; STB = 0;  end
   // initialize memory
   #10 begin k = 0; repeat (16) begin M0.M4.RAM[k] = 4'hf; k = k + 1; end end
   #20 begin Data_in = 4'hf; Addr_in  = 0; select = 1; end
   #30 begin clear = 1; WE = 0; end
     // load memory
   #40 begin
     count = 1;
     CS = 0;
     begin
       repeat (16) @ (negedge clock) Data_in = Data_in + 1;
       count = 0;
       @ (negedge clock) CS = 1;
     end
   end
   #200 count = 1;              // Establish address
   #240 count = 0;
   #250 WE = 1;
   #260 CS = 0;                 // Read from memory
   #280 clear = 0;
   #290 clear = 1;
   #300 count = 1;              // Establish address
   #340 begin s1 = 1; s0 = 1; count = 0; end
   #390 CS = 0;
   #400 clear = 0;          // Clear the registers
   #410 clear = 1;
   #420 begin count = 1; CS = 0; end        // Accumulate values
   #490 begin count = 0; CS = 1; end
 join
endmodule

module Flip_flop_7474 (output reg Q, input D, CLK, preset, clear);
  always @ (posedge CLK, negedge preset , negedge clear)
  if (!preset)              Q <= 1'b1;
  else if (!clear)          Q <= 1'b0;
  else                      Q <= D;
endmodule
```

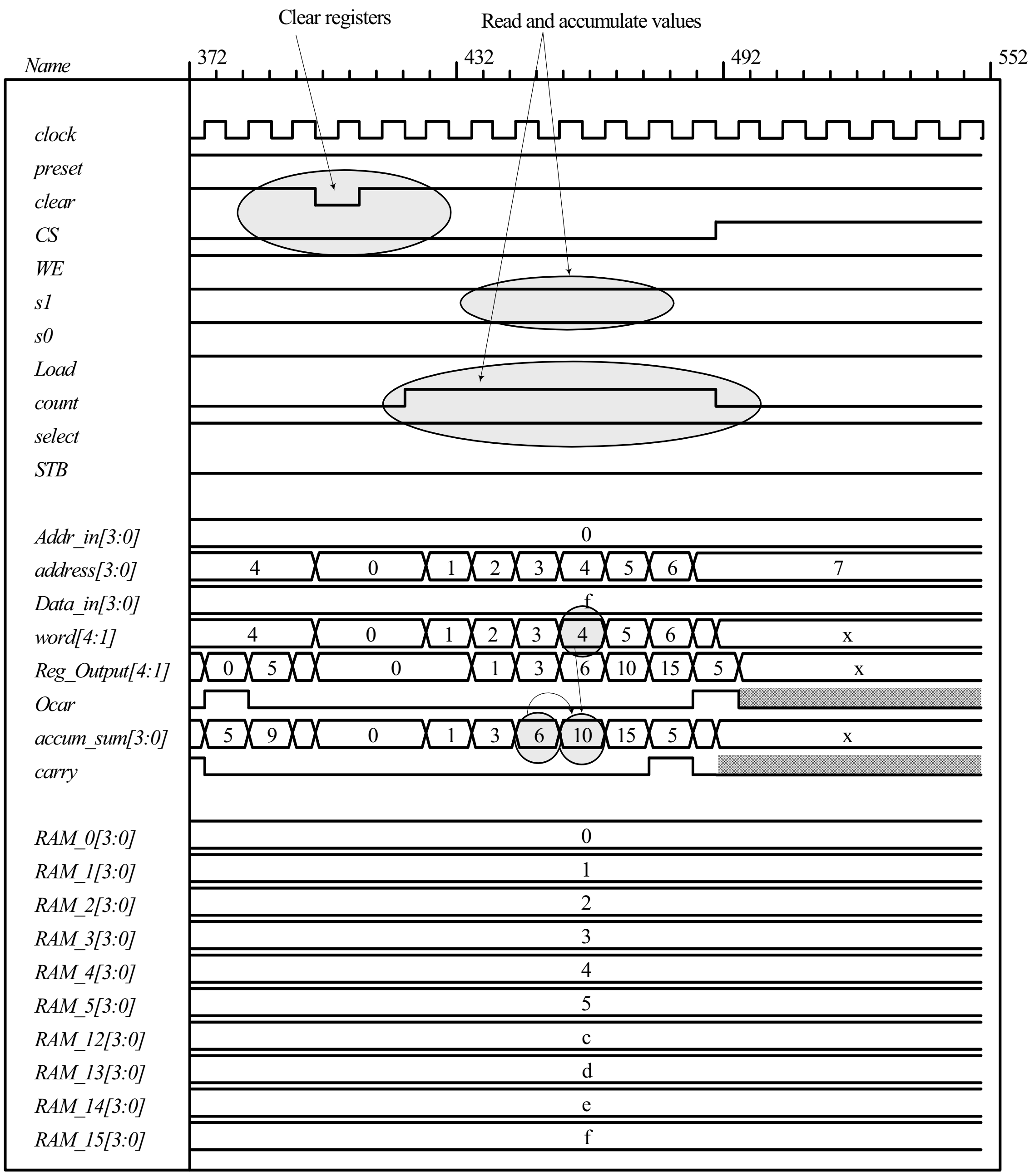
```
module Adder_7483 (
  output S4, S3, S2, S1, C4,
  input A4, A3, A2, A1, B4, B3, B2, B1, C0, VCC, GND
);
// Note: connect VCC and GND to supply1 and supply0 in the test bench
  wire [4: 1] sum;
  wire [4: 1] A = {A4, A3, A2, A1};
  wire [4: 1] B = {B4, B3, B2, B1};
  assign S4 = sum[4];
  assign S3 = sum[3];
  assign S2 = sum[2];
  assign S1 = sum[1];
  assign {C4, sum} = A + B + C0;
endmodule

module Mux_74157 (
  output reg Y1, Y2, Y3, Y4,
  input A1, A2, A3, A4, B1, B2, B3, B4, SEL, STB
);
  wire [4: 1] In_A = {A1, A2, A3, A4};
  wire [4: 1] In_B = {B1, B2, B3, B4};

  always @ (In_A, In_B, SEL, STB)
    if (STB) {Y1, Y2, Y3, Y4} = 4'b0;
    else if (SEL) {Y1, Y2, Y3, Y4} = In_B;
    else {Y1, Y2, Y3, Y4} = In_A;
endmodule

module Counter_74161 (
  output    QD, QC, QB, QA,     // Data output
  output    COUT,               // Output carry
  input     D, C, B, A,         // Data input
  input     P, T,               // Active high to count
            L,             // Active low to load
            CK,                 // Positive edge sensitive
            CLR                 // Active low to clear
);
reg [3: 0] A_count;
assign QD = A_count[3];
assign QC = A_count[2];
assign QB = A_count[1];
assign QA = A_count[0];
assign COUT = ((P == 1) && (T == 1) && (L == 1) && (A_count == 4'b1111));
```

```
always @ (posedge CK, negedge CLR)
if (CLR == 0)                   A_count <= 4'b0000;
else if (L == 0)                A_count <= {D, C, B, A};
else if ((P == 1) && (T == 1))  A_count <= A_count + 1'b1;
else                            A_count <= A_count; // redundant statement
endmodule

module RAM_74189 (output S4, S3, S2, S1, input D4, D3, D2, D1, A3, A2, A1, A0, CS, WE);
// Note: active-low CS and WE
  wire [3: 0]   address = {A3, A2, A1, A0};
  reg [3: 0]    RAM [0: 15];          // 16 x 4 memory
  wire [4: 1]   Data_in = { D4, D3, D2, D1}; // Input word
  tri [4: 1]    Data;                 // Output data word, three-state output
  assign S1 = Data[1];                // Output bits
  assign S2 = Data[2];
  assign S3 = Data[3];
  assign S4 = Data[4];

  always @ (Data_in, address, CS, WE) if (~CS && ~WE) RAM[address] = Data_in;
  assign Data = (~CS && WE) ? ~RAM[address] : 4'bz;    // Note complement of data word
endmodule

module Reg_74194 (
  output reg   QA, QB, QC, QD,
  input        A, B, C, D, SIR, SIL, s1, s0, CK, CLR
);
  always @ (posedge CK, negedge CLR)
    if (!CLR) {QA, QB, QC, QD} <= 4'b0;
    else case ({s1, s0})
      2'b00: {QA, QB, QC, QD} <= {QA, QB, QC, QD};
      2'b01: {QA, QB, QC, QD} <= {SIR, QA, QB, QC};
      2'b10: {QA, QB, QC, QD} <= {QB, QC, QD, SIL};
      2'b11: {QA, QB, QC, QD} <= {A, B, C, D};
    endcase
endmodule
```

Simulation results: initializing memory to 4'hf, then writing to memory. Note: the values of the inputs are ambiguous until the clear signal is asserted. Signals Ocar and carry are ambiguous because the output of memory is high-z until memory is read is read.


Name
0
60
120
180
clock
preset
clear
CS
WE
s1
s0
Load
count
select
STB
Addr_in[3:0]
address[3:0]
Data_in[3:0]
word[4:1]
Reg_Output[4:1]
Ocar
accum_sum[3:0]
carry
RAM_0[3:0]
RAM_1[3:0]
RAM_2[3:0]
RAM_3[3:0]
RAM_4[3:0]
RAM_5[3:0]
RAM_12[3:0]
RAM_13[3:0]
RAM_14[3:0]
RAM_15[3:0]
Initialize memory
Write to memory


Sequence through addresses and accumulate the sum
Clear registers
Name
258
318
378
438
clock
preset
clear
CS
WE
s1
s0
Load
count
select
STB
Addr_in[3:0]
address[3:0]
Data_in[3:0]
word[4:1]
Reg_Output[4:1]
Ocar
accum_sum[3:0]
carry
RAM_0[3:0]
RAM_1[3:0]
RAM_2[3:0]
RAM_3[3:0]
RAM_4[3:0]
RAM_5[3:0]
RAM_12[3:0]
RAM_13[3:0]
RAM_14[3:0]
RAM_15[3:0]



Supplement to Experiment #17.

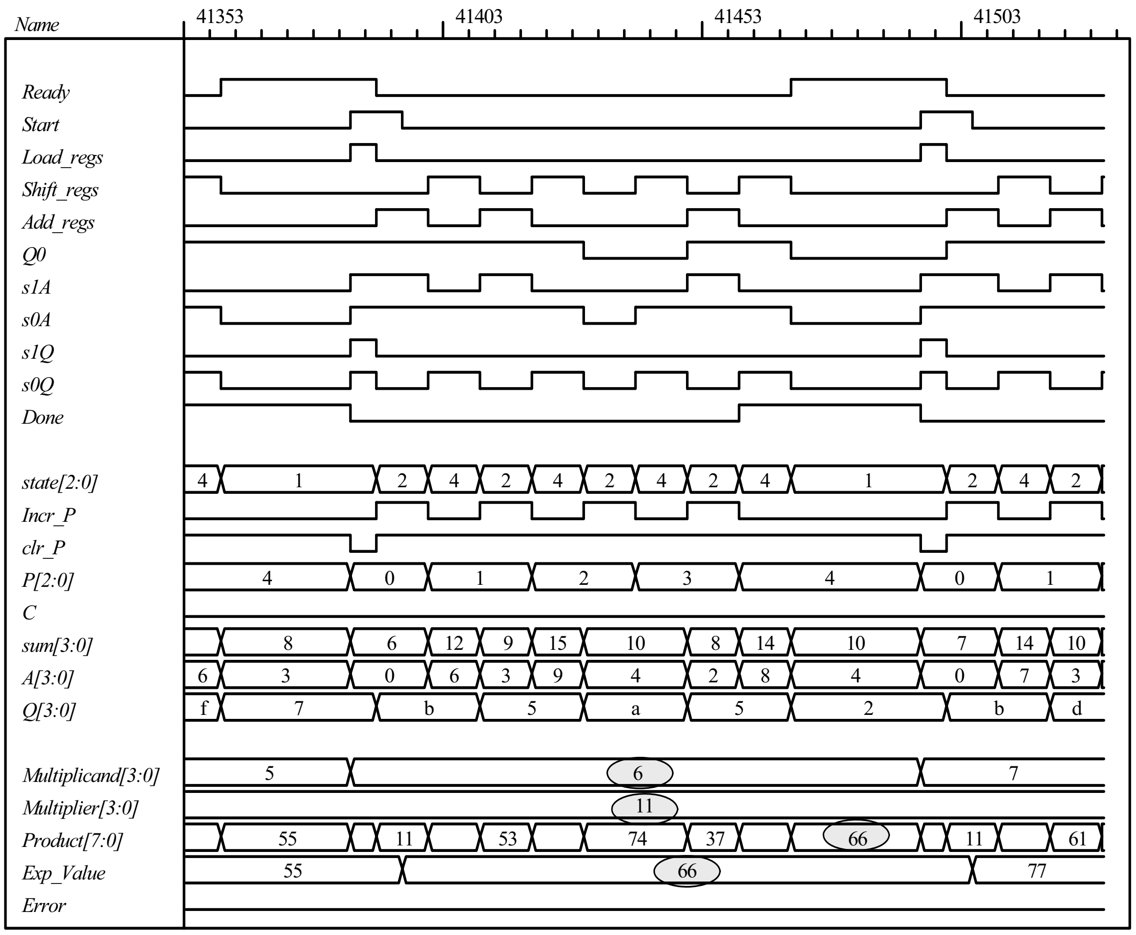
The HDL behavioral descriptions of the components in the block diagram of Fig. 11.23 are described in the solutions of previous experiments, along with their test benches and simulations results: 74161 in Experiment 10; 7483 in Experiment 7(a); 74194 in Experiment 14; and 7474 in Experiment 8(a). The structural description of the parallel adder instantiates these components to show how they are interconnected (see the solution to the supplement for Experiment 17 for a similar procedure). A test bench and simulation results for the integrated unit are given below.

```
// Control unit is obtained by modifying the solution to Prob. 8.24.
// Datapath is implemented with a structural  HDL model and IC components.
// LOAD condition for 74194: s1 = 1, s0 = 1
// SHIFT condition: s1 = 0, s0 = 1
// NO CHANGE condition: s1 = 0, s0 = 0

module Supp_11_17_Par_Mult # (parameter dp_width = 4)
(
 output    [2*dp_width - 1: 0]      Product,
 output                     Ready,
 input     [dp_width - 1: 0]        Multiplicand, Multiplier,
 input                      Start, clock, reset_b, VCC, GND
);
 wire Load_regs, Incr_P, Add_regs, Shift_regs, Done, Q0;

 Controller M0 (
  Ready, Load_regs, Incr_P, Add_regs, Shift_regs, Start, Done, Q0,
  clock, reset_b);

 Datapath M1(Product, Q0, Done, Multiplicand, Multiplier,
 Start, Load_regs, Incr_P, Add_regs, Shift_regs, clock, reset_b, VCC, GND);
endmodule

module Controller (
 output Ready,
 output reg Load_regs, Incr_P, Add_regs, Shift_regs,
 input Start, Done, Q0, clock, reset_b
);

 parameter   S_idle =    3'b001,          // one-hot code
             S_add = 3'b010,
             S_shift = 3'b100;
 reg    [2: 0]   state, next_state;      // sized for one-hot
 assign          Ready = (state == S_idle);

 always @ (posedge clock, negedge reset_b)
  if (~reset_b) state <= S_idle; else state <= next_state;
 always @ (state, Start, Q0, Done) begin
  next_state = S_idle;
  Load_regs = 0;
  Incr_P = 0;
  Add_regs = 0;
  Shift_regs = 0;
  case (state)
   S_idle:if (Start) begin next_state = S_add; Load_regs = 1; end
   S_add:     begin next_state = S_shift; Incr_P = 1; if (Q0) Add_regs = 1; end
   S_shift:   begin
               Shift_regs = 1;
               if (Done) next_state = S_idle;
               else next_state = S_add;
             end
   default:   next_state = S_idle;
  endcase
 end
endmodule

module Datapath #(parameter dp_width = 4, BC_size = 3) (
 output [2*dp_width - 1: 0] Product, output Q0, output Done,
 input [dp_width - 1: 0] Multiplicand, Multiplier,
 input Start, Load_regs, Incr_P, Add_regs, Shift_regs, clock, clear, VCC, GND
);
```

```
  wire C;
  wire Cout, Sum3, Sum2, Sum1, Sum0, P3, P2, P1, P0, A3, A2, A1, A0;
  wire Q3, Q2, Q1;
  wire [dp_width -1: 0] A = {A3, A2, A1, A0};
  wire [dp_width -1: 0] Q = {Q3, Q2, Q1, Q0};
  assign Product = {C, A, Q};
   wire [ BC_size -1: 0] P = {P3, P2, P1, P0};

// Registers must be controlled separately to execute add and shift operations correctly.
// LOAD condition for 74194: s1 = 1, s0 = 1
// SHIFT condition: s1 = 0, s0 = 1
// NO CHANGE condition: s1 = 0, s0 = 0

  wire B3 = Multiplicand[3];        // Data word to adder
  wire B2 = Multiplicand[2];
  wire B1 = Multiplicand[1];
  wire B0 = Multiplicand[0];
  wire Q3_in = Multiplier[3];
  wire Q2_in = Multiplier[2];
  wire Q1_in = Multiplier[1];
  wire Q0_in = Multiplier[0];
  assign Done = ({P3, P2, P1, P0} == dp_width);       // Counts bits of multiplier
  wire s1A = Load_regs || Add_regs;              // Controls for A register
  wire s0A = Load_regs || Add_regs || Shift_regs;
  wire s0Q = Load_regs || Shift_regs;            // Controls for Q register
  wire s1Q = Load_regs;
   wire Pout;                        // Unused
  wire clr_P = clear && ~Load_regs;
  Flip_flop_7474   M0_C (C, Cout, clock, VCC, clr_P);
  Adder_7483 M1 (Sum3, Sum2, Sum1, Sum0, Cout, A3, A2, A1, A0, B3, B2, B1, B0, GND, VCC, GND);
  Counter_74161  M3_P (P3, P2, P1, P0, Pout, GND, GND, GND, GND, Incr_P, Incr_P, VCC, clock,
clr_P);
  Reg_74194  M4_A (A3, A2, A1, A0, Sum3, Sum2, Sum1, Sum0, C, GND, s1A, s0A, clock, clr_P);
  Reg_74194  M5_Q (Q3, Q2, Q1, Q0, Q3_in, Q2_in, Q1_in, Q0_in, A0, GND, s1Q, s0Q, clock, clear);

endmodule

module t_Supp_11_17_Par_Mult;
  parameter        dp_width = 4;           // Width of datapath
  wire  [2 * dp_width - 1: 0]   Product;
  wire             Ready;
  reg    [dp_width - 1: 0]      Multiplicand, Multiplier;
  reg              Start, clock, reset_b;
  integer              Exp_Value;
  reg              Error;
  supply0          GND;
  supply1          VCC;
  Supp_11_17_Par_Mult M0 (Product, Ready, Multiplicand, Multiplier, Start, clock, reset_b, VCC, GND);
  wire [dp_width -1: 0] sum = {M0.M1.Sum3, M0.M1.Sum2, M0.M1.Sum1, M0.M1.Sum0};
  initial #115000 $finish;
  initial begin clock = 0; #5 forever #5 clock = ~clock; end
  initial fork
    reset_b = 1;
    #2 reset_b = 0;
    #3 reset_b = 1;
  join
  always @ (negedge Start) begin
    Exp_Value = Multiplier * Multiplicand;
    //Exp_Value = Multiplier * Multiplicand +1;    // Inject error to confirm detection
  end
```

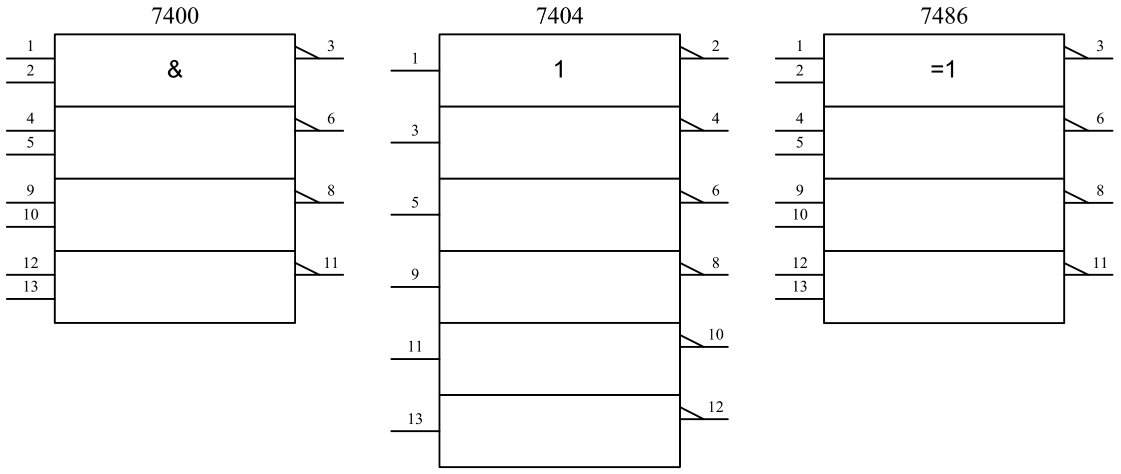
```
always @ (posedge Ready) begin
   # 1 Error <= (Exp_Value ^ Product) ;
  end
  initial begin
   #5 Multiplicand = 0;
   Multiplier = 0;
   repeat (32) #10 begin
      Start = 1;
       #10 Start = 0;
       repeat (32) begin
         Start = 1;
         #10 Start = 0;
         #100 Multiplicand = Multiplicand + 1;
       end
     Multiplier = Multiplier + 1;
   end
  end
endmodule

module Flip_flop_7474 (output reg Q, input D, CLK, preset, clear);
  always @ (posedge CLK, negedge preset , negedge clear)
  if (!preset)              Q <= 1'b1;
  else if (!clear)          Q <= 1'b0;
  else                      Q <= D;
endmodule

module Adder_7483 (
  output S4, S3, S2, S1, C4,
  input A4, A3, A2, A1, B4, B3, B2, B1, C0, VCC, GND
);
// Note: connect VCC and GND to supply1 and supply0 in the test bench
  wire [4: 1] sum;
  wire [4: 1] A = {A4, A3, A2, A1};
  wire [4: 1] B = {B4, B3, B2, B1};
  assign S4 = sum[4];
  assign S3 = sum[3];
  assign S2 = sum[2];
  assign S1 = sum[1];
  assign {C4, sum} = A + B + C0;
endmodule

module Counter_74161 (
  output   QD, QC, QB, QA,     // Data output
  output       COUT,          // Output carry
  input    D, C, B, A,     // Data input
  input        P, T,             // Active high to count
         L,            // Active low to load
         CK,             // Positive edge sensitive
         CLR             // Active low to clear
);
reg [3: 0] A_count;
assign QD = A_count[3];
assign QC = A_count[2];
assign QB = A_count[1];
assign QA = A_count[0];
assign COUT = ((P == 1) && (T == 1) && (L == 1) && (A_count == 4'b1111));
```

```
always @ (posedge CK, negedge CLR)
if (CLR == 0)            A_count <= 4'b0000;
else if (L == 0)              A_count <= {D, C, B, A};
else if ((P == 1) && (T == 1))     A_count <= A_count + 1'b1;
else                 A_count <= A_count; // redundant statement
endmodule

module Reg_74194 (
  output reg QA, QB, QC, QD,
  input A, B, C, D, SIR, SIL, s1, s0, CK, CLR
);
  always @ (posedge CK, negedge CLR)
    if (!CLR) {QA, QB, QC, QD} <= 4'b0;
    else case ({s1, s0})
      2'b00: {QA, QB, QC, QD} <= {QA, QB, QC, QD};
      2'b01: {QA, QB, QC, QD} <= {SIR, QA, QB, QC};
      2'b10: {QA, QB, QC, QD} <= {QB, QC, QD, SIL};
      2'b11: {QA, QB, QC, QD} <= {A, B, C, D};
    endcase
endmodule
```



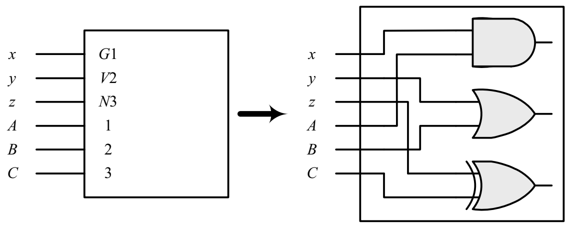
CHAPTER 12

12.1

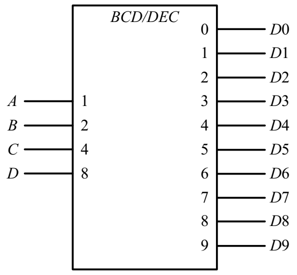


12.2 See textbook.

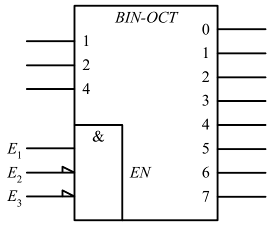
12.3



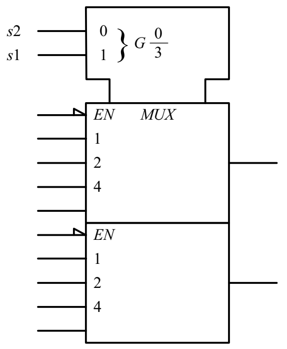
12.4 BCD-to-decimal decoder (similar to IC 7442)



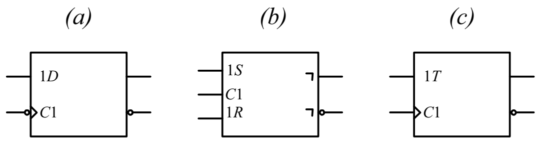
12.5 Similar to 7438:



12.6 IC type 74153.

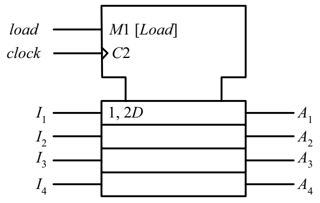


12.7



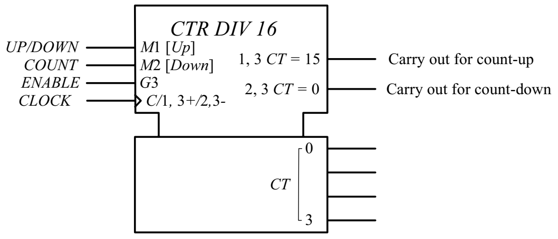
12.8 The common control block is used when the circuit has one or more inputs that are common to all lower sections.

12.9

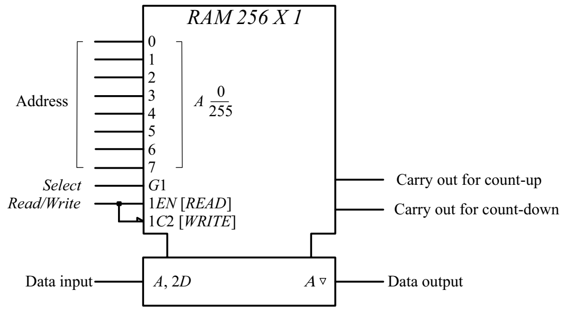


12.10 See textbook.

12.11



12.12


RAM 256 X 1
Address
0
1
2
3
4
5
6
7
A 0/255
Select
Read/Write
G1
1EN [READ]
1C2 [WRITE]
Carry out for count-up
Carry out for count-down
Data input
A, 2D
A ▿
Data output