CMR INSTITUTE OF TECHNOLOGY

USN | | | | | | | | | | |

Internal Assessment Test 2 – November 2016

Sub:	Digital Electronics						Code:	15EC33
Date:	/11/2016	Duration:	90 mins	Max Marks:	50	Sem: 3	Branch:	ECE/TCE

Note: **Answer to the point. Sketch figures wherever necessary. Q3 is compulsory.**

Q1 (a) State the advantages of look ahead carry adder over ripple carry adder using equations and circuit diagram (Assume 4 bit adder for example).

[10](CO3-L3)

OR

Q1 (b) Design a 4 bit BCD adder using IC 7483.

Q1 (c) Implement 12 bit comparator using IC 7485.

[5 + 5](CO2-L3, CO2-L3)

Q2.(a) Implement the following Boolean functions -

(i) $f(a,b,c) = \sum m(1,3,5,6)$ using 4:1 MUX

(ii) $f(a,b,c,d) = \sum m(0,1,5,6,7,9,10,15)$ using 8:1 MUX.

Q2 (b) Implement the following functions using two IC 74138

$$f_1(p,q,r,s) = \sum m(0,4,8,10,14,15) \ \& \ f_2(p,q,r,s) = \pi M(1,3,5,7,9,11,13)$$

OR

Q2 (c) i) With the aid of block diagram clearly distinguish between a decoder and an encoder.

ii) Implement 4:16 decoder using Decoder 3:8.

Q2 (d) Implement full subtractor using single 3:8 decoder.

[2.5 + 2.5 + 5] (CO2-L2,CO2-L3)

Q3. (a) Explain the construction of master-slave JK flip flop and list its benefits.

(b) Explain an application of SR Latch with the required diagrams.

[5+5](CO4-L2)

Q4. (a) Explain Universal shift register.

[10](CO5-L2)

OR

Q4 (b) Draw the circuit and timing diagram for a 4-bit johnson counter with the starting value of 1000.

(c) Design asynchronous mod 8 binary ripple counter. Show its circuit and timing diagram.

[5+5](CO6-L4)

Q5 (a) Design a Mod -6 synchronous counter using clocked JK Flip Flop. Let the count sequence be 0 -> 2 -> 3 -> 6 -> 5 -> 1 -> 0 …

OR

Q5 (b) Design and implement a synchronous 3 bit up/ down counter using D flip flop.

[10](CO5-L3)

--ALL THE BEST—

CO – course outcome, Ln – Bloom's Taxonomy Level

Q1. (a) State the advantages of Look ahead carry adder over ripple carry adder using equations and circuit diagram (Assume 4 bit adder for example)

Ans:

5.11 Fast Adder (LOOK AHEAD CARRY ADDER)

The parallel adder discussed in the last paragraph is ripple carry type in which the carry output of each full-adder stage is connected to the carry input of the next higher-order stage. Therefore, the sum and carry outputs of any stage cannot be produced until the input carry occurs; this leads to a time delay in the addition process. This delay is known as *carry propagation delay*, which can be best explained by considering the following addition.

```
  0 1 0 1
+ 0 0 1 1
---------
  1 0 0 0
```

Addition of the LSB position produces a carry into the second position. This carry, when added to the bits of the second position (stage), produces a carry into the third position. The latter carry, when added to the bits of the third position, produces a carry into the last position. The key thing to notice in this example is that the sum bit generated in the last position (MSB) depends on the carry that was generated by the addition in the previous positions. This means that, adder will not produce correct result until LSB carry has propagated through the intermediate full-adders. This represents a time delay that depends on the propagation delay produced in an each full-adder. For example, if each full-adder is considered to have a propagation delay of 30 ns, then S_3 will not reach its correct value until 90 ns after LSB carry is generated. Therefore, total time required to perform addition is 90+30 = 120 ns.

Obviously, this situation becomes much worse if we extend the adder circuit to add a greater number of bits. If the adder were handling 16-bit numbers, the carry propagation delay could be 480 ns.

One method of speeding up this process by eliminating inter stage carry delay is called **look ahead-carry addition.** This method utilizes logic gates to look at the lower-order bits of the augend and addend to see if a higher-order carry is to be generated. It uses two functions : carry generate and carry propagate.

Consider the circuit of the full-adder shown in Fig. 5.22. Here, we define two functions : carry generate and carry propagate.

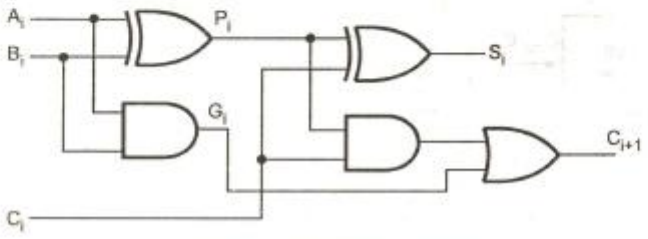


Fig. 5.22 Full-adder circuit

$$P_i = A_i \oplus B_i$$

$$G_i = A_i B_i$$

The output sum and carry can be expressed as

$$S_i = P_i \oplus C_i$$

$$C_{i+1} = G_i + P_i C_i$$

G_i is called a carry generate and it produces on carry when both A_i and B_i are one, regardless of the input carry. P_i is called a carry propagate because it is term associated with the propagation of the carry from C_i to C_{i+1}.

Now the Boolean function for the carry output of each stage can be written as follows.

$$C_2 = G_1 + P_1 C_1$$

$$C_3 = G_2 + P_2 C_2 = G_2 + P_2 (G_1 + P_1 C_1)$$

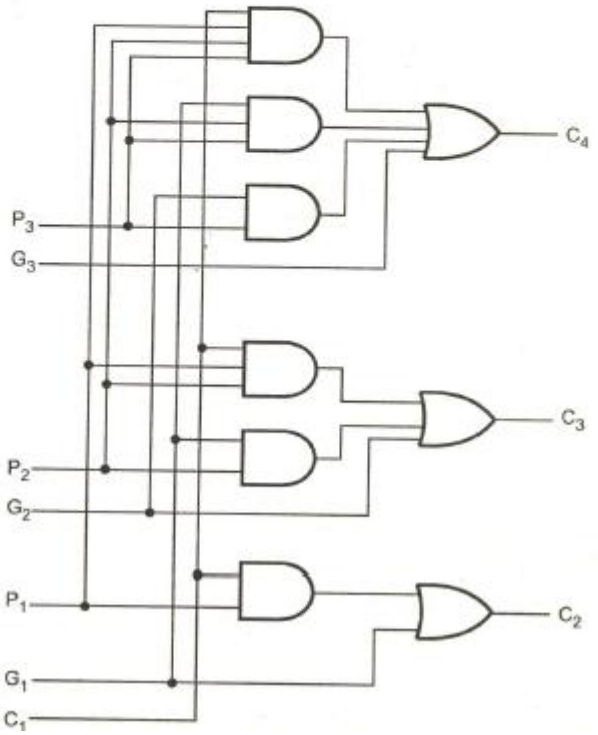
$$= G_2 + P_2 G_1 + P_2 P_1 C_1$$

$$C_4 = G_3 + P_3 C_3 = G_3 + P_3 (G_2 + P_2 G_1 + P_2 P_1 C_1)$$

$$= G_3 + P_3 G_2 + P_3 P_2 G_1 + P_3 P_2 P_1 C_1$$

From the above Boolean function it can be seen that C_4 does not have to wait for C_3 and C_2 to propagate; in fact C_4 is propagated at the same time as C_2 and C_3.

The Boolean functions for each output carry are expressed in sum-of product form, thus they can be implemented using AND-OR logic or NAND-NAND logic. Fig. 5.23 shows implementation of Boolean functions for C_2, C_3 and C_4 using AND-OR logic.



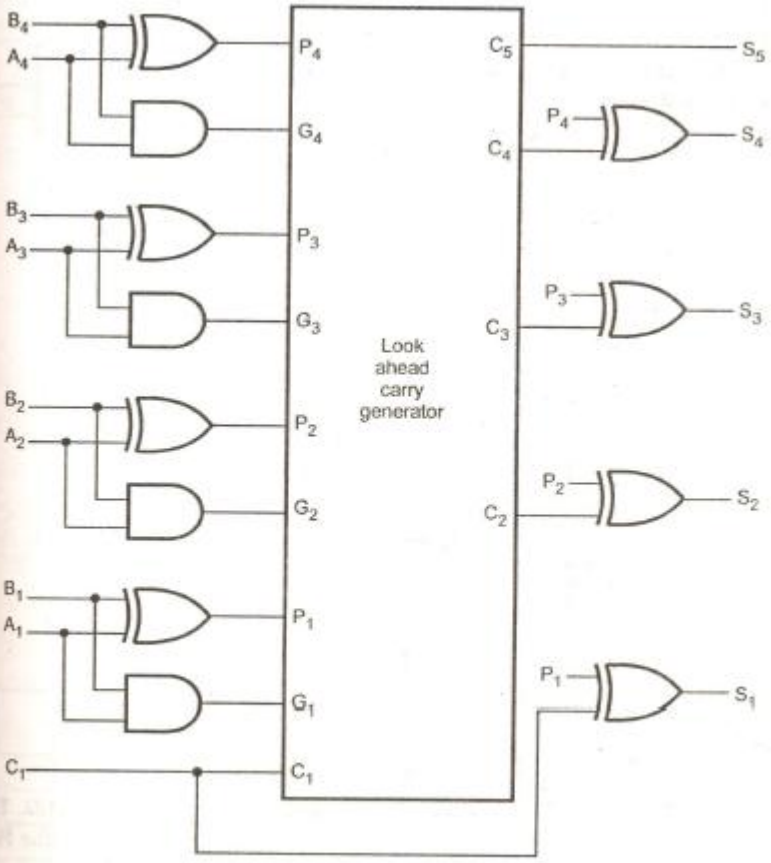


Fig. 5.24 4-bit parallel adder with look ahead carry generator

Q1 (b) Design a 4 bit BCD adder using IC 7483

Inputs				Output
S_3	S_2	S_1	S_0	Y
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	0
0	1	0	1	0
0	1	1	0	0
0	1	1	1	0
1	0	0	0	0
1	0	0	1	0
1	0	1	0	1
1	0	1	1	1
1	1	0	0	1
1	1	0	1	1
1	1	1	0	1
1	1	1	1	1

Table 5.5

S_3S_2 \ S_1S_0	00	01	11	10
00	0	0	0	0
01	0	0	0	0
11	1	1	1	1
10	0	0	1	1

$$Y = S_3S_2 + S_3S_1$$

Y = 1 indicates sum is greater than 9. We can put one more term, C_{out} in the above expression to check whether carry is one. If any one condition is satisfied we add 6 (0110) in the sum.

With this design information we can draw the block diagram of BCD adder, as shown in the Fig. 5.32.

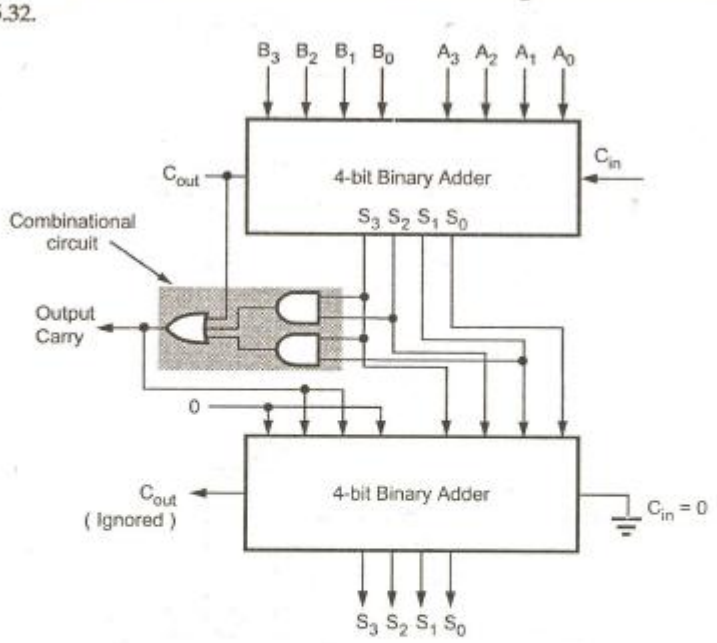


Fig. 5.32 Block diagram of BCD adder

As shown in the Fig. 5.32, the two BCD numbers, together with input carry, are first added in the top 4-bit binary adder to produce a binary sum. When the output carry is equal to zero (i.e. when sum ≤ 9 and C_{out} = 0) nothing (zero) is added to the binary sum. When it is equal to one (i.e. when sum > 9 or C_{out} = 1), binary 0110 is added to the binary sum through the bottom 4-bit binary adder. The output carry generated from the bottom binary adder can be ignored, since it supplies information already available at the output-carry terminal.

Q1 (c) Implement 12 bit comparator using IC 7485.

The design shown here 8 bit, for 12 bit design another IC 7485 is cascaded in a similar manner.

Solution : Fig. 4.69 shows an 8-bit comparator using two 7485 ICs.

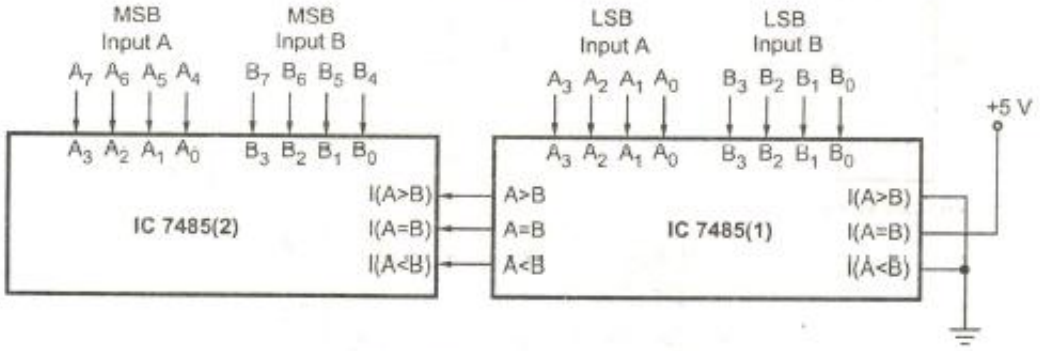
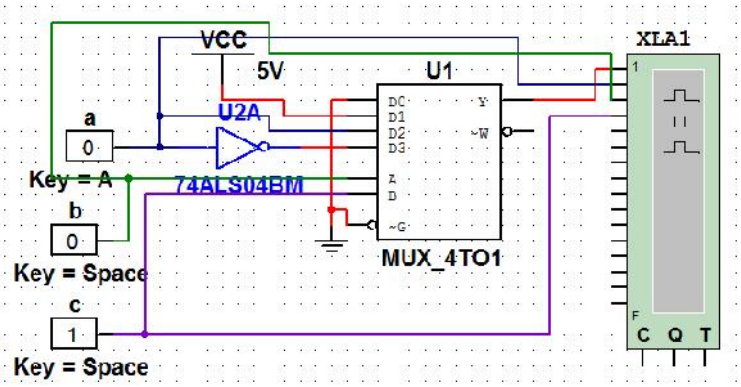


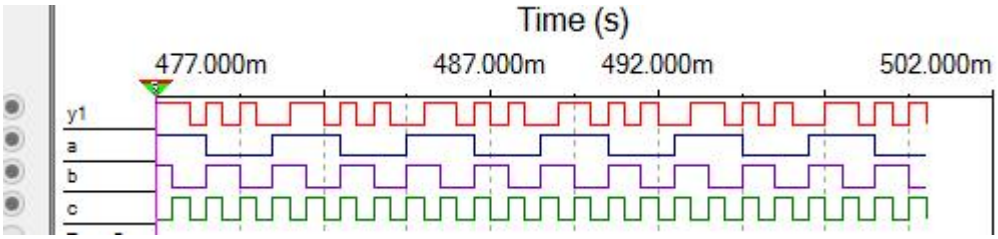
Fig. 4.69 8-bit comparator

Q2 a (i)) $f(a,b,c) = \sum m(1,3,5,6)$ using 4:1 MUX

Using VEM Technique

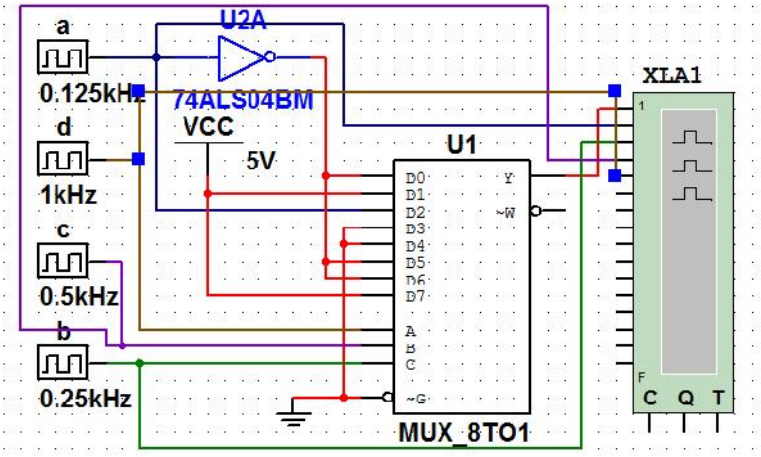
	0	①	2	③
	4	⑤	⑥	7
	0	1		

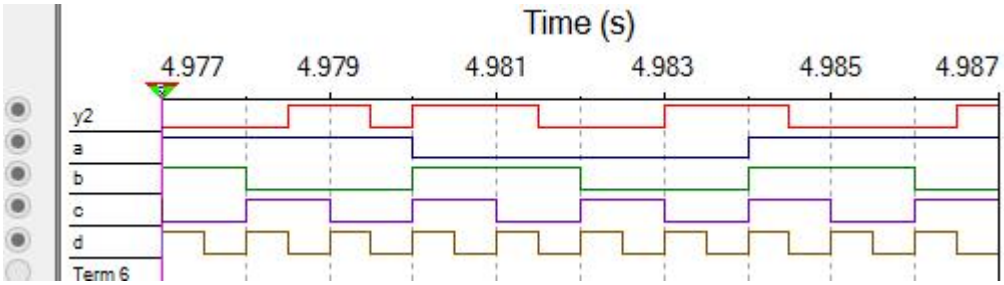




Q2 a (ii) $f(a,b,c,d) = \sum m(0,1,5,6,7,9,10,15)$ using 8:1 MUX.

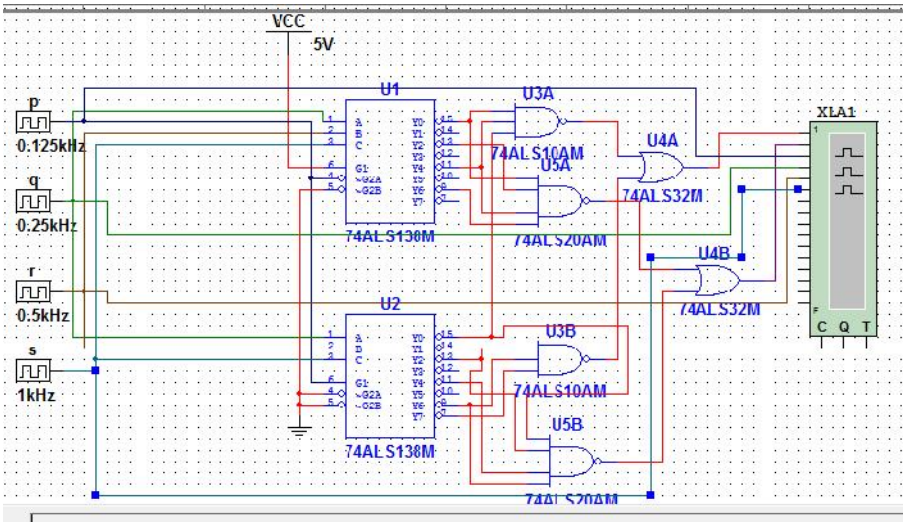
	⓪	①	2	3	4	⑤	⑥	⑦
	8	⑨	⑩	11	12	13	14	⑮
		1		0	0			1

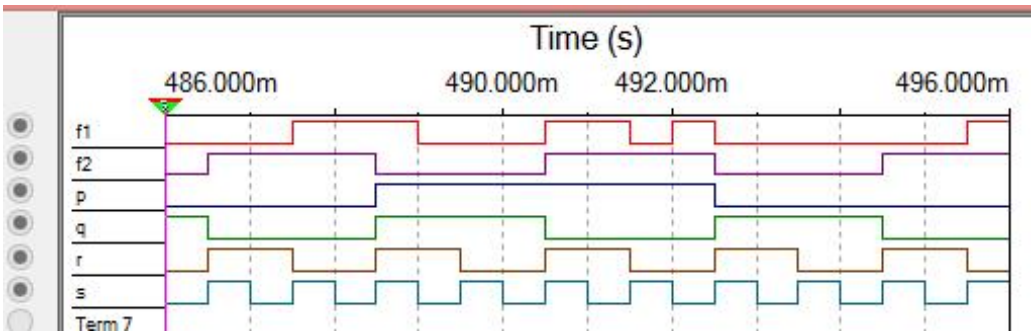




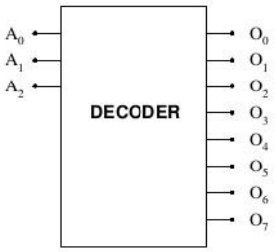
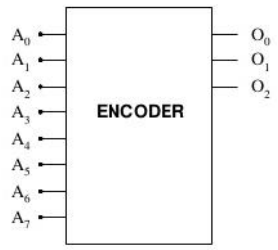
Q2 (b) Implement the following functions using two IC 74138

$$f_1(p,q,r,s) = \sum m(0,4,8,10,14,15) \;\&\; f_2(p,q,r,s) = \pi M(1,3,5,7,9,11,13)$$

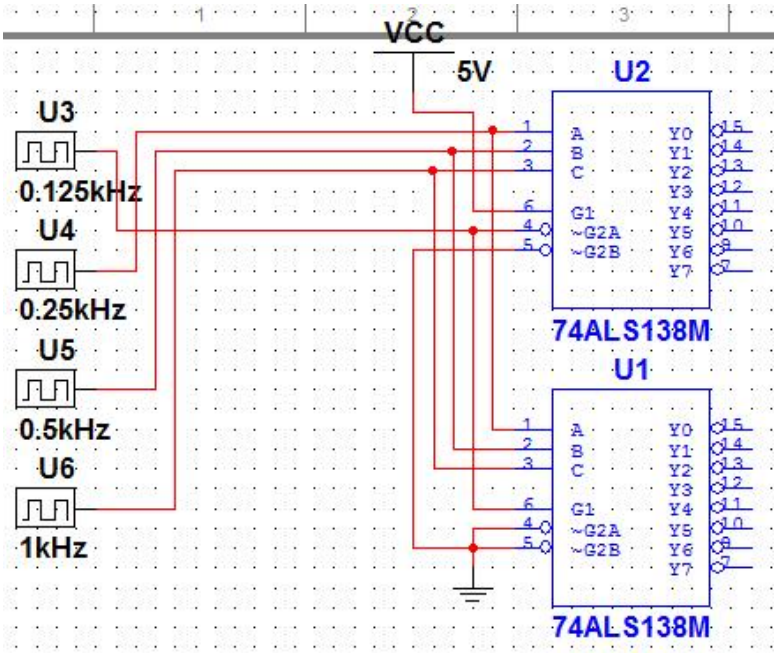




Q2 c i)

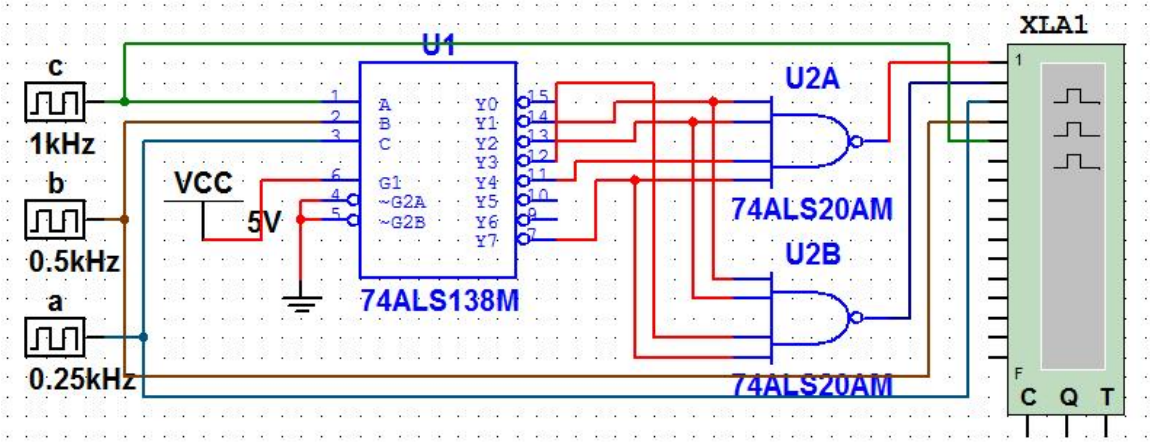
Decoders	Encoder
Decoders have multiple input and output lines. If n is number of input then output ≤ 2^n. Therefore number of input < number of output.	Encoders have multiple input and output lines where number of input > number of output.
Only one output is active for a given input.	Multiple outputs can be active for a given input.
	

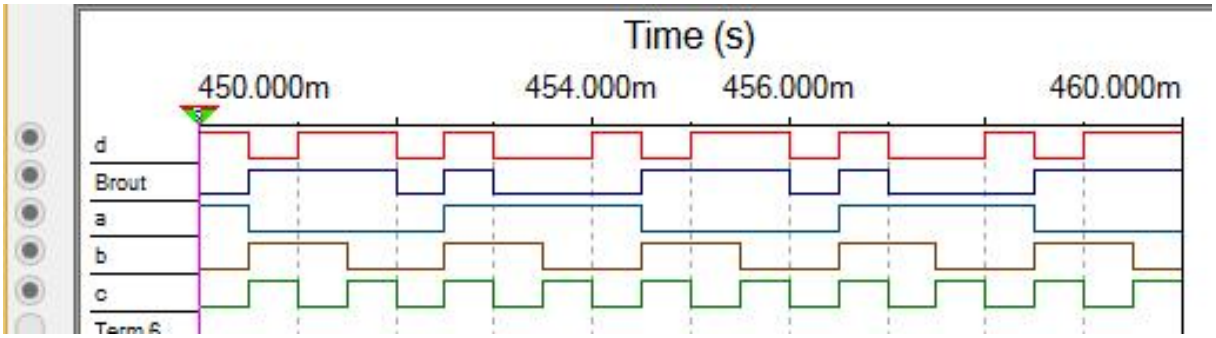
Q2 c ii)



Q2 d) Full Subtractor

Decimal Number	A	B	Br_{in}	D	Br_{out}
0	0	0	0	0	0
1	0	0	1	1	1
2	0	1	0	1	1
3	0	1	1	0	1
4	1	0	0	1	0
5	1	0	1	0	0
6	1	1	0	0	0
7	1	1	1	1	1





Q3. (a) Explain the construction of master slave JK flip flop and list its benefits.

6.4.2 The Master-Slave *JK* Flip-Flop

Since the output state of a master-slave *SR* flip-flop is undefined upon returning the control input to 0 when $S = R = 1$, it is necessary to avoid this condition. The *master-slave JK flip-flop*, on the other hand, does allow its two information input lines to be simultaneously 1. This results in the toggling of the output of the flip-flop. That is, if the present state is 0, then the next state is 1; while if the present state is 1, then the next state is 0. The logic diagram of a master-slave *JK* flip-flop is shown in Fig. 6.14*a*. The *J* and *K* inputs have the effect of setting and resetting the flip-flop, respectively, and hence are analogous to the *S* and *R* inputs of the master-slave *SR* flip-flop. In addition, two and-gates are used to sense and steer the state of the slave.

To see how this flip-flop works, assume the master-slave *JK* flip-flop of Fig. 6.14*a* is in its 1-state, the control signal, i.e., the clock, is 0, and that $J = K = 1$. Thus, the master and slave latches are both in the 1-state with $Q = Q_S = 1$ and $\overline{Q} = \overline{Q}_S = 0$. As a result of the feedback lines, the output of the *J*-input and-gate is logic-0 and the output of the *K*-input and-gate is logic-1. The net effect is that $S = 0$ and $R = 1$ at the inputs to the master latch, although these inputs cannot affect the state of the master at this time since $C = 0$. If the clock is now changed from 0 to 1, then the master resets; while the slave, being disabled, remains in its 1-state. However, upon the clock returning to 0, the content of the master is transferred to the slave, causing the new state of the master-slave *JK* flip-flop to become the 0-state. Thus, the output of the master-slave *JK* flip-flop toggled when $J = K = 1$ as the result of the control signal.

Now assume the master-slave *JK* flip-flop is in its 0-state, again $J = K = 1$, and the control signal, i.e., the clock, is low. Thus, $Q = Q_S = 0$ and $\overline{Q} = \overline{Q}_S = 1$. In this case the output of the *J*-input and-gate is logic-1; while the output of the *K*-input and-gate is logic-0. At the master input terminals, $S = 1$ and $R = 0$. Hence, when the clock is changed from 0 to 1, the master enters its 1-state, which is subsequently transferred to the slave when the clock changes from 1 to 0. Again, the state of the master-slave *JK* flip-flop toggled. The toggling behavior of the flip-flop when $J = K = 1$ is indicated by the fourth row in the function table shown in Fig. 6.14*b*.

Consider now the third row of the function table that indicates that a 1 on just the *J* input line has the effect of setting the flip-flop. To see this, assume the master-slave *JK* flip-flop is in its 1-state when the clock is low. Thus, $Q = Q_S = 1$ and $\overline{Q} = \overline{Q}_S = 0$. Since the slave is enabled and in its 1-state, the master must also be in its 1-state, i.e., $Q_M = 1$ and $\overline{Q}_M = 0$. If $J = 1$ and $K = 0$, then the outputs of both and-gates are logic-0 since they each have a 0 on one of their inputs, i.e., the upper and-gate has $\overline{Q}_S = 0$ and the lower and-gate has $K = 0$. Consequently, at the input terminals of the master, $S = R = 0$. When the clock becomes 1, the state of the master does not change, i.e., it remains in its 1-state. Upon returning the clock to 0, the slave, which in turn takes on the value of the master, also remains in its 1-state.

On the other hand, if the master and slave latches are in their 0-states when $J = 1$, $K = 0$, and the clock is low, then $Q = Q_M = Q_S = 0$ and $\overline{Q} = \overline{Q}_M = \overline{Q}_S = 1$.

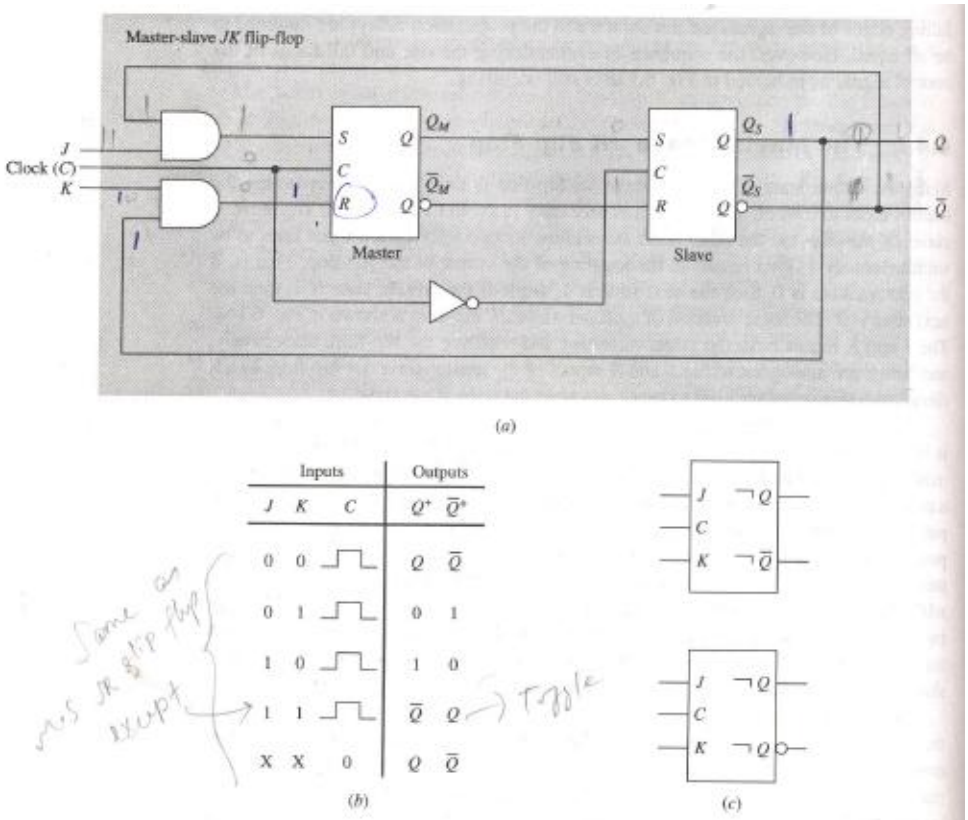


Figure 6.14 Master-slave *JK* flip-flop. (*a*) Logic diagram using gated *SR* latches. (*b*) Function table where Q^+ denotes the output Q in response to the inputs. (*c*) Two logic symbols.

The output of the *J*-input and-gate is logic-1 and the output of the *K*-input and-gate is logic-0. Thus, $S = 1$ and $R = 0$ at the inputs to the master latch. When the clock goes high, the master is set. The 1-state of the master is then subsequently transferred to the slave when the clock returns to 0. In summary, regardless of its present state when $J = 1$ and $K = 0$, the master-slave *JK* flip-flop enters or remains in its 1-state upon the occurrence of the pulse signal on the control line. This corresponds to the third row of the function table.

By a similar argument, if $J = 0$ and $K = 1$, then the master-slave *JK* flip-flop enters or remains in its 0-state after a clock pulse has occurred. This resetting effect is described by the second row of the function table.

Considering the remaining rows of the function table, the first row indicates that the master-slave *JK* flip-flop retains its current state when $J = K = 0$ during a clock pulse. Similarly, the last row indicates that whenever the clock is low, i.e., $C = 0$, the state of the flip-flop does not change.

In Fig. 6.14*c* two symbols are shown for the master-slave *JK* flip-flop. Again, the postponed-output indicator is used to symbolize that the output change occurs coincident with the falling edge of the control signal, i.e., when the control signal changes from 1 to 0.

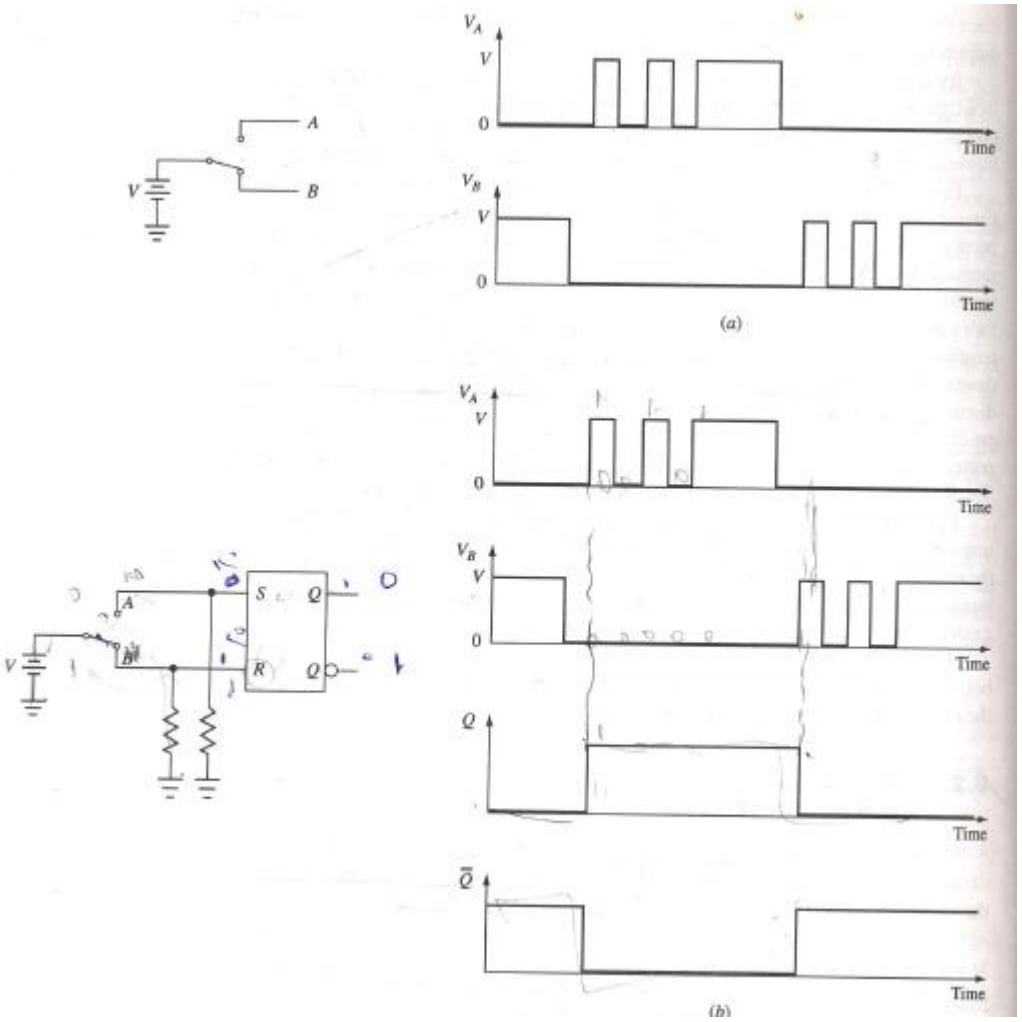
For ease of the above analysis, the logic-1 values on the *J* and *K* lines were assumed to be applied prior to the application of the clock pulse. In actuality, these values can occur anytime while the control signal is 1 since the master, being a latch, is enabled during that time.

A timing diagram illustrating the behavior of a master-slave *JK* flip-flop is shown in Fig. 6.15. Again, for simplicity, propagation delays are assumed to be equal and the finite slopes of the rising and falling edges of the signals are not shown. In addition, manufacturer's constraints regarding minimum width of the signals, i.e., minimum time durations that signals are applied, and setup and hold times of the information signals relative to the control signal must be adhered to for proper operation of master-slave flip-flops. It is assumed these constraints are satisfied in the timing diagram of Fig. 6.15.

Q3 (b) Explain an application of SR latch with the required diagrams

6.2.2 An Application of the *SR* Latch: A Switch Debouncer

A common problem involving switches is the occurrence of *contact bounce*. This is illustrated in Fig. 6.3*a*. As indicated by the waveforms, with the center contact of the switch in its lower position, the voltage at terminal *B* is $+V$ volts, while the voltage at terminal *A* is zero. Now if the center contact is moved from its lower position to its upper position, then it is noted that the voltage at terminal *B* first becomes zero, followed by the voltage at terminal *A* becoming $+V$ volts when the center contact reaches the upper terminal. However, as a result of contact bounce, the center contact of the switch leaves terminal *A*, causing the output voltage at that terminal to return to zero, and then upon returning to terminal *A*, causing the voltage at terminal *A* to become $+V$ volts again. This opening and closing effect, due to the springiness of the contacts, may occur several times before the center contact of the switch remains in its upper position. It is important to note that during contact bounce, the center contact does not return all the way to terminal *B*. Similarly, as indicated by the waveforms of Fig. 6.3*a*, contact bounce again occurs when the switch is moved from its upper position to its lower position. The effect of contact

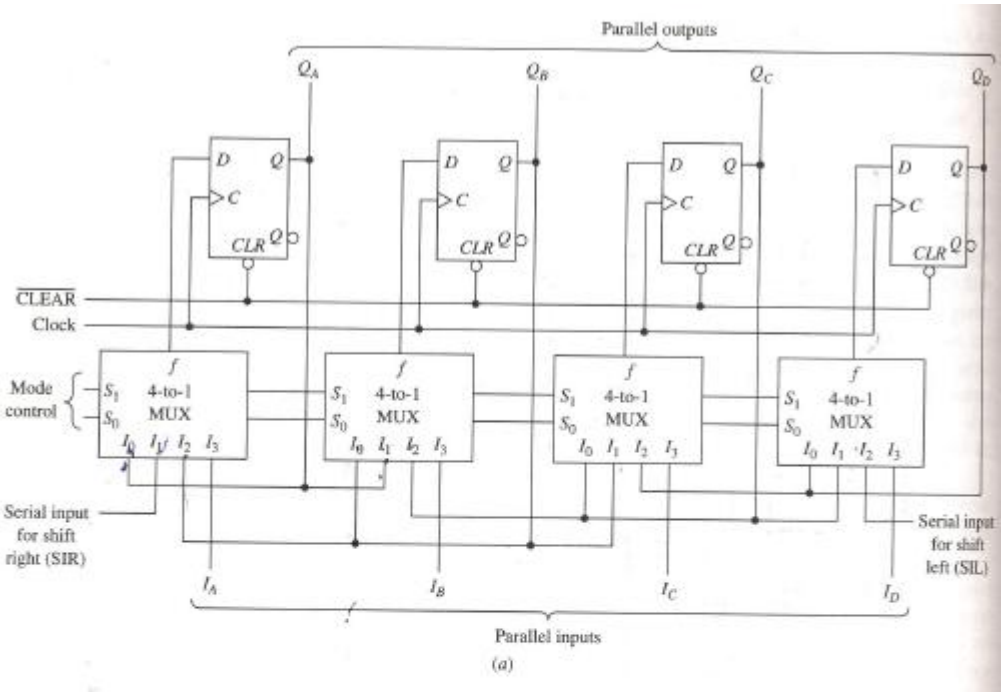


bounce is normally undesirable. For example, in the case of push-button keys on a keyboard, contact bounce may cause a system to respond as though a key was depressed several times in succession.

A very simple, but important, application of the *SR* latch is to eliminate the effect of contact bounce. A *switch debouncer* circuit and corresponding waveforms

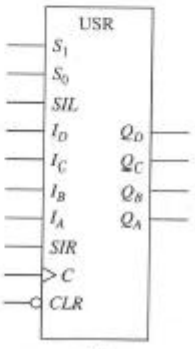
are shown in Fig. 6.3b. Assume positive logic so that $+V$ volts corresponds to logic-1 and ground to logic-0. By use of the two pull-down resistors, logic-0 values are ensured at the S and R terminals of the SR latch whenever the center contact of the switch is not connected to either terminals A or B, i.e., whenever the switch is open. Thus, when the center contact moves from its lower position to its upper position, the SR latch remains in its reset state until the center contact reaches terminal A, at which time the Q output of the SR latch becomes 1. If the switch now opens, as a result of contact bounce, then the 0 input to the S and R terminals of the latch causes the Q and $\overline{Q}$ outputs to remain unchanged. Hence, by use of the SR latch, the effect of contact bounce is eliminated. In a similar manner, the effect of contact bounce is also eliminated when the switch moves from its upper position to its lower position.

Q4 (a) Explain Universal shift register.



Select lines S_1	S_0	Register operation
0	0	Hold
0	1	Shift right
1	0	Shift left
1	1	Parallel load

(b)

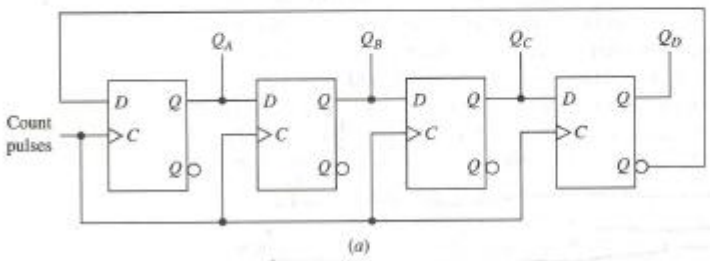


(c)

Figure 6.29 Universal shift register. (*a*) Logic diagram. (*b*) Mode control. (*c*) Symbol.

Q4 (b) Draw the circuit diagram and timing diagram for a 4 bit johnson counter with thw starting value of 1000.

A variation of the ring counter is the *switch-tail counter*, also known as the *twisted-ring counter* or *Johnson counter*. This counter is illustrated in Fig. 6.38*a* and its counting sequence is given in Fig. 6.38*b* assuming the counter starts in the $Q_AQ_BQ_CQ_D = 0000$ state. In this counter, the complement of the rightmost flip-flop serves as the input to the leftmost flip-flop in the shift-right register configuration.



Q_A	Q_B	Q_C	Q_D	And-gate inputs
0	0	0	0	$\bar{Q}_A\bar{Q}_D$
1	0	0	0	Q_AQ_B
1	1	0	0	$Q_B\bar{Q}_C$
1	1	1	0	$Q_C\bar{Q}_D$
1	1	1	1	Q_AQ_D
0	1	1	1	$\bar{Q}_AQ_B$
0	0	1	1	$\bar{Q}_BQ_C$
0	0	0	1	$\bar{Q}_CQ_D$
0	0	0	0	

etc.

(b)

Figure 6.38 Mod-8 twisted-ring counter. (*a*) Logic diagram. (*b*) Counting sequence.

Timing diagram need to write for the johnson counter design

Q4 (c) Design asynchronous mod 8 binary ripple counter. Show its circuit and timing diagram.

Ans: Design shown here is mod 16 (4-bit) binary ripple counter. Mod 8 can be designed in a similar way.

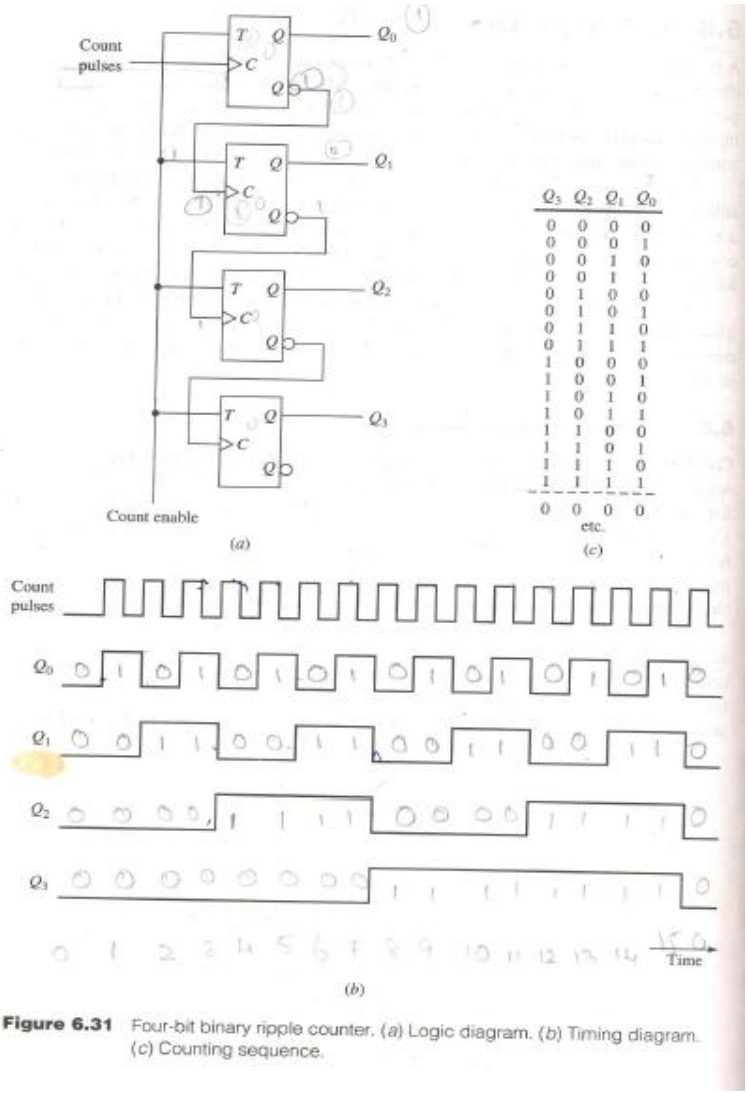


Figure 6.31 Four-bit binary ripple counter. (*a*) Logic diagram. (*b*) Timing diagram. (*c*) Counting sequence.

Q5 a) Mod 6 synchronous counter 0 → 2 → 3 → 6 → 5 → 1 → 0

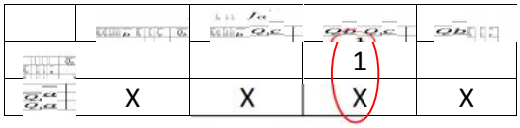
Assume that invalid state will reset counter to 0

Excitation Table for J K Flip Flop

Q	**Q^+**	**J**	**K**
0	0	0	X
0	1	1	X
1	0	X	1
1	1	X	0

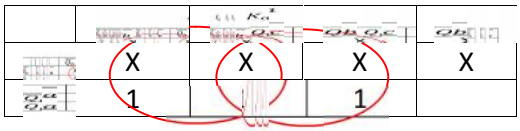
Present state				Next State				Excitation Table					
Decimal Value	Q_a	Q_b	Q_c	Decimal Value	Q_a	Q_b	Q_c	J_a	K_a	J_b	K_b	J_c	K_c
0	0	0	0	2	0	1	0	0	X	1	X	0	X
1	0	0	1	0	0	0	0	0	X	0	X	X	1
2	0	1	0	3	0	1	1	0	X	X	0	1	X
3	0	1	1	6	1	1	0	1	X	X	0	X	1
4	1	0	0	0	0	0	0	X	1	0	X	0	X
5	1	0	1	1	0	0	1	X	1	0	X	X	0
6	1	1	0	5	1	0	1	X	0	X	1	0	X
7	1	1	1	0	0	0	0	X	1	X	1	X	1

J_a



$$J_a = Q_b\,Q_c$$

K_a

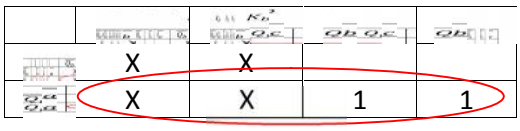


$$K_a = \overline{Q_b} + Q_c$$

J_b

$$J_b = \overline{Q_a}\,\overline{Q_c}$$

K_b



$$K_b = Q_a$$

J_c

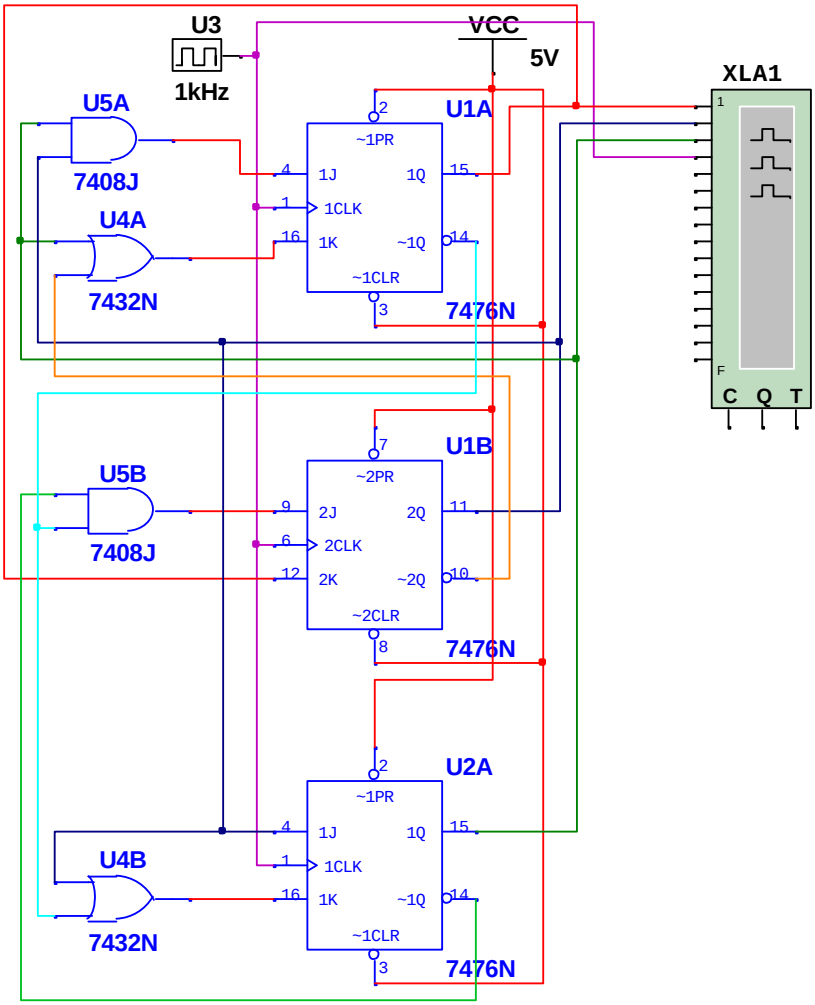
		X	X	1
		X	X	1

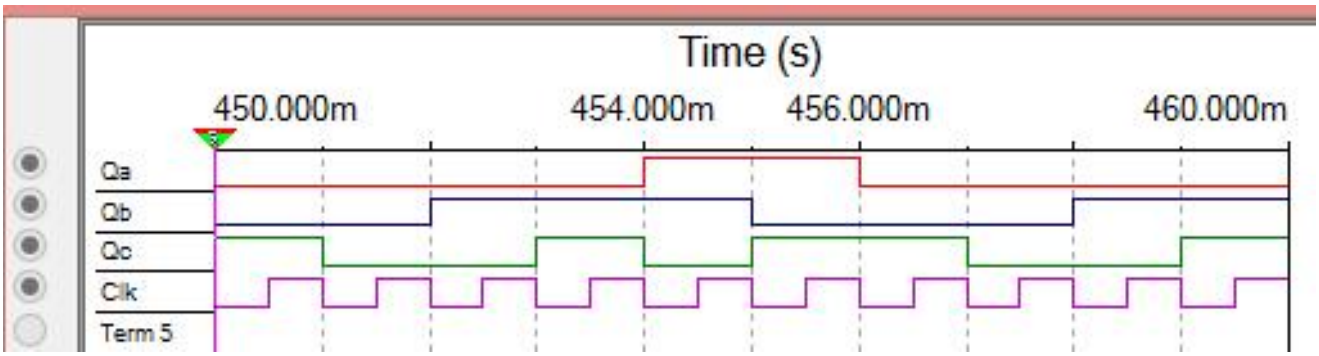
$$J_c = Q_b\, Q_c$$

K_c

	X	1	1	X
	X		1	X

$$K_c = \overline{\quad}_a + Q_b$$





Q5 b) Mod 8 synchronous up/ down counter

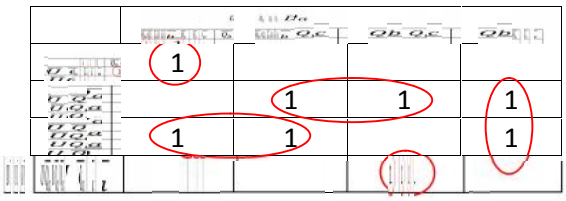
Excitation Table for D Flip Flop

Q	Q^+	D
0	0	0

0	1	1
1	0	0
1	1	1

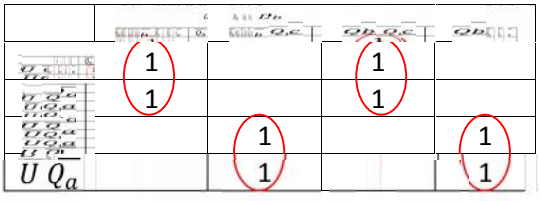
Present state					Next State				Excitation Table		
Decimal Value		Q_a	Q_b	Q_c	Decimal Value	Q_a	Q_b	Q_c	D_a	D_b	D_c
0	0	0	0	0	7	1	1	1	1	1	1
1	0	0	0	1	0	0	0	0	0	0	0
2	0	0	1	0	1	0	0	1	0	0	1
3	0	0	1	1	2	0	1	0	0	1	0
4	0	1	0	0	3	0	1	1	0	1	1
5	0	1	0	1	4	1	0	0	1	0	0
6	0	1	1	0	5	1	0	1	1	0	1
7	0	1	1	1	6	1	1	0	1	1	0
8	1	0	0	0	1	0	0	1	0	0	1
9	1	0	0	1	2	0	1	0	0	1	0
10	1	0	1	0	3	0	1	1	0	1	1
11	1	0	1	1	4	1	0	0	1	0	0
12	1	1	0	0	5	1	0	1	1	0	1
13	1	1	0	1	6	1	1	0	1	1	0
14	1	1	1	0	7	1	1	1	1	1	1
15	1	1	1	1	0	0	0	0	0	0	0

$$D_a$$



$$D_a = \bar{U}\,\overline{Q_a}\,\overline{Q_b}\,\overline{Q_c} + U\,\overline{Q_a}\,Q_b\,Q_c + UQ_a\,\overline{Q_b} + \bar{U}Q_aQ_c + Q_aQ_b\,\overline{Q_c}$$

$$D_b$$



$$D_b = \bar{U}\,\overline{Q_a}\,Q_b + \bar{U}Q_bQ_c + U\,\overline{Q_b}\,Q_c + UQ_b\,\overline{Q_c}$$

$$D_b = \bar{U}\,\overline{Q_b}\,\overline{Q_c} + Q_bQ_c) + U(\overline{Q_b}Q_c + Q_b\,\overline{Q_c})$$

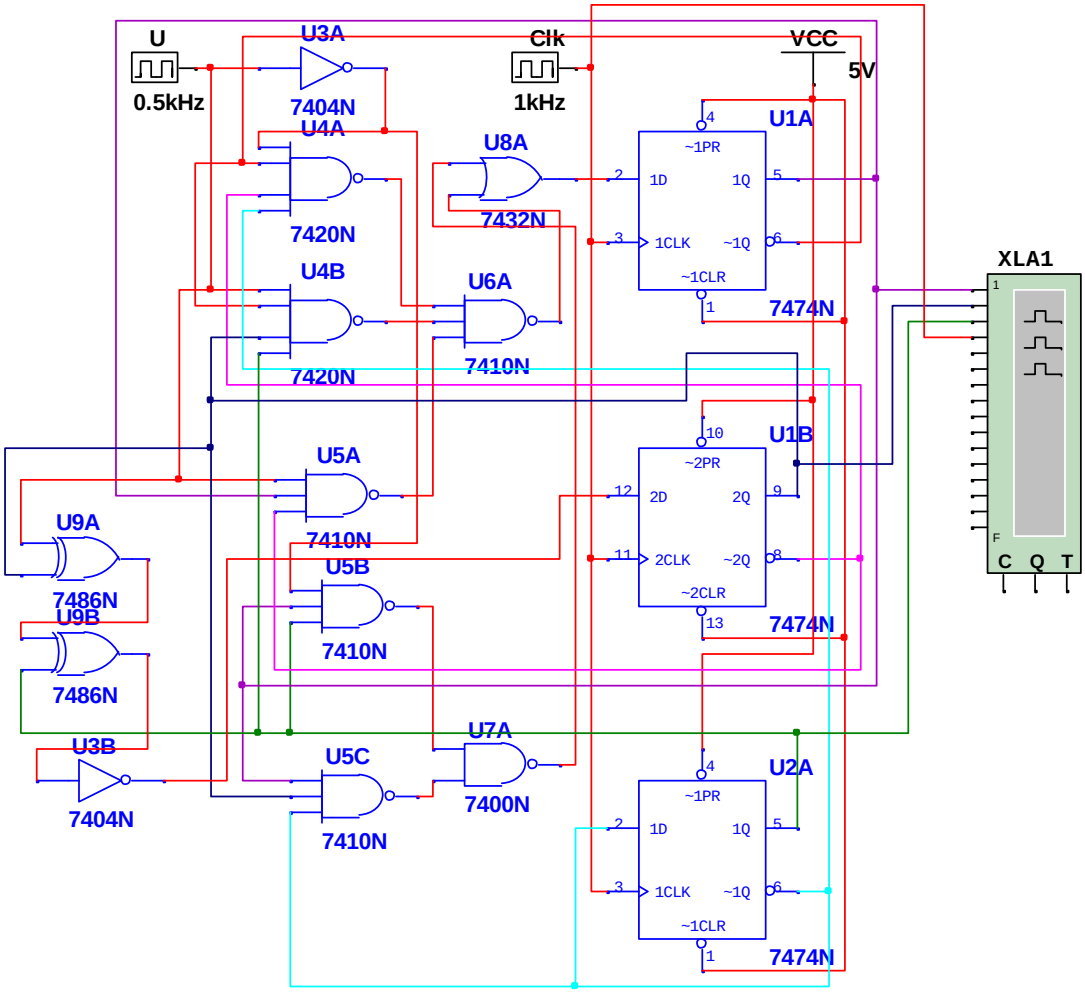
$$D_b = \overline{U}\ \overline{(Q_b \quad \oplus Q_c)}$$

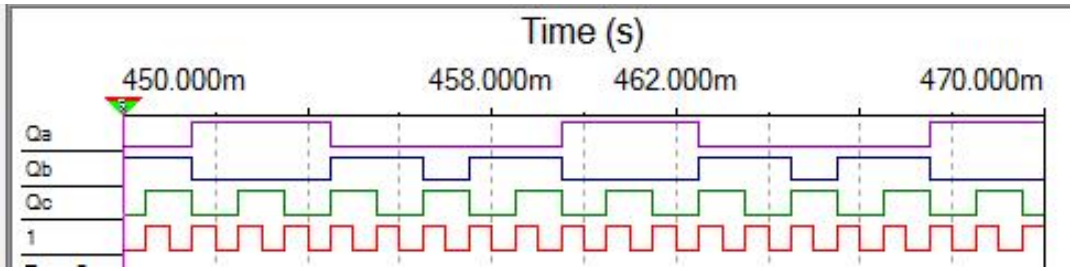
$$D_b = \overline{\ni (Q_b \in \quad)}$$

D_c

	1			1
	1			1
	1			1
	1			1

$$D_a = \overline{\quad c}$$




Time (s)
450.000m
458.000m
462.000m
470.000m
Qa
Qb
Qc
1