7. Counters & Registers

7.1 Introduction to Counters

- A counter is an **array of flipflops that advances from state to state** in response to an event. This event is usually a single cycle of a clock waveform.

- Each flipflop represents a single bit. Thus a counter with two flipflops can have 4 (i.e. 2^2) possible outputs or states. With three flipflops, it can have 8 (i.e. 2^3) possible states and so on.

- Counters have numerous applications in digital systems:

  - They could be used to build the circuit for a digital watch or a digital alarm clock.

  - They could be used as a driver for a set of traffic lights, which is based on a repetitive fixed sequence of changes (or states).

  - They could also be used as a program counter that steps through the addresses (or instructions) of a software program.

- The number of states through which a counter cycles before returning to a starting state (not necessarily zero) is called the **Modulo** (or **MOD**) of the counter.

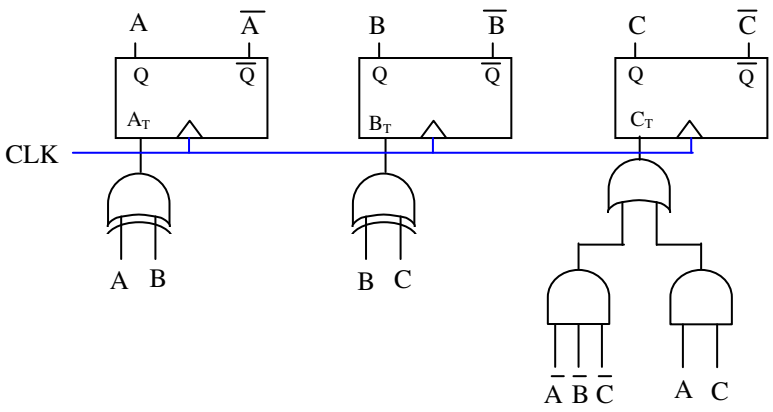
- In the next section of the notes we are going to **analyse a counter** to see how it operates, i.e. we will determine the count sequence for a given design.

- Then we will examine how to **design a counter** given the count sequence.

7.2 Analysing a Counter

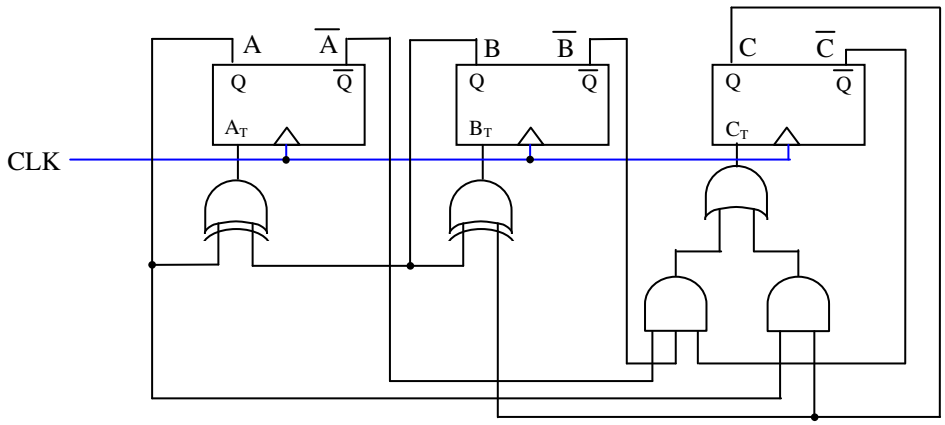
- The best way to illustrate the analysis of a counter is using an example.

- ***Ex. 7.1 Determine the operation of the following circuit:***



- Note that the parameters A, B and C are in fact the output states of the flipflops. This system has technically only one input, i.e. the clock signal.

- The current representation of the circuit is convenient for analytical purposes. In reality the circuit should be presented as follows:



- In order to fully analyse the circuit, we need to consider all combinations of states (or outputs) of the flipflops, i.e. from ABC = 000 to 111.

- From this we can determine the count sequence for the circuit. In order words, we can determine the sequence of states that the counter will transition through before repeating.

- So, taking the first state combination:

  ABC = 000,

  we can determine from the combinational circuitry (i.e. the gates leading to the inputs of the flipflops) that the inputs to the Toggle flipflops will be:

  A_T = 0, **B_T = 0** and **C_T = 1**

- Recall the operation of a Toggle flipflop – T = 0 implies no change while T = 1 complements the existing output.

- Hence, we can determine the next output for the circuit to be:

  A(τ) = 0 (A = 0, T = 0 $\Rightarrow$ no change)

  B(τ) = 0 (B = 0, T = 0 $\Rightarrow$ no change)

  C(τ) = 1 (C = 0, T = 1 $\Rightarrow$ complement existing output, i.e. $\overline{C}$ = 1)

- Thus the next output for the circuit is:

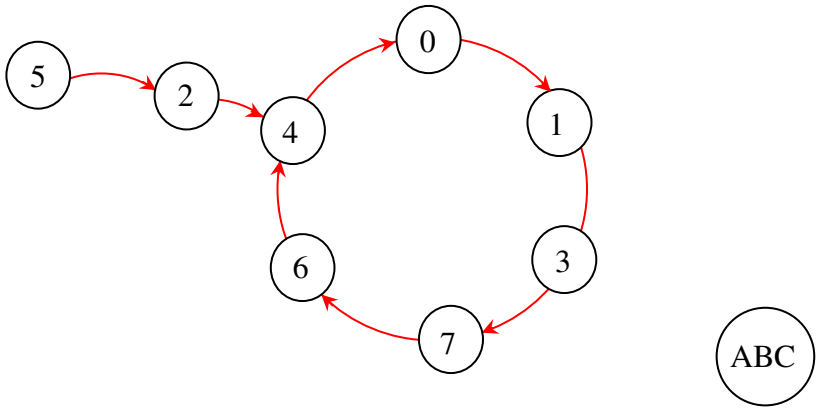
  A(τ)B(τ)C(τ) = 001

- We repeat this process for the remaining state combinations to obtain the following table:

Present State	Flipflop Inputs			**Next State**
	$A \oplus B$	$B \oplus C$	$\bar{A}\bar{B}\bar{C}+AC$	
A B C	A_T	B_T	C_T	A(τ) B(τ) C(τ)
0 0 0				
0 0 1				
0 1 0				
0 1 1				
1 0 0				
1 0 1				
1 1 0				
1 1 1				

- In order to visually illustrate the behaviour of the counter circuit, we use a state diagram to show the sequence of states.

- Using the above table, we can easily obtain the following state diagram:



- Note, in the above state diagram, the state is given by the output combination of ABC.

- The control variable in this case is the clock (CLK) and is omitted from the diagram as this is the standard control variable for counters.

- In other words, when CLK = 1 we will transition from one state to the next.

- In this case we can clearly see that the next state depends primarily on the present state, unlike combinational circuits.

7.3 Designing a Counter

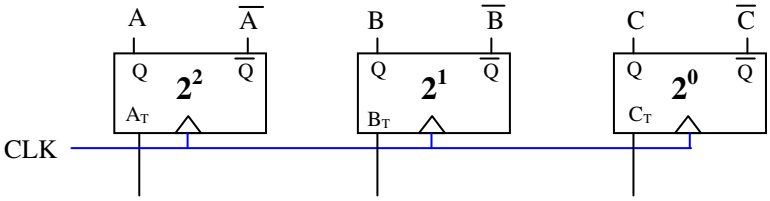
- Now, we are going to take the state sequence obtained in the previous example as our starting point and we are going to design a counter to achieve this sequence.

- ***Ex. 7.2 Design a MOD 6 counter with a repeating sequence 0 1 3 7 6 4, using T flipflops. Check for possible lockout.***

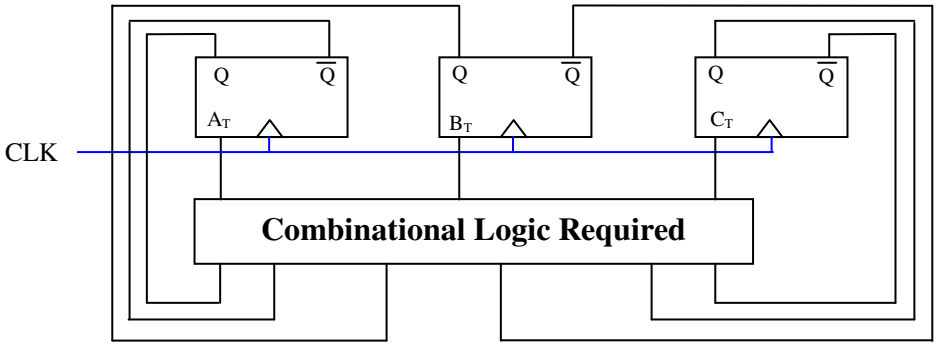
- **Step 1** – We need to determine the number of flipflops required.

- Since the sequence does not contain a number greater than 7 (i.e. 111 in binary), we only need to use **three** flipflops.

- We will label these A, B and C and, arbitrarily, choose A to represent the MSB. Hence:



- The design is reduced to determining the logic necessary to generate A_T, B_T and C_T from the states of A, B and C to produce the new values for A, B and C, i.e.:



- **Step 2** – We know the sequence of states required (given in question), so we now need to determine the sequence of inputs to each of the flipflops in order to produce the required output.

- Hence, for example, let's take the first two states which are 0 and 1 or in binary ABC = 000 and 001.

- So, taking each bit in turn:

A goes from 0 to 0 $\Rightarrow$ no change $\Rightarrow$ $\mathbf{A_T = 0}$
B goes from 0 to 0 $\Rightarrow$ no change $\Rightarrow$ $\mathbf{B_T = 0}$
C goes from 0 to 1 $\Rightarrow$ output needs to be toggled $\Rightarrow$ $\mathbf{C_T = 1}$

- Hence, the flipflop inputs required to transition from state 000 to state 001 is given by:

$$\mathbf{A_T\ B_T\ C_T = 0\ 0\ 1}$$

- Repeating this process for the remaining sequence of states (i.e. from, 1 to 3, 3 to 7, etc.), we obtain the following table:

State	A B C	Flipflop Inputs A_T B_T C_T
0	**0 0 0**	**0 0 1**
1	**0 0 1**	
3	0 1 1	
7	1 1 1	
6	1 1 0	
4	1 0 0	
0	*0 0 0*	

Operation	**T**
Stay at 0	**0**
Stay at 1	**0**
Go to 0	**1**
Go to 1	**1**

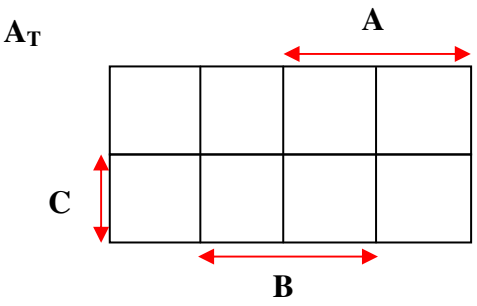
- The requirements table for the T flipflop is given above for the sake of convenience.

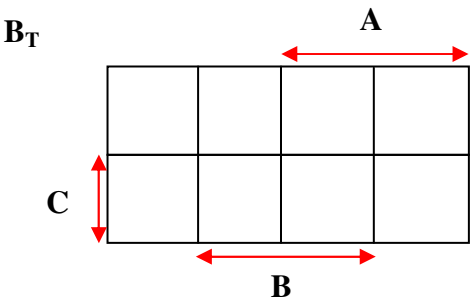
- Since the sequence repeats, we need to ensure that we transition from state 4 back to the first state, i.e. state 0.

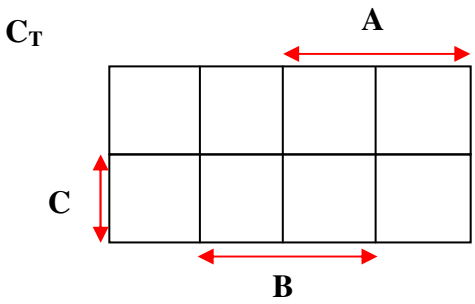
- **Step 3** – Once we know the sequence of inputs to each of the flipflops, we then need to generate a Karnaugh Map (KM) for each of the flipflop inputs and obtain a minimal combinational logic circuit for implementing this sequence.

- A quick glance at the above table shows that: $A_T = \sum(3,4)$, $B_T = \sum(1,6)$, $C_T = \sum(0,7)$

- Generate a KM for each of A_T, B_T, and C_T. We will treat the unused states ***2*** and ***5*** as don't care terms for now, as they are not part of the required sequence.







- In engineering design, it is always good practice to self check a design to make sure that the design process has been carried out correctly.
- In this case, it provides a check to ensure that the minimisation was done correctly.
- We self check the design by effectively analysing our circuit (as in section 7.1 of the notes) to see if it produces the required sequence.
- Consider the first output of the sequence ABC = 000. Passing this through our circuit design gives:

$$A_T = A \oplus B = 0 \oplus 0 = 0$$
$$B_T = B \oplus C = 0 \oplus 0 = 0$$
$$C_T = \overline{A \oplus C} = \overline{0 \oplus 0} = 1$$

i.e. $\mathbf{A_T\ B_T\ C_T = 0\ 0\ 1}$

- Hence, if the current state is ABC = 000 and the flipflop inputs are $A_T\ B_T\ C_T = 001$ then the next state will be:

$$\mathbf{A(\tau)\ B(\tau)\ C(\tau) = 0\ 0\ 1}$$

- We now repeat the process by passing this state through our circuit, by generating the new flipflop inputs and by determining the next state.
- Continuing in this manner we obtain the following self check table:

State	A B C	A_T $A \oplus B$	B_T $B \oplus C$	C_T $\overline{A \oplus C}$
0	**0 0 0**	**0**	**0**	**1**
1	**0 0 1**			
3				
7				
6				
4				
0				

- We also need to check the unused states. Although they are not part of the main cycle, they may, nevertheless, cause problems with our counter design in certain circumstances.

- The unused states are 2 and 5. Passing these through our circuit design gives:

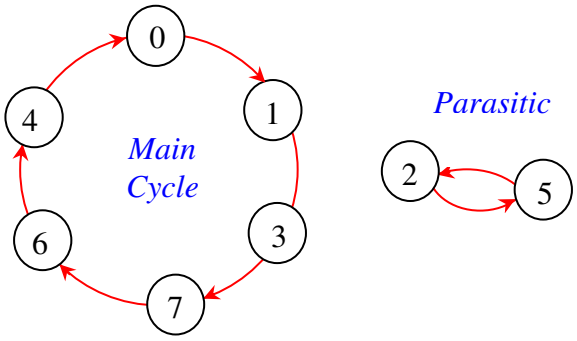
State	A B C	A_T $A \oplus B$	B_T $B \oplus C$	C_T $\overline{A \oplus C}$
2	0 1 0			

- This self check reveals that if the counter should enter one of the unused states 2 and 5, it will oscillate between these two states on successive clock cycles. It will be locked out of the *main* cycle. Hence **lockout** occurs.

- In this condition, the counter is said to be in a **parasitic cycle**.

- The counter can go into unused states by a pulse of noise (external spurious electrical intrusion) or at power up.

- Thus, the state diagram for our current counter design is as follows:



- We can correct this problem in one of two ways.

- The first way is to use *appropriate hardware* – i.e. flipflops that have asynchronous inputs such as preset and clear.

- These inputs can be used to initialise the counter to the first state of ABC = 000 before normal operation begins.

- Our second option is to solve the problem by *redesigning our circuit* to prevent lockout occurring.

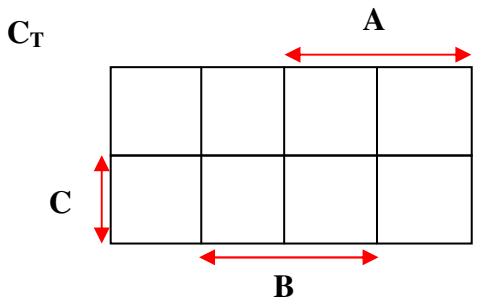
- One way to prevent lockout is to make a simple change to the existing design.

- Thus, for example we can make state 2 return to state 4 in the main cycle by forcing $C_T = 0$ for this state, as follows:

State	A B C	A_T	B_T	C_T
2	0 1 0	1	1	**0**
4	1 0 0			

- We haven't modified the sequence of inputs to the Toggle flipflops representing A and B, so the input circuitry for these remain the same.

- However , we have modified the input sequence for the Toggle flipflop representing C and so we need to revisit its Karnaugh Map, as follows:

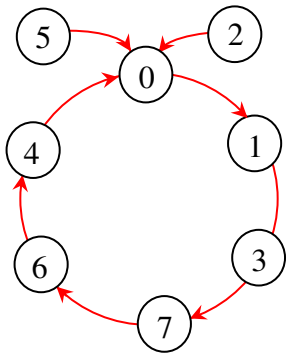


- The design and state diagram of Ex. 7.1 is, thus, realised.

An alternative redesign …

- Alternatively, the counter could be redesigned from the outset so that lockout can never occur by ensuring that all unused states return to a pre-defined state, such as ABC = 000.

- This would give the following state diagram:



- This design yields a synchronous reset to 000 on receipt of the first clock pulse.

- We could implement the full design from scratch by repeating what we have done already and including the transition of states 2 to 0 and 5 to 0.

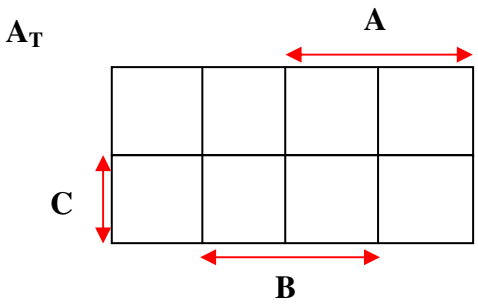
- However, as we have already carried out the design of the main sequence, we only need to consider, in this case, the transition of states 2 to 0 and 5 to 0.

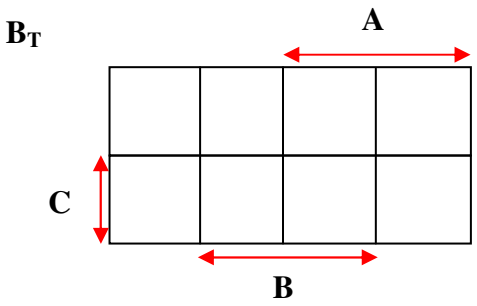
- Hence, repeating step 2 for the unused sates, we need to determine the inputs that allows the transition of states 2 to 0 and 5 to 0, as follows:

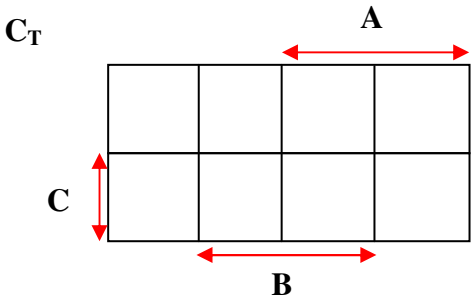
State	A B C	A_T	B_T	C_T
2 0	0 1 0 0 0 0	0	1	0
5 0	1 0 1 0 0 0	1	0	1

- Next, we redo step 3 and revisit the KMs for all three flipflops, as shown below.

- We no longer have don't care terms for the unused states as we do actually care what happens these states. Thus all the don't care terms in the previous KMs have now been assigned either a 1 or a 0, in accordance with the above table.







- In relation to both redesigns, it's worth noting the following:
- The first redesign is based on the principle of designing the counter circuit to satisfy the main cycle and only then checking to see if there's a problem with lockout.
- If there is, this problem can be resolved with the minimum of changes.
- Thus, this approach ensures *an overall minimal design* in terms of the combinational logic required.
- The second approach of ensuring that all unused states return to a prescribed state is a decision that is taken at the outset of the design.
- This approach ensures that *lockout cannot occur*, i.e. the problem should never arise. It also ensures that all *unused states return to the main cycle on one clock pulse*.
- However, these added benefits usually come at the expense of additional gates and, hence, *a non-minimal solution* (in comparison with the first design approach).

- ***Ex. 7.3 Design a BCD 2421 counter which has the sequence 0 1 2 3 4 5 6 7 8 9. Use JK flipflops and minimal NAND gating.***
- Note, here we are given a BCD 2421 weighted counter, as opposed to the conventional BCD 8421 standard weighting.
- When working with Karnaugh Maps and minterms, we use the latter weighting.
- So the first thing we need to do here is to obtain the minterm (or 8421) equivalent of the 2421 weighting. This is shown in the following table:

Decimal Value	BCD 2421	Minterm Value (8421)
0	0000	0
1	0001	1
2	0010	2
3	0011	3
4	0100	4
5	1011	11
6	1100	12
7	1101	13
8	1110	14
9	1111	15

- So the sequence we are actually generating is in fact 0, 1, 2, 3, 4, 11, 12, 13, 14 and 15.

- The unused minterms (or states) are therefore m_5, m_6, m_7, m_8, m_9, m_{10}.

- Recall the requirements table for JK flipflops:

Operation	J	K
Stay at 0	**0**	**X**
Stay at 1	**X**	**0**
Go to 0	**X**	**1**
Go to 1	**1**	**X**

- **Step 1** – We need to determine the number of flipflops required.

- Since the sequence does not contain a number greater than 15 (i.e. 1111 in binary), we need to use **four** flipflops.

- We will label these A, B, C and D and, arbitrarily, choose A to represent the MSB.

- **Step 2** – Now, we need to determine the sequence of inputs to the JK flipflops in order to produce the required output.

- Hence, we obtain the following table:

Minterm (8421)	Count (2421)	ABCD	Flipflop Inputs A_J A_K	B_J B_K	C_J C_K	D_J D_K
0	0	**0000**	**0 X**	**0 X**	**0 X**	**1 X**
1	1	**0001**	0 X	0 X	1 X	X 1
2	2	0010	0 X	0 X	X 0	1 X
3	3	0011	0 X	1 X	X 1	X 1
4	4	0100	1 X	X 1	1 X	1 X
11	5	1011	X 0	1 X	X 1	X 1
12	6	1100	X 0	X 0	0 X	1 X
13	7	1101	X 0	X 0	1 X	X 1
14	8	1110	X 0	X 0	X 0	1 X
15	9	1111	X 1	X 1	X 1	X 1

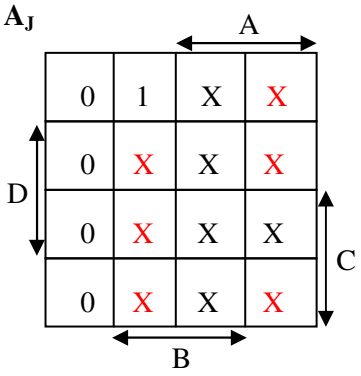
- Remember – we know the sequence of outputs, we are generating the sequence of inputs to go from one output to the next for each consecutive pair of outputs.

- Also, as the sequence repeats, the last state (1111) must return to the first state (0000).

- **Step 3** – Next, we need to put the sequence of values for each flipflop input (there are 8 inputs in total) into a Karnaugh Map in order to obtain a minimal circuit implementation.

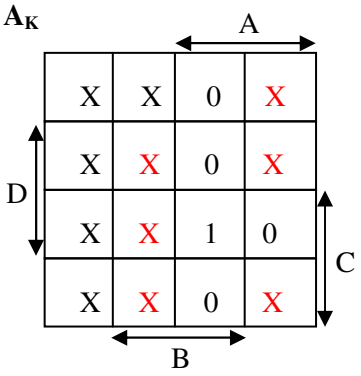
- Noting that the unused states are treated as don't care terms, the Karnaugh Maps for each of the eight inputs are as follows:

A_J



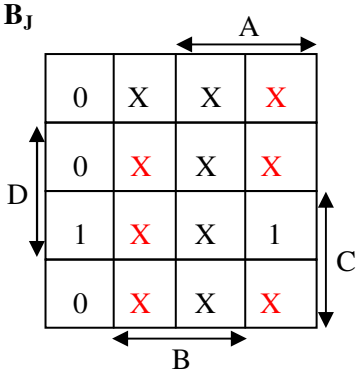
A_J =

A_K



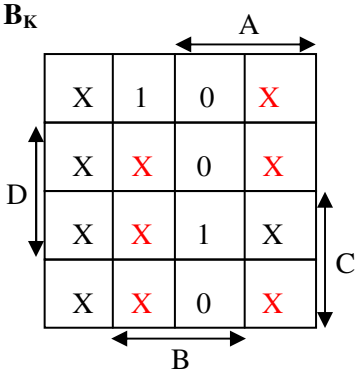
A_K =

B_J



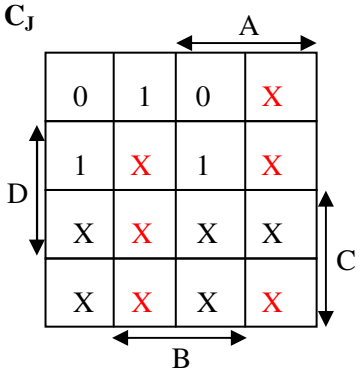
B_J =

B_K



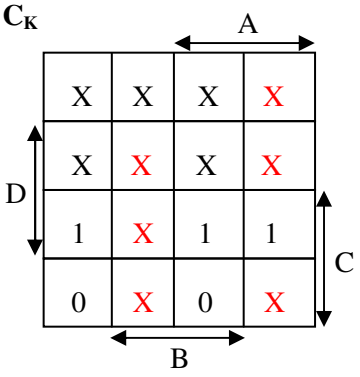
B_K =

C_J



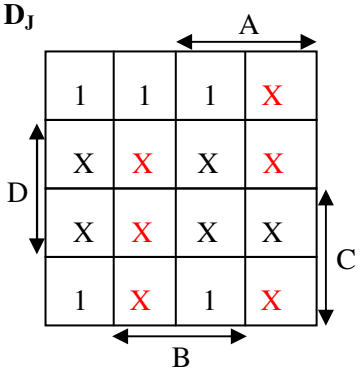
C_J =

C_K



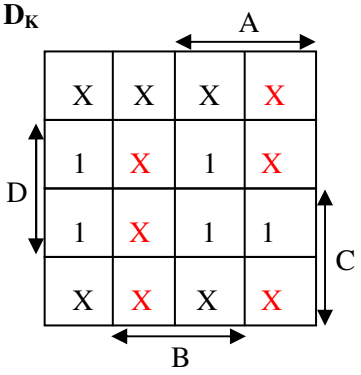
C_K =

D_J



D_J =

D_K



D_K =

- Finally, we need to use NAND implementation. Note that, by default, complements are available from the flipflops.

- Converting to NAND gives:

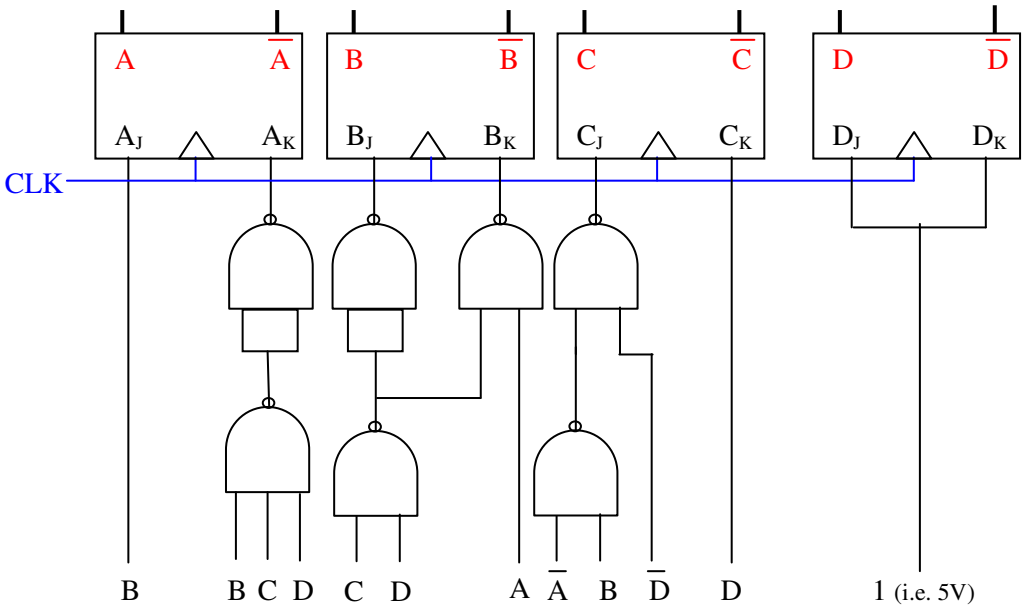
$A_J =$ $A_K =$

$B_J =$ $B_K =$

$C_J =$ $C_K =$

$D_J =$ $D_K =$

- This requires a total of 7 NAND gates, leading to the following circuit design:



- In the above diagram, all the variables (A, B, C, D) and their complements are generated by the outputs of the flipflops. For ease of viewing, these connections are omitted from the diagram.

- As is good practice in engineering design, we need to carry out a self check of our design.

- More importantly, we need to determine what the unused states do in our design, as these were not allowed for at the outset.

- Recall the concise form of the JK state table:

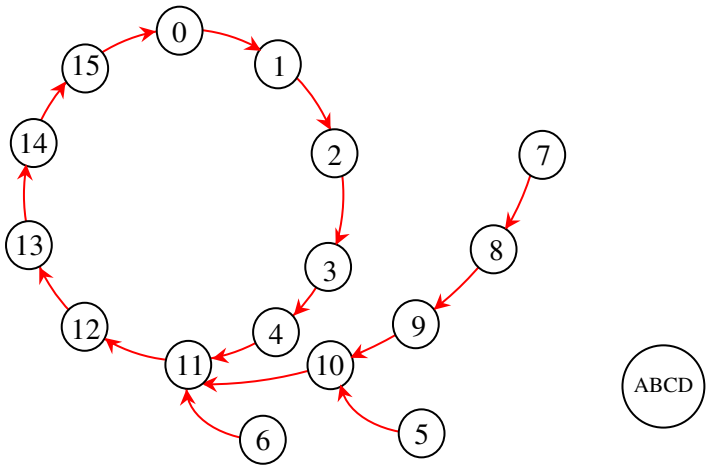
J	K	Q(τ)
0	0	**Q**
0	1	**0**
1	0	**1**
1	1	$\overline{\mathbf{Q}}$

- Carrying out a full self check of our circuit design (see previous example for more detail) we get the following table:

Minterm	Count	ABCD	Flipflop Inputs $A_J\ A_K$	$B_J\ B_K$	$C_J\ C_K$	$D_J\ D_K$
0	0	0000	0 0	0 1	0 0	1 1
1	1	0001	0 0	0 1	1 1	1 1
2	2	0010	0 0	0 1	0 0	1 1
3	3	0011	0 0	1 1	1 1	1 1
4	4	0100	1 0	0 1	1 0	1 1
11	5	1011	0 0	1 1	1 1	1 1
12	6	1100	1 0	0 0	0 0	1 1
13	7	1101	1 0	0 0	1 1	1 1
14	8	1110	1 0	0 0	0 0	1 1
15	9	1111	1 1	1 1	1 1	1 1
0	0	0000				
5	-	0101	1 0	0 1	1 1	1 1
10	-	1010	0 0	0 0	0 0	1 1
11	5	1011				
6	-	0110	1 0	0 1	1 0	1 1
11	5	1011				
7	-	0111	1 1	1 1	1 1	1 1
8	-	1000	0 0	0 0	0 0	1 1
9	-	1001	0 0	0 0	1 1	1 1
10	-	1010				

- All unused states return to the main cycle and hence **lockout does not occur** here.

- Finally, although not required for the question, the behaviour of the designed counter can be summarized by the following state diagram (expressed using minterms):



7.4 Registers

- A **register** is a **set of binary storage cells**, each storing one bit of information.
- It consists of an array of flipflops connected together with a common clock.
- While the actual content of a register is binary information, the interpretation of this information will vary considerably from one application to another.
- An *n*-bit register consists of *n* flipflops and is capable of storing binary information containing *n* bits.
- In addition to flipflops, registers may also contain some combinational logic which can perform certain data processing tasks.
- Typically, the flipflops are used to store the information and the combinational circuitry controls when and how the information is transferred into the register.

Shift registers …

- A shift register is one that moves (or shifts) its information sideways (either left to right or right to left).
- The flipflops are cascaded together (i.e. in series) with the output of one connected to the input of the next.
- The flipflops are controlled by a common clock. On each clock pulse the information is shifted by one bit to either the left or to the right.
- Note – a given binary number **multiplied by 2 corresponds to a shift to the LEFT** by one position and the new least significant bit (LSB) is set to 0. For example:

  Consider the binary number: 0100
  Shift to the left by one and insert 0 for the LSB gives: 1000

  This is the same as multiplying by 2, i.e.: 0100 **x 2** = 1000.

- Similarly – a given binary number **divided by 2 corresponds to a shift to the RIGHT** by one position and the new most significant bit (MSB) is set to 0. For example:

  Consider the binary number: 1000
  Shift to the right by one and insert 0 for the MSB gives: 0100

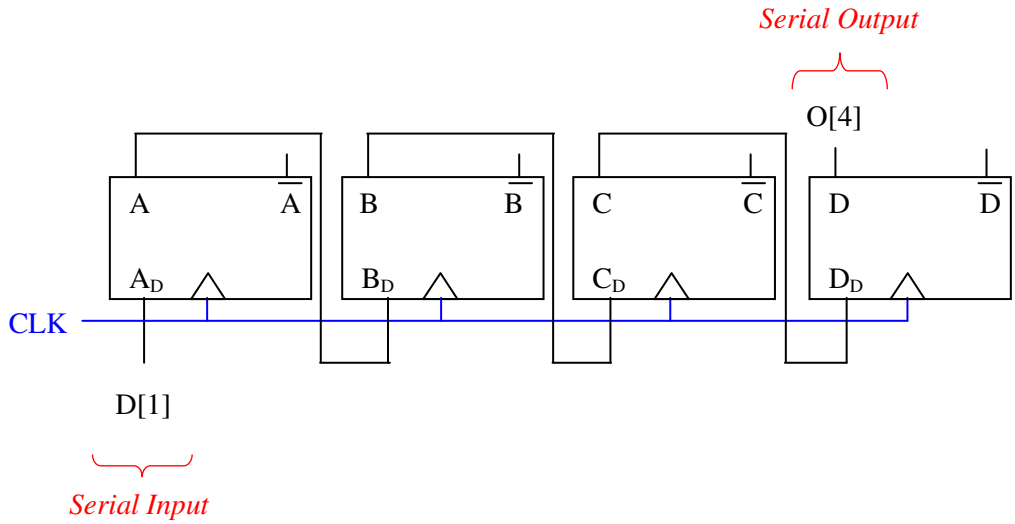
  This is the same as dividing by 2, i.e.: 1000 ÷ **2** = 0100.

Serial and Parallel Data entry …

- There are two ways by which data can be entered into registers, namely serial and parallel entry.

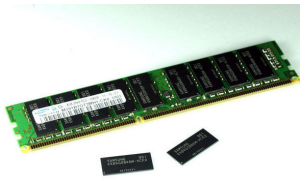
- Serial data entry is where the data is entered one bit at a time, as illustrated below:

`



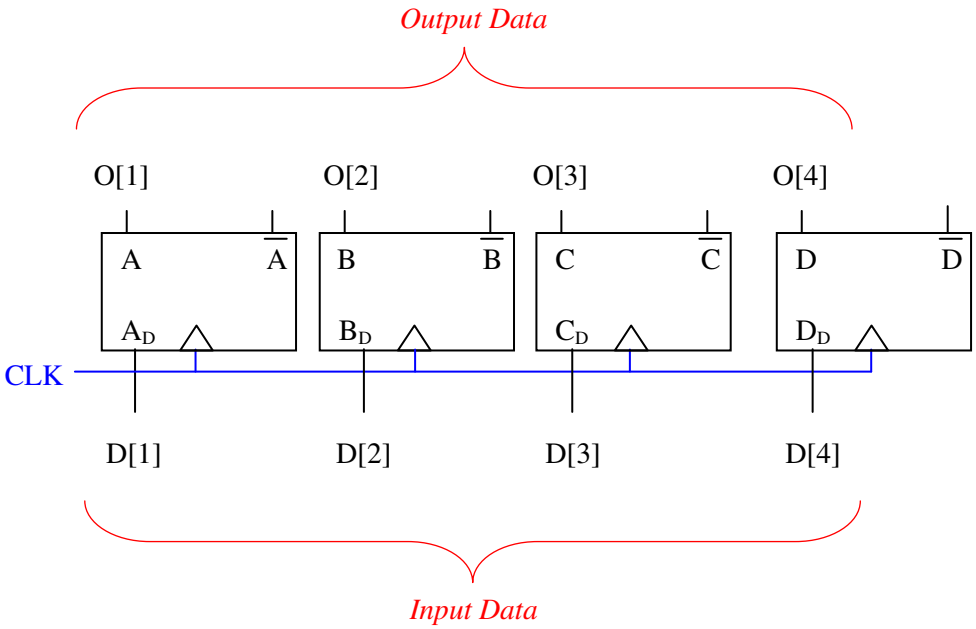
- On each clock pulse, one bit of information is entered into the flipflop A. On subsequent clock pulses, this bit gets shifted from one flipflop to the next

- The circuit shown is an example of a 4-bit shift register.



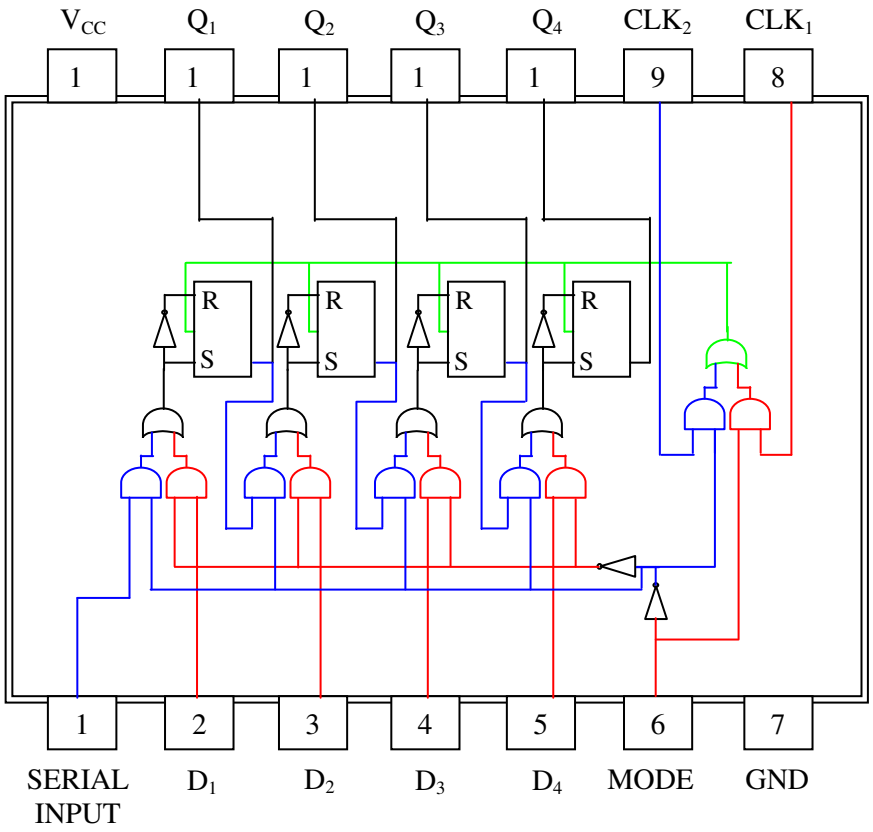
- Parallel data entry is where multiple data bits are entered to the register at the same time.

- The circuit below illustrates how 4-bits of information are transferred in parallel on a single clock pulse:



Universal Shift Register …

- Both the serial and parallel data entry modes can be incorporated into a single IC known as the ***Universal Shift Register***, as shown below:

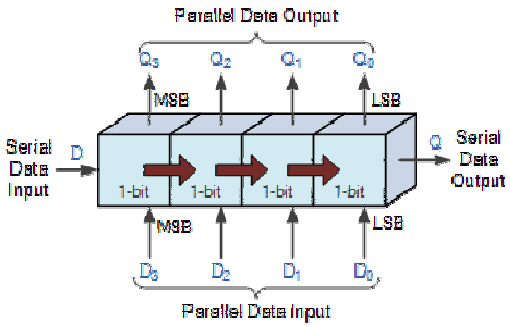


- The MODE option allows the choice of either serial or parallel mode. If mode = 1, then parallel entry is selected. If mode = 0, then serial entry is selected.

- The SR flipflops with invertors between the S and R inputs constitute D type flipflops.

- These are the master slave or edge-triggered type and it is this property which permits correct serial mode operation.

- The two AND gates and OR gate represent a ***data selector***. When mode = 1, then D_1 to D_4 is selected.



- The mode control also selects the clock input either CLK_1 or CLK_2. CLK_1 is the clock used in parallel mode and CLK_2 is the one used in serial mode.

- This device allows for easy conversion from serial to parallel. For example, a 4-bit serial word could be clocked into the shift registers in 4 cycles. When the full word is entered the mode can be changed to parallel and the parallel word can then be clocked out in one cycle.