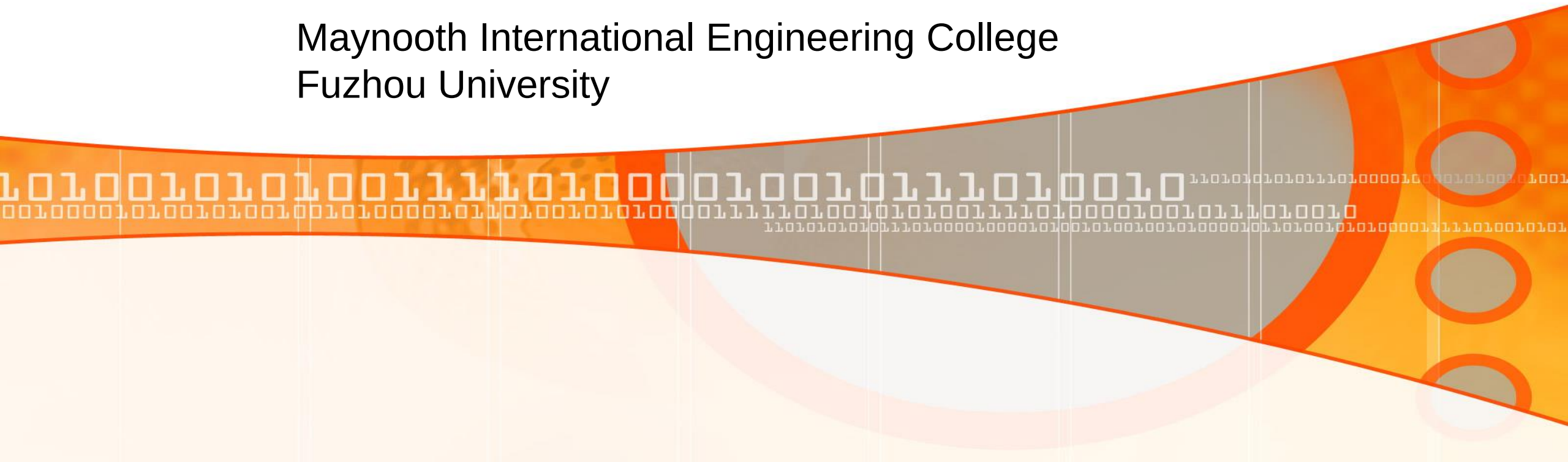
EE103 Digital Systems 1

Wong Chin Hong

106

Maynooth International Engineering College
Fuzhou University



So far ... !

- We've used Karnaugh Maps to minimise logic ...
- We've implemented circuits using NAND only or NOR only gates ...
- We've looked in detail at the SR, D, JK and T flipflops ...
- We've analysed a counter ...

COUNTER ATTACK

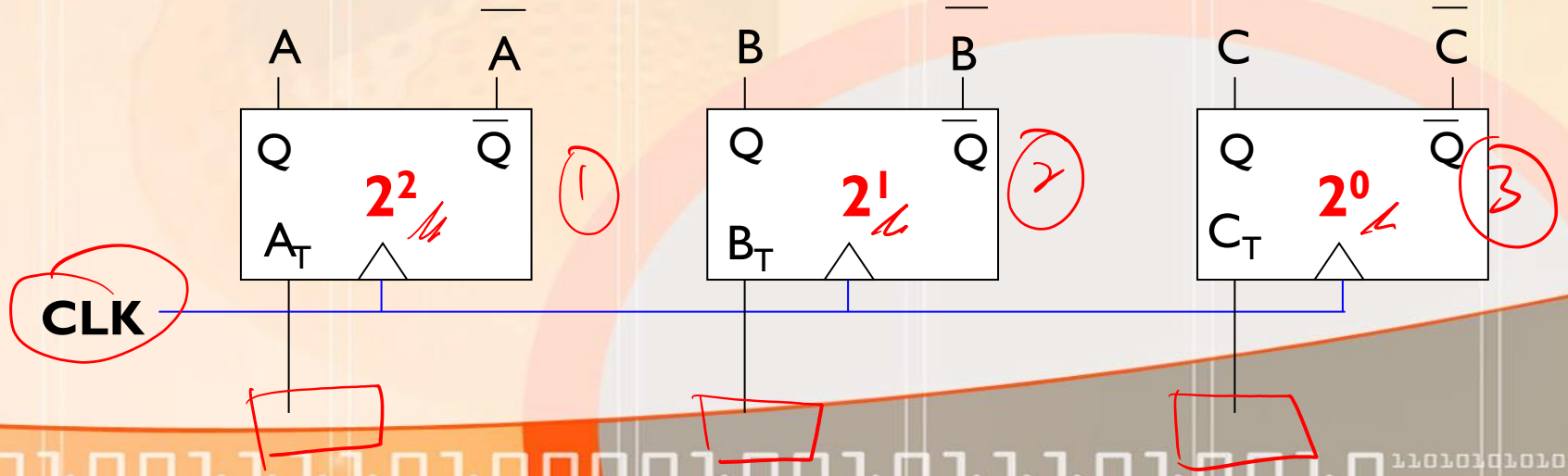
TODAY, we are going to look at designing a counter ...

Designing a Counter

- ***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.***

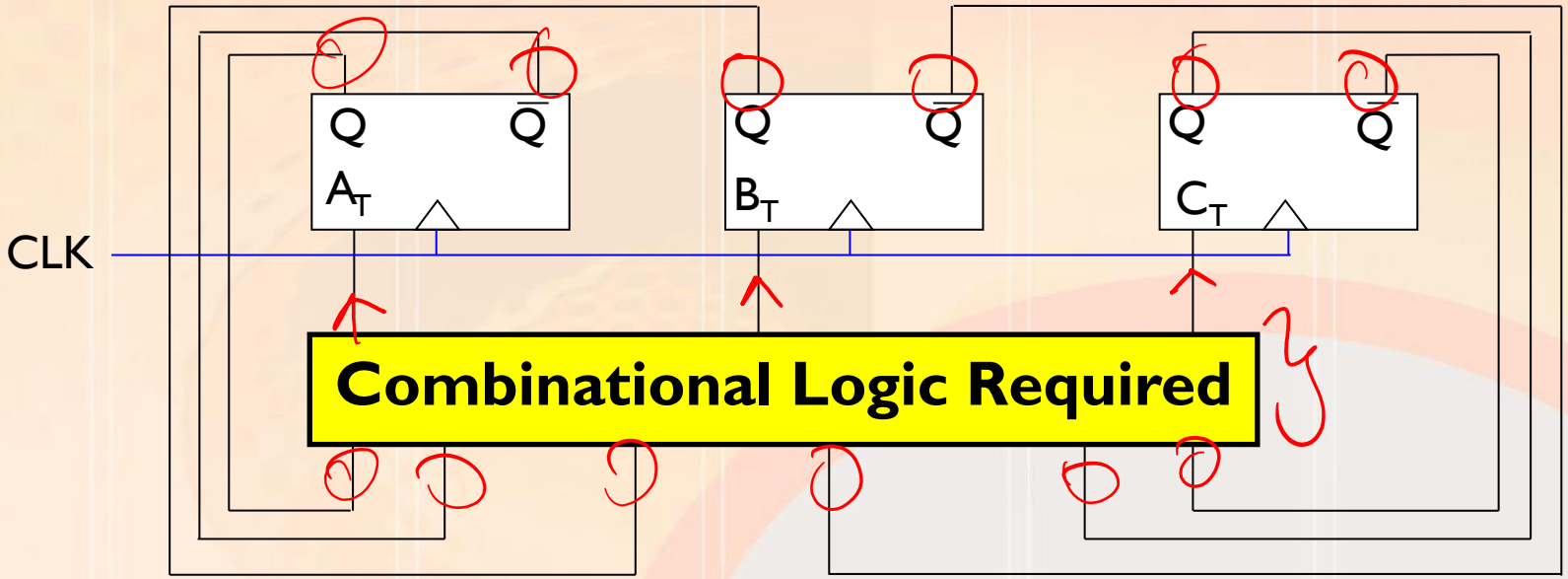
Designing a Counter

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



Designing a Counter

- 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.:



Designing a Counter

- **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**.

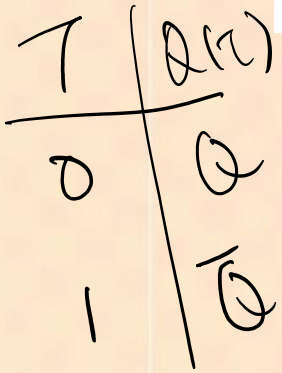
- *Recall the requirements table for the T flipflop …*

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

$Q(t) = Q$

$Q(t) = \overline{Q}$

Designing a Counter



- So, **taking each bit in turn**:

A goes from 0 to 0 ⇒ no change ⇒ $A_T = 0$
B goes from 0 to 0 ⇒ no change ⇒ $B_T = 0$
C goes from 0 to 1 ⇒ output needs to be toggled
⇒ $C_T = 1$

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

$$A_T\ B_T\ C_T = 0\ 0\ 1$$

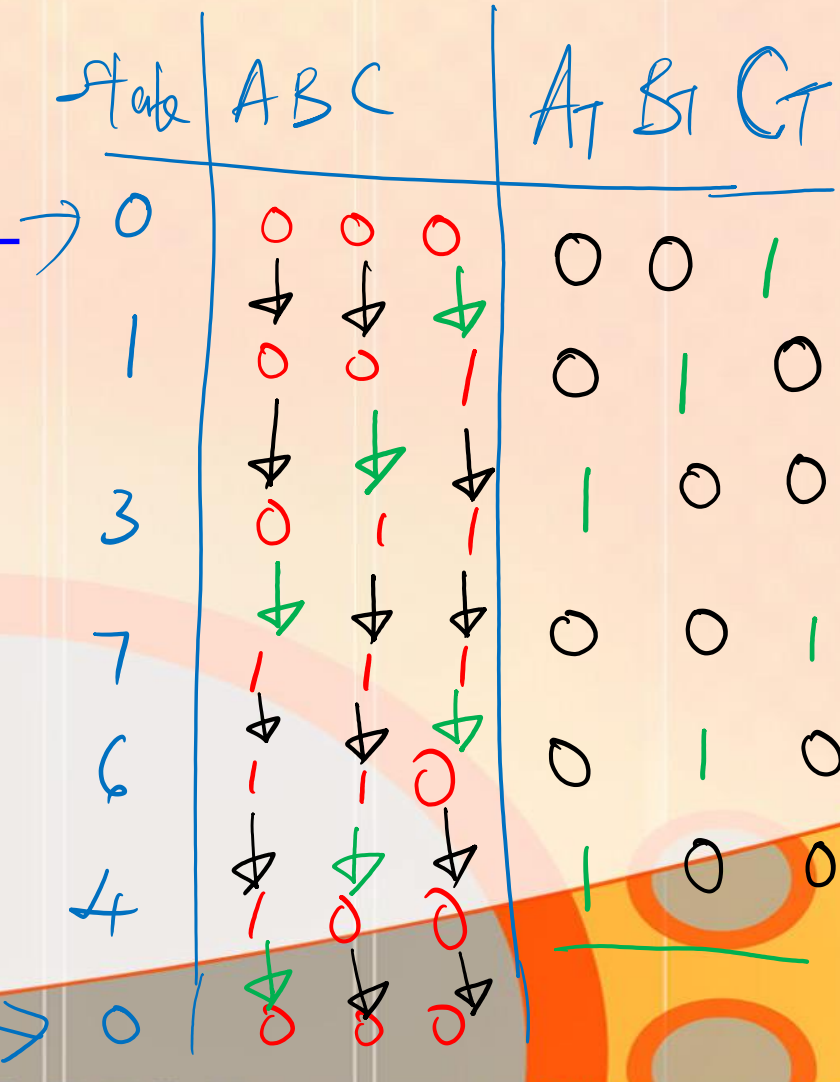
- Repeating this process for the remaining sequence of states (i.e. from, 1 to 3, 3 to 7, etc.), we obtain the following table:

Designing a Counter

T	Q(t+1)
0	Q
1	$\bar{Q}$

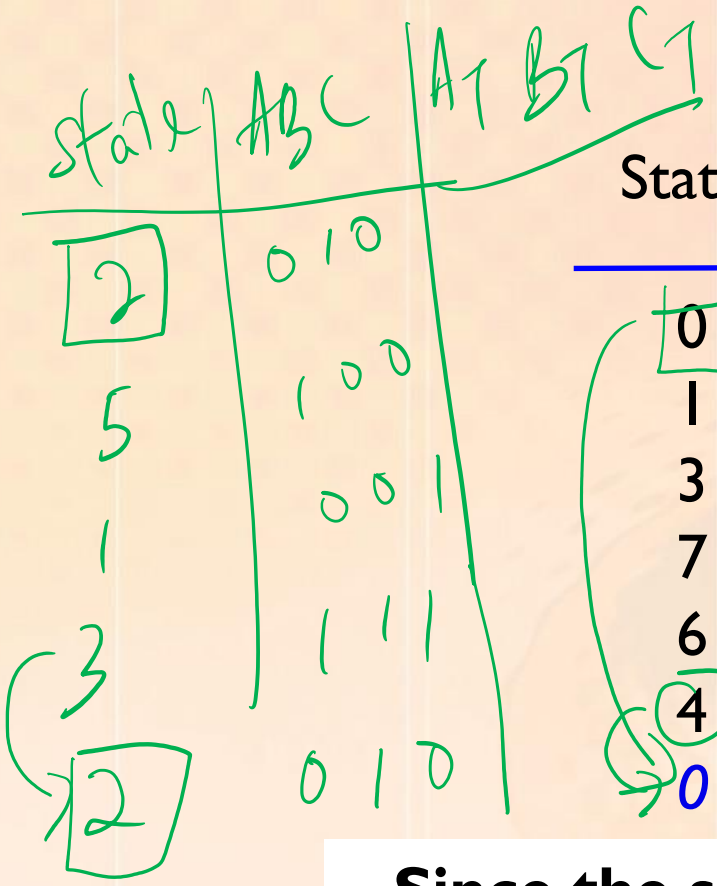
State	A B C	Flipflop Inputs A_T B_T C_T
0	0 0 0	0 0 1
1	0 0 1	0 1 0
3	0 1 1	1 0 0
7	1 1 1	0 0 1
6	1 1 0	0 1 0
4	1 0 0	1 0 0
0	0 0 0	

given



Designing a Counter

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



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

Designing a Counter

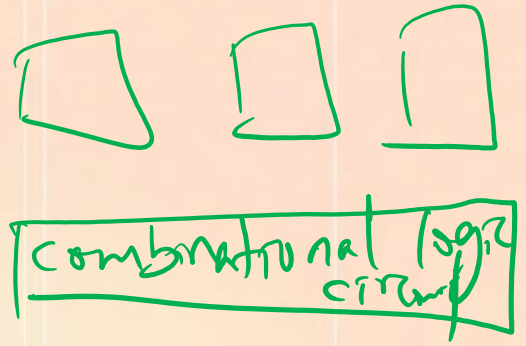
- **STEP 3** – Once we know the sequence of inputs to each of the flipflops, we then need to **generate a Karnaugh Map (KM) for each of the flipflop inputs** and obtain a minimal combinational logic circuit for implementing this sequence.

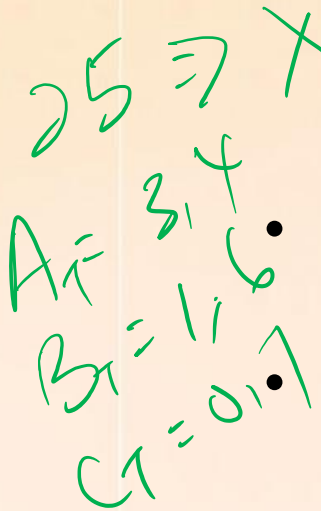
- A quick glance at the previous table shows that:

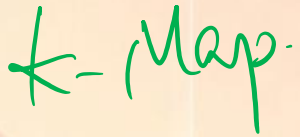
$$A_T = \Sigma(3,4),\ B_T = \Sigma(1,6),\ C_T = \Sigma(0,7)$$

- 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.

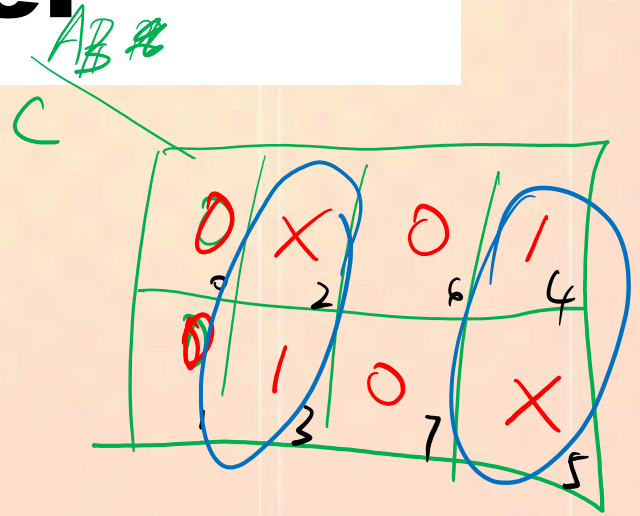






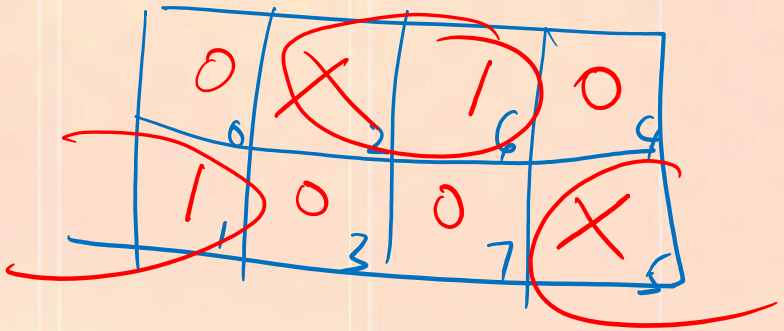
Designing a Counter

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



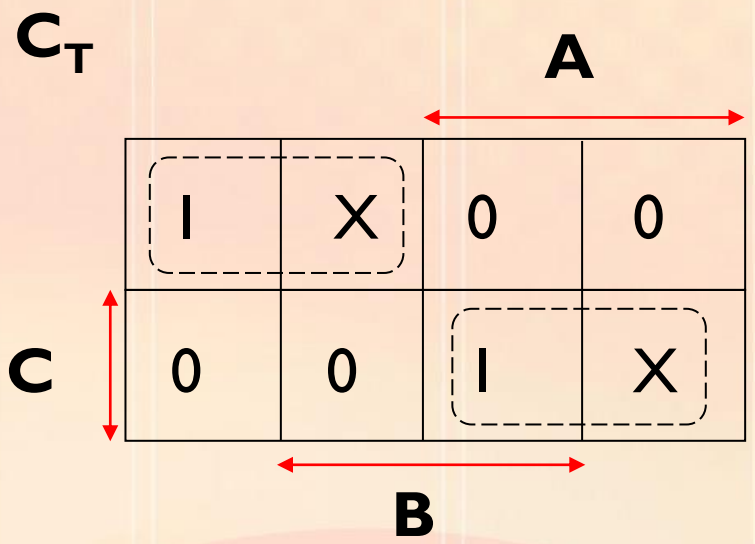
Designing a Counter

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



Designing a Counter

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



$$C_T = \bar{A}\bar{C} + AC$$

$$= \overline{A \oplus C}$$

Designing a Counter

- 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 to see if it produces the required sequence.

Designing a Counter

- Consider the first output of the sequence ABC = 000. Passing this through our circuit design gives:

$$\mathbf{A_T} = A \oplus B = 0 \oplus 0 = \mathbf{0}$$

$$\mathbf{B_T} = B \oplus C = 0 \oplus 0 = \mathbf{0}$$

Designing a Counter

- Consider the first output of the sequence ABC = 000. Passing this through our circuit design gives:

$$A_T = A \oplus B = 0 \oplus 0 = \mathbf{0}$$

$$B_T = B \oplus C = 0 \oplus 0 = \mathbf{0}$$

$$C_T = \overline{A} \oplus \overline{C} = 0 \oplus 0 = \mathbf{1}$$

$$A_T\ B_T\ C_T = 0\ 0\ 1$$

- Hence, if the current state is **ABC = 000** and the flipflop inputs are $A_T\ B_T\ C_T = 0\ 0\ 1$ then the next state will be:

$$A(\tau)\ B(\tau)\ C(\tau) = 0\ 0\ 1$$

Designing a Counter

- We now repeat the process by passing this state through our circuit, by generating the new flipflop inputs and by determining the next state.

Designing a Counter

- 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	0	1	0
3	0	1	1	1	0	0
7	1	1	1	0	0	1
6	1	1	0	0	1	0
4	1	0	0	1	0	0
0	*0*	*0*	*0*			

=> are correct

Designing a Counter

- We also need to **check the unused states**.

2 & 5 X

- 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:

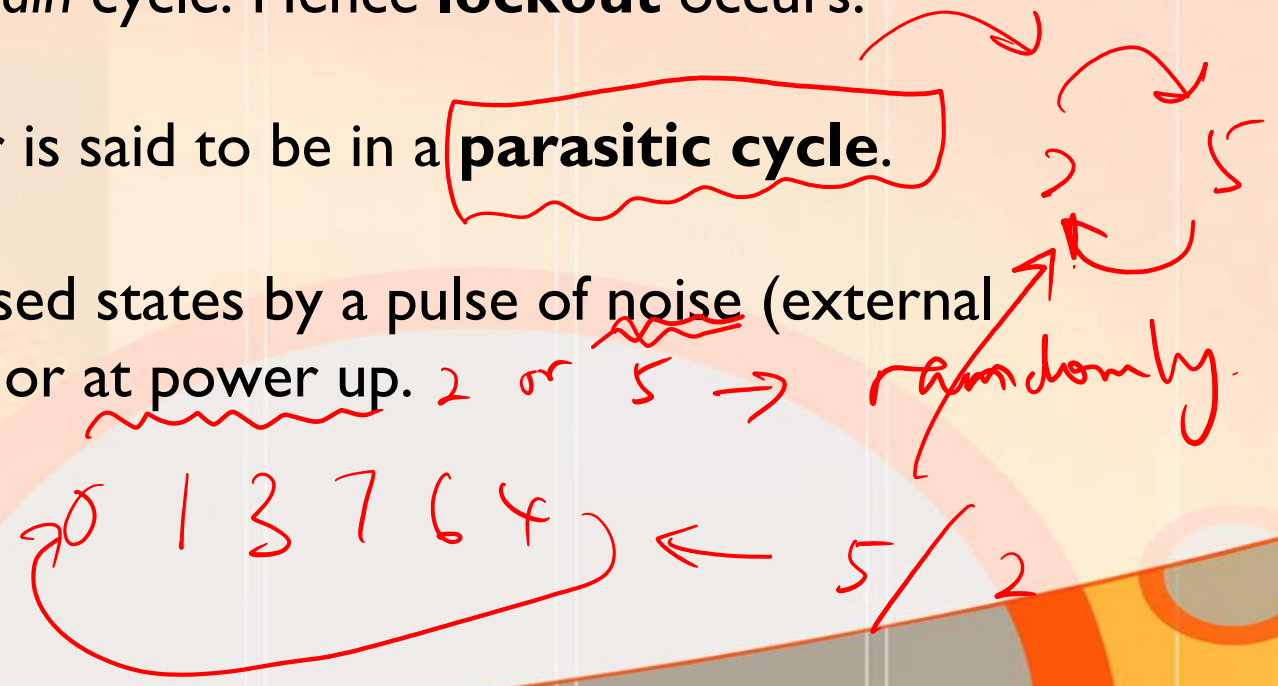
Designing a Counter

- 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	B_T	C_T
				$A \oplus B$	$B \oplus C$	$\overline{A \oplus C}$
2	0	1	0	1	1	1
5	1	0	1	1	1	1
2	0	1	0	1	1	1
5	1	0	1			

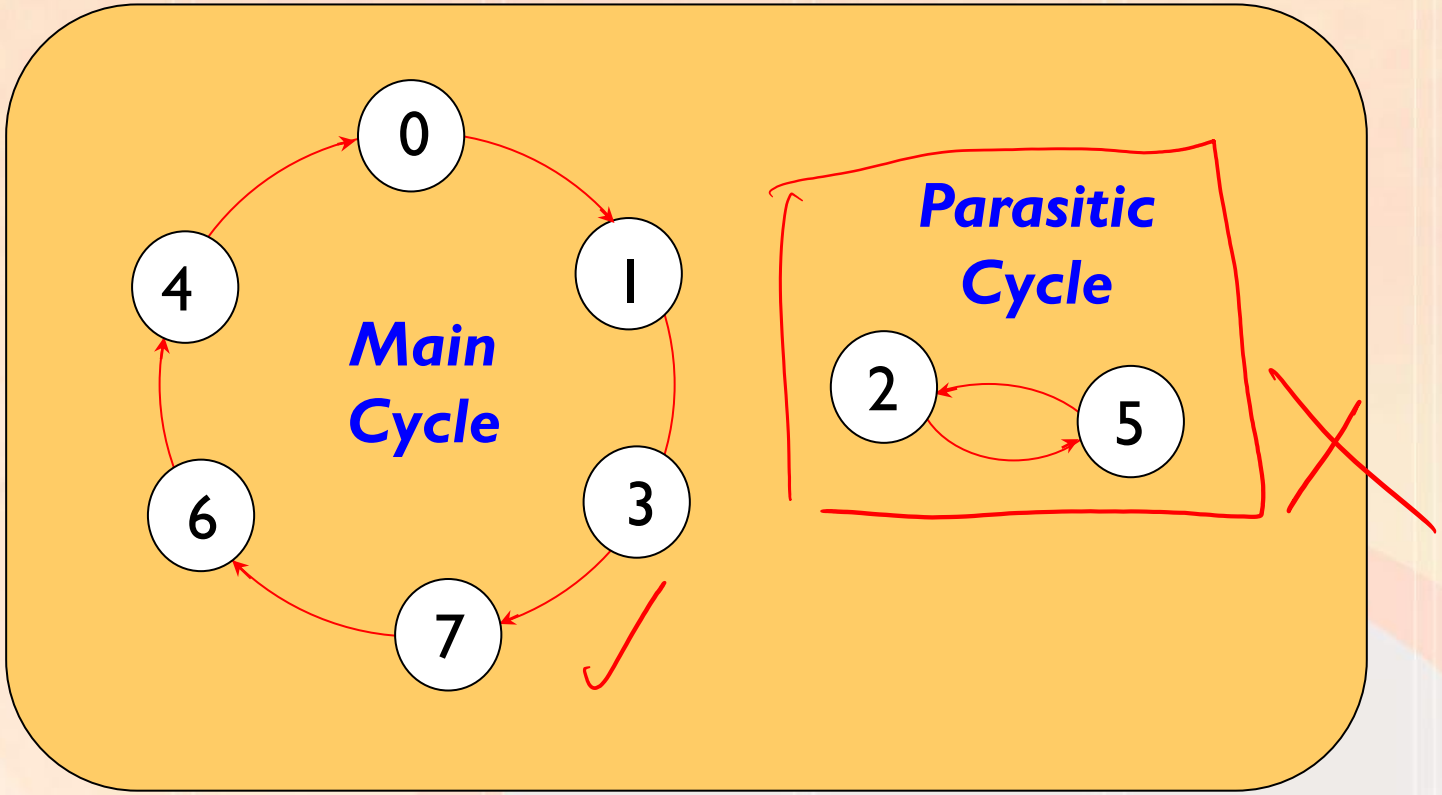
Designing a Counter

- This self check shows 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.



Designing a Counter

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



Designing a Counter

- 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.

Designing a Counter

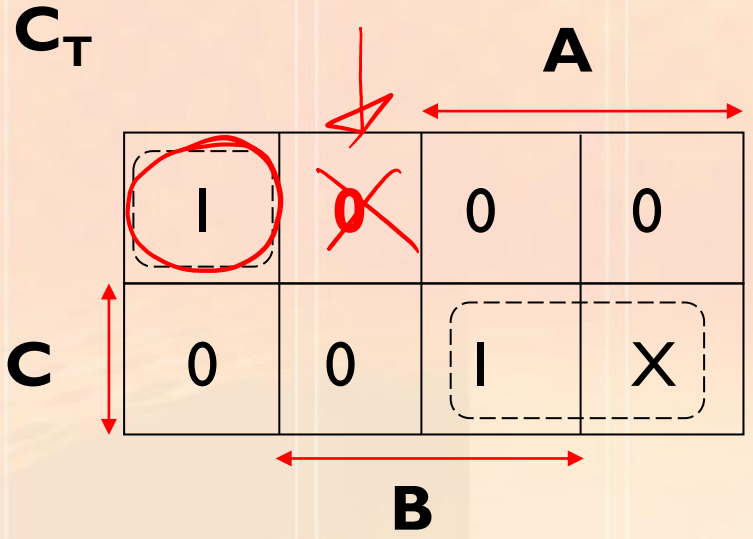
- Thus for example, we can make state 2 return to state 4 in the main cycle by forcing $\mathbf{C_T = 0}$ for this state as follows:

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 & B, so the input circuitry for these remain the same.

- However, we have modified the input sequences for the 'C' Toggle flipflop and so we need to revisit its KM as follows …

Designing a Counter

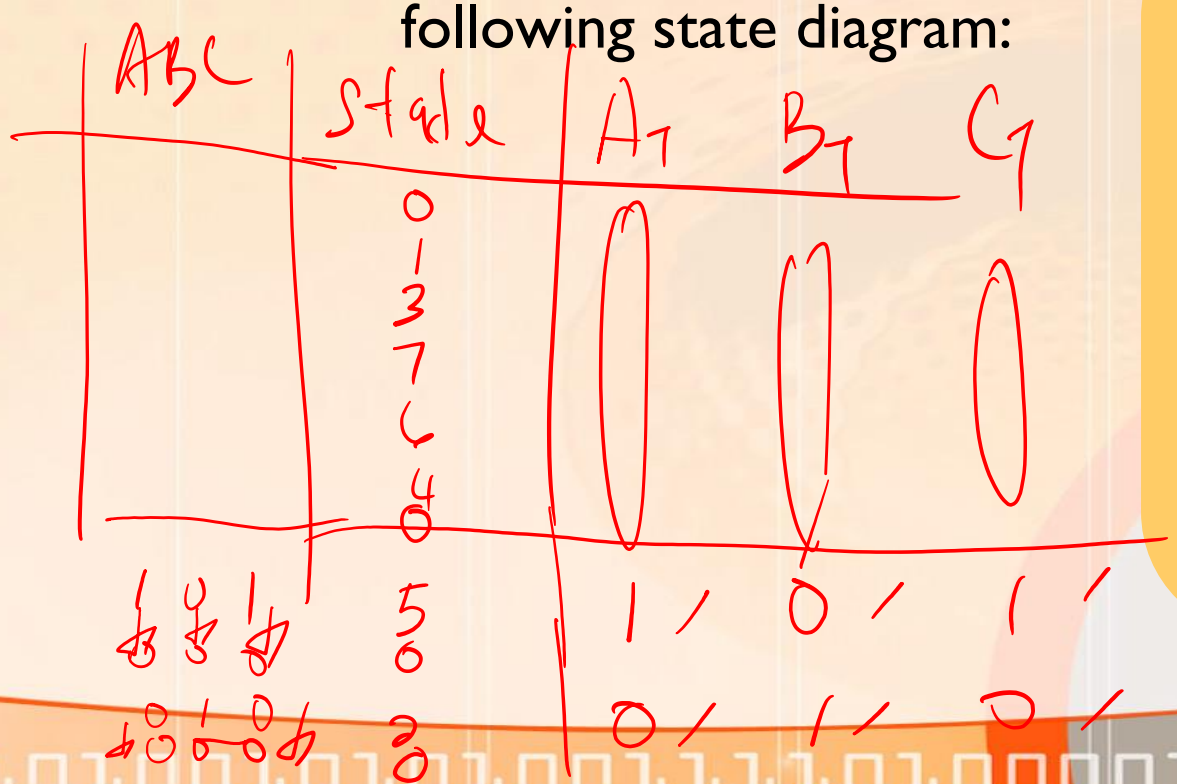


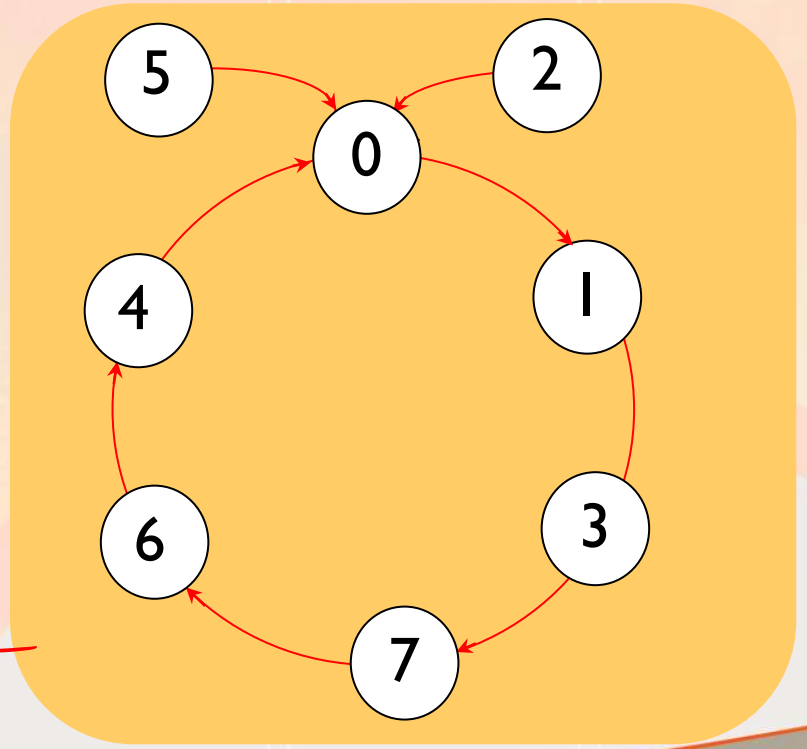
Designing a Counter

- 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.

2 5 X don't care

- This would give the following state diagram:





Designing a Counter

- 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** …

Designing a Counter

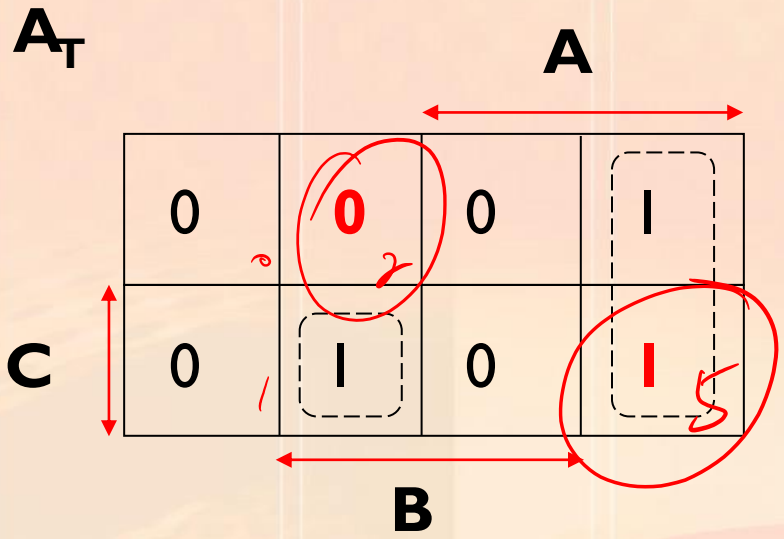
State	A	B	C	A_T	B_T	C_T
2	0	1	0	0	1	0
0	0	0	0			
5	1	0	1	1	0	1
0	0	0	0			

change all the don't care to

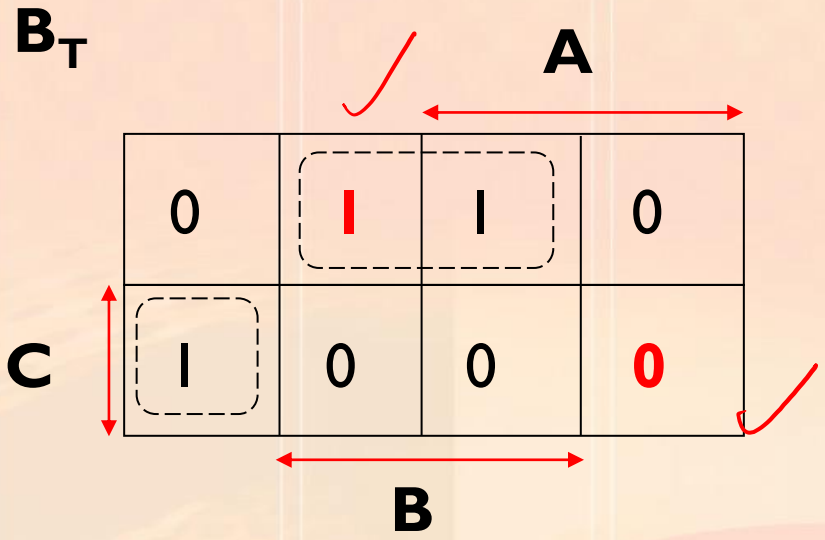
Designing a Counter

- 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.
- The new Karnaugh Maps are as follows …

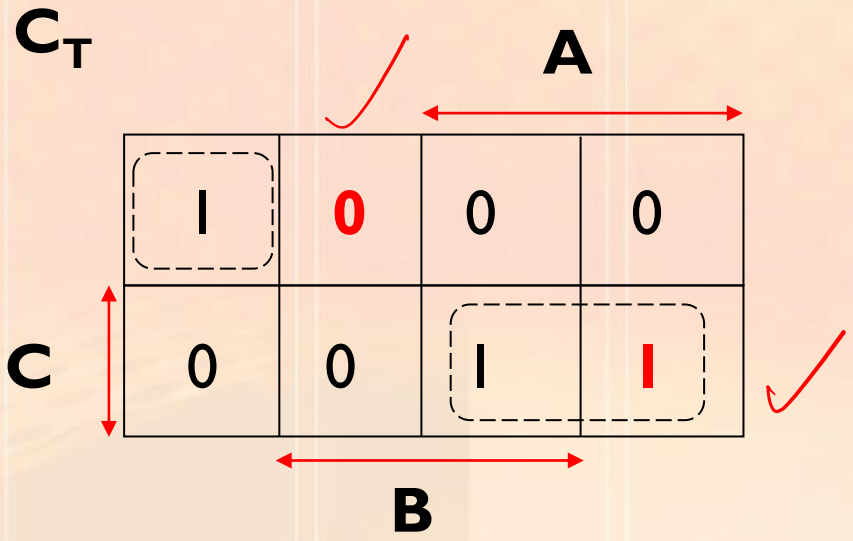
Designing a Counter



Designing a Counter



Designing a Counter



Designing a Counter

- 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.

Designing a Counter

- 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***.
- These added benefits usually come at the expense of additional gates and, hence, ***a non-minimal solution*** (in comparison with the first design approach).

Designing a Counter

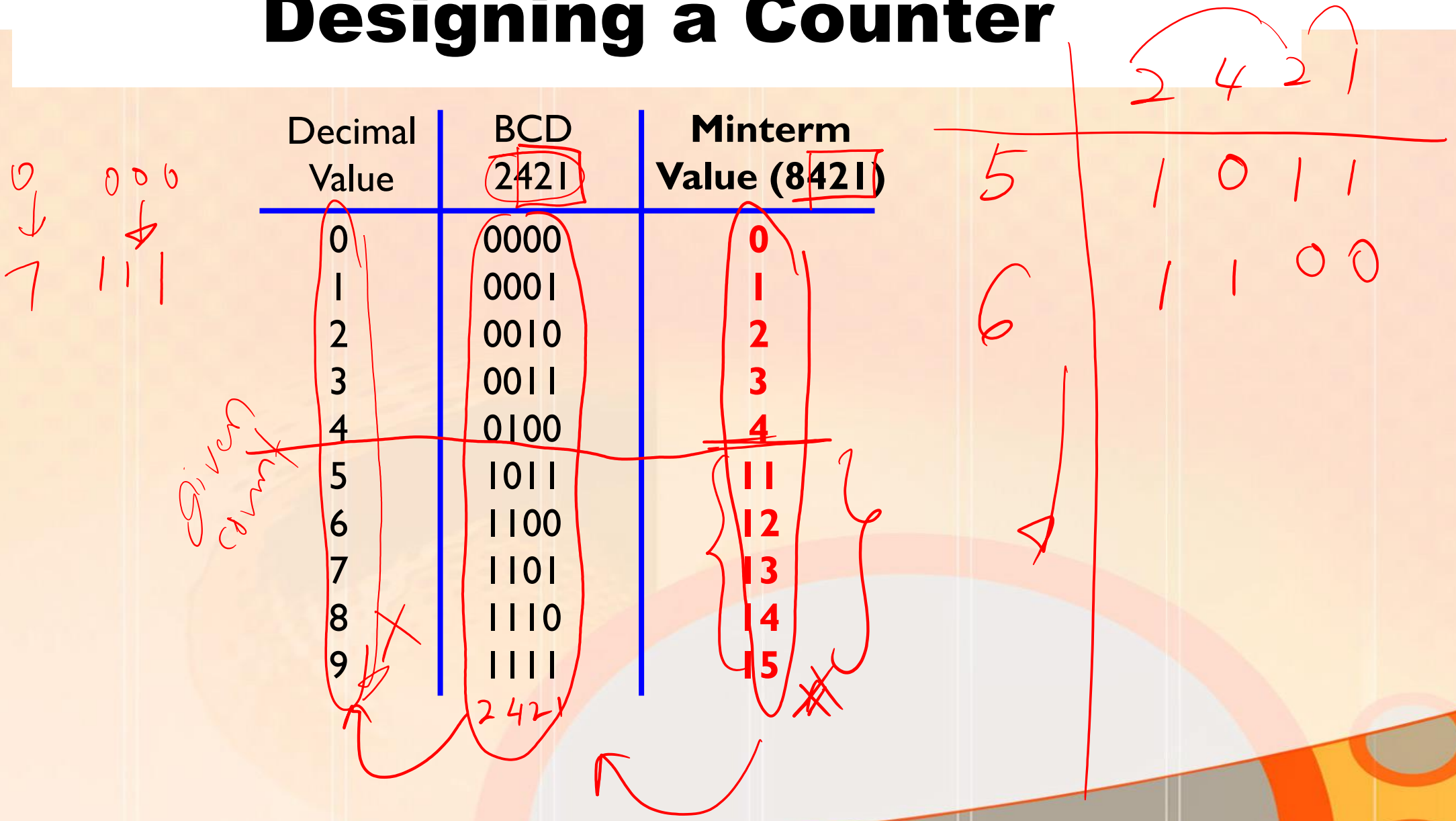
- ***Ex. 7.3 Design a BCD 2421 counter which has the sequence 0 1 2 3 4 5 6 7 8 9. Use JK flipflops and minimal NAND gating.***

Designing a Counter

- 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 …

Designing a Counter

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



Designing a Counter

- 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 $\mathbf{m_5}$, $\mathbf{m_6}$, $\mathbf{m_7}$, $\mathbf{m_8}$, $\mathbf{m_9}$, $\mathbf{m_{10}}$.

- 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

Designing a Counter

- **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 …

Designing a Counter

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				
4	4	0100				
11	5	1011				
12	6	1100				
13	7	1101				
14	8	1110				
15	9	1111				

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

Designing a Counter

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

Designing a Counter

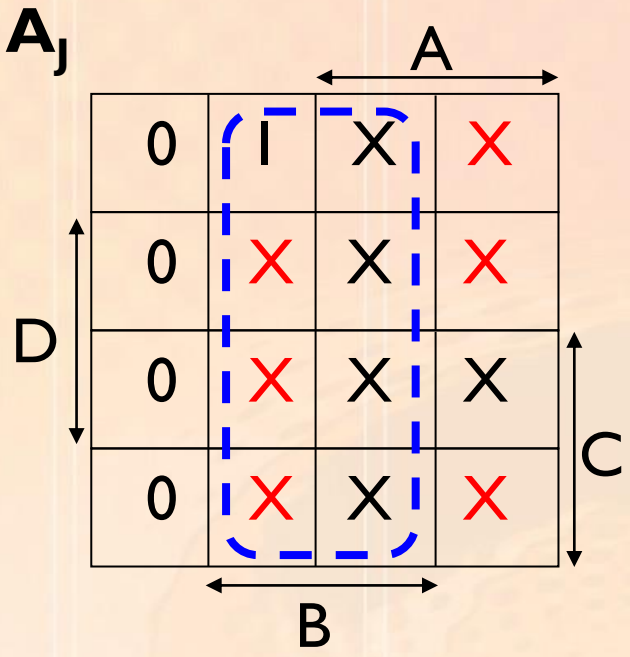
- Remember – we know the sequence of outputs, we are generating the sequence of inputs to go from one output to the next for each consecutive pair of outputs.

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

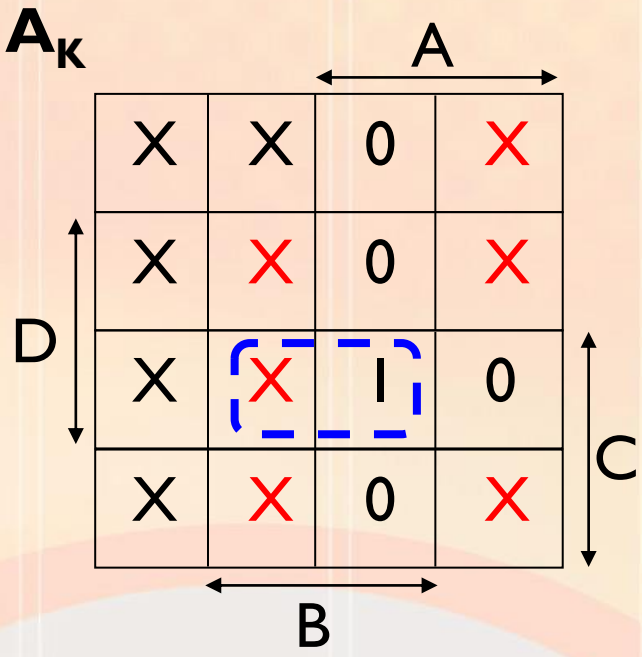
- **Step 3** – Next, we need to put the sequence of values for each flipflop input (there are 8 inputs in total) into a Karnaugh Map in order to **obtain a minimal circuit implementation**.

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

Designing a Counter

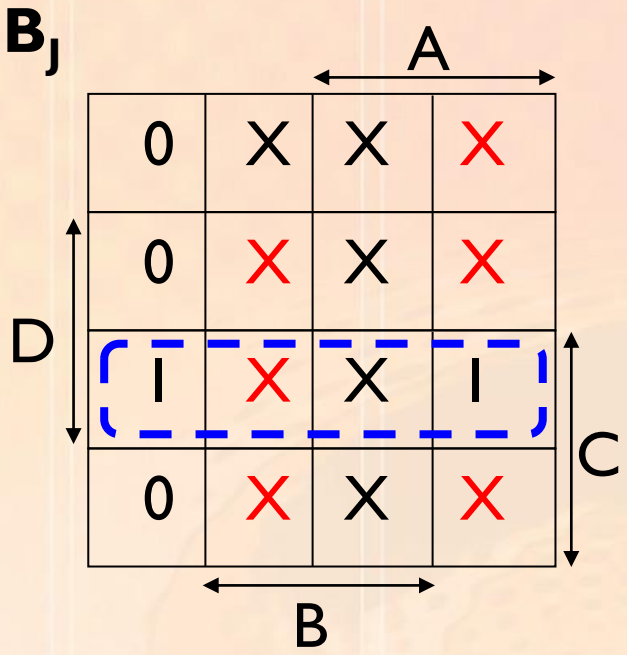


$A_J = B$

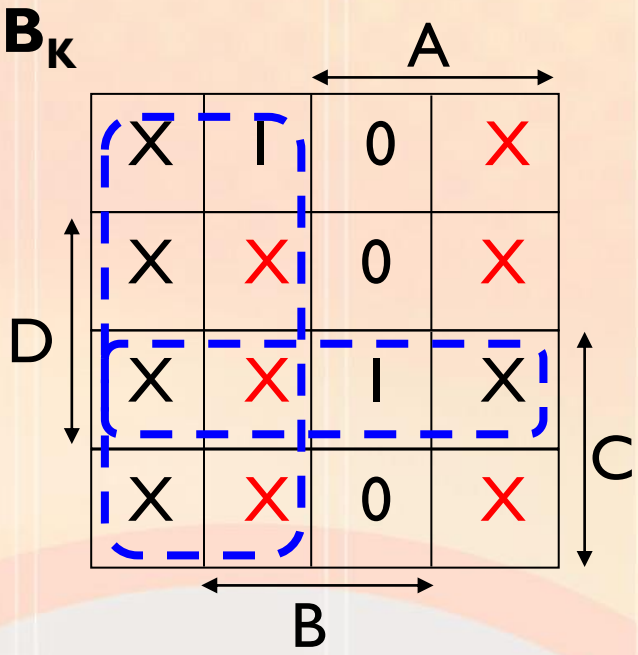


$A_K = BCD$

Designing a Counter



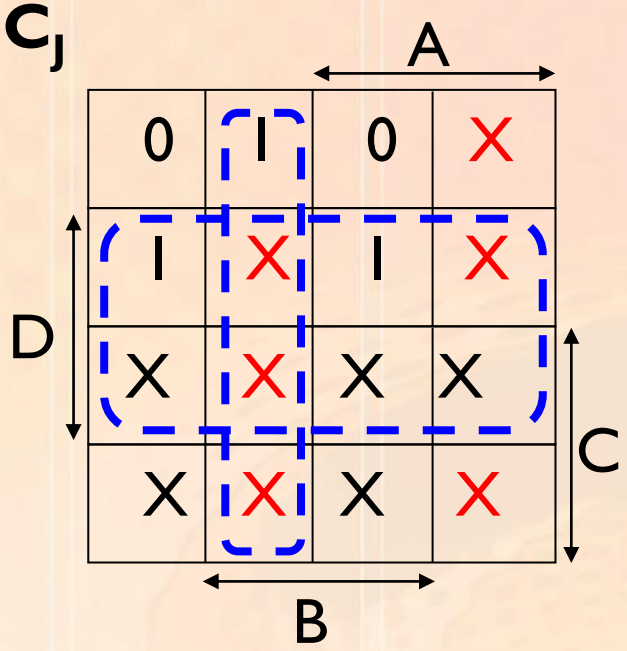
$B_J = CD$



$B_K = \overline{A} + CD$

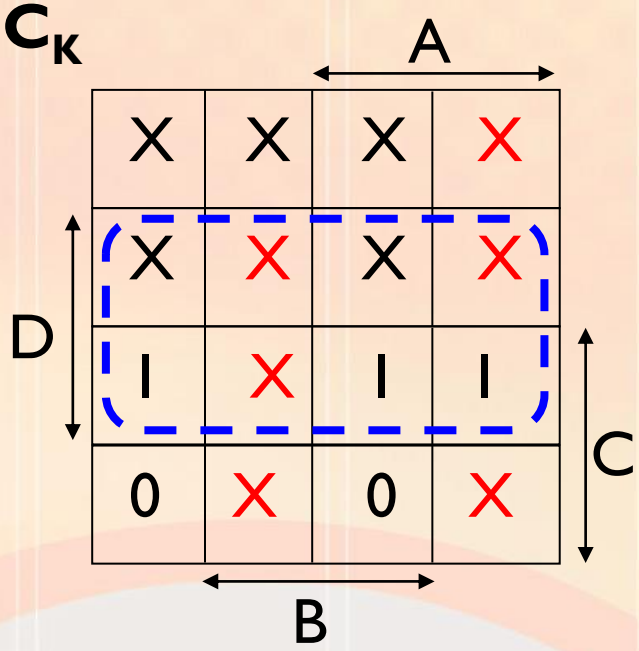
Designing a Counter

C_J



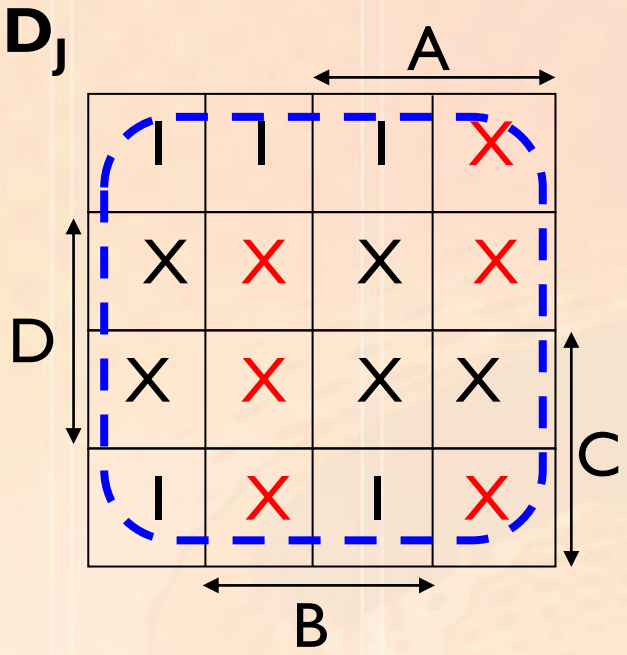
$C_J = D + \overline{A}B$

C_K

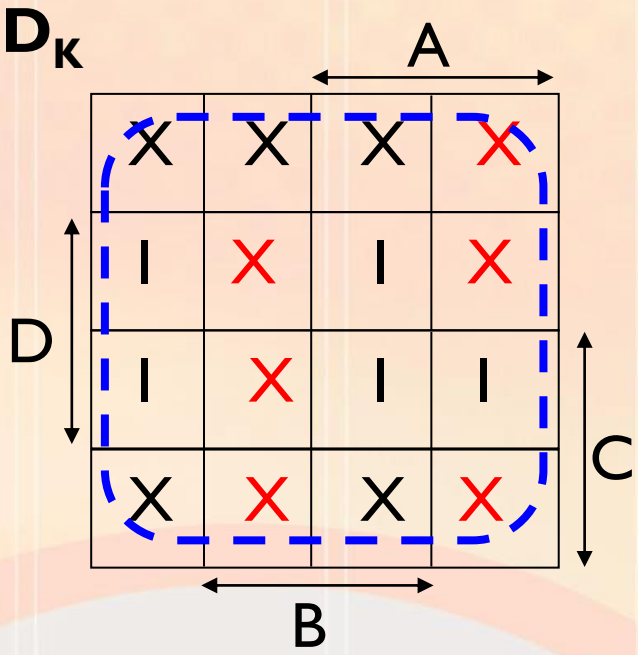


$C_K = D$

Designing a Counter



$D_J = 1$



$D_K = 1$