Week 5 - T1 2020

Sequential Circuit Design

ELEC2141: Digital Circuit Design

UNSW SYDNEY

Summary

Sequential circuits

SR, D Latches

D, JK and T Flip- flops

Analysis of sequential circuits

State equations

State tables

State diagrams

Overview

Moore and Mealy sequential circuits

State reduction

State assignment

Sequential circuit design

Circuit synthesis

Moore to Mealy/Mealy to Moore conversion

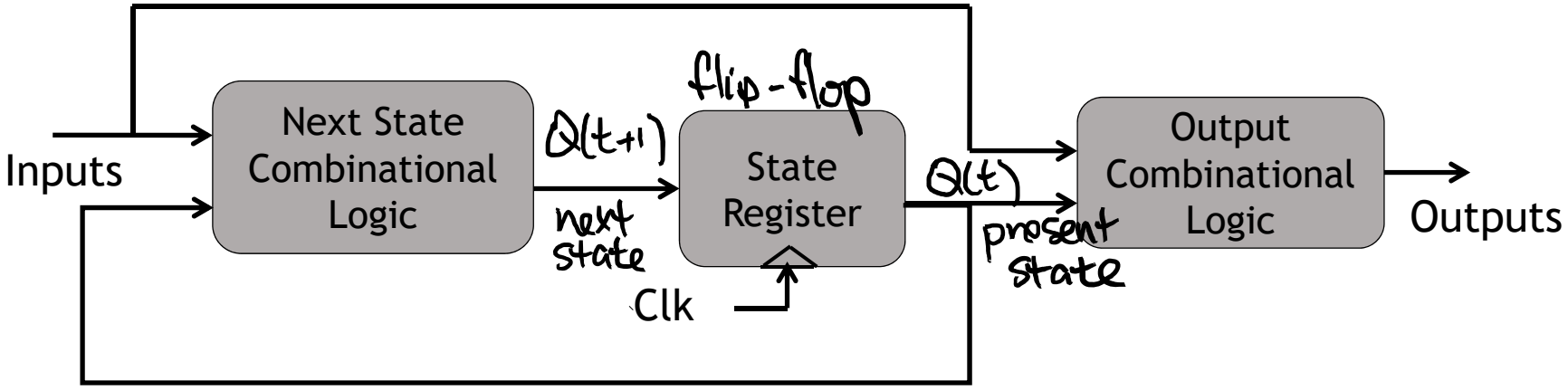
Reading: Mano - Chapter 4, 4.3-4.4

Mealy and Moore models

Two models of sequential circuits: Mealy and Moore

Mealy models: the output is a function of both the present state and the input

Also referred to as *Mealy Finite State Machines (FSM)*



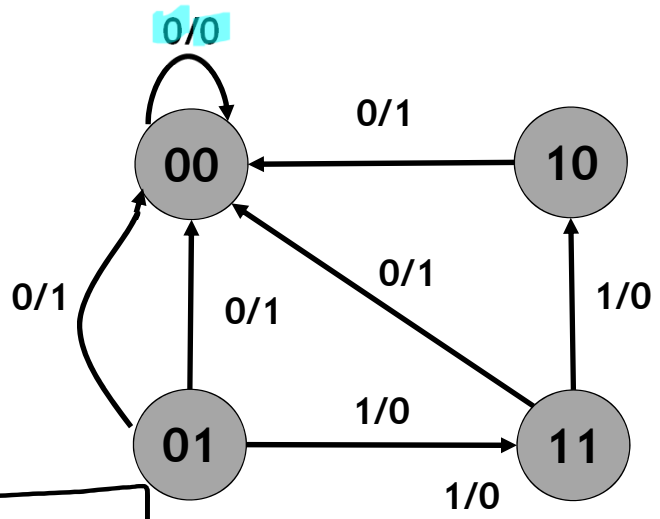
Mealy Finite Machine

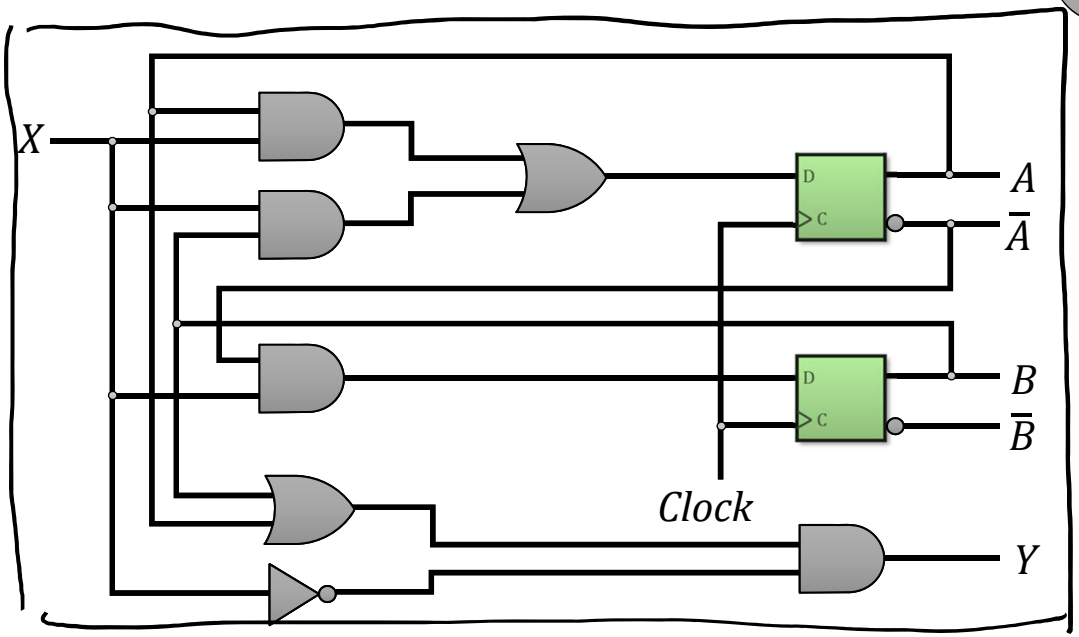
UNSW SYDNEY

Mealy FSM example

Present state		Next state				Output	
		$X=0$		$X=1$		$X=0$	$X=1$
A	B	A	B	A	B		
0	0	0	0	0	1	0	0
0	1	0	0	1	1	1	0
1	0	0	0	1	0	1	0
1	1	0	0	1	0	1	0

$$Y(t) = [A + B]\overline{x}$$





FSM

UNSW SYDNEY

Mealy models

The outputs may change during the clock cycle

Momentary false values (glitches) may occur due to the delay encountered from the time that the inputs change and from the time that flip-flop outputs change

In order to solve mitigate these problems,

- Inputs must be synchronized with the clock
- Outputs must be sampled immediately before the clock edge
- Inputs are changed at the inactive edge of the clock

Thus, in a Mealy circuit, the output value is the value that is present immediately before the active edge of the clock

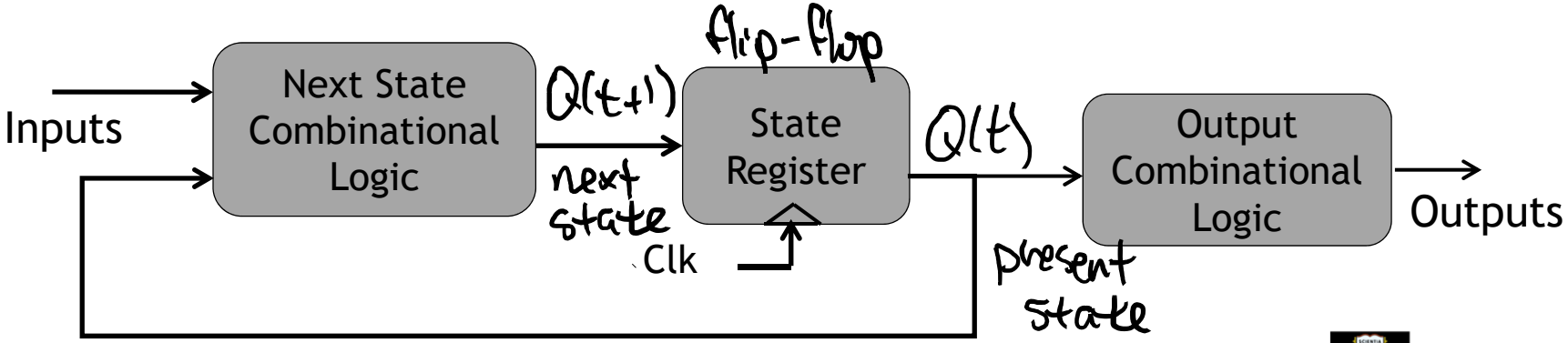
Moore model

Moore Model: the output is a function of only the present state

Also known as *Moore Finite State Machine*

State tables do not need separate output listing for various input combinations.

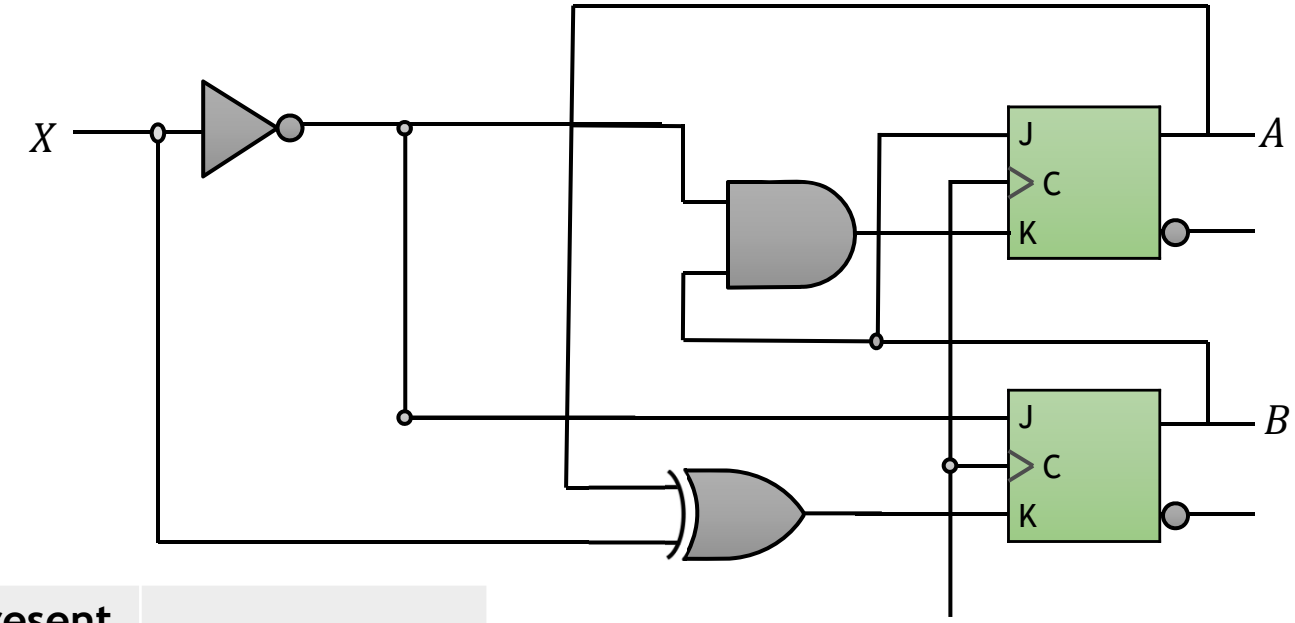
Output value is indicated inside the circle together with the present state.



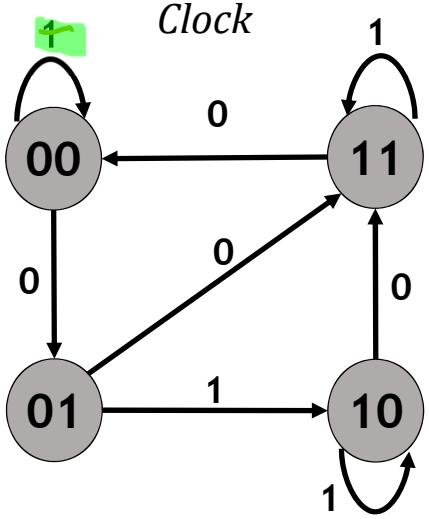
Moore Finite Machine

UNSW SYDNEY

Moore FSM example

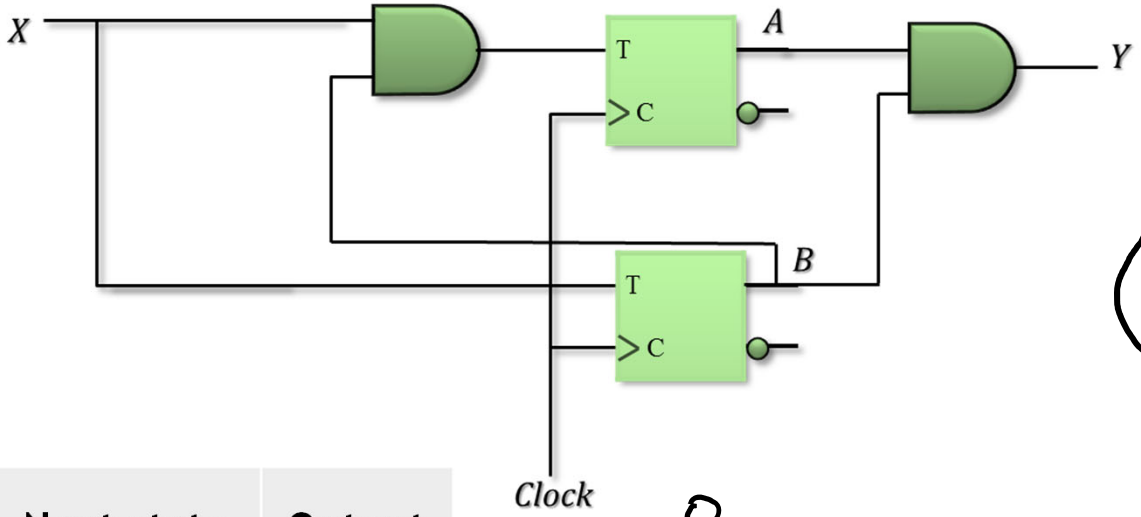


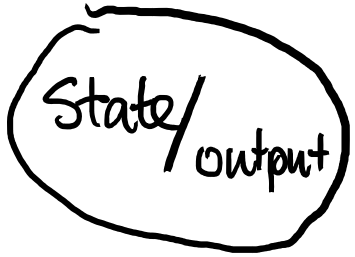
Present state		Next state			
		$X=0$		$X=1$	
A	B	A	B	A	B
0	0	0	1	0	0
0	1	1	1	1	0
1	0	1	1	1	0
1	1	0	0	1	1



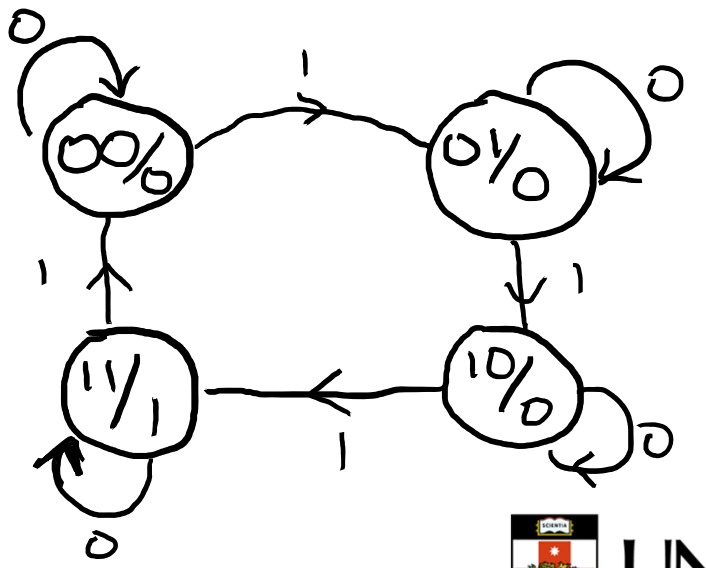
UNSW SYDNEY

Moore FSM example





Present state		Next state				Output
		$X=0$		$X=1$		
A	B	A	B	A	B	Y
0	0	0	0	0	1	0
0	1	0	1	1	0	0
1	0	1	0	1	1	0
1	1	1	1	0	0	1



State reduction

State reduction is a necessary to realize sequential circuits with a fewer numbers of flip-flops and hence reduce the cost of a circuit

State-reduction algorithms reduce the number of states in a state table while maintaining the same input-output requirements and relationships

Steps in state reduction

1. Produce state table from the state diagram
2. Apply the following algorithm:

   Two states are said to be equivalent if, for each member of the set of inputs, they give exactly the same output and send the circuit either to the same state or equivalent state
3. Construct *implication table* to check each pair of states for possible equivalence

Implication table

Implication tables consist of squares, one for every possible pair of states, with spaces to list any possible implied states

Present state	Next state		Output	
	$X=0$	$X=1$	$X=0$	$X=1$
a	c	b	0	1
b	d	a	0	1
c	a	d	1	0
d	b	d	1	0

Present states a and b have the same output for the same input

Their next states are c and d for $X=0$ and b and a for $X=1$

The characteristics of equivalent states are that if (a,b) imply (c,d) and (c,d) imply (a,b), then both pairs of states are equivalent

Implication table

When this relationship exist, we say *(a,b) implies (c,d)*

That is if a and b are equivalent, then c and d have to equivalent

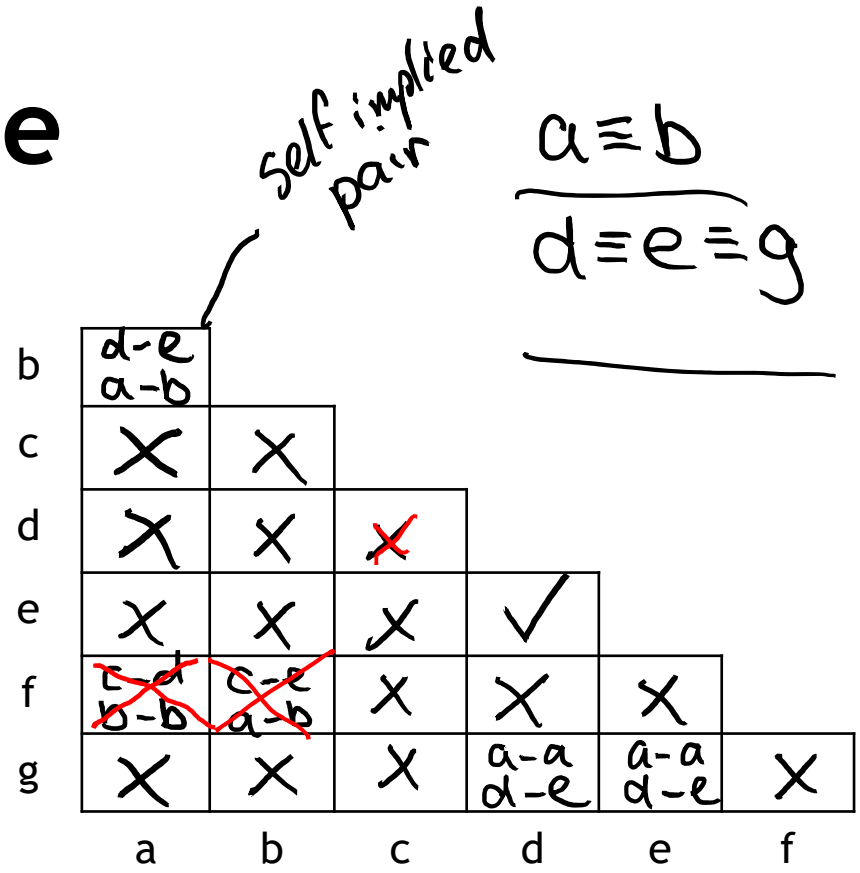
We can also observe that (c,d) implies (a,b)

> *The characteristics of equivalent states is that if (a,b) imply (c,d) and (c,d) imply (a,b), then both pairs of states are equivalent*

The previous state table can be reduced to only two states

Implication table

Present state	Next state		Output	
	$X=0$	$X=1$	$X=0$	$X=1$
a	d	b	0	0
b	e	a	0	0
c	g	f	0	1
d	a	d	1	0
e	a	d	1	0
f	c	b	0	0
g	a	e	1	0



Two states that are not equivalent are marked with a cross (X) in the corresponding square whereas their equivalence is recorded with a check mark (✓)

UNSW SYDNEY

Steps in filling implication table

Place a cross in any square corresponding to non-equivalent states (i.e. different outputs)

Enter the implied states in the remaining squares (top-down - then on to next column to the right)

Record check marks for equivalent states

Make successive passes through the table to place a cross and check marks

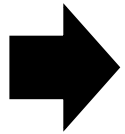
Continue procedure until no additional squares can be crossed out

	a	b	c	d	e	f
b	d-e b-a					
c	×	×				
d	×	×	×			
e	×	×	×	✓		
f	~~c-d~~ ~~b-b~~	~~c-e~~ ~~a-b~~	×	×	×	
g	×	×	×	a-a d-e	a-a d-e	×

Implication table result

b → a
e, g → d

Present state	Next state		Output	
	$X=0$	$X=1$	$X=0$	$X=1$
a	d	a	0	0
~~a~~	~~d~~	~~a~~	~~0~~	~~0~~
c	d	f	0	1
d	a	d	1	0
d	a	d	1	0
f	c	a	0	0
~~d~~	~~a~~	~~d~~	~~1~~	~~0~~

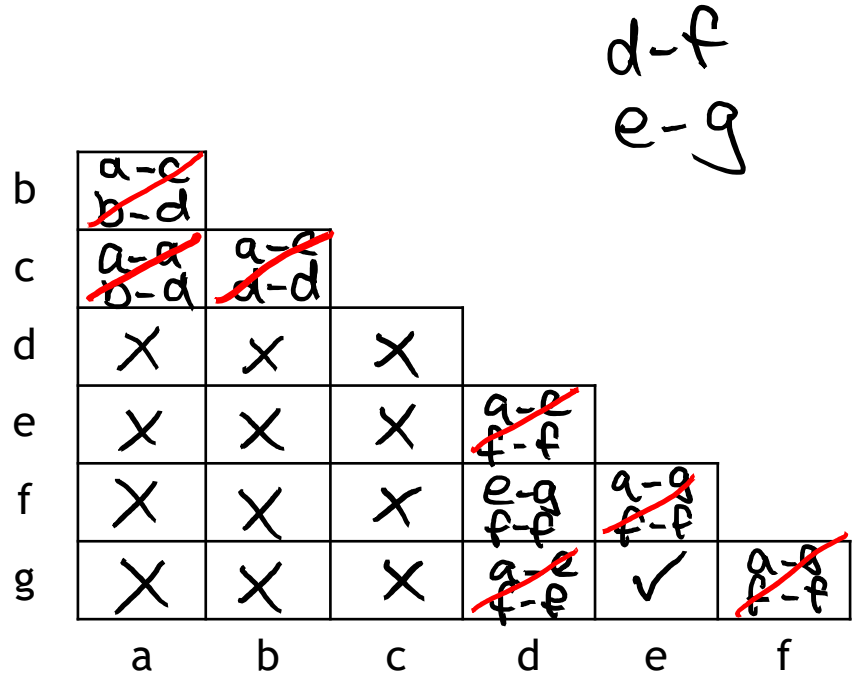


Present state	Next state		Output	
	$X=0$	$X=1$	$X=0$	$X=1$
a	d	a	0	0
c	d	f	0	1
d	a	d	1	0
f	c	a	0	0

UNSW SYDNEY

Implication table example

Present state	Next state		Output	
	$X=0$	$X=1$	$X=0$	$X=1$
a	a	b	0	0
b	c	d	0	0
c	a	d	0	0
d	e	f	0	1
e	a	f	0	1
f	g	f	0	1
g	a	f	0	1



state: 2^n − no. f/f

7 states − 4 states

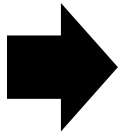
3 f/f 2 f/f

UNSW SYDNEY

Implication table example

f → d
g → e

Present state	Next state		Output	
	$X=0$	$X=1$	$X=0$	$X=1$
a	a	b	0	0
b	c	d	0	0
c	a	d	0	0
d	e	d	0	1
e	a	d	0	1
~~d~~	~~e~~	~~d~~	~~0~~	~~1~~
~~e~~	~~a~~	~~d~~	~~0~~	~~1~~



Present state	Next state		Output	
	$X=0$	$X=1$	$X=0$	$X=1$
a	a	b	0	0
b	c	d	0	0
c	a	d	0	0
d	e	d	0	1
e	a	d	0	1

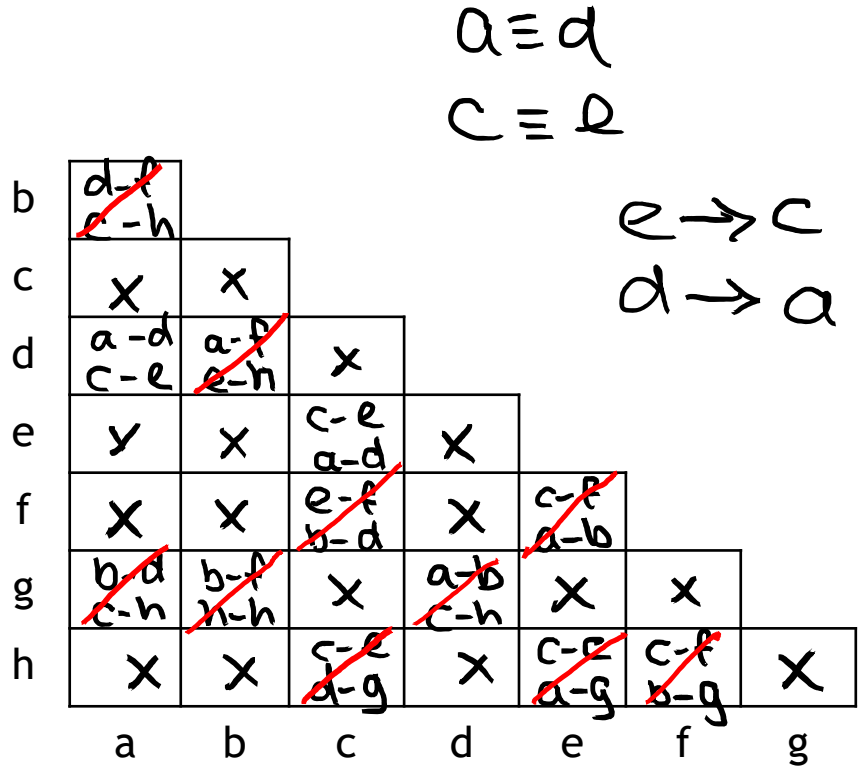
5

UNSW SYDNEY

Implication table problem

Moore model

Present state	Next state		Output
	$X=0$	$X=1$	
a	~~d~~ a	c	0
b	f	h	0
c	~~e~~ c	~~d~~ a	1
~~d~~	~~a~~	~~e~~ c	~~0~~
~~e~~	~~c~~	~~a~~	~~1~~
f	f	b	1
g	b	h	0
h	c	g	1



State assignment

Traffic light

Red 00
Yellow 01
Green 10

Necessary to assign unique binary code to the states to design a sequential circuit with physical components

For m states, codes must contain n bits, where $2^n \geq m$

Eight states can be assigned with three bits codes, binary numbers from 000 through 111

If the state table has seven states, binary numbers 000 to 110 can be used to assign the states; the remaining state is unused

For five states, five binary numbers can be used; the remaining three states are unused

The unused states are treated as "*don't care conditions*" during design

State assignment / coding

State	Assignment 1, Binary	Assignment 2, Gray Code	Assignment 3, One-Hot
a	000	000	00001
b	001	001	00010
c	010	011	00100
d	011	010	01000
e	100	100	10000

Assignment 1 is often used and easy to apply

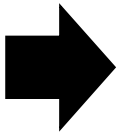
Assignment 2 makes it easier for Boolean function to be placed in the map for simplification

Assignment 3 provides faster machines and simpler decoding logic for the next state and output

State assignment example

Using binary code for state assignment, the reduced state table in the last example can be given as

Present state	Next state		Output	
	$X=0$	$X=1$	$X=0$	$X=1$
a	a	b	0	0
b	c	d	0	0
c	a	d	0	0
d	e	d	0	1
e	a	d	0	1



Present state	Next state		Output	
	$X=0$	$X=1$	$X=0$	$X=1$
000	000	001	0	0
001	010	011	0	0
010	000	011	0	0
011	100	011	0	1
100	000	011	0	1

Sequential circuit design

Procedure

1. **Specification:** Derive the state diagram from the word description and specifications of the desired operation.
2. **Formulation:** Obtain state table from the state diagram. Reduce the number of states.
3. **State Assignment:** Assign binary values to the states. Obtain the binary-coded state stable.
4. **Flip-Flop Selection:** Choose the type of flip-flops to be used D, JK, T
5. **Equation Determination:** Derive the simplified flip-flop input equations and output equations.
6. **Optimization:** Optimize the flip-flop input equations and output equations.
7. **Technology mapping:** Draw the logic diagram using flip-flops, ANDs, ORs, and inverters. Transform the logic diagram to the appropriate gate technology.
8. **Verification:** Verify the design implementation with simulation.

UNSW SYDNEY

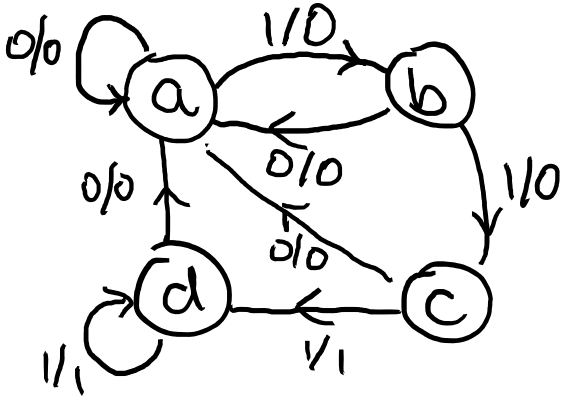
Sequential circuit design example

We need to design a circuit that detects a sequence of three or more consecutive 1's in a string of bits coming from an input line (i.e. the input is in serial bit stream)

X: 1 0 0 0 1 1 1 0 0 1 1 1 0 0 0
0 0 0 0 0 1 0 0 0 1 0 0 0

State diagram and state table

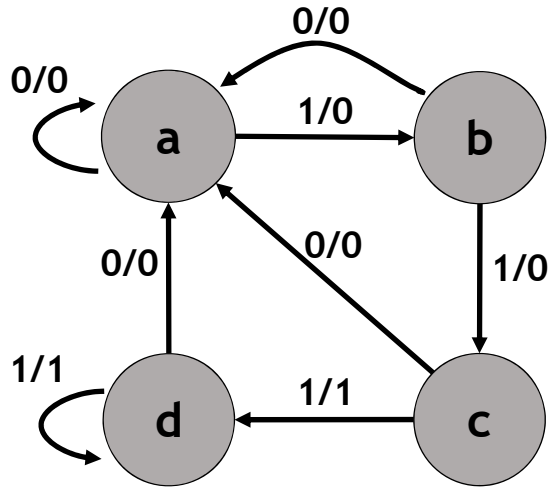
a - no 1's
b - one 1
c - two 1's
d - three 1's



Present state	Next state		Output	
	$X=0$	$X=1$	$X=0$	$X=1$
a	a	b	0	0
b	a	c	0	0
c	a	d	0	1
d	a	d	0	1

UNSW SYDNEY

Sequential circuit design example



Present state	Next state		Output	
	$X=0$	$X=1$	$X=0$	$X=1$
a	a	b	0	0
b	a	c	0	0
c	a	~~d~~ c	0	1
~~d~~ c	a	~~d~~ c	0	1

2. State reduction

c=d

d->c

	a	b	c
b	a-a b-c		
c	×	×	
d	×	×	✓

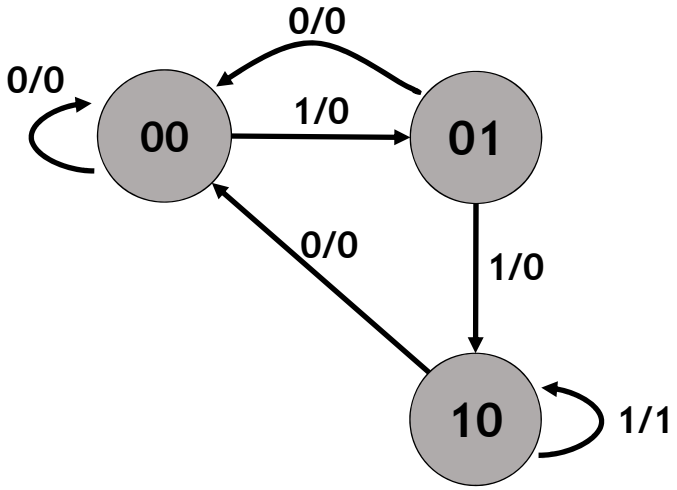
3. State assignment

	Binary	Grey	One-Hot
a	00	00	001
b	01	01	010
c	10	11	100

UNSW SYDNEY

Sequential circuit design example

Binary coded state diagram and table



Present state	Next state $Y(t+1)$ $Z(t+1)$		Output F	
	$X=0$	$X=1$	$X=0$	$X=1$
00	00	01	0	0
01	00	10	0	0
10	00	10	0	1

Synthesis using D flip-flops

The next state equation is the same as the D flip-flop input equation (taking flip flops Y and Z)

$$D_Y = Y(t+1) \text{ and } D_Z = Z(t+1)$$

UNSW SYDNEY

Sequential circuit design example

Present state	Next state		Output	
	$X=0$	$X=1$	$X=0$	$X=1$
00	00	01	0	0
01	00	10	0	0
10	00	10	0	1

Present state		Input	Next state		Output
			D_Y	D_Z	
Y	Z	X	$Y(t+1)$	$Z(t+1)$	F
0	0	0	0	0	0
0	0	1	0	1	0
0	1	0	0	0	0
0	1	1	1	0	0
1	0	0	0	0	0
1	0	1	1	0	1
1	1	0	x	x	x
1	1	1	x	x	x

$XY + XZ$

D_Y

Y \ ZX	00	01	11	10
0	0	0	1	0
1	0	1	X	X

$X\bar{Y}Z$

D_Z

Y \ ZX	00	01	11	10
0	0	1	0	0
1	0	0	X	X

F

Y \ ZX	00	01	11	10
0	0	0	0	0
1	0	1	X	X

XY

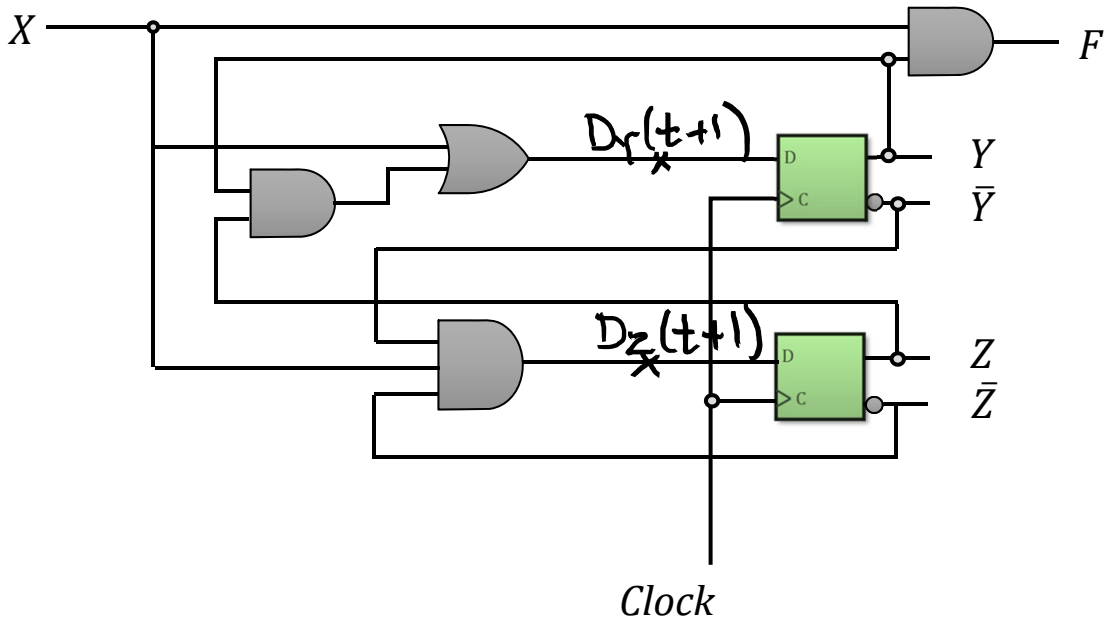
UNSW SYDNEY

Sequential circuit design example

Flip flop equations

$$\boldsymbol{D_Y} = XY + XZ$$
$$\boldsymbol{D_Z} = X\bar{Y}\bar{Z}$$
$$\boldsymbol{F} = XY$$

Logic diagram



UNSW SYDNEY

Excitation Tables

When D flip-flops are employed, the input equations are obtained directly from the next state

However, for JK and T flip-flops, the input equations need to be derived indirectly from the state table

An excitation table that lists the required inputs for a given change of state in state table is thus needed

Excitation table for JK flip-flop

Q(t)	Q(t+1)	J	K
0	0	0	X
0	1	1	X
1	0	X	1
1	1	X	0

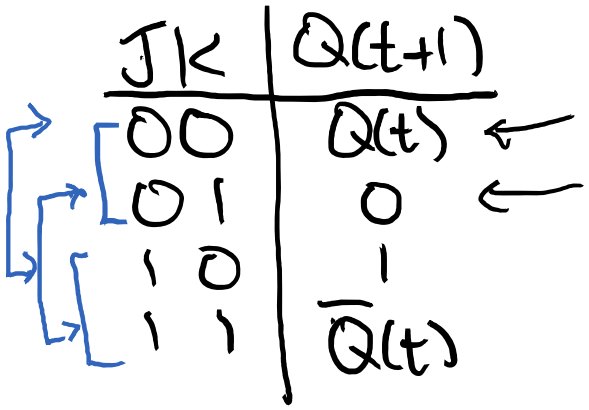
Excitation table for T flip-flop

Q(t)	Q(t+1)	T
0	0	0
0	1	1
1	0	1
1	1	0

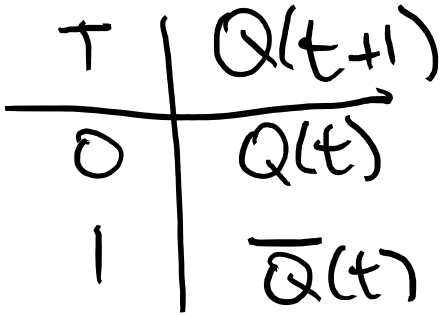
UNSW SYDNEY

Excitation - characteristic tables

Q(t)	Q(t+1)	J	K
0	0	0	X
0	1	1	X
1	0	X	1
1	1	X	0



Q(t)	Q(t+1)	T
0	0	0
0	1	1
1	0	1
1	1	0



UNSW SYDNEY

Synthesis using JK flip-flops

The excitation table is used to obtain the required inputs from the state table

For previous design problem:

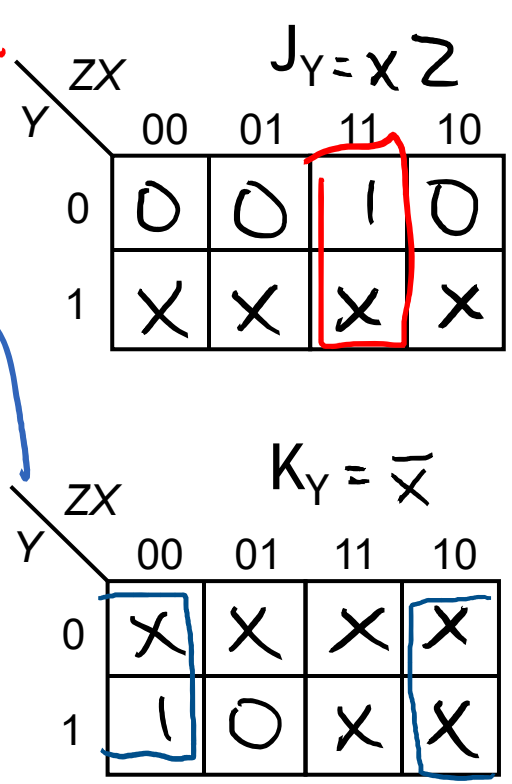
Present state		Input	Next state		Flip-Flop Inputs			
Y	Z	X	$Y(t+1)$	$Z(t+1)$	J_Y	K_Y	J_Z	K_Z
0	0	0	0	0	0	X	0	X
0	0	1	0	1	0	X	1	X
0	1	0	0	0	0	X	X	1
0	1	1	1	0	1	X	X	1
1	0	0	0	0	X	1	0	X
1	0	1	1	0	X	0	0	X
1	1	0	X	X	X	X	X	X
1	1	1	X	X	X	X	X	X

Q(t)	Q(t+1)	J	K
0	0	0	X
0	1	1	X
1	0	X	1
1	1	X	0

Synthesis using JK flip-flops

The required JK input equations can then be obtained from the state table

Present state		Input	Next state		Flip-Flop Inputs			
Y	Z	X	$Y(t+1)$	$Z(t+1)$	J_Y	K_Y	J_Z	K_Z
0	0	0	0	0	0	X	0	X
0	0	1	0	1	0	X	1	X
0	1	0	0	0	0	X	X	1
0	1	1	1	0	1	X	X	1
1	0	0	0	0	X	1	0	X
1	0	1	1	0	X	0	0	X
1	1	0	X	X	X	X	X	X
1	1	1	X	X	X	x	X	X



UNSW SYDNEY

Synthesis using JK flip-flops

The required JK input equations can then be obtained from the state table

Present state		Input	Next state		Flip-Flop Inputs			
Y	Z	X	$Y(t+1)$	$Z(t+1)$	J_Y	K_Y	J_Z	K_Z
0	0	0	0	0	0	X	0	X
0	0	1	0	1	0	X	1	X
0	1	0	0	0	0	X	X	1
0	1	1	1	0	1	X	X	1
1	0	0	0	0	X	1	0	X
1	0	1	1	0	X	0	0	X
1	1	0	X	X	X	X	X	X
1	1	1	X	X	X	x	X	X

$J_Z = X$

Y \ ZX	00	01	11	10
0	0	1	X	X
1	0	0	X	X

$K_Z = Z$

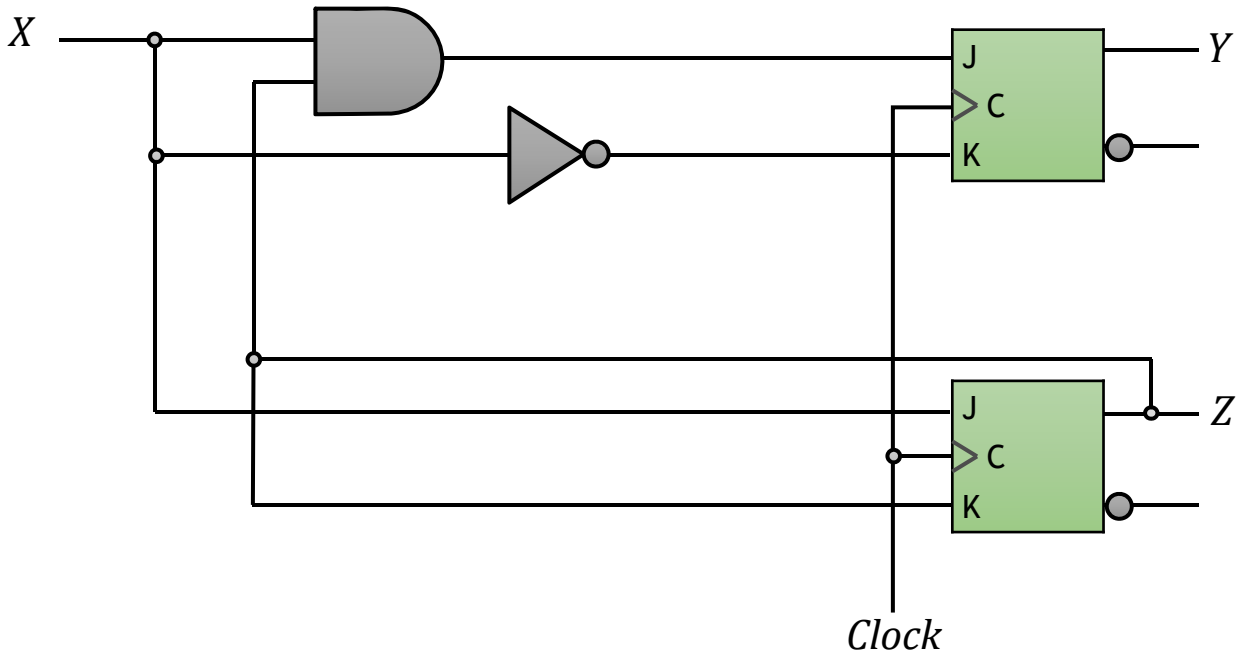
Y \ ZX	00	01	11	10
0	X	X	1	1
1	X	X	X	X

UNSW SYDNEY

Synthesis using JK flip-flops

$$
\begin{cases}
J_Y = XZ \\
K_Y = \bar{X} \\
J_Z = X \\
K_Z = Z
\end{cases}
$$

Logic diagram

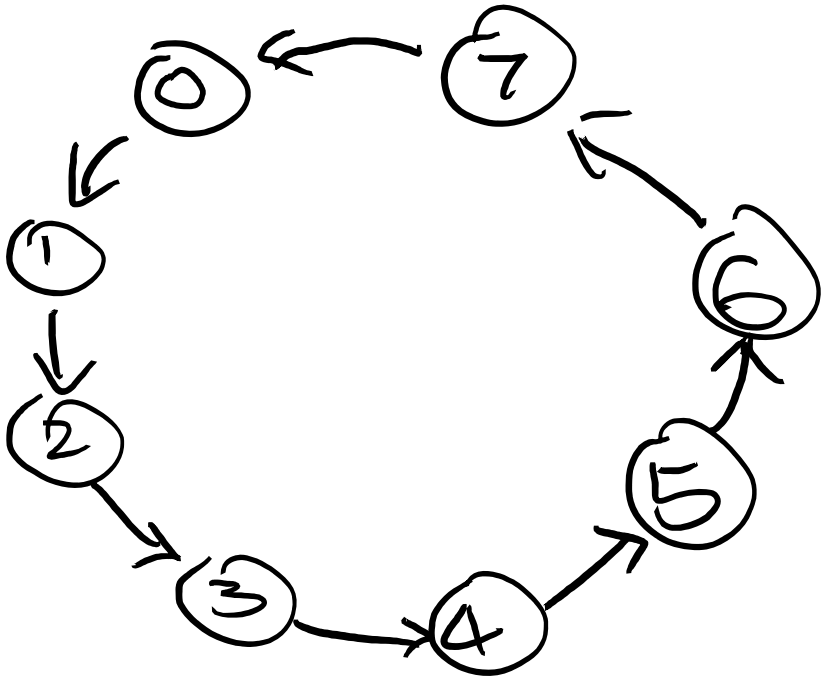


UNSW SYDNEY

Synthesis using T flip-flops

Design a three-bit counter consisting of three T flip-flops that can count in binary form from 0 to 7

State diagram:



State assgn.

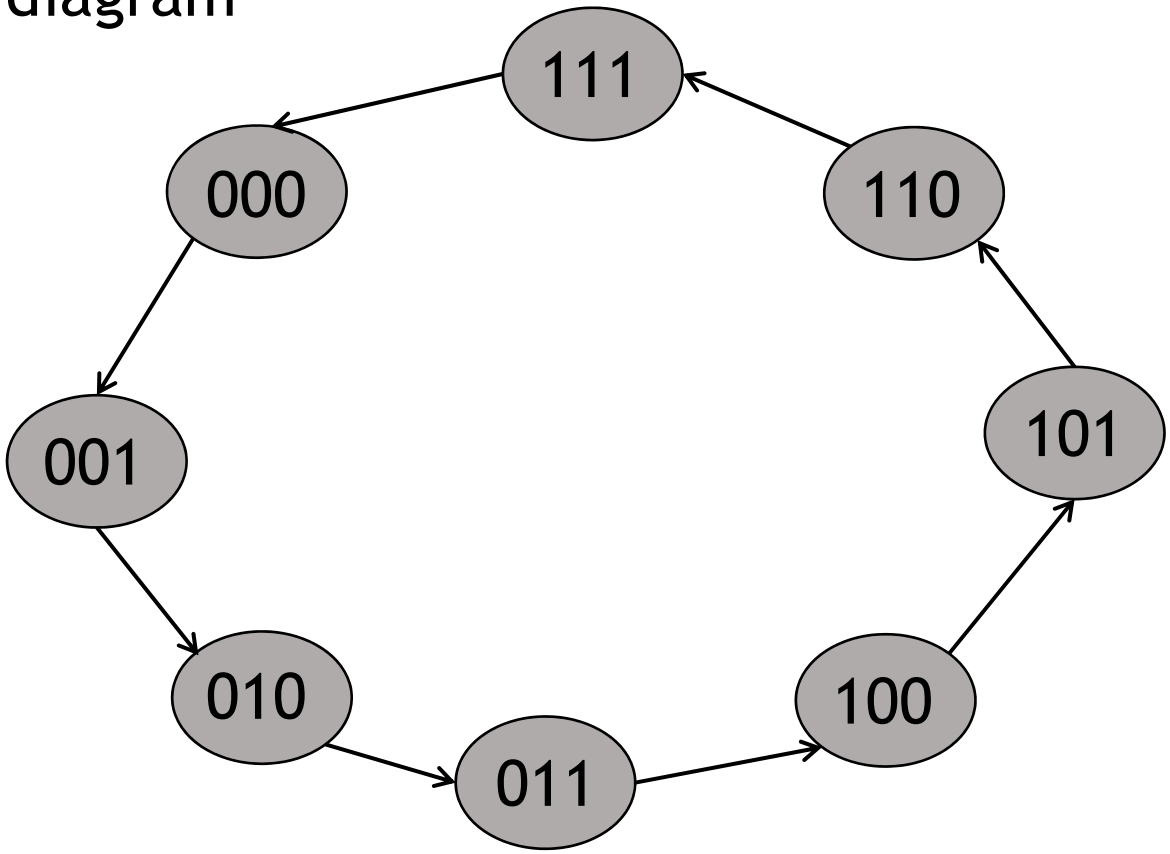
0	000
1	001
2	010
3	011
4	100
5	101
6	110
7	111

UNSW SYDNEY

Synthesis using T flip-flops

Design a three-bit counter consisting of three T flip-flops that can count in binary form from 0 to 7

State diagram



Synthesis using T flip-flops

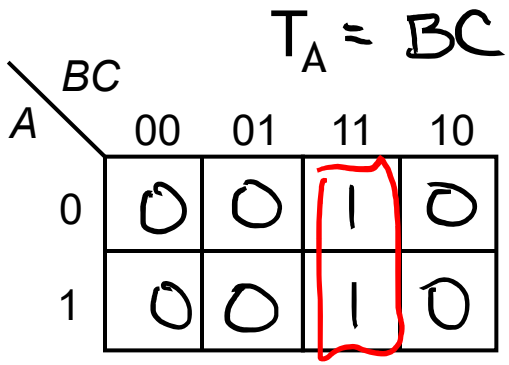
State Table

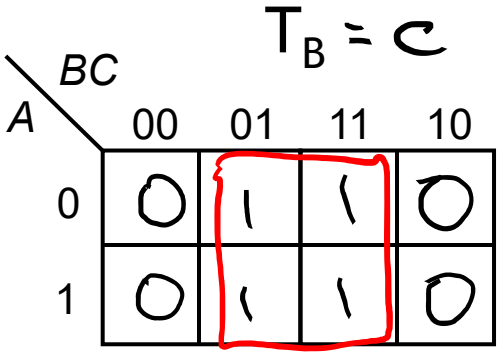
Present state			Next state			Flip-Flop Inputs		
A	B	C	$A(t+1)$	$B(t+1)$	$C(t+1)$	T_A	T_B	T_C
0	0	0	0	0	1	0	0	1
0	0	1	0	1	0	0	1	1
0	1	0	0	1	1	0	0	1
0	1	1	1	0	0	1	1	1
1	0	0	1	0	1	0	0	1
1	0	1	1	1	0	0	1	1
1	1	0	1	1	1	0	0	1
1	1	1	0	0	0	1	1	1

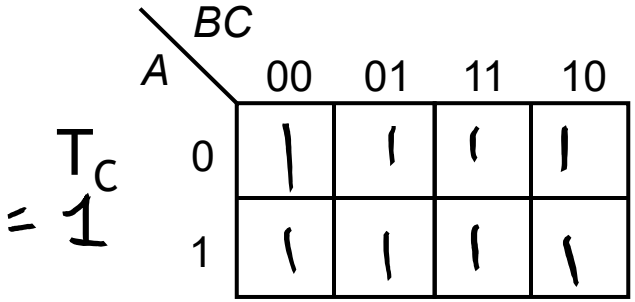
Q(t)	Q(t+1)	T
0	0	0
0	1	1
1	0	1
1	1	0

Synthesis using T flip-flops

Present state			Next state			Flip-Flop Inputs		
A	B	C	$A(t+1)$	$B(t+1)$	$C(t+1)$	T_A	T_B	T_C
0	0	0	0	0	1	0	0	1
0	0	1	0	1	0	0	1	1
0	1	0	0	1	1	0	0	1
0	1	1	1	0	0	1	1	1
1	0	0	1	0	1	0	0	1
1	0	1	1	1	0	0	1	1
1	1	0	1	1	1	0	0	1
1	1	1	0	0	0	1	1	1



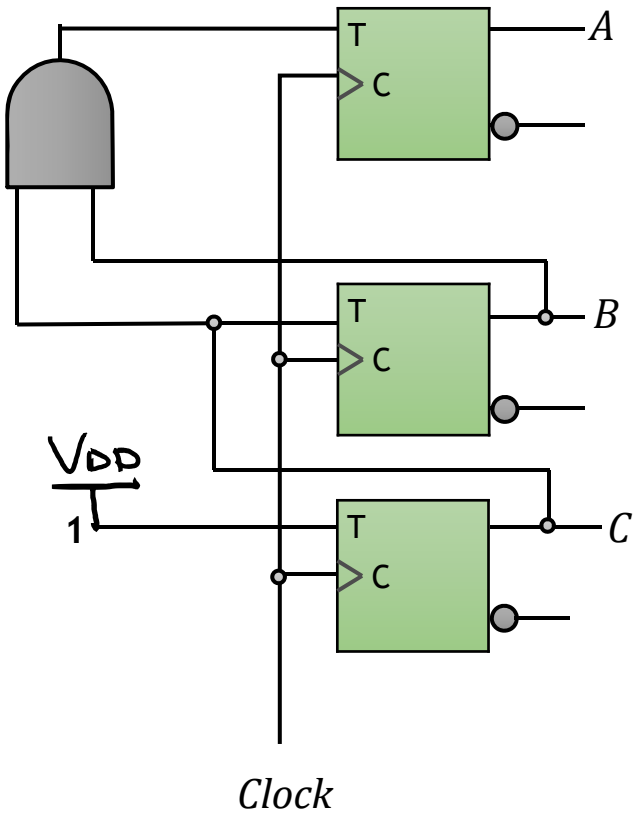




UNSW SYDNEY

Synthesis using T flip-flops

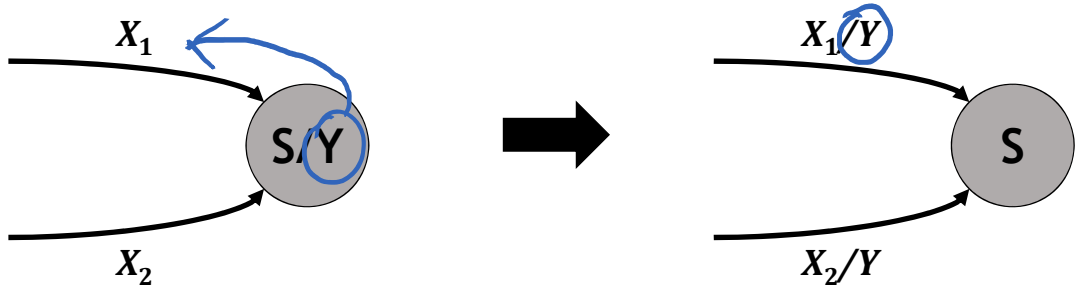
$$\begin{cases} T_A = BC \\ T_B = C \\ T_C = 1 \end{cases}$$



Moore to Mealy Conversion

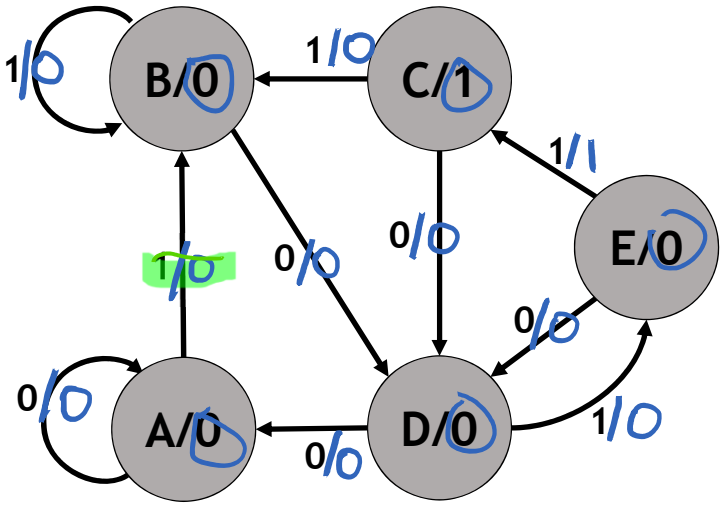
To convert a Moore machine to a Mealy one:

1. Pull the outputs back in all *incoming* transitions in the state diagram



2. Duplicate the outputs of any state to all occurrences of that state as a *next-state* in the state table

Moore to Mealy Example



Current State	Next State		Output Z
	$X=0$	$X=1$	
A	A	B	0
B	D	B	0
C	D	B	1
D	A	E	0
E	D	C	0

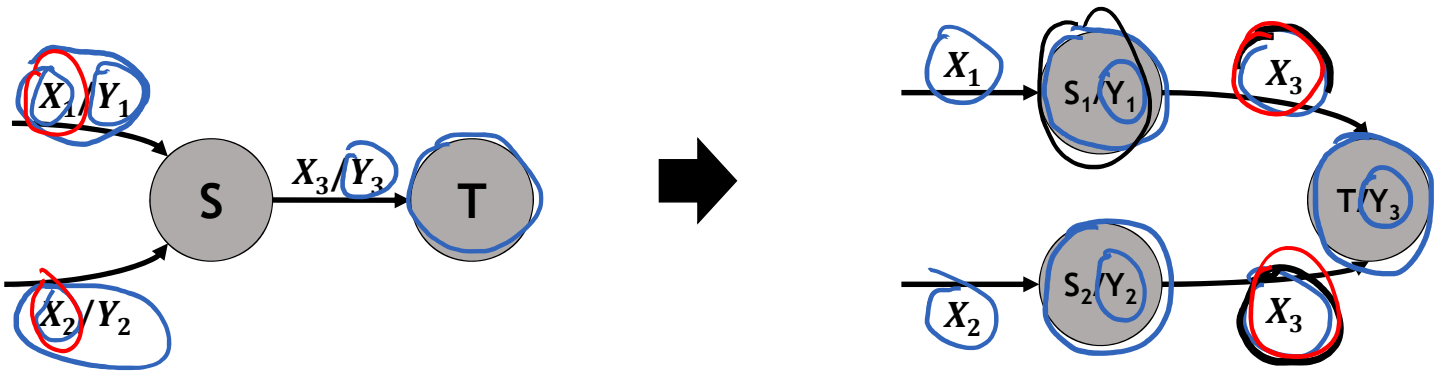
Current State	Next State, Output Z	
	$X=0$	$X=1$
A	A, 0	B, 0
B	D, 0	B, 0
C	D, 0	B, 0
D	A, 0	E, 0
E	D, 0	C, 1

UNSW SYDNEY

Mealy to Moore Conversion

To convert a Mealy machine to a Moore one:

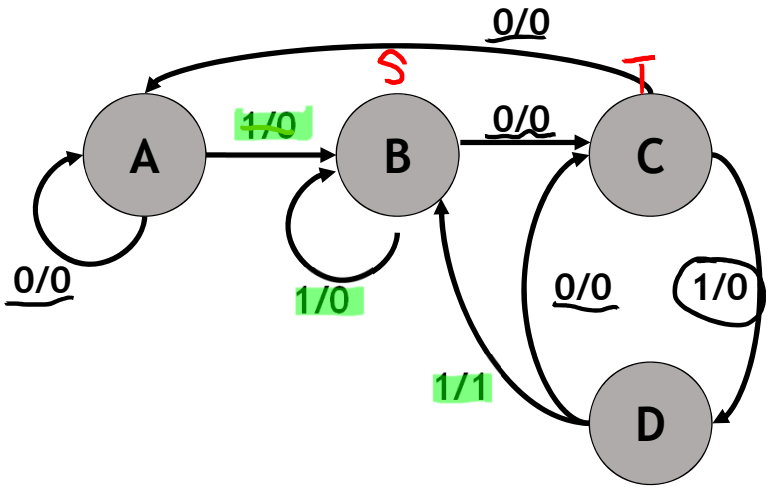
1. Split each state to the number of *different* outputs coming in, and push the outputs into the state in the state diagram

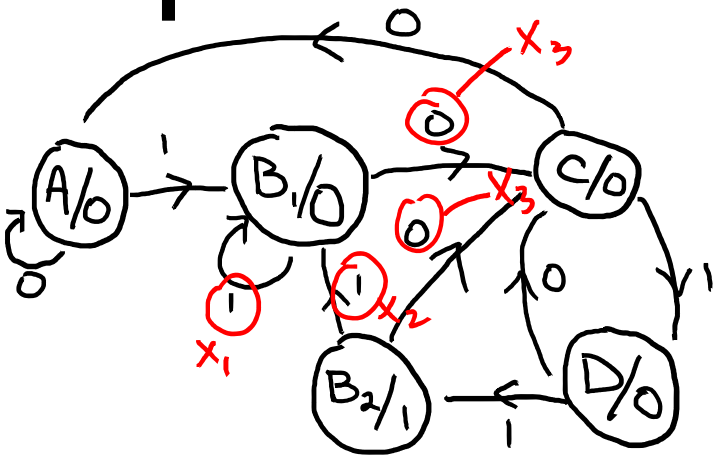


2. Create *new states* corresponding to all state-output combinations and fill in a new state table

UNSW SYDNEY

Mealy to Moore Example





Current State	Next State, Output Z	
	$X=0$	$X=1$
A	A, 0	B, 0
B	C, 0	B, 0
C	A, 0	D, 0
D	C, 0	B, 1

Current State	Next State		Output Z
	$X=0$	$X=1$	
A	A	B_1	0
B_1	C	B_1	0
C	A	D	0
D	C	B_2	0
B_2	C	B_1	1