EE103 Digital Systems 1

Wong Chin Hong

106

Maynooth International Engineering College
Fuzhou University

EE103 - Learning Outcomes

At the end of this module, you will be able to:

1. Conduct basic arithmetic with binary numbers.
2. Perform Boolean algebra.
3. Minimise logic using Karnaugh Maps.
4. Implement a logic circuit using only NAND / NOR gates.
5. Describe the operation of basic flip-flops.
6. Design a synchronous counter.
7. Distinguish between different programmable logic devices.

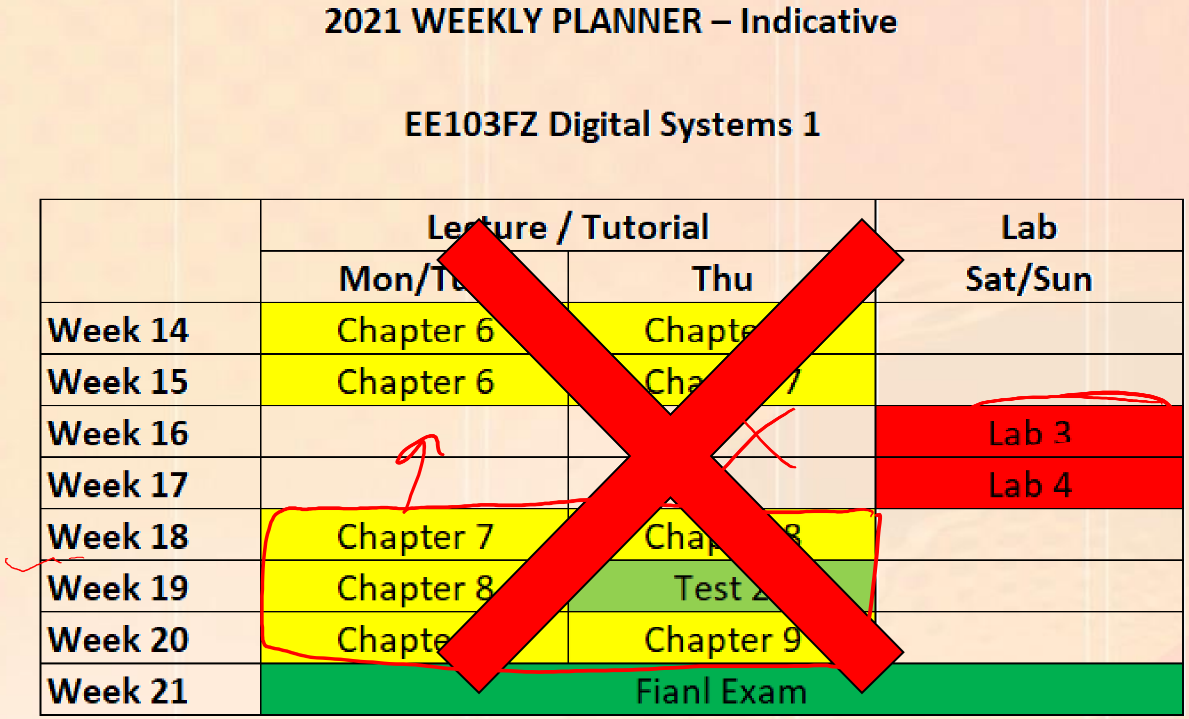
EE103 - Assessment Method

Assessment Type	Percentage	Week
Lab	10%	16 & 17
Tests x 2	20%	17 ~~18~~ Thursday.
Final Exam	70%	
	100%	

Textbook:

- Mano, M. Morris, Digital design, (2nd ed.), Prentice-Hall, 1991.

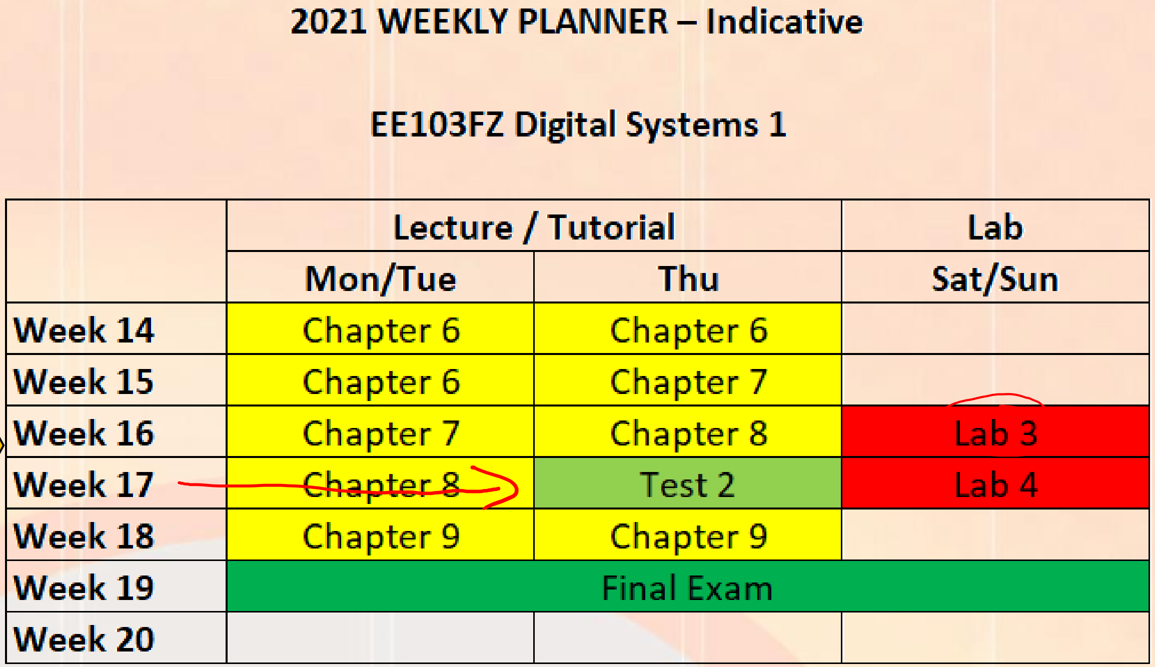
EE103 - Assessment Method



2021 WEEKLY PLANNER – Indicative

EE103FZ Digital Systems 1

	Lecture / Tutorial		Lab
	Mon/Tu	Thu	Sat/Sun
Week 14	Chapter 6	Chapte	
Week 15	Chapter 6	Cha 7	
Week 16			Lab 3
Week 17			Lab 4
Week 18	Chapter 7	Chap 8	
Week 19	Chapter 8	Test 2	
Week 20	Chapte	Chapter 9	
Week 21	Fianl Exam		



2021 WEEKLY PLANNER – Indicative

EE103FZ Digital Systems 1

	Lecture / Tutorial		Lab
	Mon/Tue	Thu	Sat/Sun
Week 14	Chapter 6	Chapter 6	
Week 15	Chapter 6	Chapter 7	
Week 16	Chapter 7	Chapter 8	Lab 3
Week 17	~~Chapter 8~~	Test 2	Lab 4
Week 18	Chapter 9	Chapter 9	
Week 19	Final Exam		
Week 20			

EE103 - Module Detail

- 22 – lectures, tutorials (13 completed)
- (2 x 1 hr) Class tests (1 completed)
- 12 hours of laboratories (4 x 3hr) … to start later in the semester (2 completed)
- 70% for final exam, 10% for labs & 20% for class tests.

EE103 – Some Advice

- Be punctual to class.
- Participate in class discussion.
- Pay attention when the lecturer is talking in front.
- No playing hand phone during class.

EE103 – Some Advice

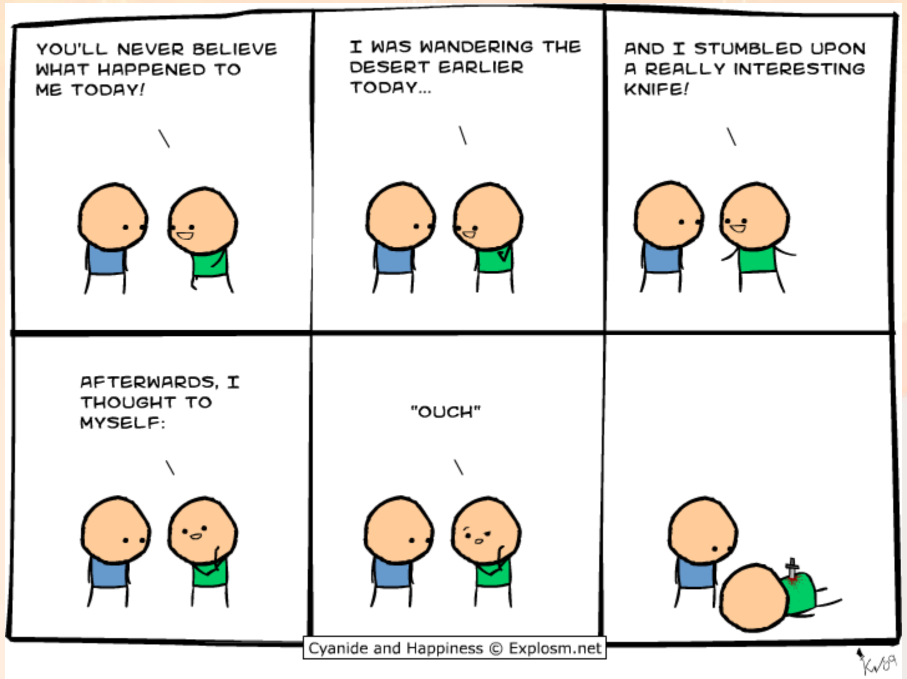
- Check your emails, Teams announcement and chat regularly
- Check Moodle (EE103 module)
- Ask plenty of questions – *before, during and after class!*
- Study hard, but have fun also ☺

So …

Let’s begin …

So far … !

- We’ve introduced the concept of Karnaugh Maps and used these to minimise logic …
- We’ve looked at implementing logic circuits using NAND only or NOR only gates …


YOU'LL NEVER BELIEVE WHAT HAPPENED TO ME TODAY!
I WAS WANDERING THE DESERT EARLIER TODAY...
AND I STUMBLED UPON A REALLY INTERESTING KNIFE!
AFTERWARDS, I THOUGHT TO MYSELF:
"OUCH"
Cyanide and Happiness © Explosm.net

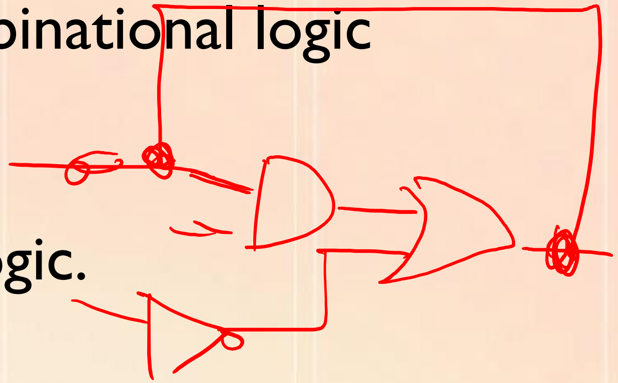
NOW, we are going to introduce Sequential Logic and look at Flipflops …

Sequential Logic

- **Combinational logic** is where the output of a digital circuit is produced by combining the inputs in a manner defined by the logic.

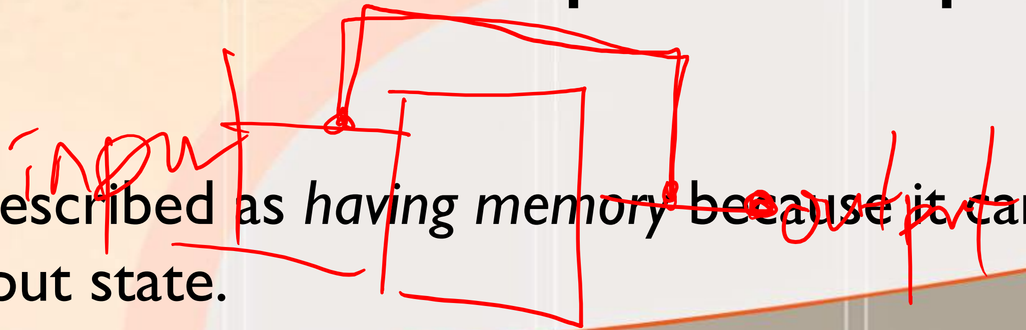
- Until now, all our circuits have been based on combinational logic only.

- However many digital systems contain sequential logic.



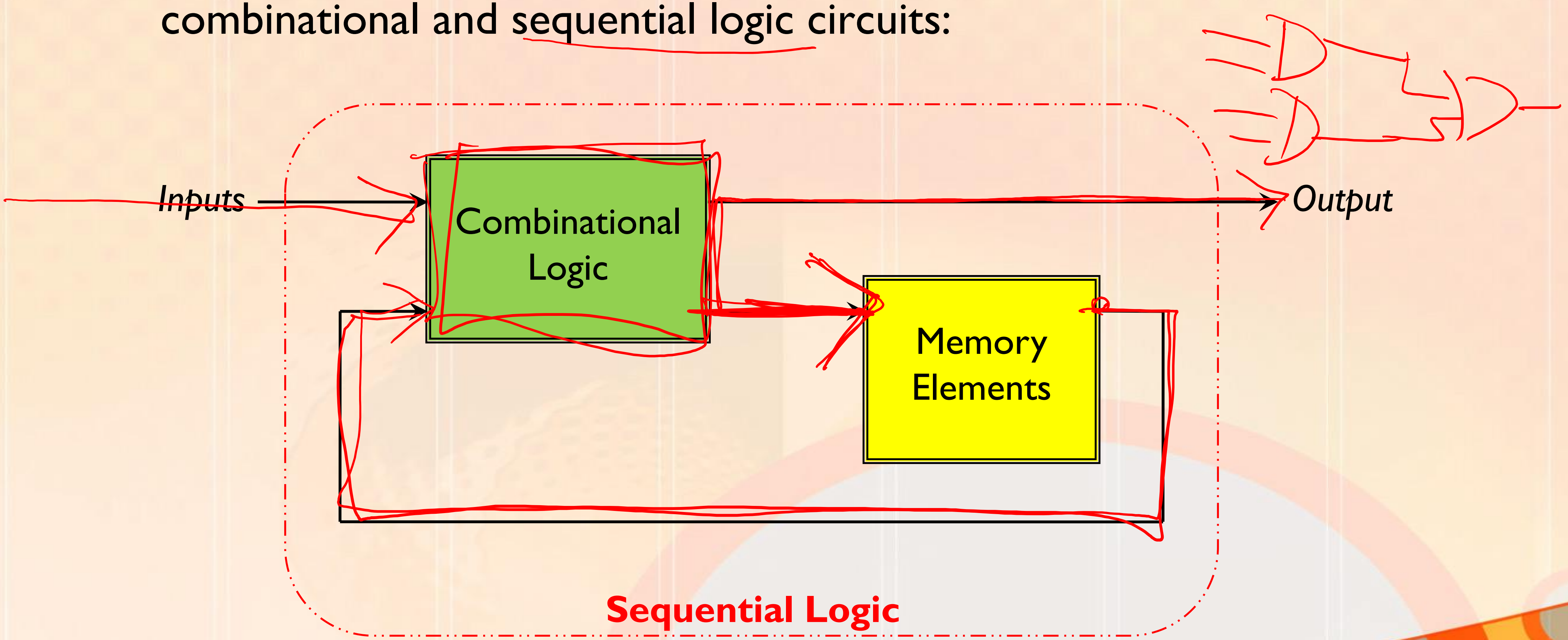
- **Sequential logic** is where the **output** of a digital circuit **depends on both the inputs and also on the previous output state** of the circuit.

- Such a system is described as *having memory* because it can remember its output state.



Sequential Logic

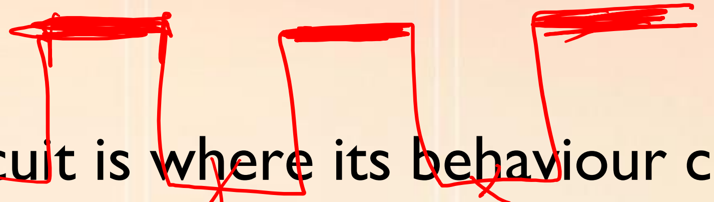
- The following diagram illustrates the fundamental difference between combinational and sequential logic circuits:



Sequential Logic

- Sequential systems are categorised in two ways, namely synchronous and asynchronous.

- A **synchronous** sequential circuit is where its behaviour depends on knowledge of the inputs **at a discrete instant of time,** usually determined by a **clock**.



- An **asynchronous** sequential circuit is where its behaviour can be affected by changes in the input signals at **any instant of time**.

always
~~avail~~ output

- One of the most basic but fundamental sequential circuit is the flip-flop, which is a bistable device, i.e. it settles in one of two possible states (typically SET and RESET).

0 → 0
↳ 1 → 0
0/1 set → 1
0/1 → 0
0 → 1
1 → 1

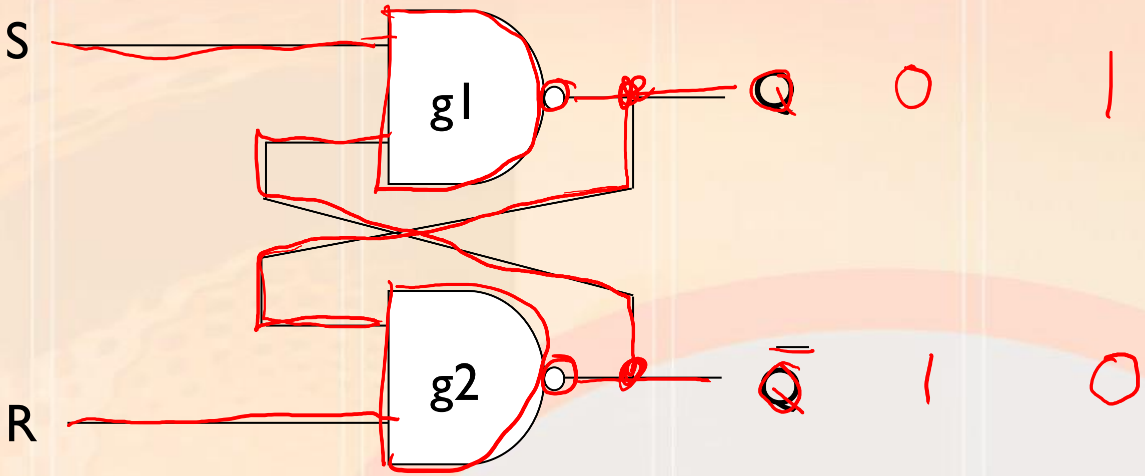
Sequential Logic

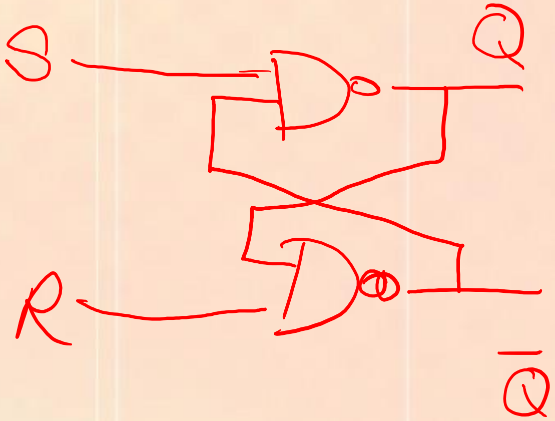
- We are now going to look at four common types of flipflops, namely SR, D, JK and T.

Asynchronous SR Flipflop

- The following circuit represents a NAND implementation of an **asynchronous SR flipflop**:

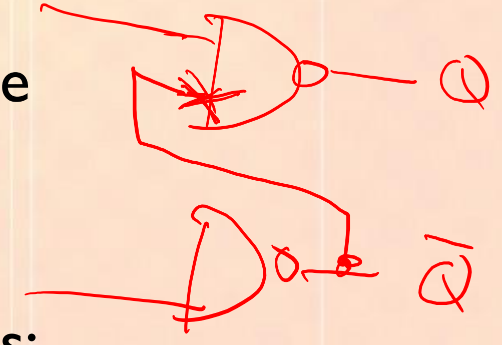
X CLK





Asynchronous SR Flipflop

- The circuit has two inputs S and R and two outputs Q and $\overline{Q}$. Note how the outputs are both fed back to act as inputs to the NAND gates.



- Recall, once again, the truth table of the NAND gate as follows:

A	B	f
0	0	1
0	1	1
1	0	1
1	1	0

B
$\overline{B}$

A	f	
0	1	Disabled
1	$\overline{B}$	Enabled

Asynchronous SR Flipflop

tau.

NAND

- We can work out the **next output $Q(\tau)$** of the SR circuit for all combinations of the inputs S, R and **Q** (**the previous output**).

current Q

Next $Q(\tau)$

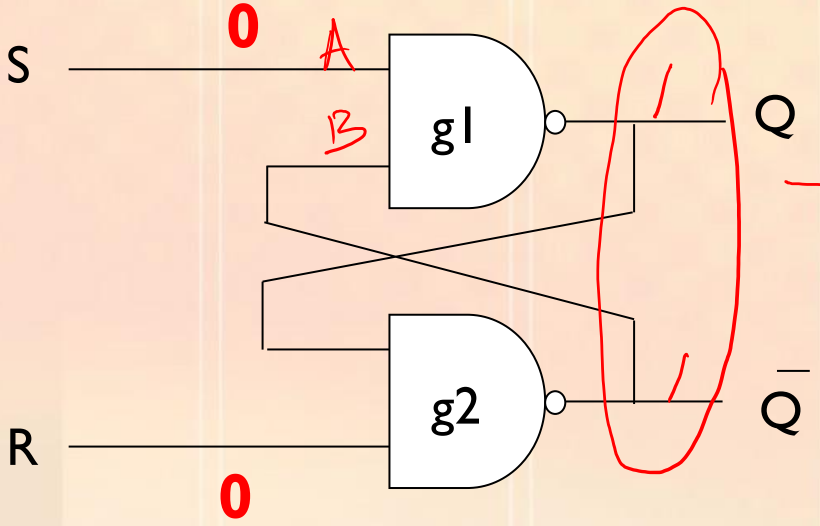
output.

- This is similar to the truth table approach for combinational logic circuits, but we now refer to the table as a **state table**.

- Let's examine the circuit in relation to each combination of inputs in turn …

Asynchronous SR Flipflop

- Let **S = 0** and **R = 0** …



$Q \neq \bar{Q}$

1 → 0

0 → 1

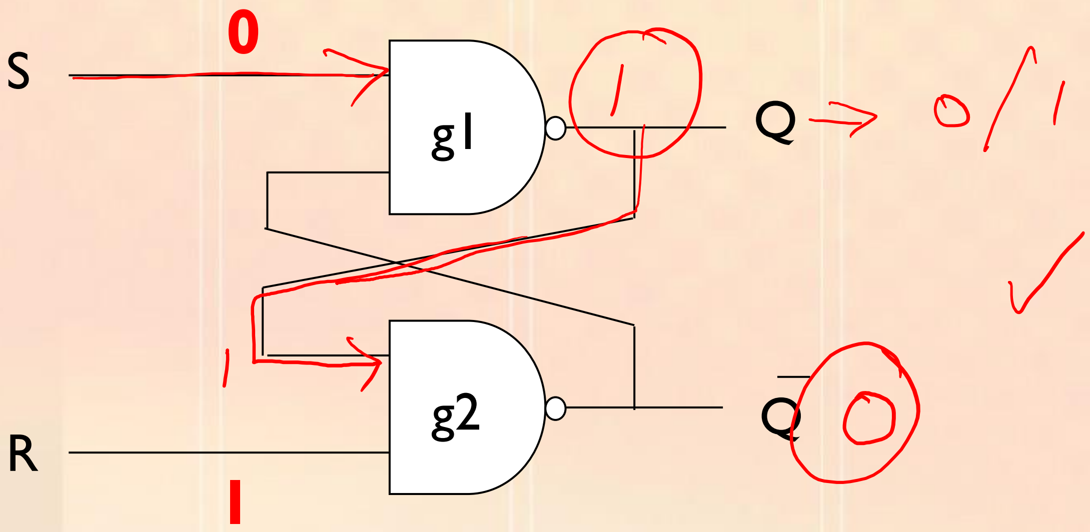
→ 1 ~~→~~ 1

0 ~~→~~ 0

A	B	NAND
0	0	1
0	1	1
1	0	1
1	1	0

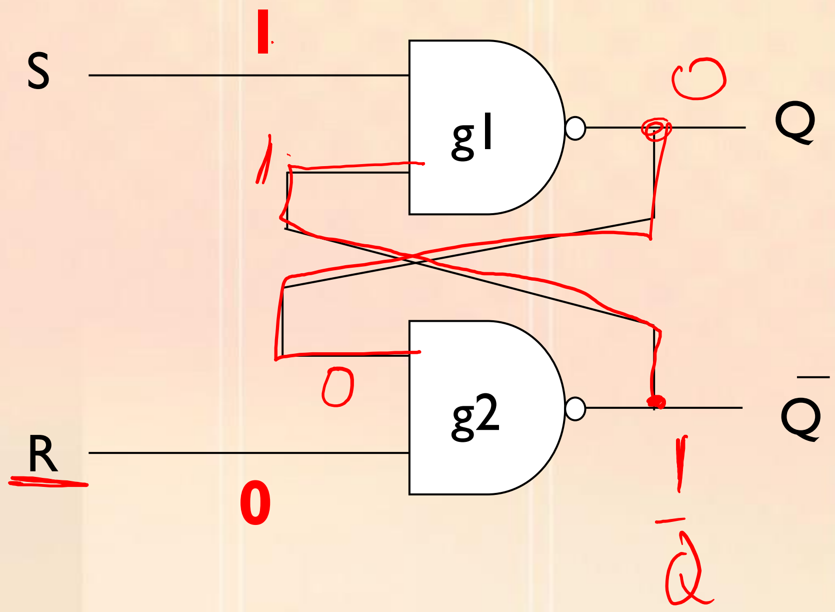
Asynchronous SR Flipflop

- Let **S = 0** and **R = 1** …



Asynchronous SR Flipflop

- Let **S = 1** and **R = 0** …



0	0	1
0	1	1
1	0	1
1	1	$\boxed{0}$

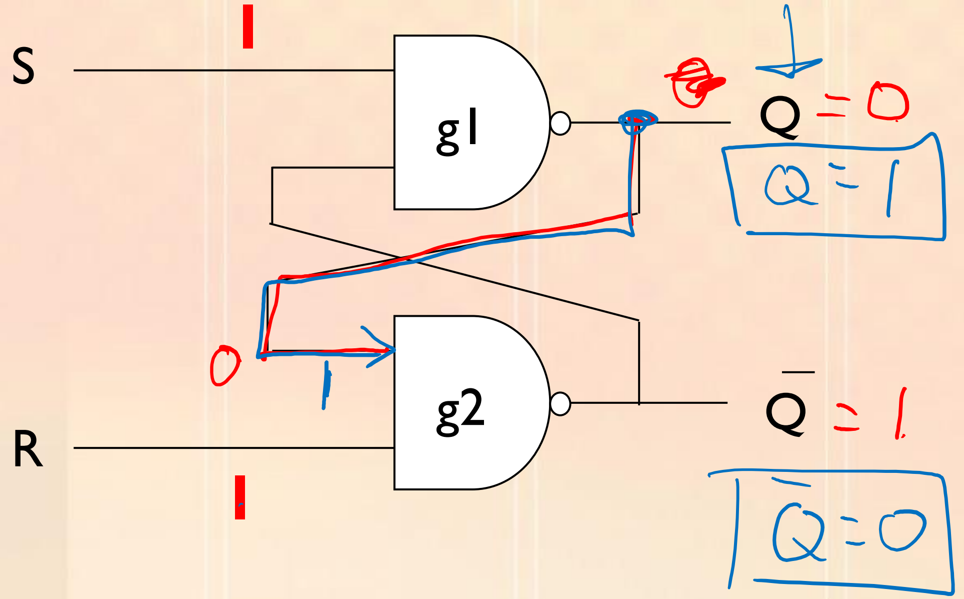
1

Q(t)

Asynchronous SR Flipflop

- Let **S = 1** and **R = 1** …

set Q → 0/1

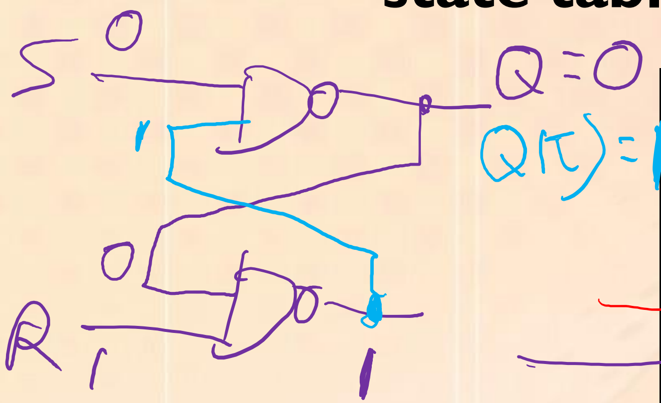


0	0	1
0	1	1
1	0	1
1	1	0

Asynchronous SR Flipflop

- Thus, by taking each combination of the inputs, and working through the circuit, as we have done, we can derive the following **state table**:

$\overline{Q}$



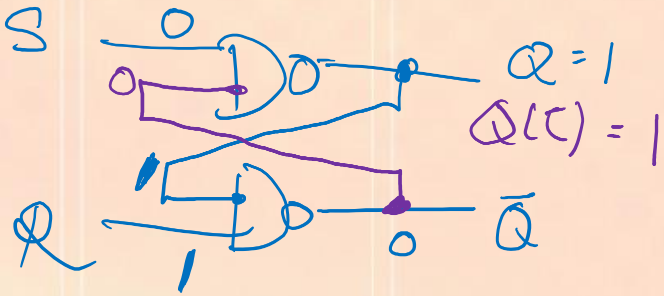
S	R	Q	Q(τ)
0	0	0	-
0	0	1	-
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

set

reset

stay

stay



Asynchronous SR Flipflop

- Thus, by taking each combination of the inputs, and working through the circuit, as we have done, we can derive the following **state table**:

S	R	Q	Q(τ)
0	0	0	-
0	0	1	-
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

Undefined

Q is Set to 1

Q is Reset to 0

Q stay

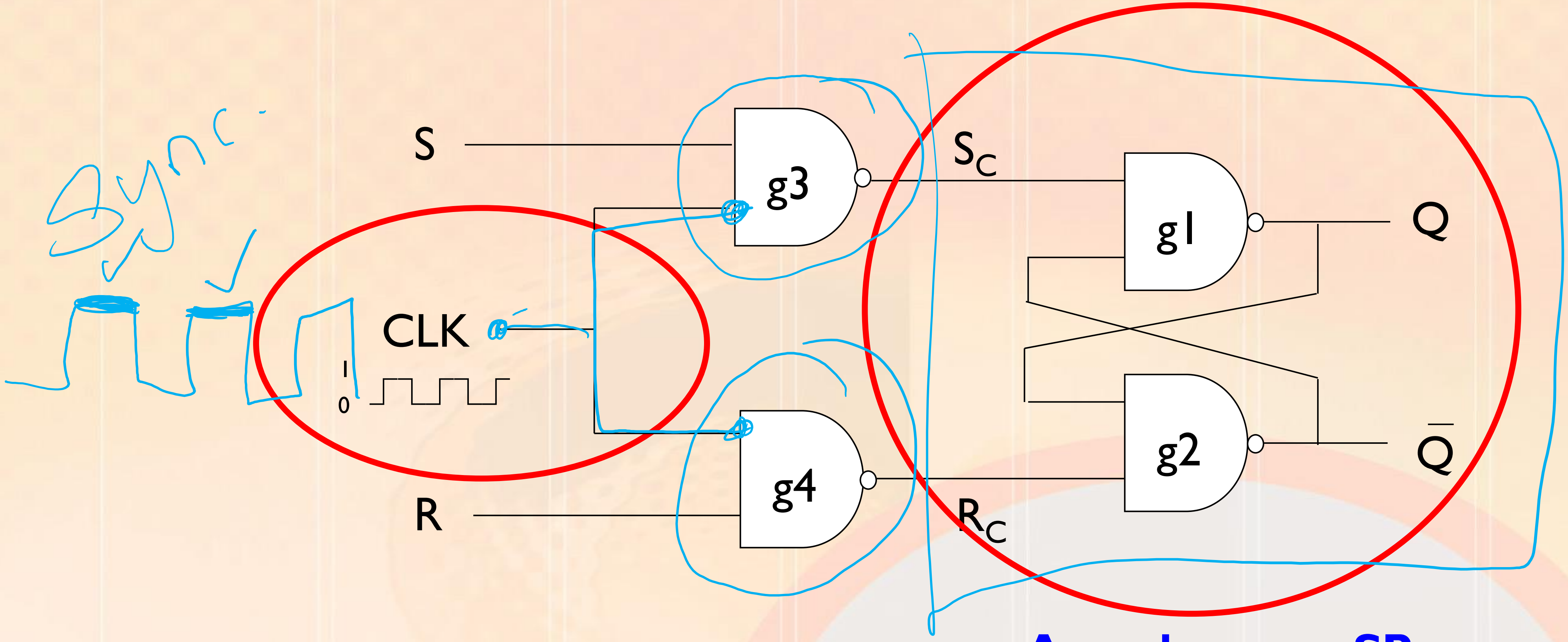
Synchronous SR Flipflop

- As it stands, the previous design will react to change in the input **at any instant** of time, i.e. it is asynchronous. X Time X CLK

✓ Time ✓ CLK

- A **synchronous** version of the flipflop (i.e. one controlled by a clock) can be achieved through the addition of two NAND gates as follows:

Synchronous SR Flipflop



CLOCK Signal

Asynchronous SR

Synchronous SR Flipflop

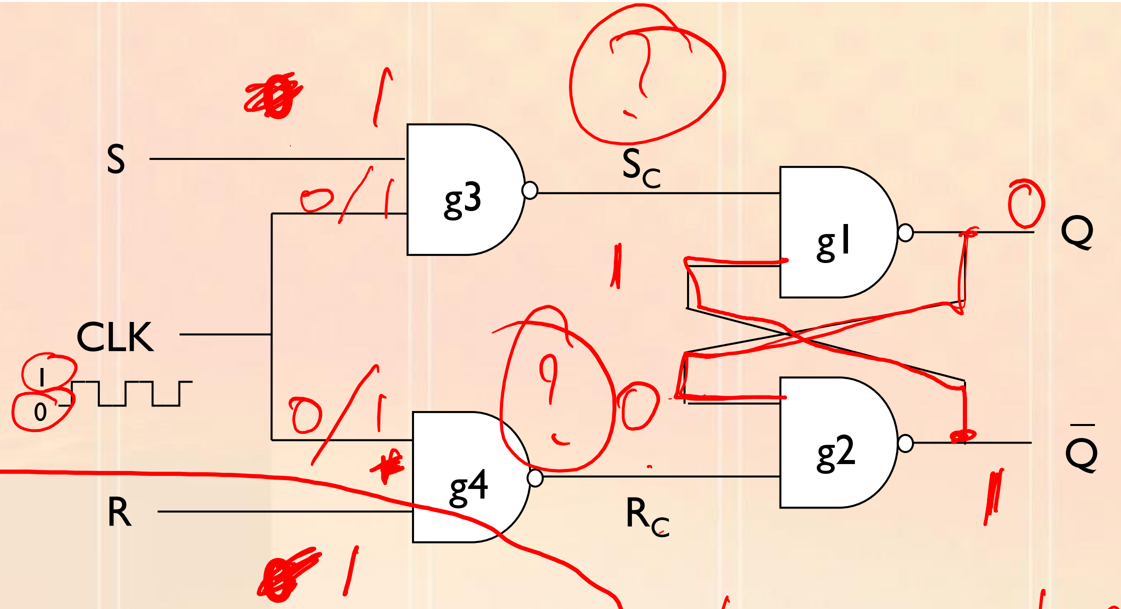
- Now, let us examine the operation of this circuit …

Synchronous SR Flipflop

Async.

- If **S = 0** and **R = 0** …

Sc	Rc	Q	Q(t)
0	0	0	—
		1	—
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1



Sync.

A	B	
0	0	
0	1	1
1	0	1
1	1	0

S	R	Q	Q(t)	S	R	Q	Q(t)
0	0	0	0	1	0	0	1
		1	1			1	1
0	1	0	0	1	1	0	—
		1	0			1	—

Synchronous SR Flipflop

- Hence, we can derive the following **state table for the synchronous SR flipflop:**

Async

S	R	Q	Q(τ)
0	0	0	-
		1	-
0	1	0	0
		1	0
1	0	0	1
		1	1
1	1	0	0
		1	1

Sync.

S	R	Q	Q(τ)
0	0	0	**0**
0	0	1	**1**
0	1	0	**0**
0	1	1	**0**
1	0	0	**1**
1	0	1	**1**
1	1	0	**-**
1	1	1	**-**

Q

Q is Reset to 0

Q is Set to 1

Undefined

Synchronous SR Flipflop

- For convenience, we can also obtain a **concise form of the state table** as follows:

S	R	Q	Q(τ)
0	0	0	**0**
0	0	1	**1**
0	1	0	**0**
0	1	1	**0**
1	0	0	**1**
1	0	1	**1**
1	1	0	**-**
1	1	1	**-**

undefine

S	R	Q(τ)
0	0	**Q**
0	1	**0**
1	0	**1**
1	1	**-**

undefine.

Synchronous SR Flipflop

- The flipflop is SET by S = 1, R = 0 and is RESET by S = 0, R = 1 and hence the name SR flipflop.

- Note that the output values are only set when the clock is enabled, i.e. $\boxed{\text{CLK = 1.}}$

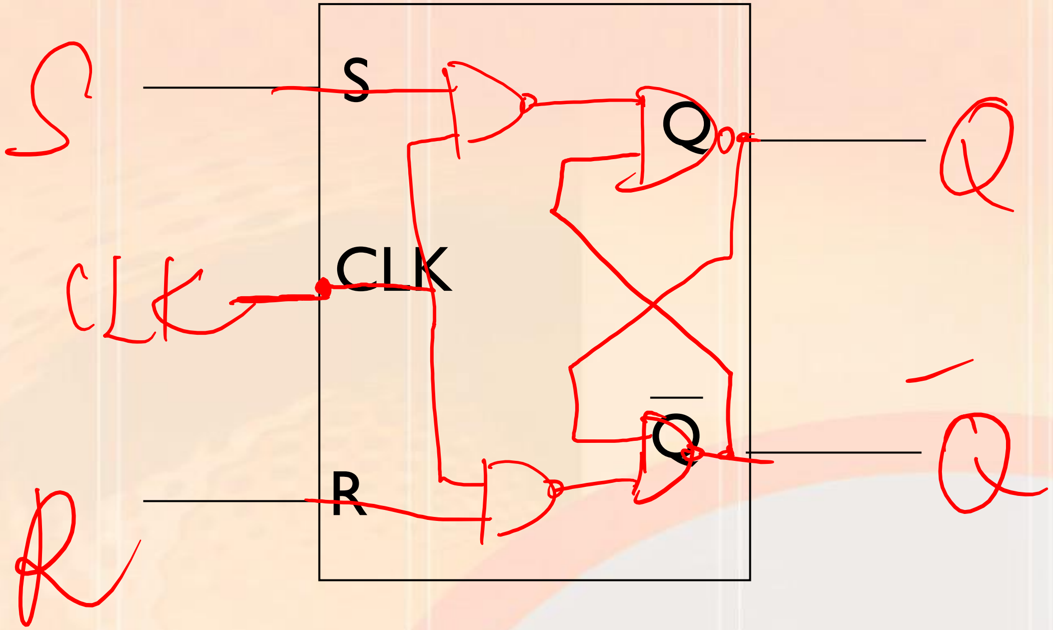
Ignore

- When the clock is disabled (i.e. $\boxed{\text{CLK = 0}}$), the S_C and R_C are set to 1, thereby maintaining the previous output values. Any change in S and R will therefore not affect the output.

- The circuit is now synchronous.

Synchronous SR Flipflop

- The **standard symbol** for the flipflop is:



Synchronous SR Flipflop

→ 0 → 0
→ 0 → 1
→ 1 → 1
→ 1 → 0
4

- An alternative means for expressing the behaviour of the flipflop is in terms of a **requirements table**.

- This particular table is useful when we consider the design of counters later in this module.

$Q \rightarrow Q(\tau)$

- The requirements table basically defines the possible transitions between states. As we have only 2 possible states (or outputs), we thus have 4 possible transitions as follows:

GO from 0 to 1
GO from 1 to 0
STAY at 0
STAY at 1

Synchronous SR Flipflop

- By carefully analysing the state table, we can derive the following requirements table for the SR flipflop, where X represents don’t care terms:

S	R	Q	Q(τ)
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	-
1	1	1	-

Operation	S	R
Stay at 0	0	X
Stay at 1	X	0
Go to 0	0	1
Go to 1	1	0