Digital Electronics: Sequential Logic

Introduction, Latches and Flip-Flops

Introduction

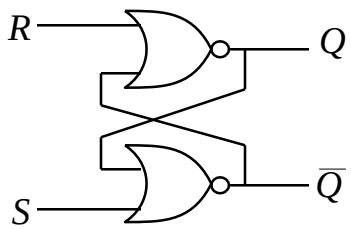
- The logic circuits discussed previously are known as *combinational*, in that the output depends only on the condition of the latest inputs
- However, we will now introduce a type of logic where the output depends not only on the latest inputs, but also on the condition of earlier inputs. These circuits are known as *sequential*, and implicitly they contain *memory* elements

Memory Elements

- A memory stores data – usually one bit per element
- A snapshot of the memory is called the *state*
- A one bit memory is often called a *bistable*, i.e., it has 2 stable internal states
- *Flip-flops* and *latches* are particular implementations of bistables

RS Latch

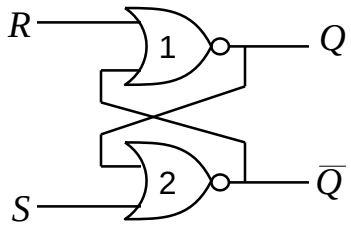
- An RS latch is a memory element with 2 inputs: Reset (R) and Set (S) and 2 outputs: Q and $\overline{Q}$.



S R	Q'	$\overline{Q}'$	comment
0 0	Q	$\overline{Q}$	hold
0 1	0	1	reset
1 0	1	0	set
1 1	0	0	illegal

Where Q' is the next state and Q is the current state

RS Latch - Operation

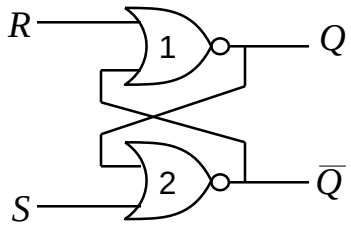


NOR truth table

a	b	y	
0	0	1	b complemented
0	1	0	
1	0	0	always 0
1	1	0	

- R = 1 and S = 0
  - Gate 1 output in 'always 0' condition, $Q = 0$
  - Gate 2 in 'complement' condition, so $\overline{Q} = 1$
- This is the (R)eset condition

RS Latch - Operation

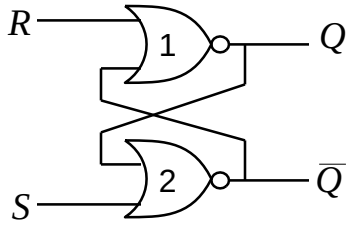


NOR truth table

a	b	y	
0	0	1	b complemented
0	1	0	
1	0	0	always 0
1	1	0	

- S = 0 and R to 0
  - Gate 2 remains in 'complement' condition, $\overline{Q} = 1$
  - Gate 1 into 'complement' condition, $Q = 0$
- This is the hold condition

RS Latch - Operation

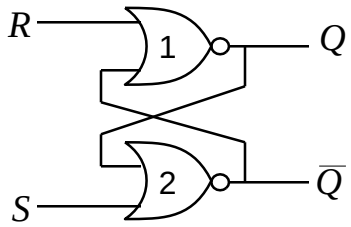


NOR truth table

a	b	y	
0	0	1	b complemented
0	1	0	
1	0	0	always 0
1	1	0	

- S = 1 and R = 0
  - Gate 1 into 'complement' condition, $Q = 1$
  - Gate 2 in 'always 0' condition, $\overline{Q} = 0$
- This is the (S)et condition

RS Latch - Operation



NOR truth table

a	b	y	
0	0	1	b complemented
0	1	0	
1	0	0	always 0
1	1	0	

- S = 1 and R = 1
  - Gate 1 in 'always 0' condition, $Q = 0$
  - Gate 2 in 'always 0' condition, $\overline{Q} = 0$
- This is the illegal condition

RS Latch – State Transition Table

- A *state transition table* is an alternative way of viewing its operation

Q	S	R	Q'	comment
0	0	0	0	hold
0	0	1	0	reset
0	1	0	1	set
0	1	1	0	illegal
1	0	0	1	hold
1	0	1	0	reset
1	1	0	1	set
1	1	1	0	illegal

- A state transition table can also be expressed in the form of a *state diagram*

RS Latch – State Diagram

- A *state diagram* in this case has 2 states, i.e., Q=0 and Q=1
- The state diagram shows the input conditions required to transition between states. In this case we see that there are 4 possible transitions
- We will consider them in turn

RS Latch – State Diagram

Q S R	Q'	comment
0 0 0	0	hold
0 0 1	0	reset
0 1 0	1	set
0 1 1	0	illegal
1 0 0	1	hold
1 0 1	0	reset
1 1 0	1	set
1 1 1	0	illegal

$Q = 0 \quad Q' = 0$

From the table we can see:

$\bar{S}.\bar{R} + \bar{S}.R + S.R =$

$\bar{S}.(\bar{R} + R) + S.R = \bar{S} + S.R =$

$(\bar{S} + S).(\bar{S} + R) = \bar{S} + R$

$Q = 1 \quad Q' = 1$

From the table we can see:

$\bar{S}.\bar{R} + S.\bar{R} = \bar{R}.(\bar{S} + S) =$

$\bar{R}$

RS Latch – State Diagram

Q S R	Q'	comment
0 0 0	0	hold
0 0 1	0	reset
0 1 0	1	set
0 1 1	0	illegal
1 0 0	1	hold
1 0 1	0	reset
1 1 0	1	set
1 1 1	0	illegal

$Q = 1 \quad Q' = 0$

From the table we can see:

$\bar{S}.R + S.R =$

$R.(\bar{S} + S) = R$

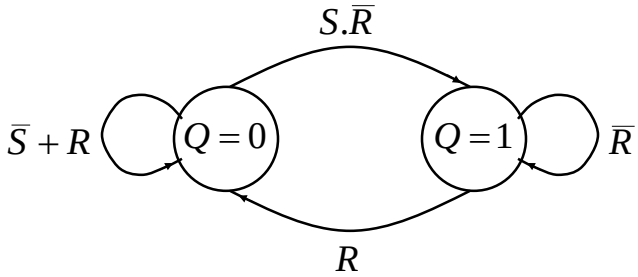
$Q = 0 \quad Q' = 1$

From the table we can see:

$S.\bar{R}$

RS Latch – State Diagram

- Which gives the following state diagram:



- A similar diagram can be constructed for the $\overline{Q}$ output
- We will see later that state diagrams are a useful tool for designing sequential systems

Clocks and Synchronous Circuits

- For the RS latch we have just described, we can see that the output state changes occur directly in response to changes in the inputs. This is called *asynchronous* operation
- However, virtually all sequential circuits currently employ the notion of *synchronous* operation, that is, the output of a sequential circuit is constrained to change only at a time specified by a global *enabling* signal. This signal is generally known as the system *clock*

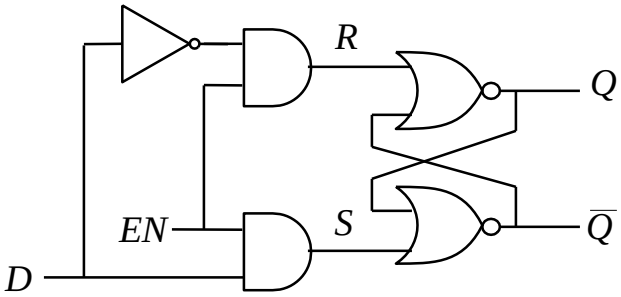
Clocks and Synchronous Circuits

- The Clock: What is it and what is it for?
  - Typically it is a square wave signal at a particular frequency
  - It imposes order on the state changes
  - Allows lots of states to appear to update simultaneously
- How can we modify an asynchronous circuit to act synchronously, i.e., in synchronism with a clock signal?

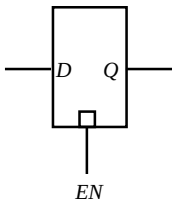
Transparent D Latch

- We now modify the RS Latch such that its output state is only permitted to change when a valid enable signal (which could be the system clock) is present
- This is achieved by introducing a couple of AND gates in cascade with the R and S inputs that are controlled by an additional input known as the *enable* (EN) input.

Transparent D Latch



Symbol

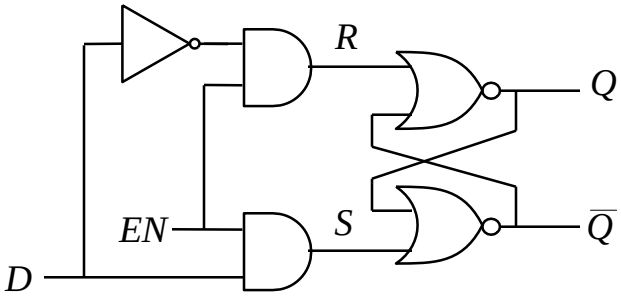


- See from the AND truth table:
  - if one of the inputs, say a is 0, the output is always 0
  - Output follows b input if a is 1
- The complement function ensures that R and S can never be 1 at the same time, i.e., illegal avoided

AND truth table

a	b	y
0	0	0
0	1	0
1	0	0
1	1	1

Transparent D Latch



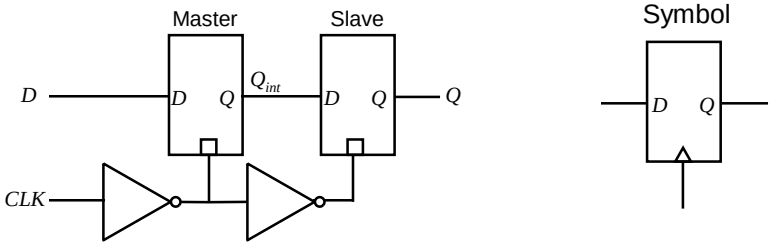
D	EN	Q'	$\overline{Q}'$	comment
X	0	Q	$\overline{Q}$	RS hold
0	1	0	1	RS reset
1	1	1	0	RS set

- See Q follows D input provided EN=1. If EN=0, Q maintains previous state

Master-Slave Flip-Flops

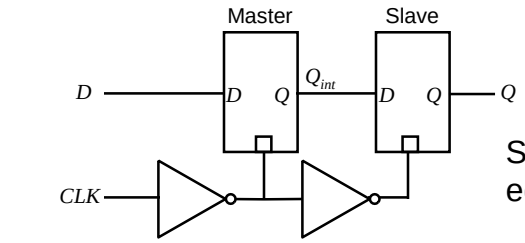
- The transparent D latch is so called '*level*' triggered. We can see it exhibits transparent behaviour if *EN*=1. It is often more simple to design sequential circuits if the outputs change only on the either rising (positive going) or falling (negative going) '*edges*' of the clock (i.e., enable) signal
- We can achieve this kind of operation by combining 2 transparent D latches in a so called *Master-Slave* configuration

Master-Slave D Flip-Flop

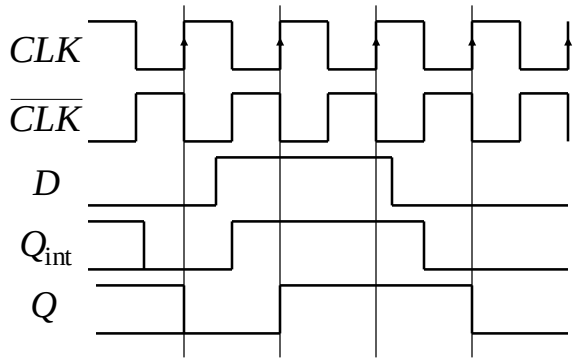


- To see how this works, we will use a timing diagram
- Note that both latch inputs are effectively connected to the clock signal (admittedly one is a complement of the other)

Master-Slave D Flip-Flop



See Q changes on rising edge of CLK



Note propagation delays have been neglected in the timing diagram

D Flip-Flops

- The Master-Slave configuration has now been superseded by new F-F circuits which are easier to implement and have better performance
- When designing synchronous circuits it is best to use truly edge triggered F-F devices
- We will not consider the design of such F-Fs on this course

Other Types of Flip-Flops

- Historically, other types of Flip-Flops have been important, e.g., J-K Flip-Flops and T-Flip-Flops
- However, J-K FFs are a lot more complex to build than D-types and so have fallen out of favour in modern designs, e.g., for field programmable gate arrays (FPGAs) and VLSI chips

Other Types of Flip-Flops

- Consequently we will only consider synchronous circuit design using D-type FFs
- However for completeness we will briefly look at the truth table for J-K and T type FFs

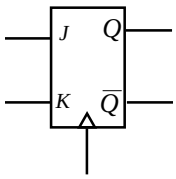
J-K Flip-Flop

- The J-K FF is similar in function to a clocked RS FF, but with the illegal state replaced with a new 'toggle' state

J K	Q' $\overline{Q}'$	comment
0 0	Q $\overline{Q}$	hold
0 1	0 1	reset
1 0	1 0	set
1 1	$\overline{Q}$ Q	toggle

Where Q' is the next state and Q is the current state

Symbol



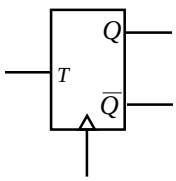
T Flip-Flop

- This is essentially a J-K FF with its J and K inputs connected together and renamed as the T input

T	Q' $\overline{Q}'$	comment
0	Q $\overline{Q}$	hold
1	$\overline{Q}$ Q	toggle

Where Q' is the next state and Q is the current state

Symbol



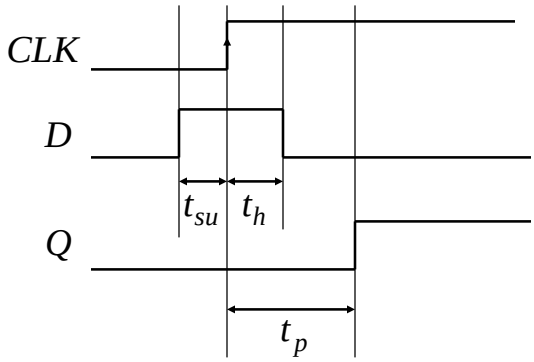
Asynchronous Inputs

- It is common for the FF types we have mentioned to also have additional so called ‘asynchronous’ inputs
- They are called asynchronous since they take effect independently of any clock or enable inputs
- Reset/Clear – force Q to 0
- Preset/Set – force Q to 1
- Often used to force a synchronous circuit into a known state, say at start-up.

Timing

- Various timings must be satisfied if a FF is to operate properly:
  - *Setup time*: Is the minimum duration that the data must be stable at the input before the clock edge
  - *Hold time*: Is the minimum duration that the data must remain stable on the FF input after the clock edge

Timing



t_{su} Set-up time

t_h Hold time

t_p Propagation delay