Latches and Flip-flops

CS207 Lecture 7

James YU

Apr. 1, 2020

南方科技大学
SOUTHERN UNIVERSITY OF SCIENCE AND TECHNOLOGY

Sequential logic

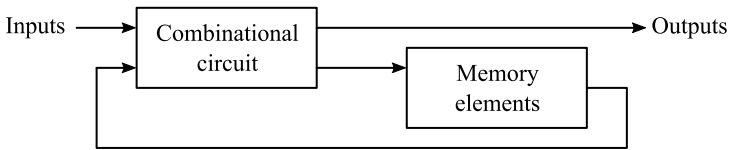
- Logic circuits for digital systems:
  - combinational logic (previous lectures),
  - sequential logic.
- Combinational?
  - Output determined by the combination of inputs.
  - Perform an operation specified by a set (combination) of Boolean functions.

Sequential logic

- Almost all electronic consumer products
  - send, receive, store, retrieve, and process information
  - depends on the ability to store binary — has memory.
- Sequential circuits
  - act as storage elements.
- The logic circuits whose outputs at any instant of time depend on the **present inputs** as well as on the **past outputs** are called *sequential circuits*.

Sequential logic

- The logic circuits whose outputs at any instant of time depend on the **present inputs** as well as on the **past outputs** are called *sequential circuits*.
- A combinational circuit, but with
  - memory elements connected in a feedback path.
  - Outputs are binary functions of not only inputs, but also the present state of the circuit.
  - A memory element is a medium in which one bit of information (`0` or `1`) can be stored or retained until necessary, and thereafter its contents can be replaced by a new value.



Sequential logic

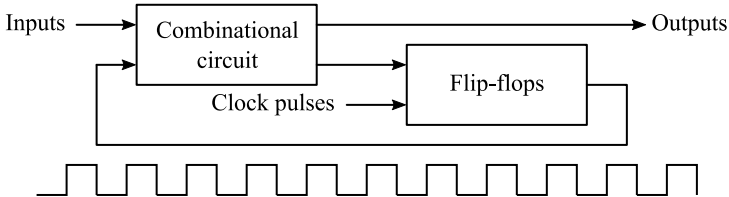
- Sequential circuits are broadly classified into two main categories, known as
  - *synchronous* or *clocked sequential circuits*, and
  - *asynchronous* or *unclocked sequential circuits*,
  - depending on the timing of their signals.
- A sequential circuit whose behavior can be defined from **the knowledge of its signal at discrete instants of time** is referred to as a synchronous sequential circuit.
  - The synchronization is achieved by a timing device known as a system clock.
  - The outputs are affected only with the application of a clock pulse.
- A sequential circuit whose behavior **depends upon the sequence in which the input signals change** is referred to as an asynchronous sequential circuit.

Sequential logic

- A sequential circuit whose behavior can be defined from **the knowledge of its signal at discrete instants of time** is referred to as a synchronous sequential circuit.
  - The synchronization is achieved by a timing device known as a system clock.
  - The outputs are affected only with the application of a clock pulse.
- Clock pulse: when changes will happen; Other signals: what changes will happen.
- Clocked sequential circuits:
  - Synchronous sequential circuits that use clock to control.
  - Synchronous because the circuit activity and the updating of storage are synchronized.

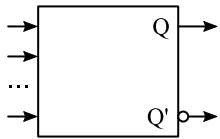
Sequential logic

- Memory element: *flip-flop*.
  - A binary storage device storing one bit;
  - Change in state happens only during clock pulse transition;
  - Loop cut when clock is inactive, no update.



Flip-flops

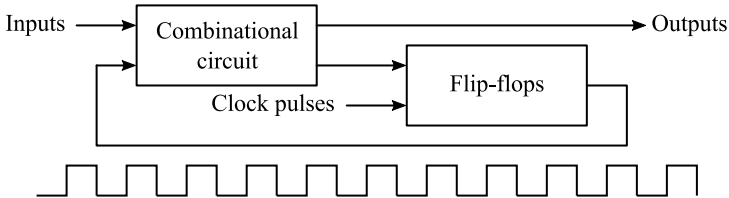
- It can have only two states, either the `1` state or the `0` state.
- The general block diagram representation of a flip-flop is shown below:



  - It has one or more inputs and two outputs.
  - The two outputs are complementary to each other.
- Normally, the state of Q is called the *state* of the flip-flop, whereas the state of Q' is called the *complementary state* of the flip-flop.

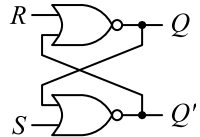
Flip-flops

- There has always been considerable confusion over the use of the terms *latch* and *flip-flop*.
  - A *flip-flop* is a device which changes its state at times when a change is taking place in the clock signal.
  - An *asynchronous latch* is continuously monitoring the input signals and changes its state at times when an input signal is changing.
  - A *synchronous latch* is continuously monitoring the input signals, but only can changes its state when a control signal is active.



SR latches

- By cross coupling a pair of NOR gates, we have a first latch: *SR latch*.

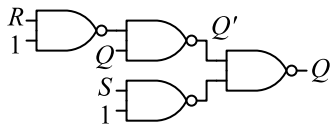


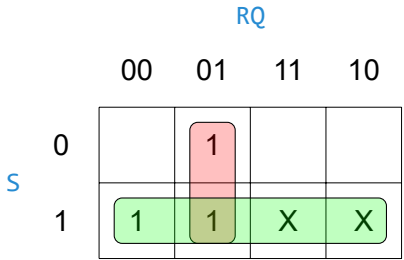
  - The set and reset inputs are labelled S and R respectively.
- The *state table* of SR latch is shown below

Present			Next
S_t	R_t	Q_t	Q_{t+1}
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	Forbidden	

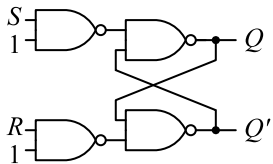
SR latches

- From the k-map: $Q_{t+1} = S + R'Q$.



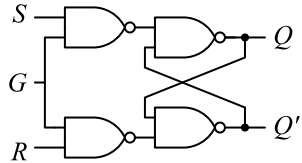


- A more conventional form:



Controlled SR latch

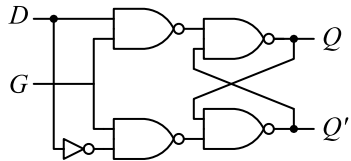
- The transparency of SR latches can be controlled by an additional signal G:



  - If $G = 0$, the outputs of first-level NAND gates are always `1`, disabling any changes in the second level gates.
  - If G makes a transition from `0` to `1`, the first-level NAND gates are enabled, making the latch active.
- A problem with SR latches: **state is indeterminate when S and R are both `1`.**

Controlled D latch

- D latch is designed to handle this problem.
  - Also called *transparent latch*, D for data.
  - Ensure the previous S and R are never equal to 1 at the same time.

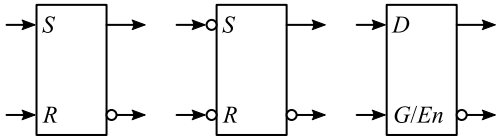


Present		Next
D_t	Q_t	Q_{t+1}
0	0	0
0	1	0
1	0	1
1	1	1

Controlled D latch

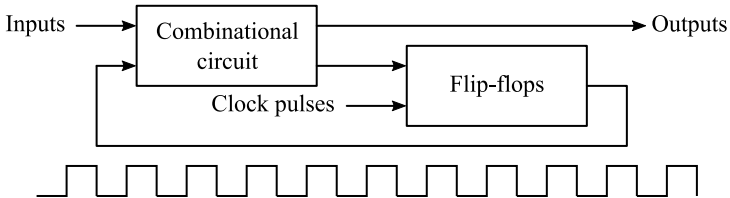
- The controlled D latch has the advantage that it only requires one data input and there is no input condition that has to be avoided.
- Transparent?
  - Data input is transferred to Q output when enable is asserted. The output follows the input.
  - When enable is de-asserted, the information is stored.

Latches



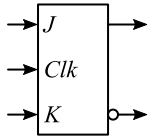
Flip-flops

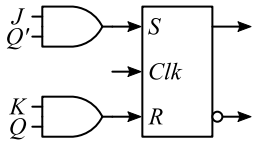
- Latch or flip-flop state change by a control input.
  - The event is called trigger.
- When latches are memories, difficulty arises:
  - State changes as soon as clock switches to 1.
  - Infinite loop during clock-1.



JK flip-flop

- Latch circuits are not suitable for operation in synchronous sequential circuits because of their transparency.
  - Flip-flops are used as the basic memory elements, which only respond to a transition on a clock input.
  - A typical example is called JK flip-flop (JKFF).

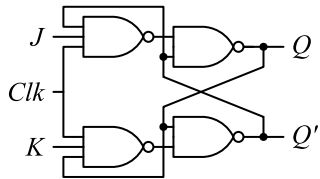




Present			Next
J_t	K_t	Q_t	Q_{t+1}
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

JK flip-flop

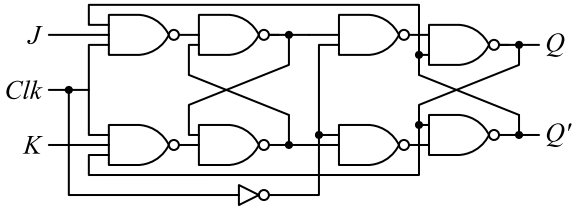
- J is S, and K is R.
- When $J = K = 1$, the flip-flop **toggles**.
- Combining the two AND gates with the SR latch circuit, we have the following reduced circuit:



- However, the above circuit alone still cannot resolve the problem with latches.
  - A master-slave JKFF can be used.

JK flip-flop

- The master-slave JKFF consists of two SR latches, the master and the slave.
  - The master is clocked in the normal way, while the slave clock is inverted:



- When the clock is `1`, the master latch works transparently.
  - However, as the slave latch is disabled at the same time, no changes will be reflected on the output.
- Upon the clock is changed to `0`, the slave is activated to reflect the data from master to the output.
  - However, the master latch is disabled: no further changes on input will be reflected.

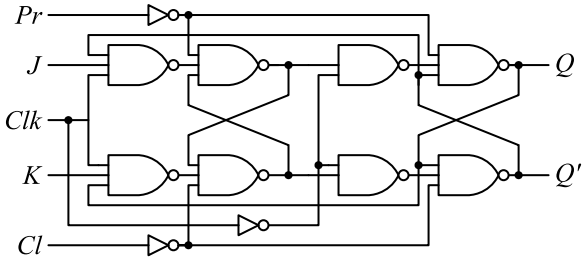
Asynchronous control

- As well as the J, K, and clock inputs, a master-slave JKFF may also have one or two additional controls to set the state of flip-flop irrespective of clock:
  - These asynchronous controls are usually called *preset* and *clear*.

Cl	Pr	Q
1	1	Forbidden
1	0	0
0	1	1
0	0	X

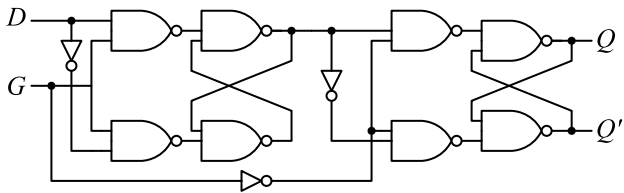
- If $Cl = 1$ and $Pr = 0$ both master and slave are cleared to 0.
- If $Cl = 0$ and $Pr = 1$ the flip-flop is preset to 1.
- Active high on Cl and Pr will override J and K.

Asynchronous control

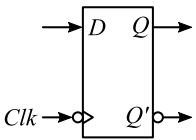


D flip-flop

- A negative edge triggered D type master-slave FF consists of a pair of D latches:

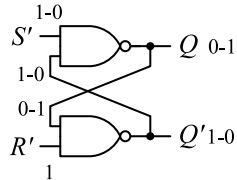


- Right is the symbolic diagram of DFF.
- The triangle is termed *dynamic input indicator*.



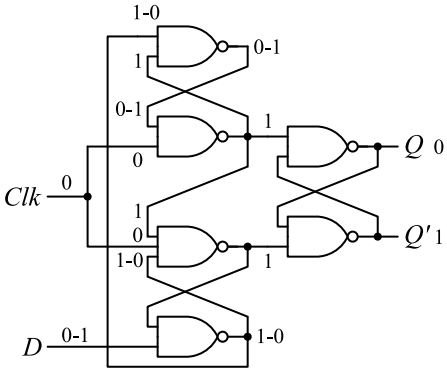
D flip-flop

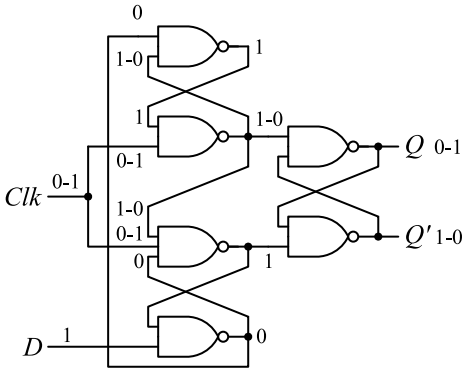
- An alternative configuration of a DFF consists of three pairs of cross-coupled NAND gates, each pair constituting a basic S'R' latch:



  - The latch is stable when $S' = R' = 1, Q = 0$.
  - To change the state, S' must make a `1`→`0` transition.

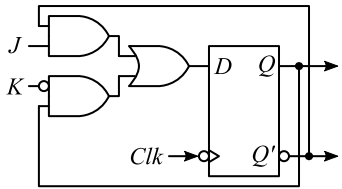
D flip-flop

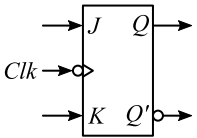

1-0
0-1
1
0-1
1
0
Q 0
Clk
0
1
0
Q' 1
1-0
1
D
0-1
1-0


0
1
1-0
1
1-0
Q 0-1
0-1
Clk
0-1
1-0
0-1
Q' 1-0
0
1
D
1
0

JK flip-flop revisit

- The previous DFF can be modified to provide the function of a JKFF as follows:



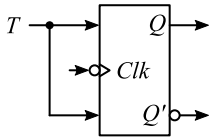


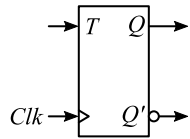
- If $J = K = 1$ and $Q' = 1$, the input to DFF is 1, the Q outputs 1.
- Think what happens when $J = 1$ and $K = 0$, and vice versa.

T flip-flop

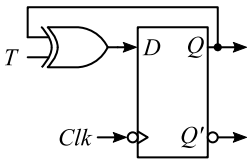
- Finally, a T flip-flop toggles the state when input $T = 1$ upon clock signal.

- It is simple to construct a TFF from JKFF:





- or DFF:



Characteristic table

- A *characteristic table* describes the logical properties of a flip-flop by describing its operation in tabular form.

J	K	Q_{t+1}	
0	0	Q_t	No change
0	1	0	Reset
1	0	1	Set
1	1	Q'_t	Complement

D	Q_{t+1}	
0	0	Reset
1	1	Set

T	Q_{t+1}	
0	Q_t	No change
1	Q'_t	Complement

Characteristic equation

- A characteristic equation describes the logical properties of a flip-flop by describing its Boolean function.
- JKFF: $Q_{t+1} = D$.
- DFF: $Q_{t+1} = JQ'_t + K'Q_t$.
- TFF: $Q_{t+1} = T \oplus Q_t$.

J	K	Q_{t+1}	
0	0	Q_t	No change
0	1	0	Reset
1	0	1	Set
1	1	Q'_t	Complement

D	Q_{t+1}	
0	0	Reset
1	1	Set

T	Q_{t+1}	
0	Q_t	No change
1	Q'_t	Complement

Verilog: Parameters

- Let's assume that we have a design which requires us to have counters of various width, but with the same functionality.
  - Normally what we do is creating counters of different widths and then use them.
- Verilog provides a powerful way to overcome this problem: it provides us with something called *parameter*.
  - These parameters are like constants local to that particular module.
- We can override the default values, either using `defparam` or by passing a new set of parameters during instantiation.
  - We call this *parameter overriding*.

Verilog: Parameters

- A parameter is defined by Verilog as a constant value declared within the module structure.
- The value can be used to define a set of attributes for the module which can characterize its behavior as well as its physical representation.
  - Defined inside a module.
  - Local scope.
  - Maybe overridden at instantiation time.
    - If multiple parameters are defined, they must be overridden in the order they were defined.
    - If an overriding value is not specified, the default parameter declaration values are used.
  - Maybe changed using the `defparam` statement.

Verilog: Parameters

```
1  module secret_number;
2  parameter my_secret = 0;
3
4  initial begin
5    $display("My secret number is %d", my_secret);
6  end
7  endmodule
8
9  module defparam_example();
10 defparam U0.my_secret = 11;
11 defparam U1.my_secret = 22;
12
13 secret_number U0();
14 secret_number U1();
15 endmodule
```

Verilog: Parameters

```
1  module secret_number;
2  parameter my_secret = 0;
3
4  initial begin
5    $display("My secret number in module is %d", my_secret);
6  end
7  endmodule
8
9  module param_overide_instance_example();
10
11 secret_number #(11) U0();
12 secret_number #(22) U1();
13 endmodule
```

Verilog: Parameters

```
module  ram_sp_sr_sw (
  clk     , // Clock Input
  address, // Address Input
  data    , // Data
  cs      , // Chip Select
  we      , // Enable
  oe        // Output En
);

parameter DATA_WIDTH = 8 ;
parameter ADDR_WIDTH = 8 ;
parameter RAM_DEPTH = 1 <<
   ADDR_WIDTH;
// Actual code of RAM here
endmodule
```

```
module  ram_controller ();

// Controller Code

ram_sp_sr_sw #(16,8,256)  ram(
   clk,address,data,cs,we,oe);

endmodule
```