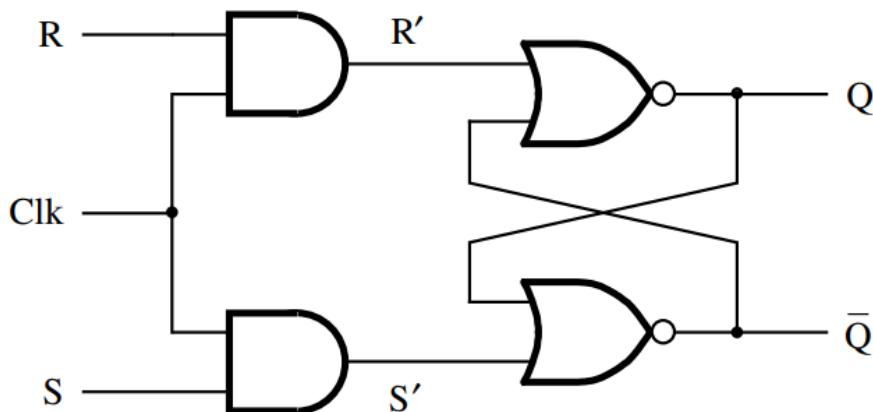


# Chapter 5: Sequential Circuits

Truong Ngoc Son, Ph.D

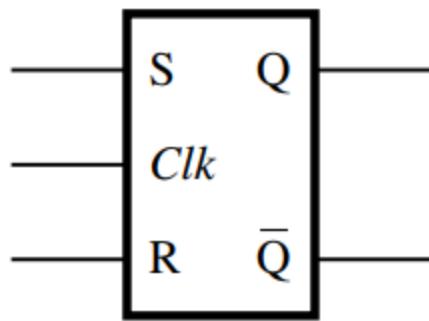
# RS Latch



(a) Circuit

Clk	S	R	$Q(t + 1)$
0	x	x	$Q(t)$ (no change)
1	0	0	$Q(t)$ (no change)
1	0	1	0
1	1	0	1
1	1	1	x

(b) Characteristic table

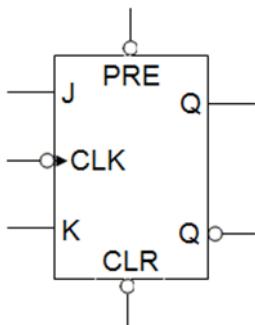


# RS Latch

- Verilog HDL

```
module RS_LATCH(  
input wire R,S,CLK,  
output reg Q,Qb  
);  
always @ (R,S,CLK) begin  
if/*((CLK==1)&&(S==0)&&(R==1))*/({CLK,S,R}  
==3'b101) begin Q=0;Qb=1;end  
else if ((CLK==1)&&(S==1)&&(R==0)) begin  
Q=1;Qb=0;end  
end  
endmodule
```

# JK Flip Flop

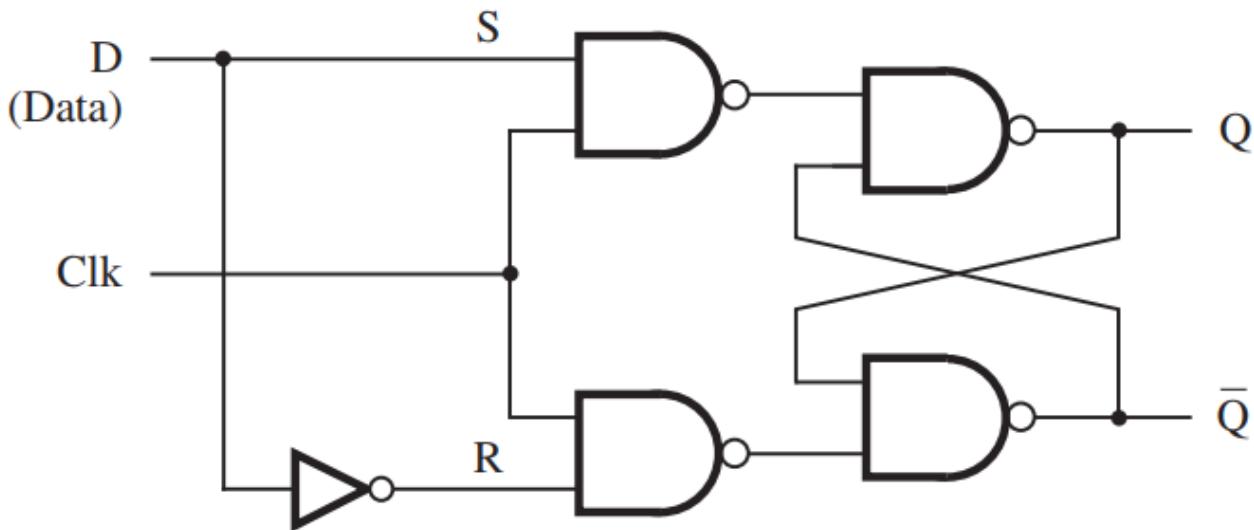


Input					Output	
Pre	CLR	CLK	J	K	Q	QD
0	0	x	x	x	1	1
0	1	x	x	x	1	0
1	0	x	x	x	0	1
1	1	0	x	x	Qo	QDo
1	1	↓	0	0	Qo	QDo
1	1	↓	0	1	0	1
1	1	↓	1	0	1	0
1	1	↓	1	1	NOT Q	NOT QD

# JK Flip Flop

- Verilog HDL

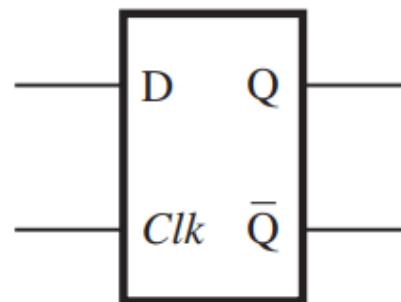
# D Latch



(a) Circuit

Clk	D	$Q(t+1)$
0	x	$Q(t)$
1	0	0
1	1	1

(b) Characteristic table



(c) Graphical symbol

# D Latch

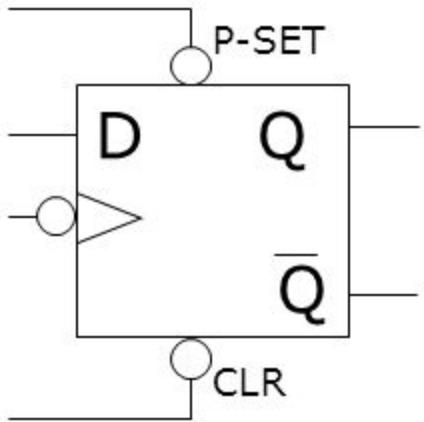
- Verilog HDL

```
module D_latch (D, Clk, Q);
    input D, Clk;
    output reg Q;

    always @(D, Clk)
        if (Clk)
            Q = D;

endmodule
```

# D Flip-Flop



P-SET	CLR	D	CLK	$Q_{n+1}$
0	1	X	X	1 (preset)
1	0	X	X	0 (clear)
0	0	X	X	? (illegal)
1	1	0	↓	0
1	1	1	↓	1

# D Flip-Flop

- The difference between a latch and a flip-flop is that a latch is asynchronous, and the outputs can change as soon as the inputs do (or at least after a small propagation delay). A flip-flop, on the other hand, is edge-triggered and only changes state when a control signal goes from high to low or low to high

```
module flipflop (D, Clock, Q);
    input D, Clock;
    output reg Q;

    always @ (posedge Clock)
        Q = D;

endmodule
```

# Latches vs. Flip-Flops

## Flip-Flop

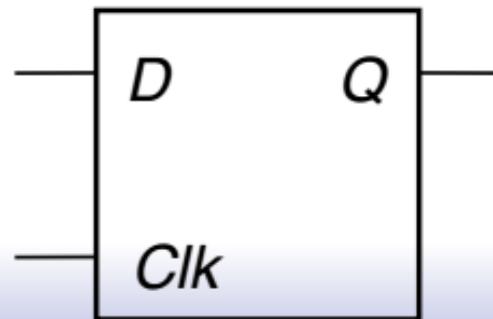
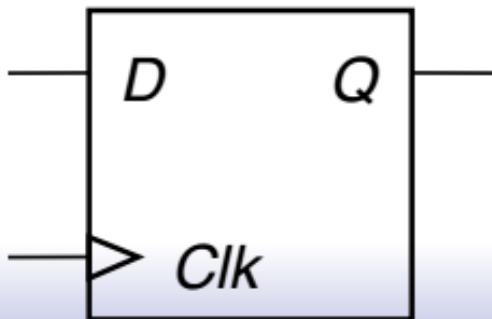
## Latch

```
module flipflop
(
    input clk,
    input d,
    output reg q
);

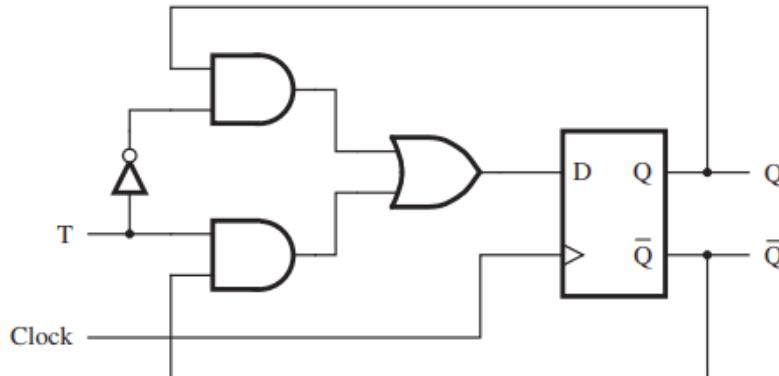
    always @(posedge clk)
    begin
        q <= d;
    end
endmodule
```

```
module latch
(
    input clk,
    input d,
    output reg q
);

    always @(clk or d)
    begin
        if( clk )
            q <= d;
    end
endmodule
```



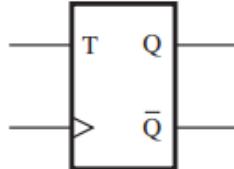
# T Flip-Flop



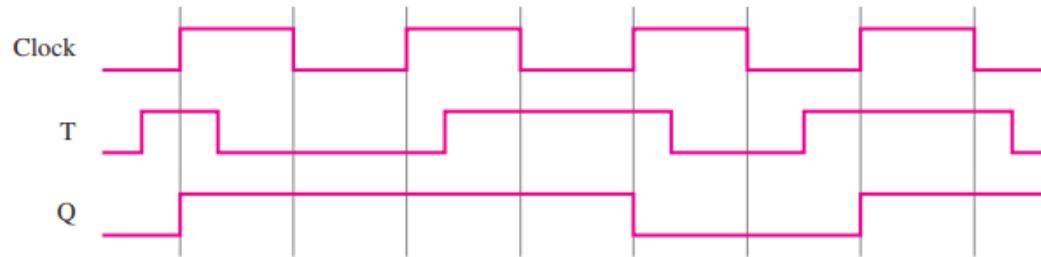
(a) Circuit

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

(b) Characteristic table



(c) Graphical symbol

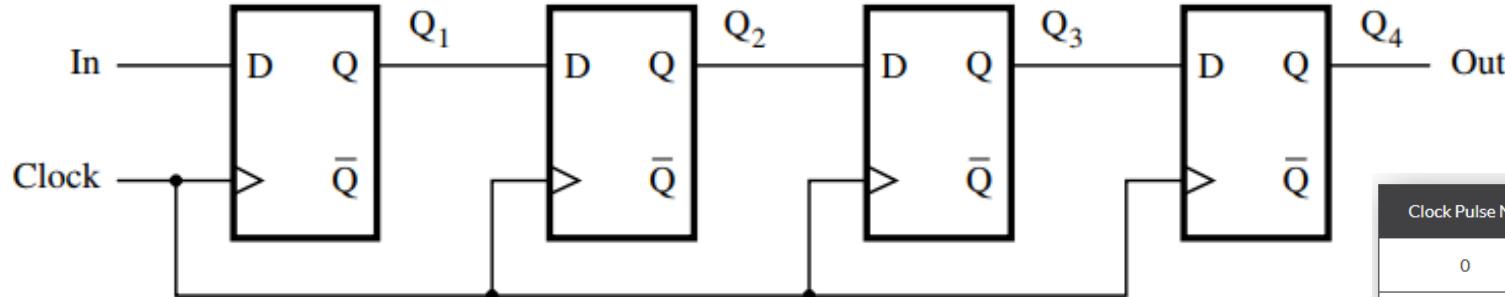


(d) Timing diagram

# T Flip-Flop

```
module T_FF(  
    input wire t, clk,  
    output reg q, qb );  
  
initial  
begin  
q=1 ;  
qb=0;  
end  
always @( posedge clk)  
if (t) begin  
q = ~q ;  
qb = !qb;  
end  
endmodule
```

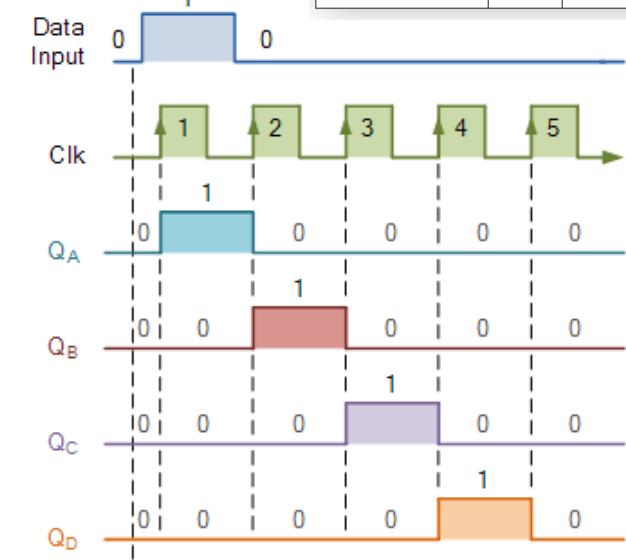
# Shift Register



(a) Circuit

	In	Q <sub>1</sub>	Q <sub>2</sub>	Q <sub>3</sub>	Q <sub>4</sub> = Out
$t_0$	1	0	0	0	0
$t_1$	0	1	0	0	0
$t_2$	1	0	1	0	0
$t_3$	1	1	0	1	0
$t_4$	1	1	1	0	1
$t_5$	0	1	1	1	0
$t_6$	0	0	1	1	1
$t_7$	0	0	0	1	1

Clock Pulse No	QA	QB	QC	QD
0	0	0	0	0
1	1	0	0	0
2	0	1	0	0
3	0	0	1	0
4	0	0	0	1
5	0	0	0	0



# Shift Register

- Instance shift register using D-FF.

```
module DFF(  
input wire d,clk,  
output reg q );  
always @(posedge clk)  
q = d ;  
endmodule  
// Serial input - serial output using DFF  
module SISO(  
input wire in, clk,  
output wire out) ;  
// signal declaration  
wire q1,q2,q3 ;  
//module instance  
DFF ff1 (in,clk,q1);  
DFF ff2 (q1,clk,q2);  
DFF ff3 (q2,clk,q3);  
DFF ff4 (q3,clk,out);  
endmodule
```

# Module instance review

- Modules can be instantiated from within other modules. When a module is instantiated, connections to the ports of the module must be specified.
- There are two ways to make port connections.
  - Connection by name, in which variables connected to each of module inputs or outputs are specified in a set of parenthesis following the name of the ports. In this method order of connections is not significant.
  - Ordered connection. In this method the order of the ports must match the order appearing in the instantiated module.

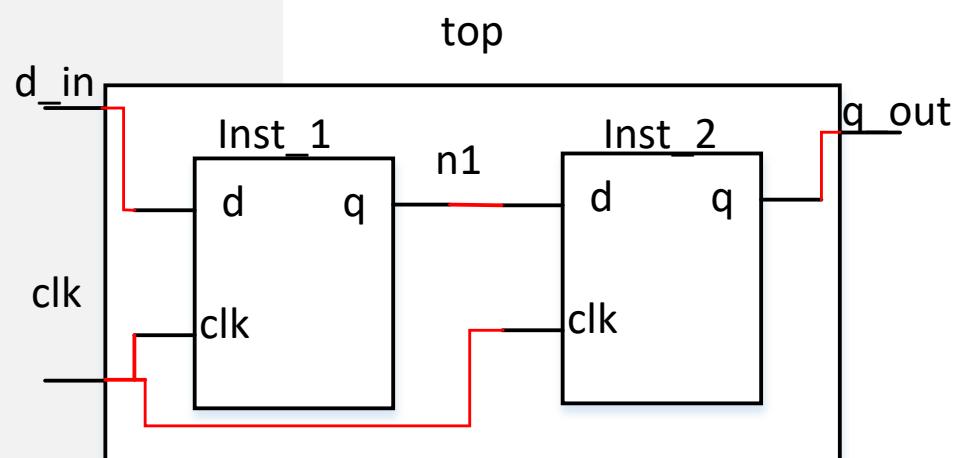
# Module instance review

- Connection by name

```
module dff (  
    input wire clk, d,  
    output reg q );  
    always @ (posedge clk) q = d;  
endmodule
```

```
module top (  
    input wire d_in, clk,  
    output wire q_out);
```

```
    wire n1;  
    dff Inst_1 (.d(d_in), .q(n1), .clk(clk));  
    dff Inst_2 (.clk(clk), .d(n1), .q(q_out));  
endmodule
```



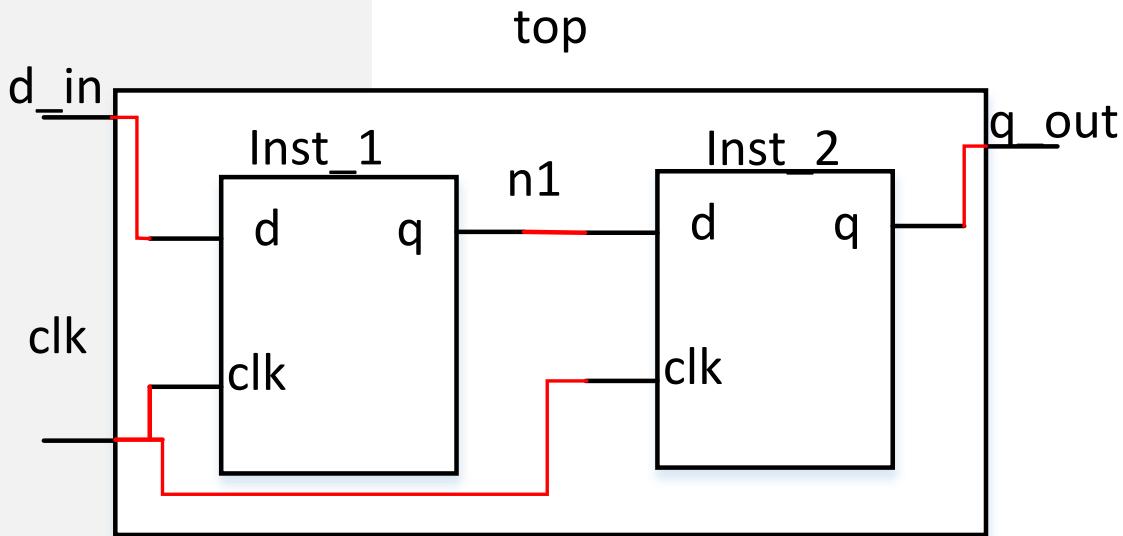
# Module instance review

- Connection by order

```
module dff (
    input wire clk, d,
    output reg q );
    always @(posedge clk) q = d;
endmodule
```

```
module top (
    input wire d_in, clk,
    output wire q_out);
    wire n1;
```

```
    dff Inst_1 (clk, d_in, n1);
    dff Inst_2 (clk, n1, q_out);
endmodule
```

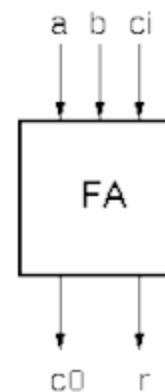


# Example - Ripple Adder

```
module FullAdder(a, b, ci, r, co);
    input a, b, ci;
    output r, co;

    assign r = a ^ b ^ ci;
    assign co = a&ci | a&b | b&ci;

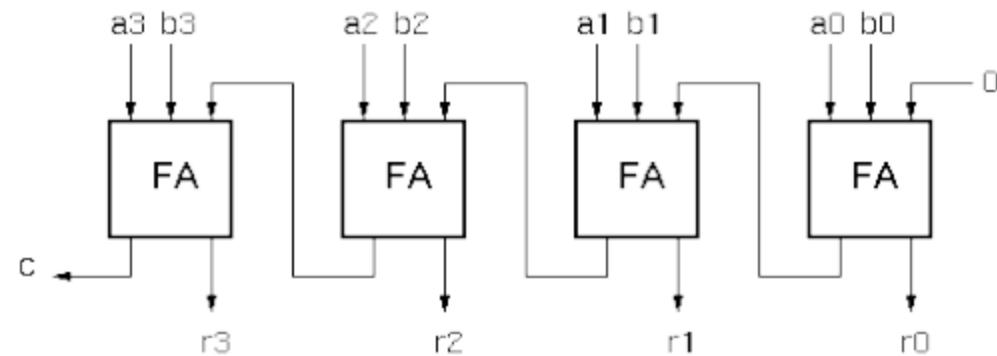
endmodule
```



```
module Adder(A, B, R);
    input [3:0] A;
    input [3:0] B;
    output [4:0] R;

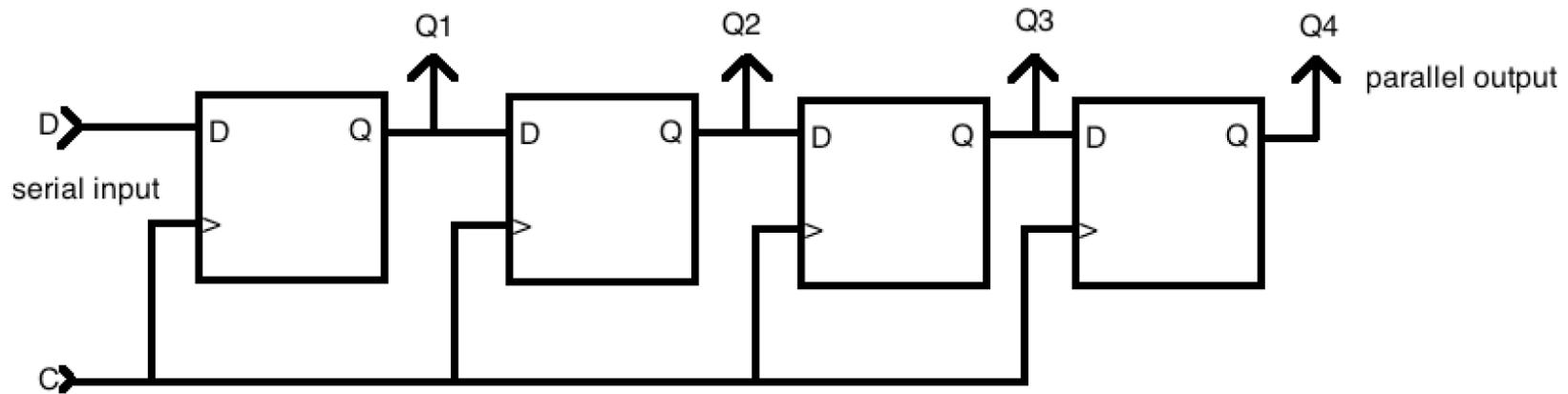
    wire c1, c2, c3;
    FullAdder
    add0(.a(A[0]), .b(B[0]), .ci(1'b0), .co(c1), .r(R[0]) ),
    add1(.a(A[1]), .b(B[1]), .ci(c1), .co(c2), .r(R[1]) ),
    add2(.a(A[2]), .b(B[2]), .ci(c2), .co(c3), .r(R[2]) ),
    add3(.a(A[3]), .b(B[3]), .ci(c3), .co(R[4]), .r(R[3]) );

endmodule
```



# Shift Register – Serial input, parallel outputs

- Instance the shift register using D-FF

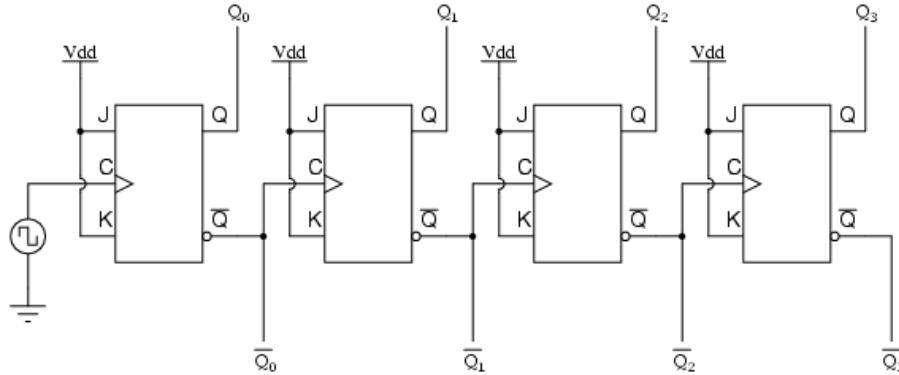


# Shift Register – Serial input parallel outputs

```
module DFF(  
input wire d,clk,  
output reg q );  
always @(posedge clk)  
q = d ;  
endmodule  
// Serial input - parallel output using DFF  
module SIPO(  
input wire in, clk,  
output wire [3:0] q) ;  
// signal declaration  
  
//module instance  
DFF ff1 (in,clk,q[0]);  
DFF ff2 (q[0],clk,q[1]);  
DFF ff3 (q[1],clk,q[2]);  
DFF ff4 (q[2],clk,q[3]);  
endmodule
```

# Counter

A simultaneous "up" and "down" counter



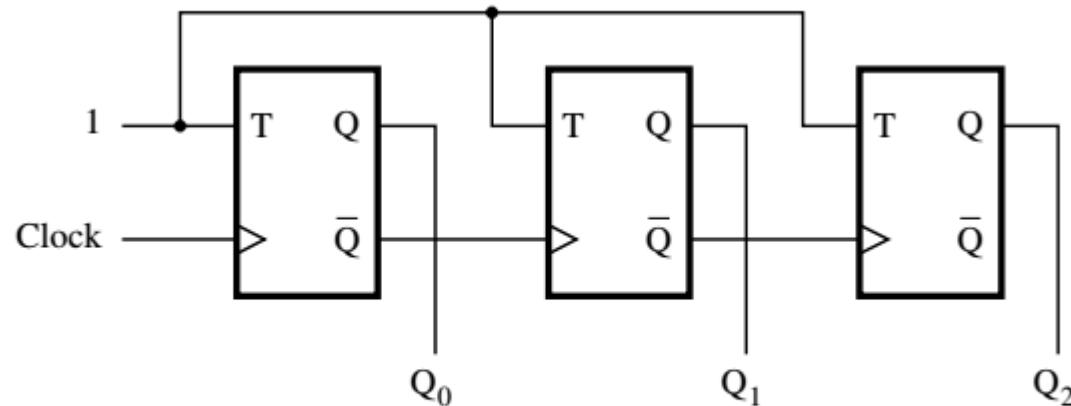
Clock cycle	$Q_2$	$Q_1$	$Q_0$
0	0	0	0
1	0	0	1
2	0	1	0
3	0	1	1
4	1	0	0
5	1	0	1
6	1	1	0
7	1	1	1
8	0	0	0

Annotations in pink highlight changes in the  $Q_1$  and  $Q_2$  columns:

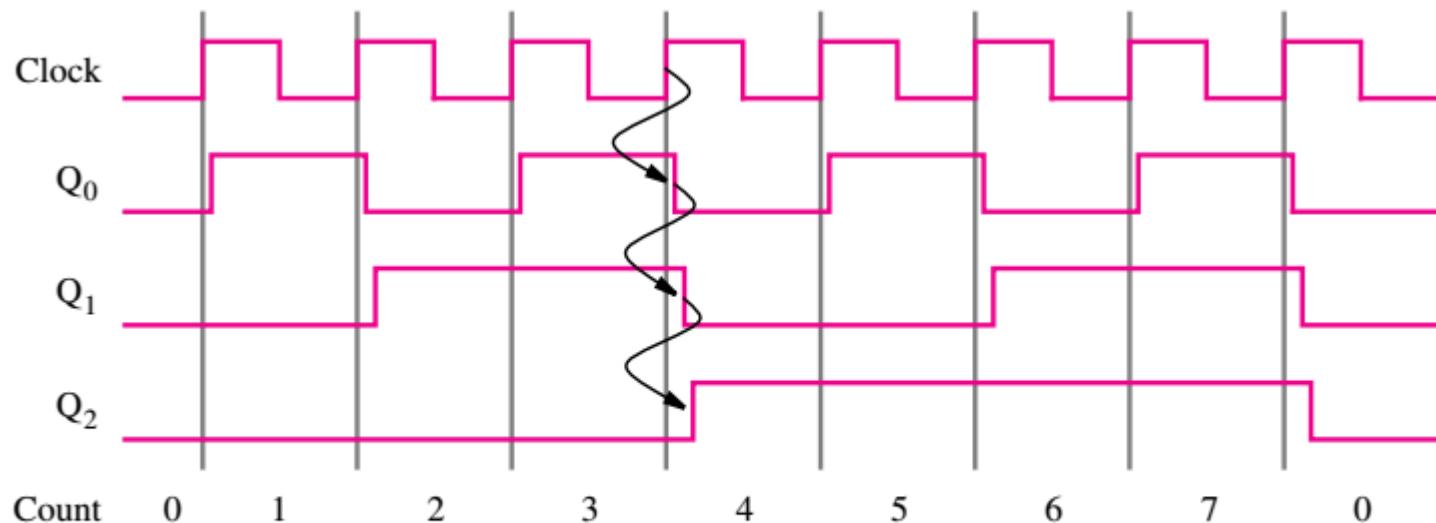
- $Q_1$  changes at cycles 2, 4, and 6.
- $Q_2$  changes at cycles 4 and 6.

Combinational circuit

# Asynchronous Counter



(a) Circuit



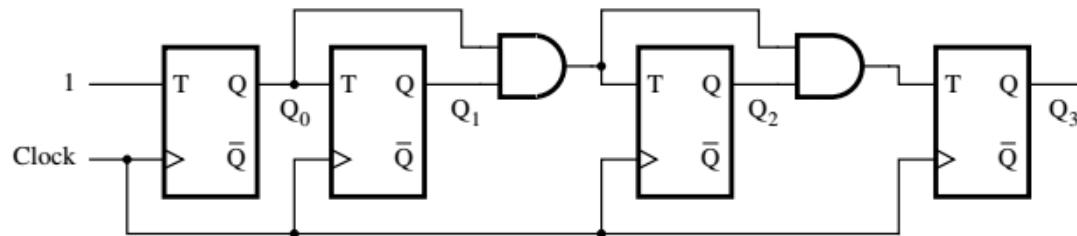
(b) Timing diagram

it

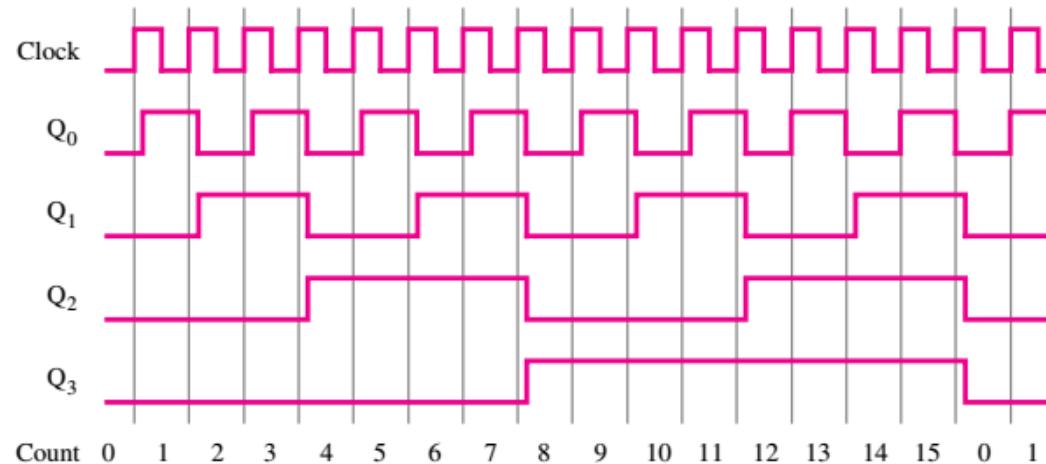
# Counter

- Instance counter with T-FF

# Synchronous counter



(a) Circuit



(b) Timing diagram

**Figure 5.21** A four-bit synchronous up-counter.

# Counter

- Instance counter with T-FF

# Example - Ripple Adder Generator

Parameters give us a way to generalize our designs. A module becomes a “generator” for different variations. Enables design/module reuse. Can simplify testing.

*Declare a parameter with default value.*

```
module Adder(A, B, R);
    parameter N = 4;
    input [N-1:0] A;
    input [N-1:0] B;
    output [N:0] R;
    wire [N:0] C;

    genvar i;
    generate
        for (i=0; i<N; i=i+1) begin:bit
            FullAdder add(.a(A[i]), .b(B[i]), .ci(C[i]), .co(C[i+1]), .r(R[i]));
        end
    endgenerate

    assign C[0] = 1'b0;
    assign R[N] = C[N];
endmodule
```

*Note: this is not a port. Acts like a “synthesis-time” constant.*

*Replace all occurrences of “4” with “N”.*

*variable exists only in the specification - not in the final circuit.*

*Keyword that denotes synthesis-time operations*

*For-loop creates instances (with unique names)*

**Adder adder4 ( ... );**

**Adder #(N(64))**      *Overwrite parameter  
N at instantiation.*

**adder64 ( ... );**

# Homework

- Design a circuit to control 8 LEDs
  - Light LEDS sequentially from left to right then turn 8 LEDs off sequentially from left to right (one-by-one).
  - The frequency is adjusted by two switches
  - The input clock is 50Mhz