

Registers & Counters

Objectives

This section deals with some simple and useful sequential circuits. Its objectives are to:

- Introduce registers as multi-bit storage devices.
- Introduce counters by adding logic to registers implementing the functional capability to increment and/or decrement their contents.
- Define shift registers and show how they can be used to implement counters that use the one-hot code.

Reading Assignment

- Sections 4.4 and 5.4

1. Registers

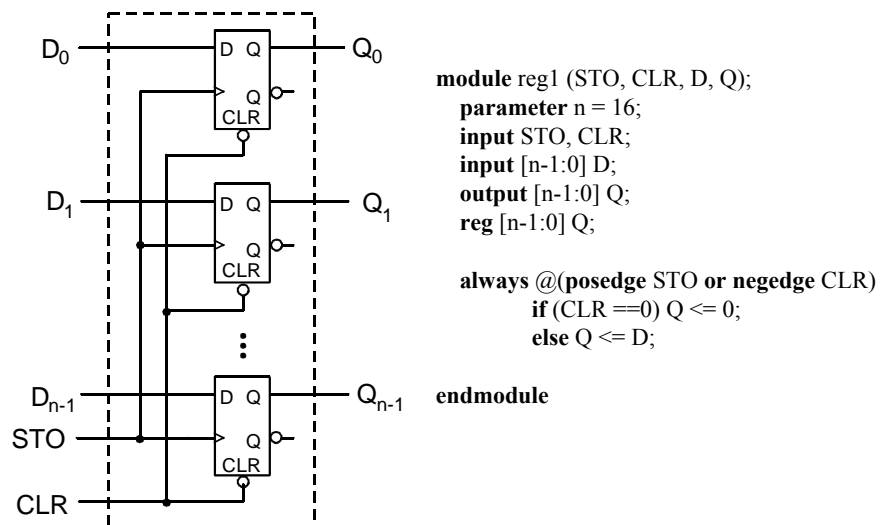
- A register is a memory device that can be used to store more than one bit of information.
- A register is usually realized as several flip-flops with common control signals that control the movement of data to and from the register.
 - Common refers to the property that the control signals apply to all flip-flops in the same way
 - A register is a generalization of a flip-flop. Where a flip-flop stores one bit, a register stores several bits
 - The main operations on a register are the same as for any storage devices, namely
 - ◆ Load or Store: Put new data into the register
 - ◆ Read: Retrieve the data stored in the register (usually without changing the stored data)

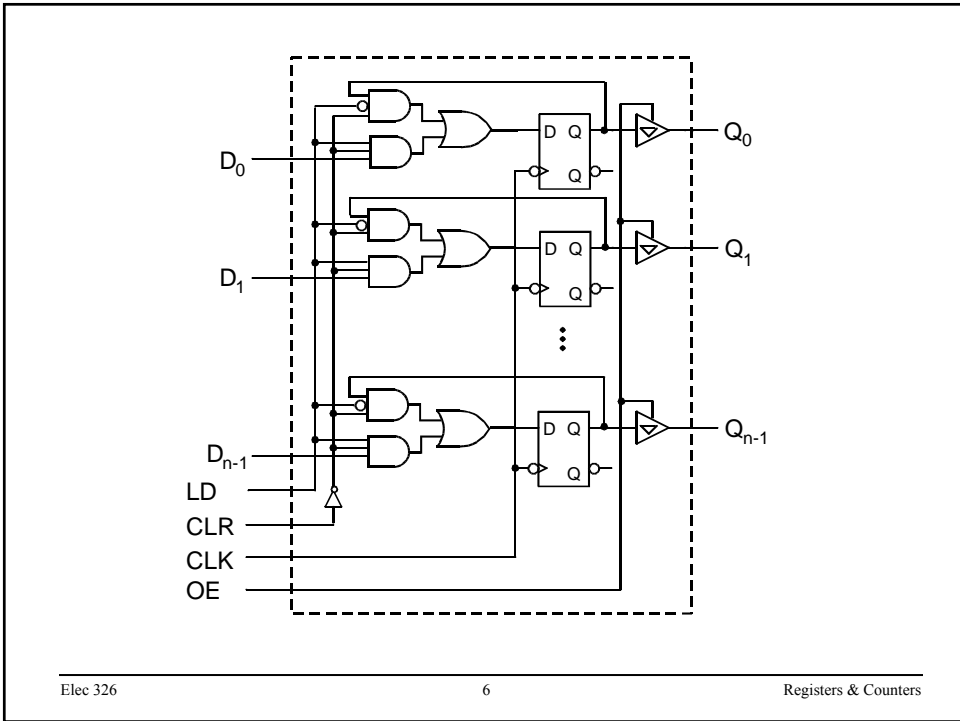
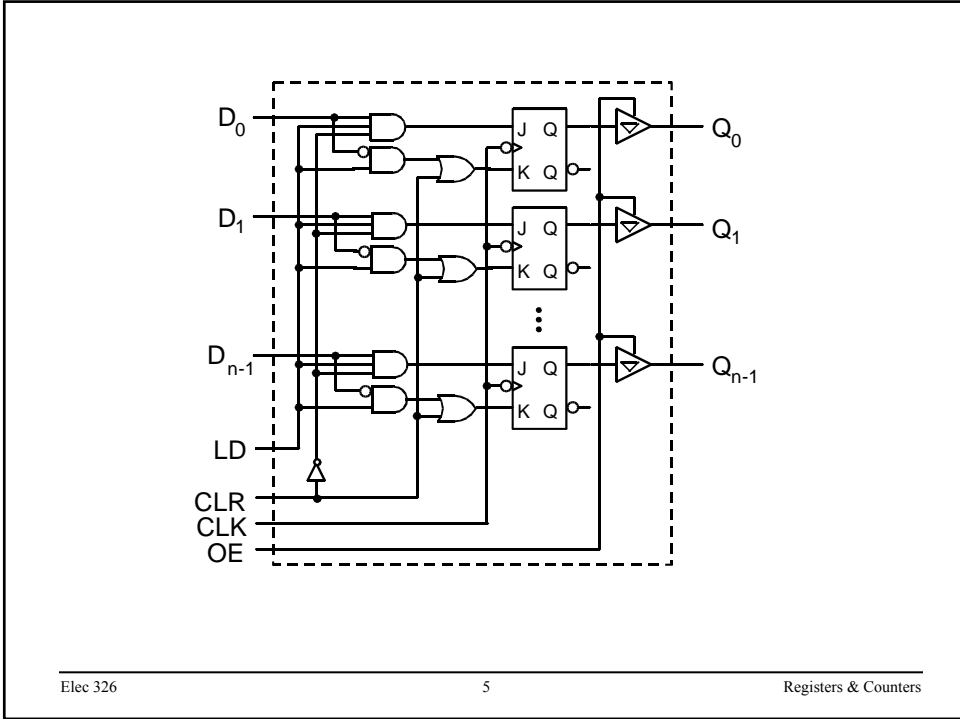
□ Control Signals

- When they are asserted, they initiate an action in the register
- *Asynchronous Control Signals* cause the action to take place immediately
- *Synchronous Control Signals* must be asserted during a clock assertion to have an effect

□ Examples

- On the following three registers, which control signals are asynchronous and which are synchronous? How are the control signals asserted?





□ Verilog description of previous two registers

```
module reg2 (CLK, CLR, LD, OE, D, Q);
  parameter n = 4;
  input CLK, CLR, LD, OE;
  input [n-1:0] D;
  output [n-1:0] Q;
  reg [n-1:0] IQ, Q;
  integer k;

  always @(posedge CLK)
    if (CLR) IQ <= 0;
    else if (LD) IQ <= D;

  always @(OE)
    if (OE) Q = IQ;
    else Q = 'bz;

endmodule
```

2. Counters

□ A counter is a register capable of incrementing and/or decrementing its contents

$$Q \leftarrow Q \text{ plus } n$$
$$Q \leftarrow Q \text{ minus } n$$

- The definition of "plus" and "minus" depend on the way the register contents encode the integers
- Binary Counters: Encode the integers with the binary number code

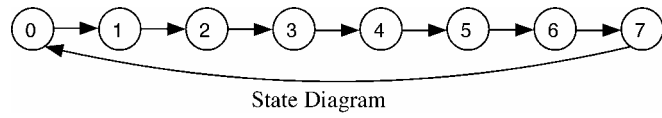
□ Example: 3-bit binary counter:

	000	
	001	
	010	
plus	011	
↓	100	
	101	
	110	
	111	
	000	
	⋮	

		↑
		minus

000	001
001	010
010	011
011	100
100	101
101	110
110	111
111	000

0	1
1	2
2	3
3	4
4	5
5	6
6	7
7	0



- What does the counter count?
- The output signals are just the state variables

□ Example: 3-bit binary up/down counter

	0	1
000	111	001
001	000	010
010	001	011
011	010	100
100	011	101
101	100	110
110	101	111
111	110	000

□ Example: Binary mod 6 counter

000	001
001	010
010	011
011	100
100	101
101	000
110	x x x
111	x x x

```

    graph LR
      0((0)) --> 1((1))
      1 --> 2((2))
      2 --> 3((3))
      3 --> 4((4))
      4 --> 5((5))
      5 --> 0
  
```

□ Design of a Binary Up Counter

Q2	Q1	Q0	
0	0	0	
0	0	1	
0	1	0	↓
0	1	1	
1	0	0	
1	0	1	
1	1	0	
1	1	1	
0	0	0	
	⋮		
	⋮		

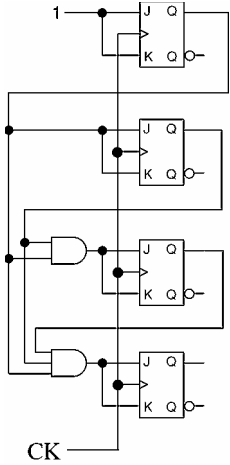
Count Sequence

Q0 Toggles every clock cycle

Q1 toggles on those clock cycles where Q0=1

Q2 toggles on those clock cycles where Q0=Q1=1

- Qi toggles on every clock cycle where Qj = 1, for i > j ≥ 0



Binary Up Counter

□ Design of a Binary Down Counter

Q2	Q1	Q0	
1	1	1	
1	1	0	
1	0	1	↓
1	0	0	
0	1	1	
0	1	0	
0	0	1	
0	0	0	
1	1	1	
⋮			
⋮			
⋮			

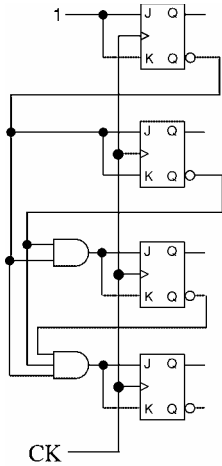
Count Sequence

Q0 Toggles every clock cycle

Q1 toggles on those clock cycles where Q0=0

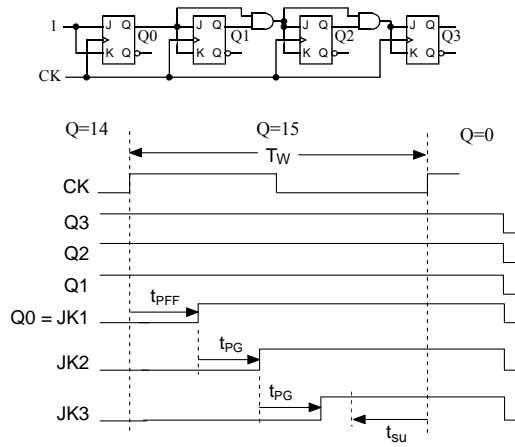
Q2 toggles on those clock cycles where Q0=Q1=0

- Qi toggles on every clock cycle where Qj = 0, for $i > j \geq 0$



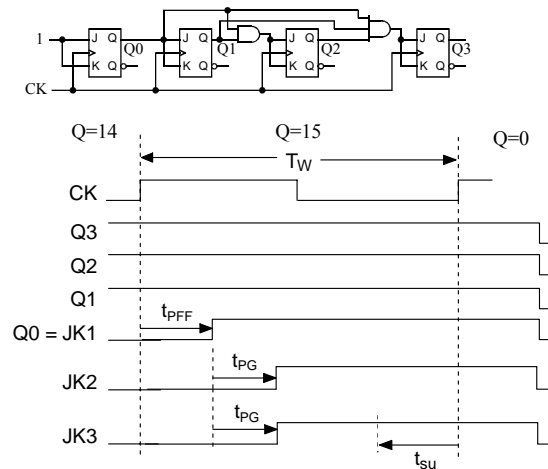
Binary Down Counter

□ Synchronous, Series-Carry Binary Counter



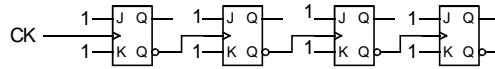
$$T_W \geq t_{PFF} + (n-2)t_{PG} + t_{su} \text{ (for } n \geq 2\text{)}$$

□ Synchronous, Parallel-Carry Binary Counter

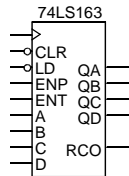


$$T_W \geq t_{PFF} + t_{PG} + t_{su} \text{ (for } n \geq 3\text{)}$$

□ Asynchronous Counters



□ Typical MSI counter chip



- LD and CLR are synchronous
- LD asserted during the rising edge of the clock loads the register from ABCD.
- CLR asserted during the rising edge of the clock clears the counter
- CLR overrides LD
- LD overrides EN
- $R_{CO} = Q_D \cdot Q_C \cdot Q_B \cdot Q_A \cdot ENT$, used for cascading chips

□ Verilog description of the 74x163

```

module V74x163 (CLK, CLR_L, LD_L, ENP, ENT, D, Q, RCO);
  input CLK, CLR_L, LD_L, ENP, ENT;
  input [3:0] D;
  output RCO;
  output [3:0] Q;
  reg [3:0] Q;
  reg RCO;

  always @(posedge CLK)
    if (CLR_L == 0) Q <= 4'b0000;
    else if (LD_L == 0) Q <= D;
    else if (ENT & ENP) Q <= Q + 1;

  always @(Q or ENT)
    if (Q == 15 && ENT == 1) RCO = 1;
    else RCO = 0;

endmodule

```

□ Verilog description of an up/down counter

```
module updowncount (R, Clock, L, E, up_down, Q);
    parameter n = 8;
    input [n-1:0] R;
    input Clock, L, E, up_down;
    output [n-1:0] Q;
    reg [n-1:0] Q;
    integer direction;

    always @(posedge Clock)
    begin
        if (up_down) direction = 1;
        else direction = -1;
        if (L) Q <= R;
        else if (E) Q <= Q + direction;
    end

endmodule
```

□ Verilog description of mod-n counters

```
module upmodn (Ck, Q);
    parameter n = 6;
    input Ck;
    output [3:0] Q;
    reg [3:0] Q;

    always @(posedge Ck)
    if (Q == n)
        Q <= 0;
    else
        Q <= Q + 1;

endmodule
```

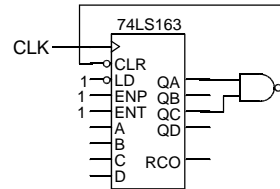
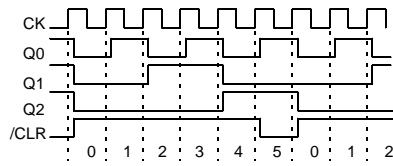
```
module dwnmodn (Ck, Q);
    parameter n = 5;
    input Ck;
    output [3:0] Q;
    reg [3:0] Q;

    always @(posedge Ck)
    if (Q == 0)
        Q <= n;
    else
        Q <= Q - 1;

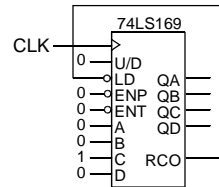
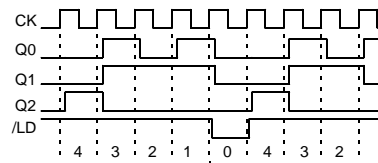
endmodule
```

□ Design of Mod n Counters

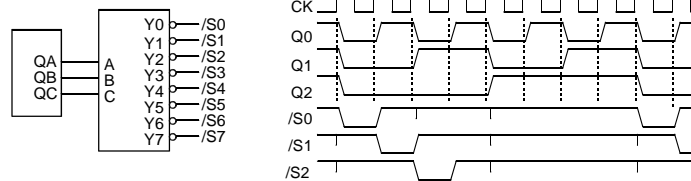
■ Mod 6 Up Counter



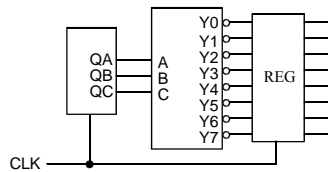
◆ Mod 5 Down Counter



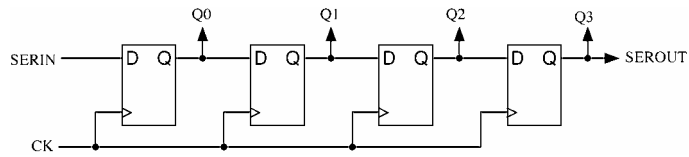
□ Decoding Binary Counter States



- The decoding spikes are hazards that can not be designed out
- The following circuit will mask the decoding spikes, at the cost of delaying the outputs one clock cycle.

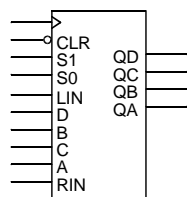


3. Shift Registers



- How would you add a control signal to control when the shift register shifted?
- How would you add parallel input capability and why would you want to?
 - ◆ What kind of control signals are needed?
- Is the shift register drawn above a left shifter or a right shifter?
- How would you make a shift register that could shift either left or right and what control signals would you need?

□ Example: 74LS194



S1	S0	Action	QA*	QB*	QC*	QD*
0	0	hold	QA	QB	QC	QD
0	1	shift right	RIN	QA	QB	QC
1	0	shift left	QB	QC	QD	LIN
1	1	load	A	B	C	D

- Shift left is from A to D
- Shift right is from D to A
- CLR is asynchronous

□ Verilog Description Of A Shift Register

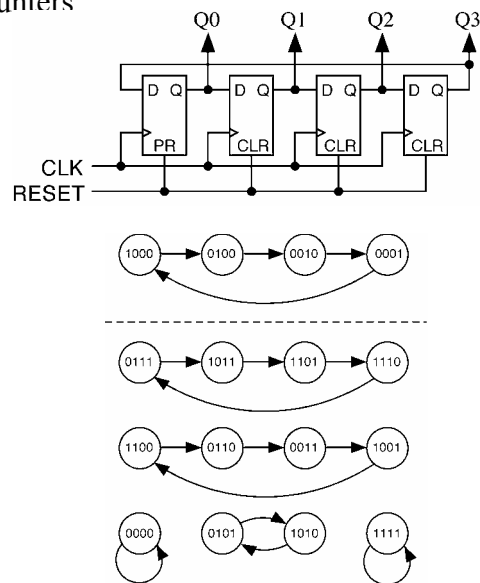
```

module shift4 (D, LD, LI, Ck, Q);
  input [3:0] D;
  input LD, LI, Ck;
  output [3:0] Q;
  reg [3:0] Q;

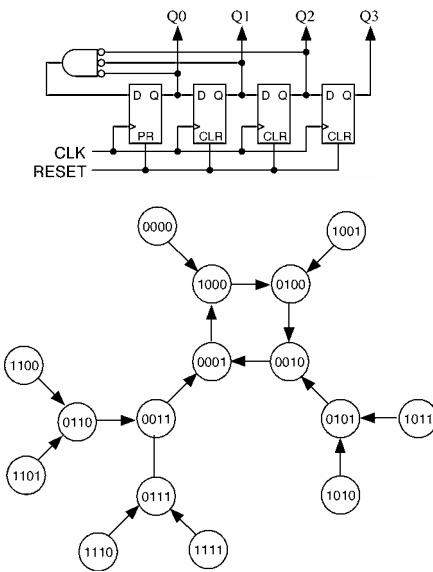
  always @(posedge Ck)
    if (LD)
      Q <= D;
    else
      begin
        Q[0] <= Q[1];
        Q[1] <= Q[2];
        Q[2] <= Q[3];
        Q[3] <= LI;
      end
    end
endmodule

```

□ Ring Counters



□ Self-Correcting Ring Counter



□ Johnson counter, switch-tail counter, moebius counter

