Lecture 12
Counters & Shift Registers

Outline

- Counters (**)
- Shift Registers (**)
- Textbook Chapter 11.1 and 11.2

Counters-Overview

- Special sequential circuits (Finite State Machine) that sequence though a set outputs.
- Examples:
  - binary counter: 000, 001, 010, 011, 100, 101, 110, 111, 000, 001, …
  - gray code counter: 000, 010, 110, 100, 101, 111, 011, 001, 000, 010, 110, …
  - one-hot counter: 0001, 0010, 0100, 1000, 0001, 0010, …
  - BCD counter: 0000, 0001, 0010, …, 1001, 0000, 0001
  - pseudo-random sequence generators: 10, 01, 00, 11, 10, 01, 00, ...

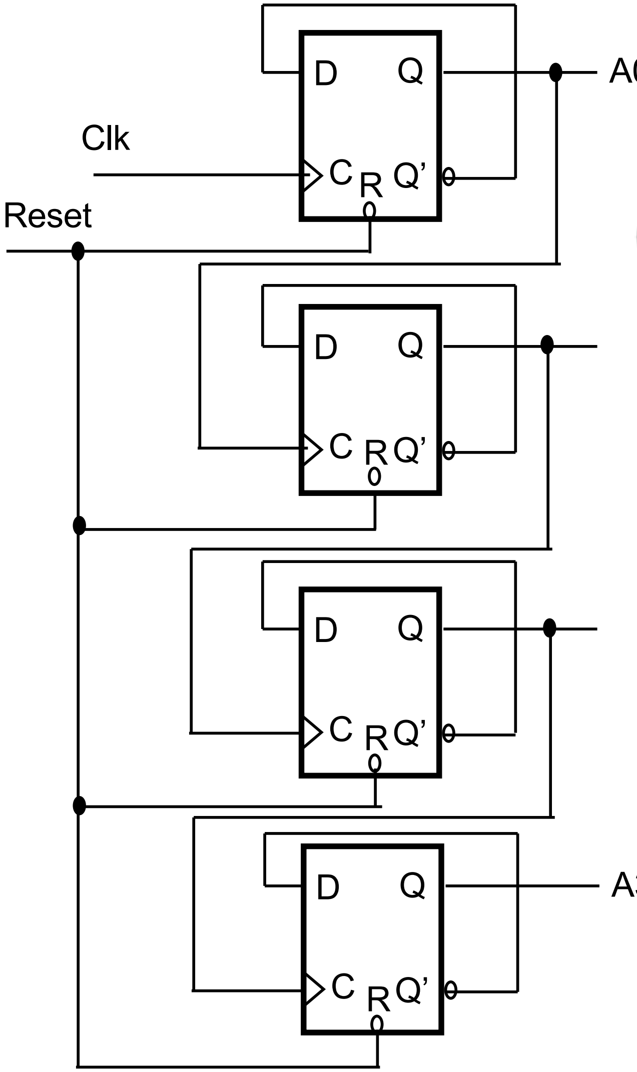
Counter in General

- In general: counters simplify controller design by
  - providing a specific number of cycles of action
  - sometimes used with a decoder to generate a sequence of control signals
- It is a special case of Finite State Machine (FSM)
- Can be asynchronous or synchronous
  - Synchronous counter is always preferred
    - Easier to implement in RTL

Ripple Counter: Asynchronous

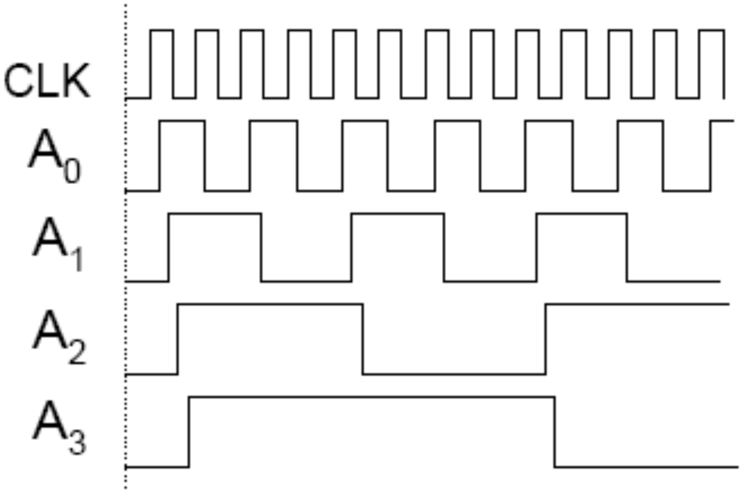
A3	A2	A1	A0
0	0	0	0
0	0	0	1
0	0	1	0
0	0	1	1
0	1	0	0
0	1	0	1
0	1	1	0
0	1	1	1
1	0	0	0
1	0	0	1
1	0	1	0
1	0	1	1
1	1	0	0
1	1	0	1
1	1	1	0
1	1	1	1

time



Each stage is ÷2 of previous

- Look at output waveforms:



Since clock inputs of some FFs are actually not driven by the clock, ripple counters are called asynchronous counters

Synchronous Counters

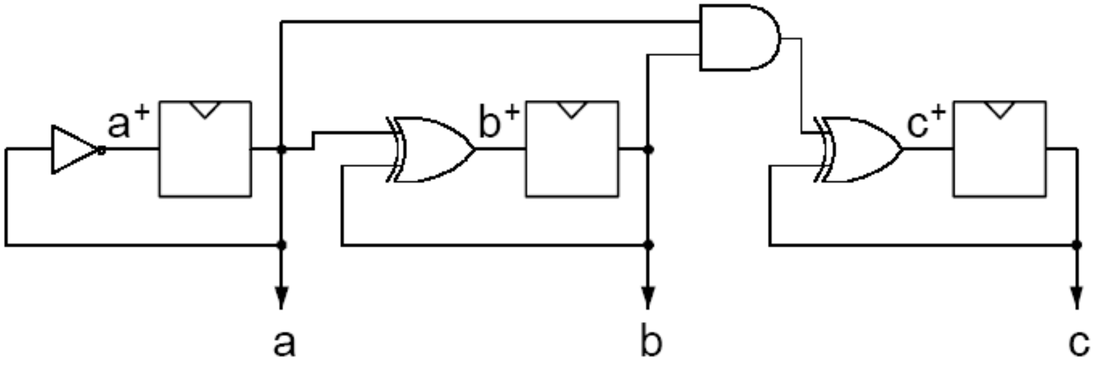
- *All outputs change with clock edge*
- Binary Counter Design: Start with 3-bit version and generalize:

c	b	a	c+	b+	a+
0	0	0	0	0	1
0	0	1	0	1	0
0	1	0	0	1	1
0	1	1	1	0	0
1	0	0	1	0	1
1	0	1	1	1	0
1	1	0	1	1	1
1	1	1	0	0	0

a+ = a’

b+ = a XOR b

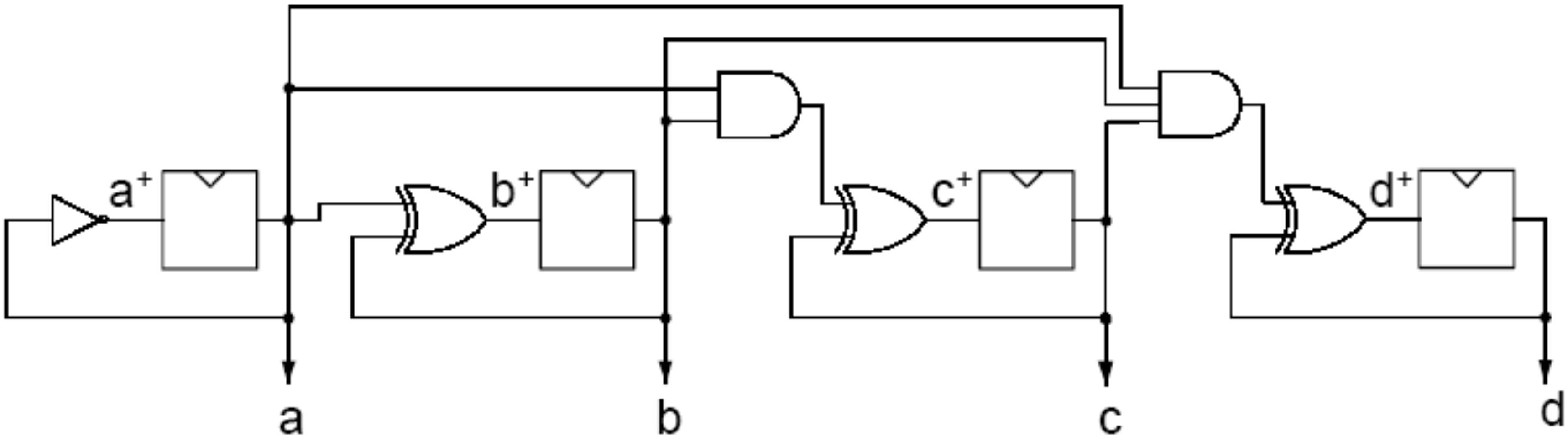
a \ cb	00	01	11	10
0	0	0	1	1
1	0	1	0	1



$$\begin{aligned} c^{+} &= a'c + abc' + b'c \\ &= c(a'+b') + c'(ab) \\ &= c(ab)' + c'(ab) \\ &= c \oplus ab \end{aligned}$$

Synchronous Counters

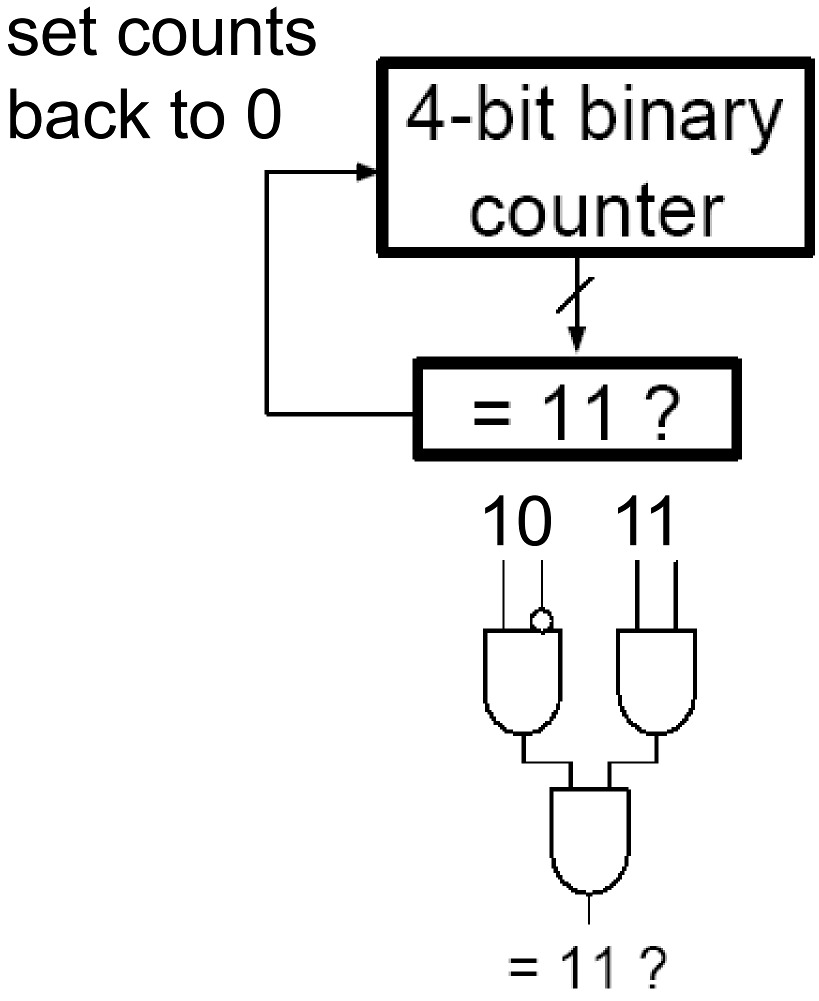
- How do we extend to n-bits?
- Extrapolate c+: d+ = d xor abc, e+ = e xor abcd



Cons: Has difficulty scaling (AND gate inputs grow with n)

Synchronous Counters

- How to stop earlier?
- Extra logic can be added to stop counting before reaching the maximum
  - – Example: count to 11



set counts back to 0

4-bit binary counter

= 11 ?

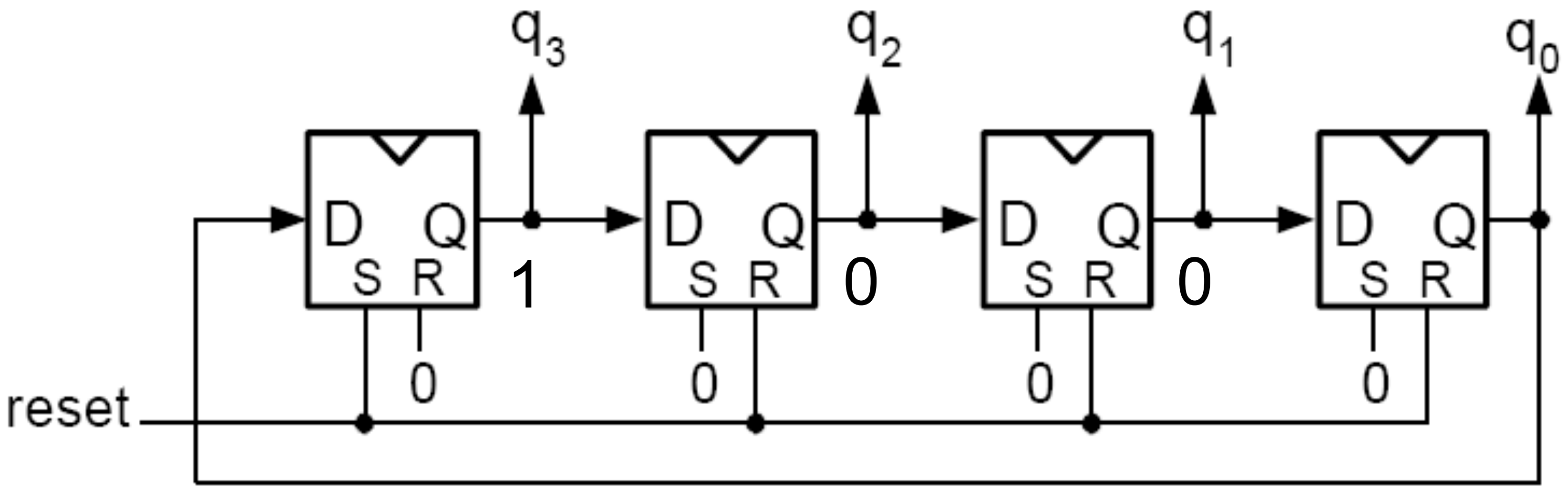
10 11

= 11 ?

Ring Counters

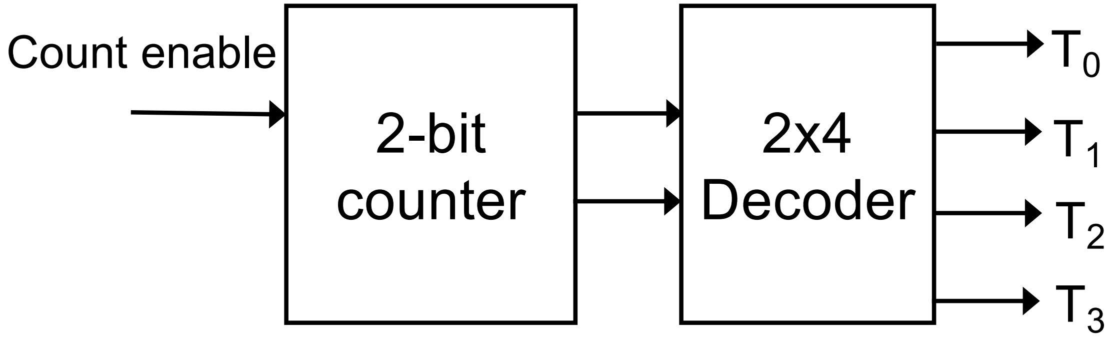
- “one-hot” counters

0001, 0010, 0100, 1000, 0001, …



Ring Counters

- Counter with a decoder at its output

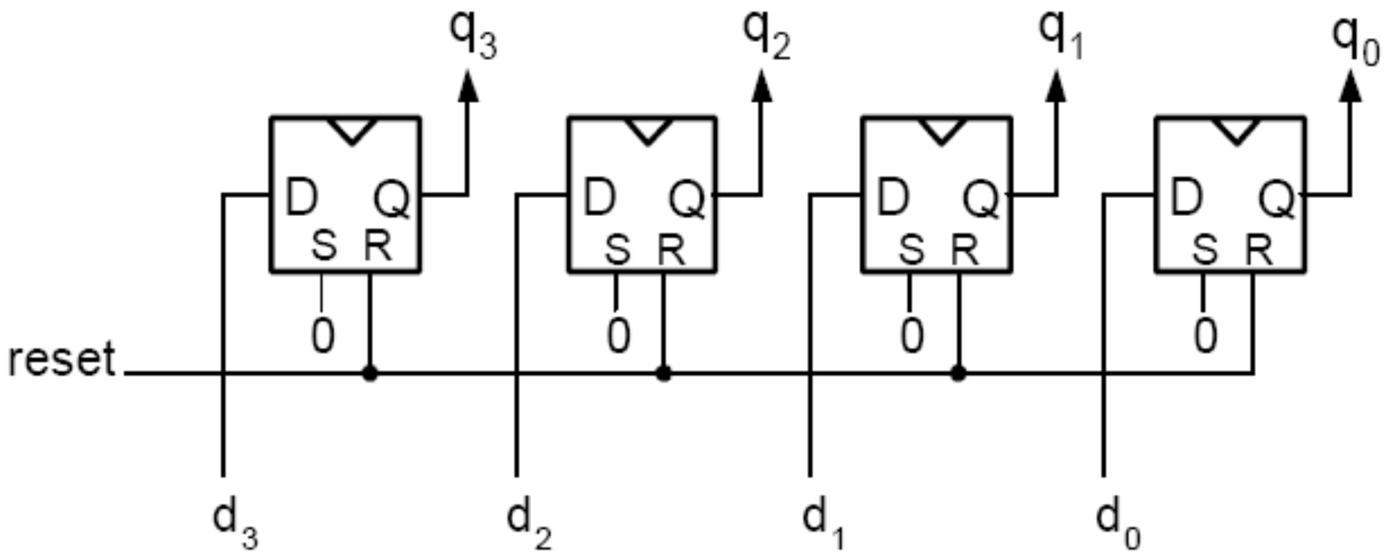


Example of Verilog Code: Counter

```
module counter (in, clk, rstb, out);
        input in, clk, rstb;
        output [3:0] out;
        reg [3:0] out;
        always @ (posedge clk or negedge rstb)
                if (rstb == 1’b0) begin
                        out <= 4’b0000;
                end
                else if (out == 4’b1111 && in== 1’b1) begin
                        out <= 4’b0000;
                end
                else if (in==1’b1) begin
                        out <= out + 1;
                end
endmodule
```

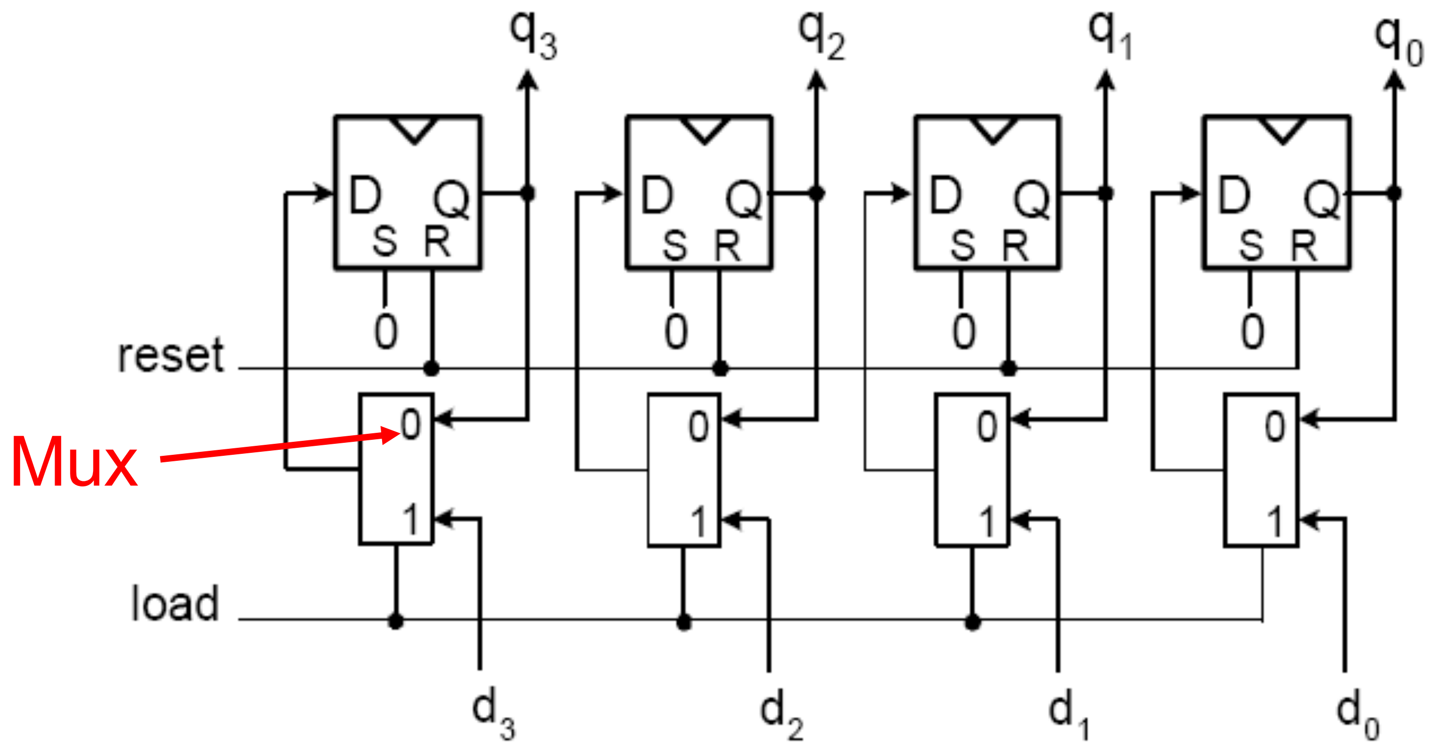
Registers-Summary

- Based on flip flops: store internal values or states



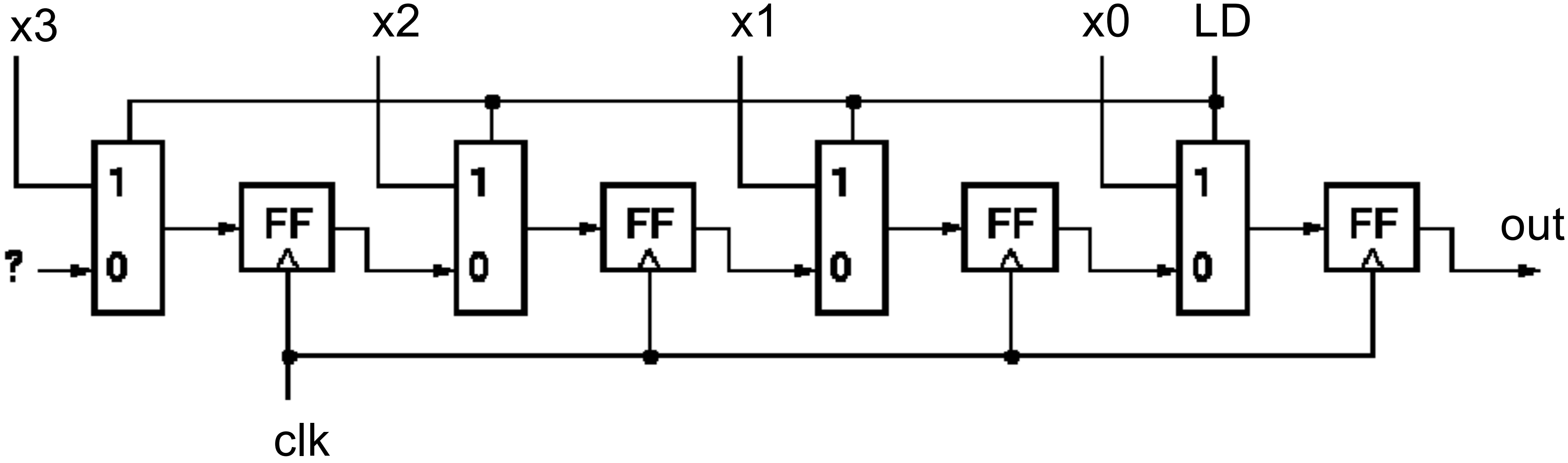
Registers

- Load Enable: load new data when load=1



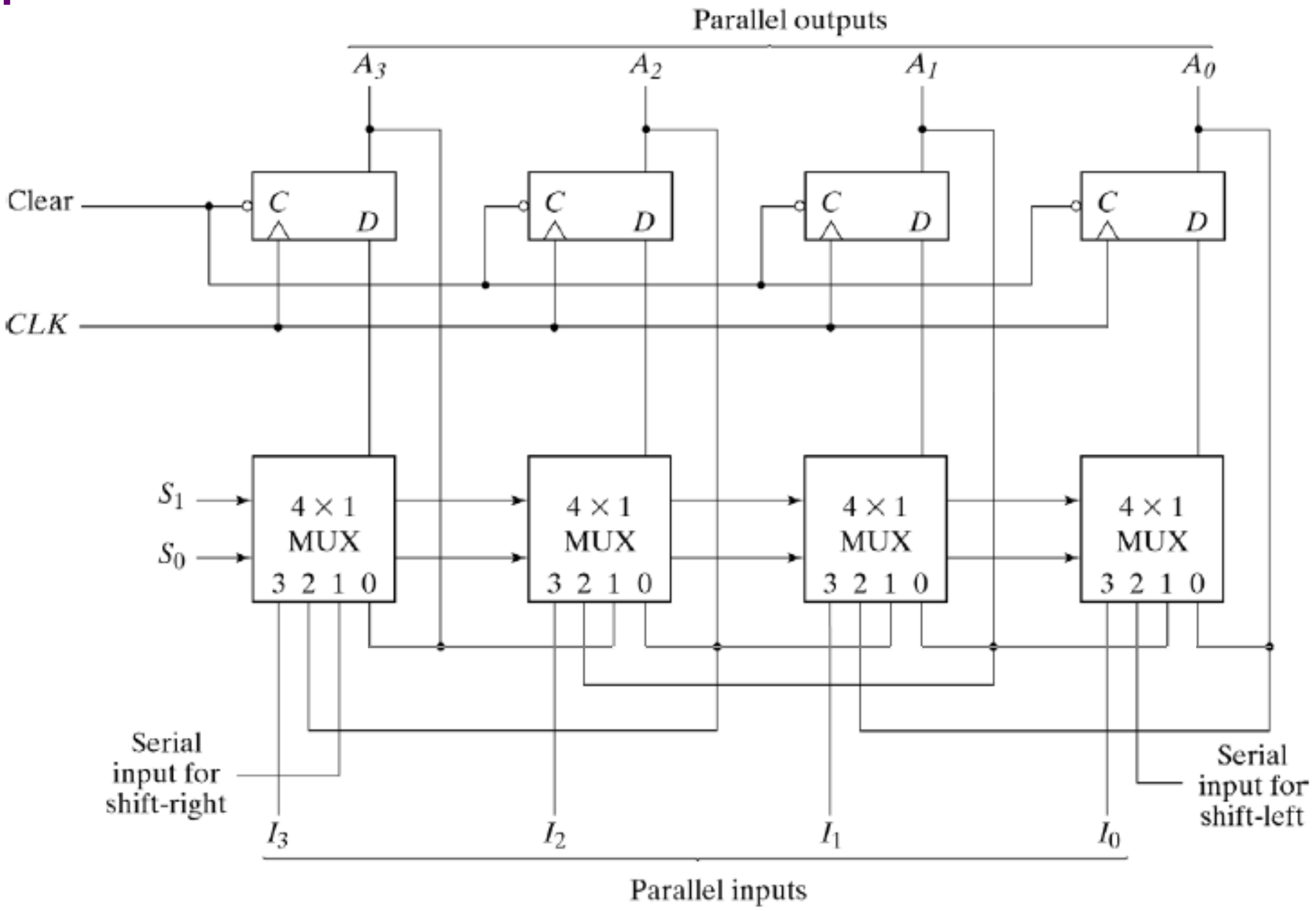
Shift Registers

- Parallel Load Shift Register



- “Parallel-to-serial converter”: parallel load data
- After loading data, shift bits by bits
- Commonly used in low speed IO, e.g. scan chains

Universal Shift Register



S1 and S0:
00: Retention
01: shift-right
10: shift-left
11: parallel load

Example of Verilog: Universal Shift Register

```
module Shift_Register( CLK,CLR,RIN,LIN,S0,S1,A,B,C,D,QA,QB,QC,QD );
    input CLK, CLR, S0, S1, RIN, LIN, A, B, C, D;
    output reg QA, QB, QC, QD;
    always @ (posedge CLK) begin
        if (CLR == 1'b1) {QA,QB,QC,QD} <= 4'b0;
        else case ({S1,S0})
                2'b00: ; // Hold
                2'b01: {QA,QB,QC,QD} <= {RIN,QA,QB,QC}; // Shift right
                2'b10: {QA,QB,QC,QD} <= {QB,QC,QD,LIN}; // Shift left
                2'b11: {QA,QB,QC,QD} <= {A,B,C,D}; // Load
                default: {QA,QB,QC,QD} <= 4'bx; // shouldn’t occur
        endcase
    end
    endmodule
```

Appendix