CSEE 3827: Fundamentals of Computer Systems, Spring 2022

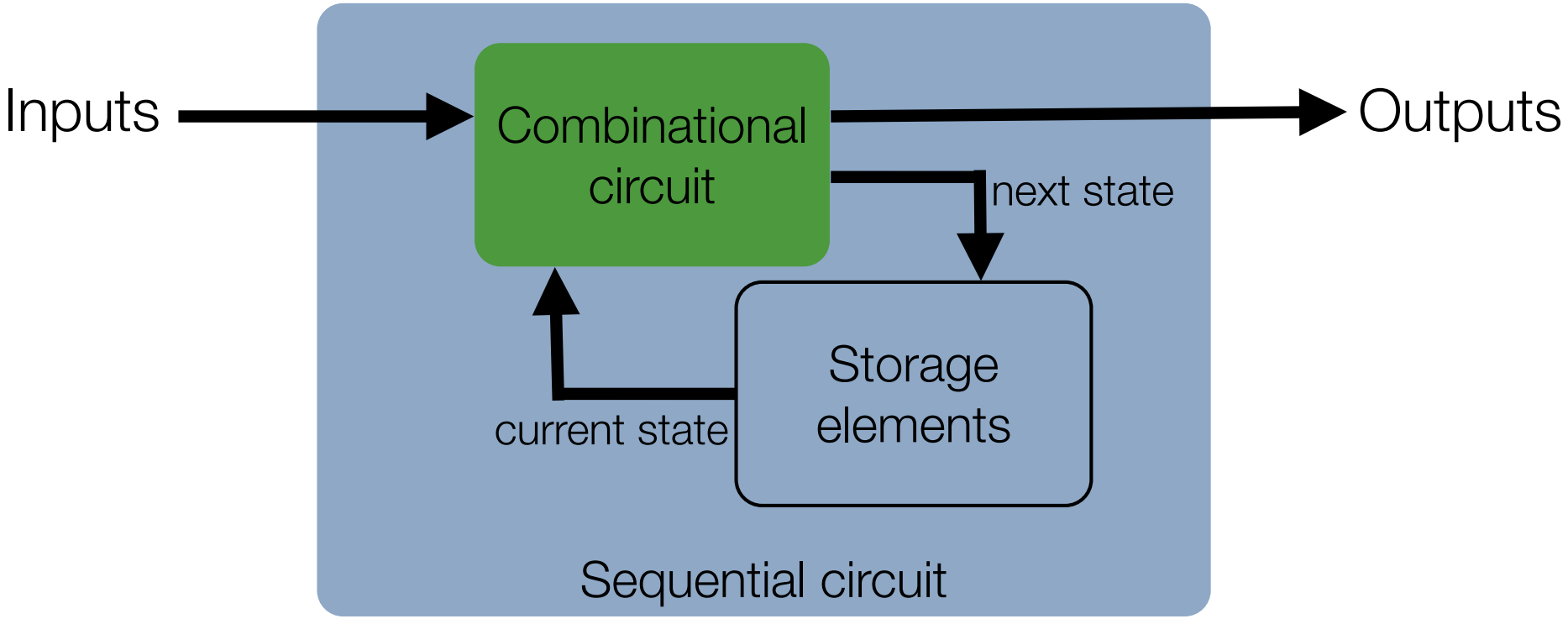
Lecture 6

Prof. Dan Rubenstein (danr@cs.columbia.edu)

Agenda (M&K 4.4-4.7, 6.1-6.3)

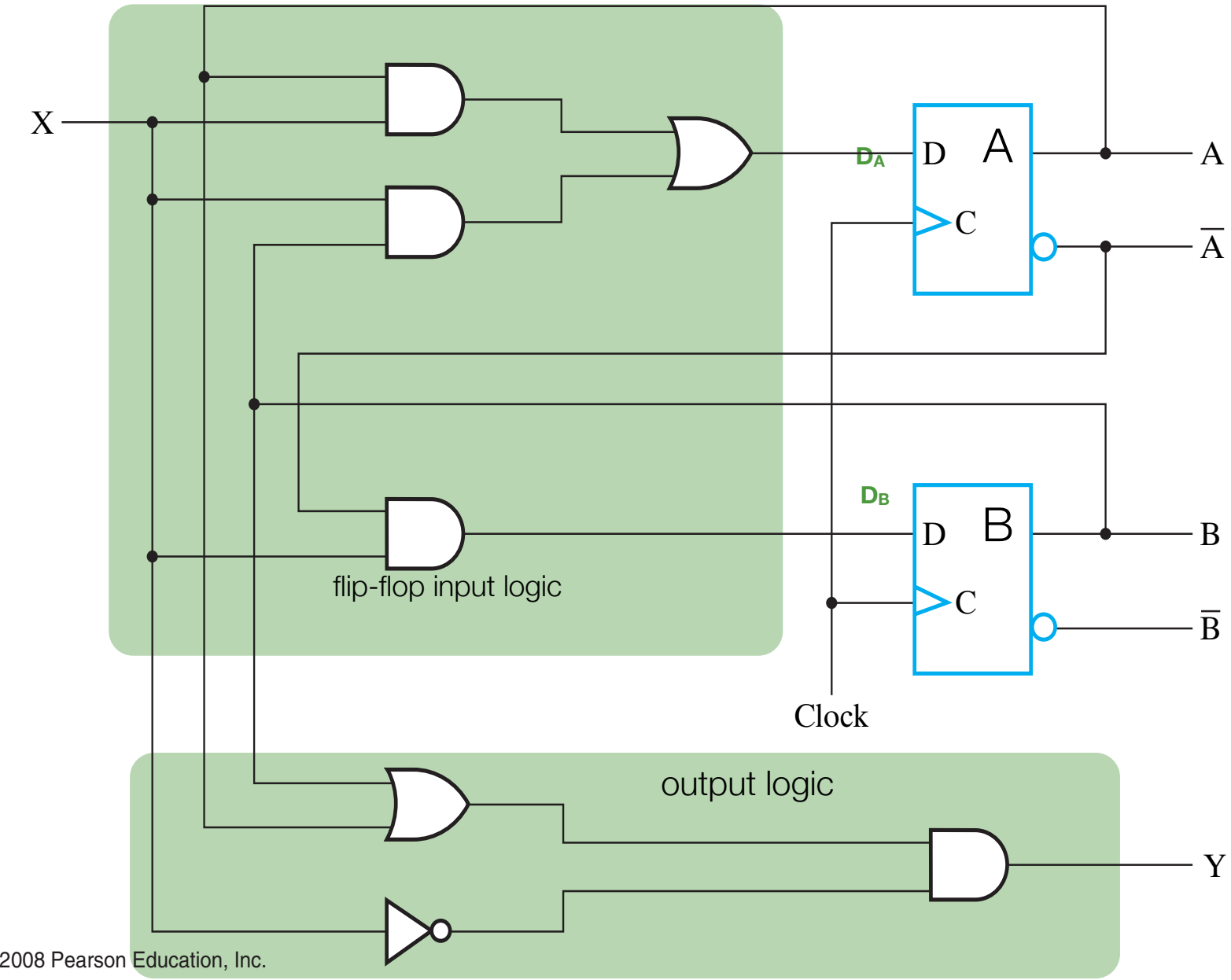
- Sequent Circuit Analysis & Design: State Machines
  - Circuit Analysis (w.r.t. time)
  - Mealy State Machines
  - Sequential Circuit Design using Flip-Flops
    - D, T, JK
- PLAs; ROM
- Register Design: Load and Transfer
  - Shifter
  - Counter
  - Adder

Recall: Sequential circuit

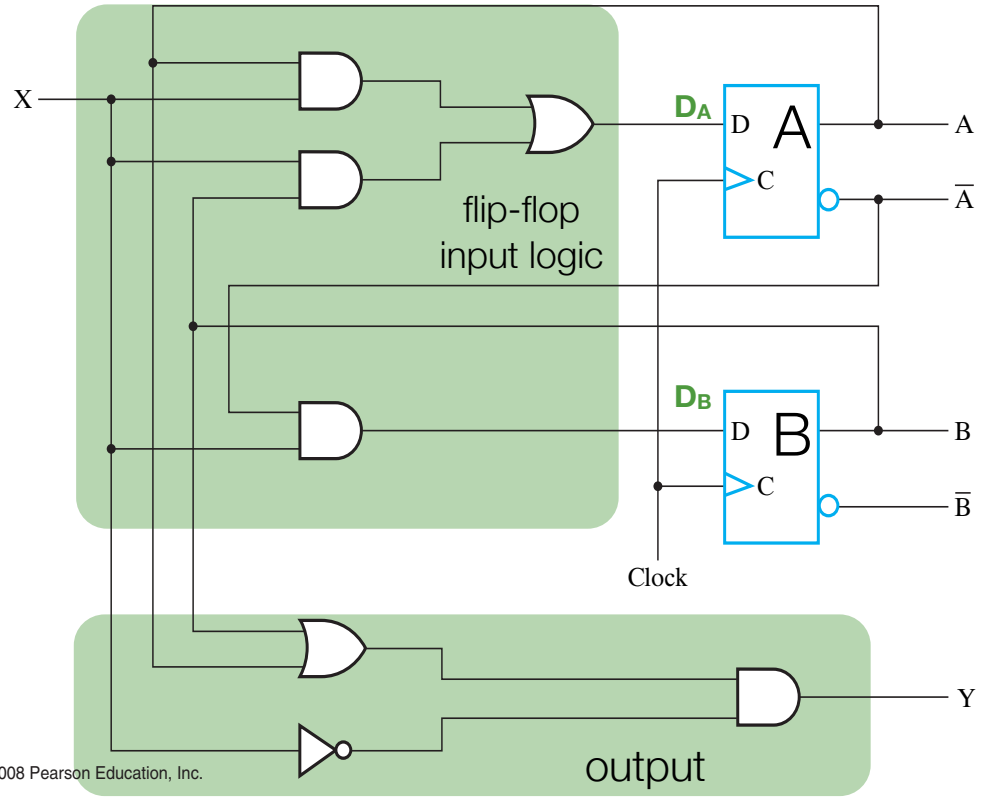


Using Flip-Flops

Sequential circuit (schematic)



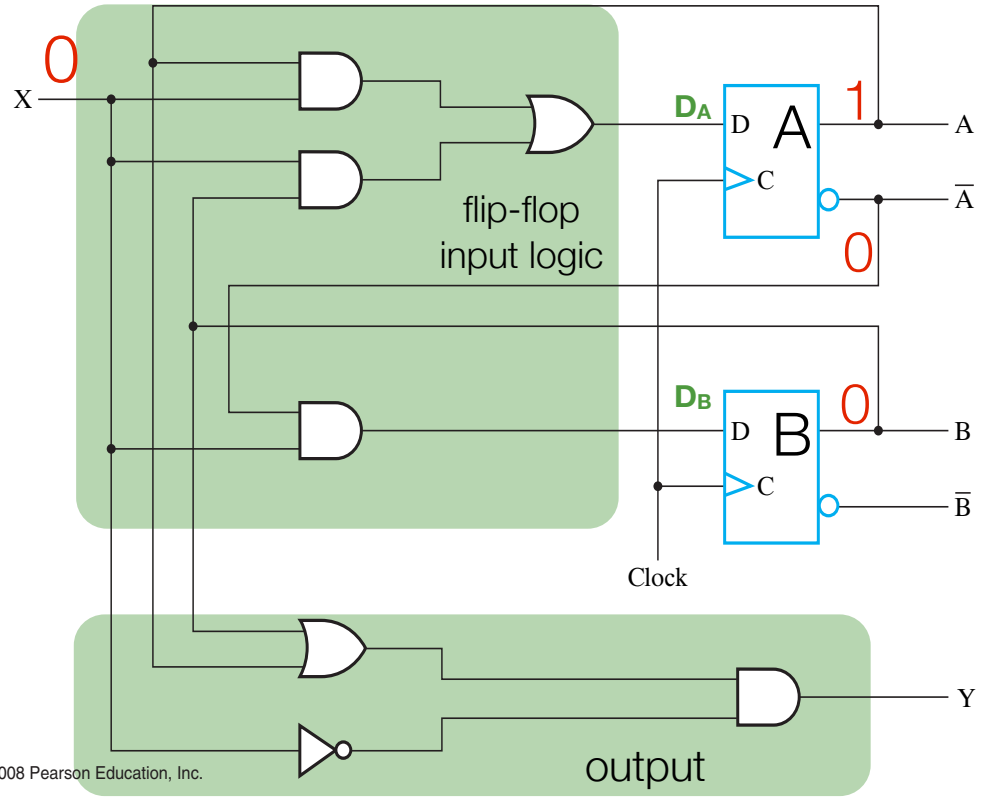
Sequential circuit (schematic)



What happens in Clock cycle t?

- X(t): an arbitrary binary input
  - can assume stays fixed for entire clock cycle
- A(t): output of top D flip-flop (fixed)
- B(t): output of bottom D flip-flop (fixed)
- Y(t): output of circuit is function of X(t), A(t), B(t)
- A(t+1): gets set (as a function of X(t), A(t), B(t))
- B(t+1): gets computed (as a function of X(t), A(t))

Sequential circuit (schematic)

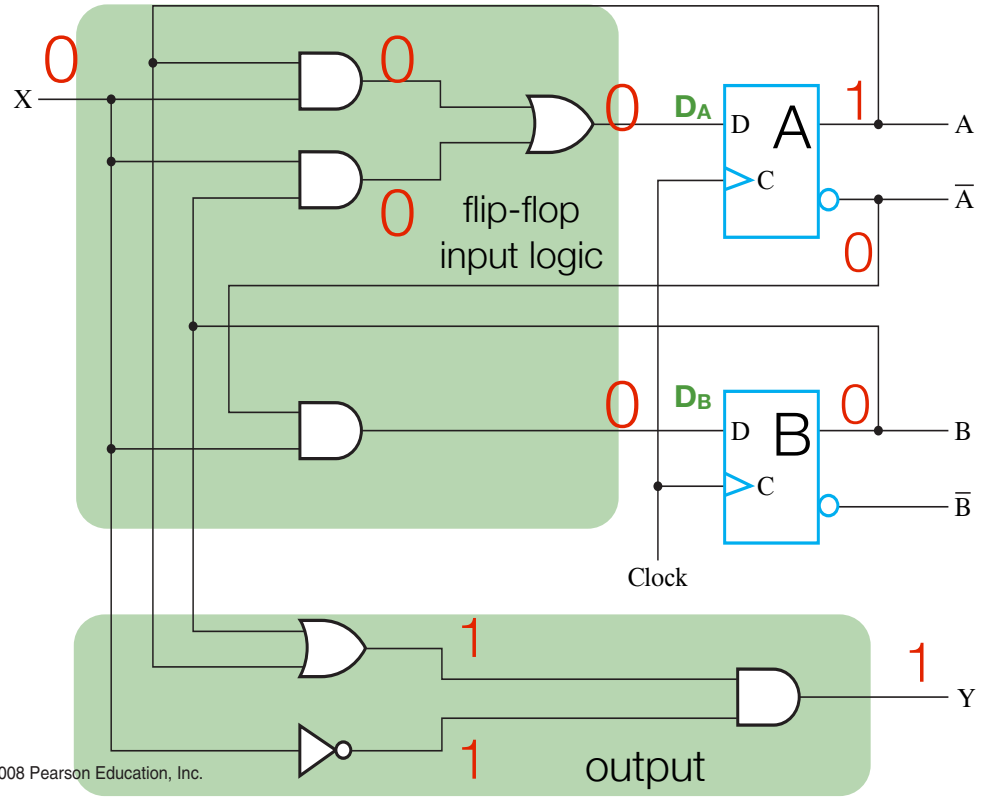


What happens in Clock cycle t?

- X(t): an arbitrary binary input
  - can assume stays fixed for entire clock cycle
- A(t): output of top D flip-flop (fixed)
- B(t): output of bottom D flip-flop (fixed)
- Y(t): output of circuit is function of X(t), A(t), B(t)
- A(t+1): gets set (as a function of X(t), A(t), B(t))
- B(t+1): gets computed (as a function of X(t), A(t))

e.g., suppose A(t)=1, B(t)=0, X(t)=0

Sequential circuit (schematic)



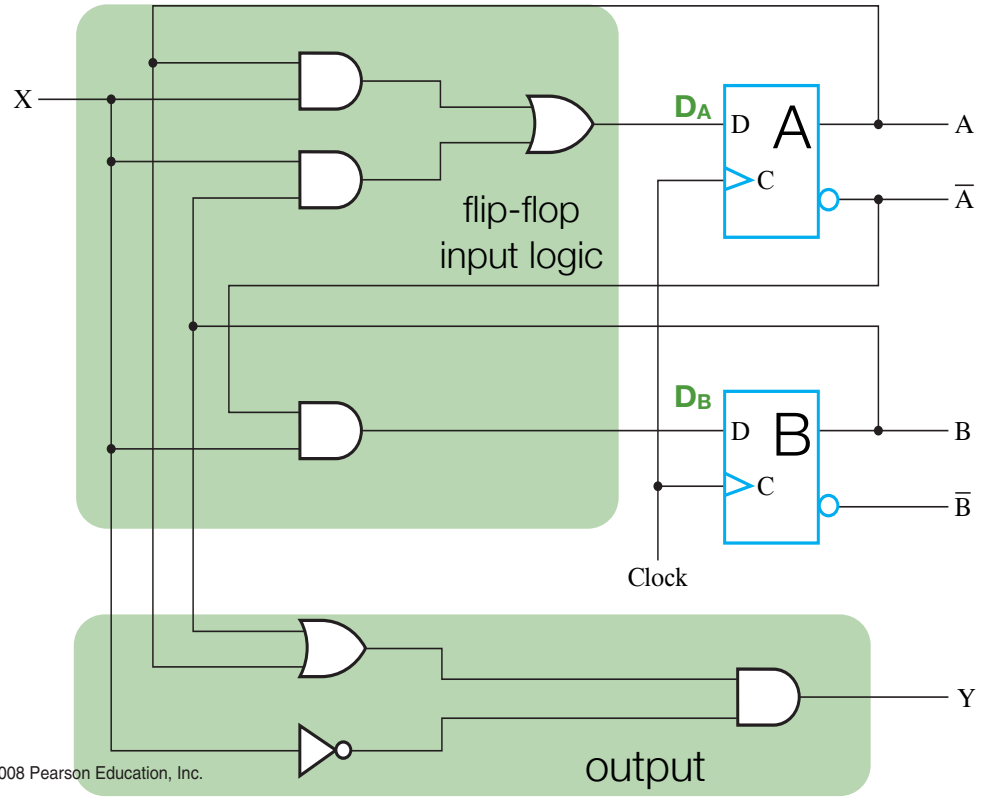
What happens in Clock cycle t?

- X(t): an arbitrary binary input
  - can assume stays fixed for entire clock cycle
- A(t): output of top D flip-flop (fixed)
- B(t): output of bottom D flip-flop (fixed)
- Y(t): output of circuit is function of X(t), A(t), B(t)
- A(t+1): gets set (as a function of X(t), A(t), B(t))
- B(t+1): gets computed (as a function of X(t), A(t))

e.g., suppose A(t)=1, B(t)=0, X(t)=0

then Y(t)=1, A(t+1)=0, B(t+1)=0

Sequential circuit (schematic)



What happens in Clock cycle t?

- X(t): an arbitrary binary input
  - can assume stays fixed for entire clock cycle
- A(t): output of top D flip-flop (fixed)
- B(t): output of bottom D flip-flop (fixed)
- Y(t): output of circuit is function of X(t), A(t), B(t)
- A(t+1): gets set (as a function of X(t), A(t), B(t))
- B(t+1): gets computed (as a function of X(t), A(t))

Algebraically:

$$A(t+1) = A(t)X(t) + X(t)B(t) \quad [A = AX+XB]$$

$$B(t+1) = \overline{A}(t)X(t) \quad [B = \overline{A}X]$$

$$Y(t) = (A(t) + B(t))\overline{X}(t) \quad [Y = (A+B)\overline{X}]$$

Sequential circuit (schematic)

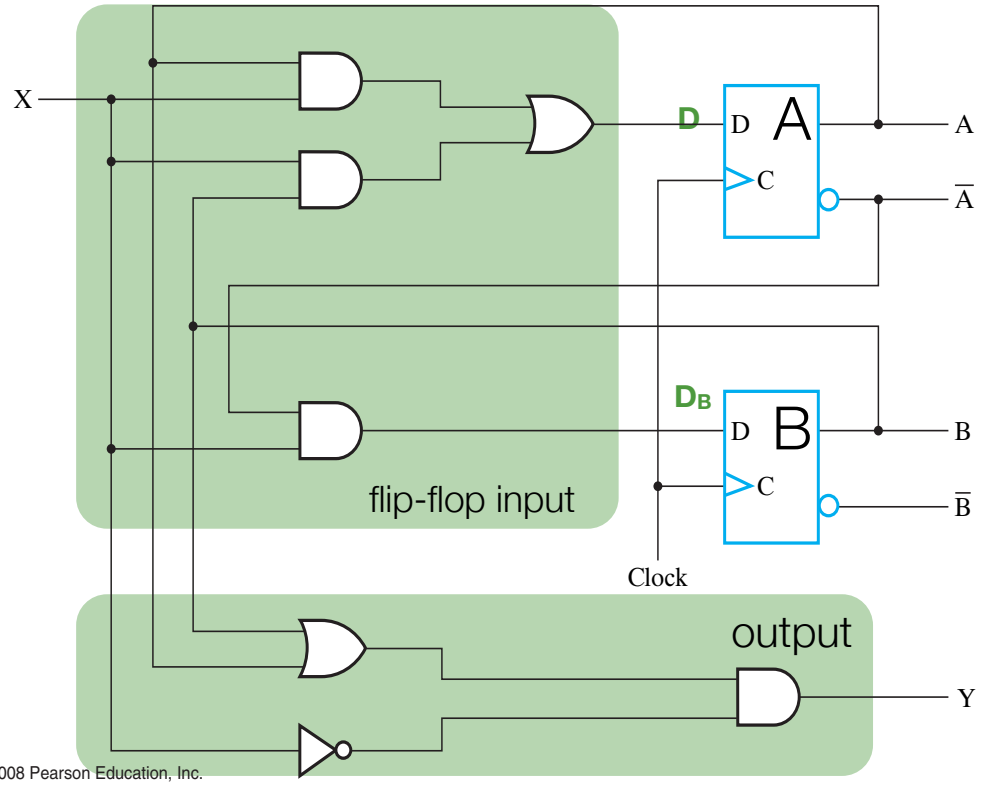


TABLE 5-1
State Table for Circuit of Figure 5-15

Present State		Input	Next State		Output
A(t)	B(t)	X(t)	A(t+1)	B(t+1)	Y(t)
0	0	0	0	0	0
0	0	1	0	1	0
0	1	0	0	0	1
0	1	1	1	1	0
1	0	0	0	0	1
1	0	1	1	0	0
1	1	0	0	0	1
1	1	1	1	0	0

← Time t → Time t+1 Time t

Sequential circuit (schematic)

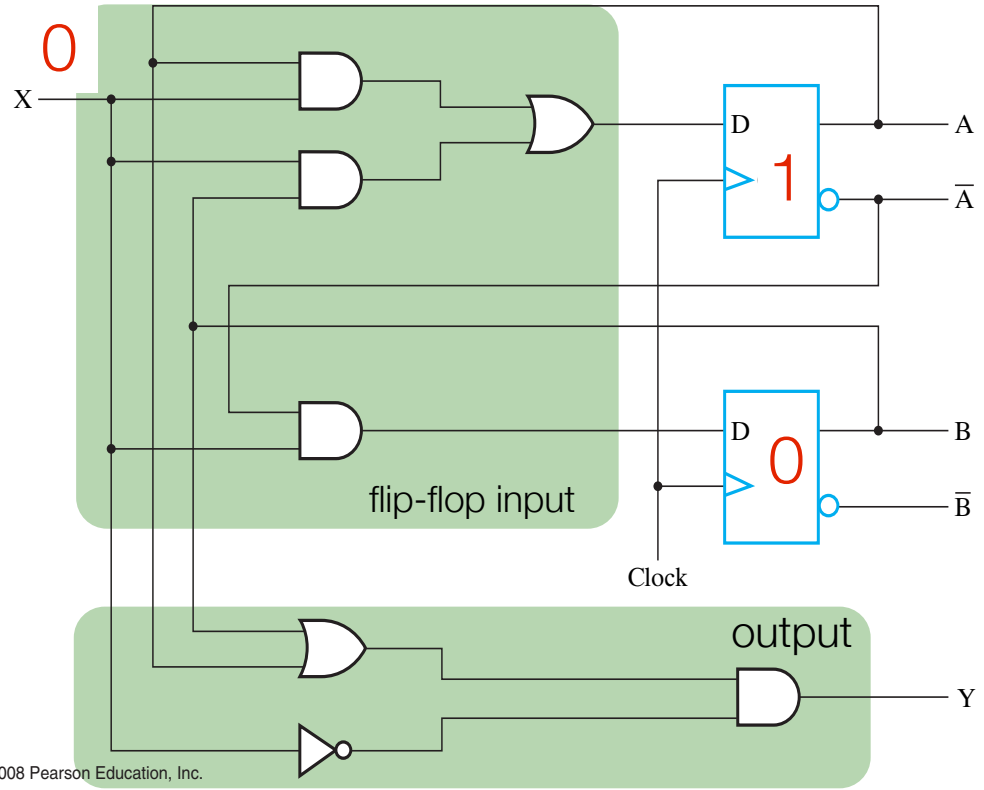


TABLE 5-1
State Table for Circuit of Figure 5-15

Present State		Input	Next State		Output
A(t)	B(t)	X(t)	A(t+1)	B(t+1)	Y(t)
0	0	0	0	0	0
0	0	1	0	1	0
0	1	0	0	0	1
0	1	1	1	1	0
1	0	0	0	0	1
1	0	1	1	0	0
1	1	0	0	0	1
1	1	1	1	0	0

⟵ Time t ⟶ Time t+1 Time t

e.g., suppose A(t)=1, B(t)=0, X(t)=0

Sequential circuit (schematic)

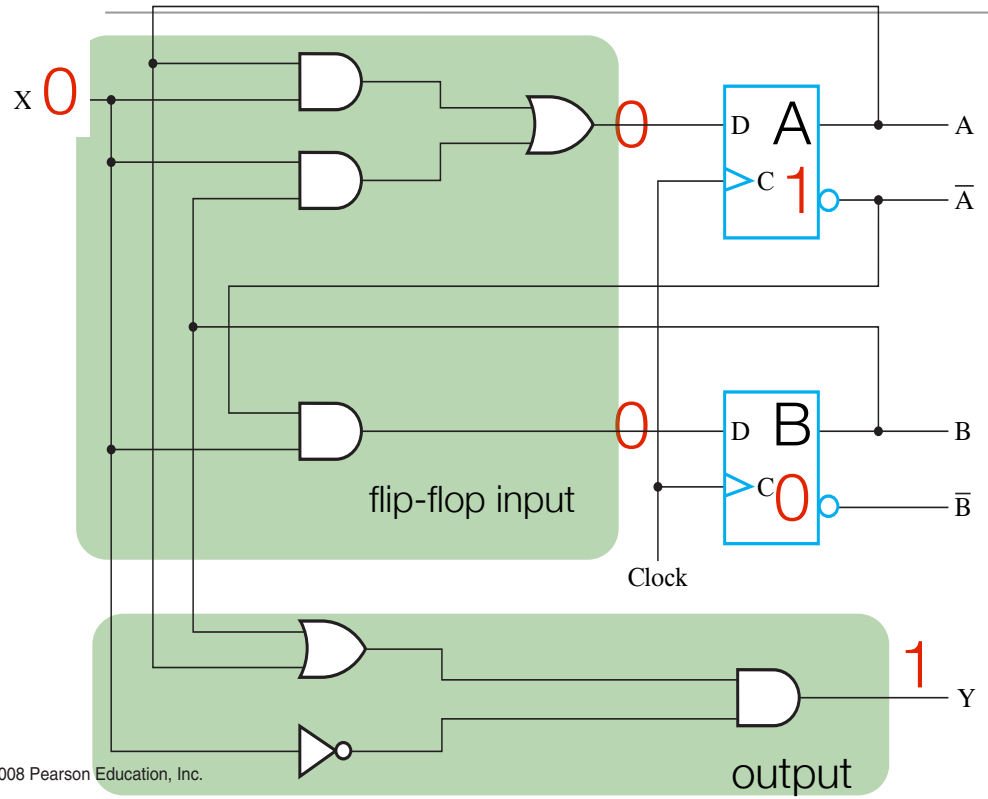


TABLE 5-1
State Table for Circuit of Figure 5-15

Present State		Input	Next State		Output
A(t)	B(t)	X(t)	A(t+1)	B(t+1)	Y(t)
0	0	0	0	0	0
0	0	1	0	1	0
0	1	0	0	0	1
0	1	1	1	1	0
1	0	0	0	0	1
1	0	1	1	0	0
1	1	0	0	0	1
1	1	1	1	0	0

← Time t → Time t+1 Time t

e.g., suppose A(t)=1, B(t)=0, X(t)=0

Then Y(t) = 1, A(t+1)=0, B(t+1)=0

Sequential circuit (schematic)

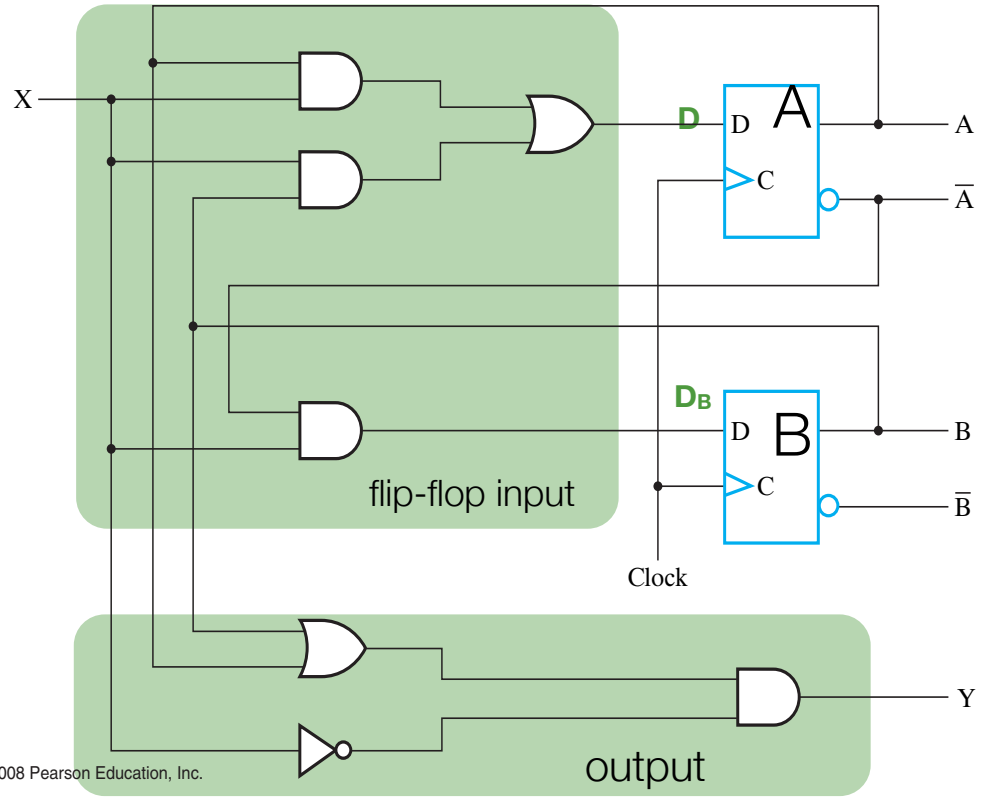


TABLE 5-1
State Table for Circuit of Figure 5-15

Present State		Input	Next State		Output
A(t)	B(t)	X(t)	A(t+1)	B(t+1)	Y(t)
0	0	0	0	0	0
0	0	1	0	1	0
0	1	0	0	0	1
0	1	1	1	1	0
1	0	0	0	0	1
1	0	1	1	0	0
1	1	0	0	0	1
1	1	1	1	0	0

← Time t → Time t+1 Time t

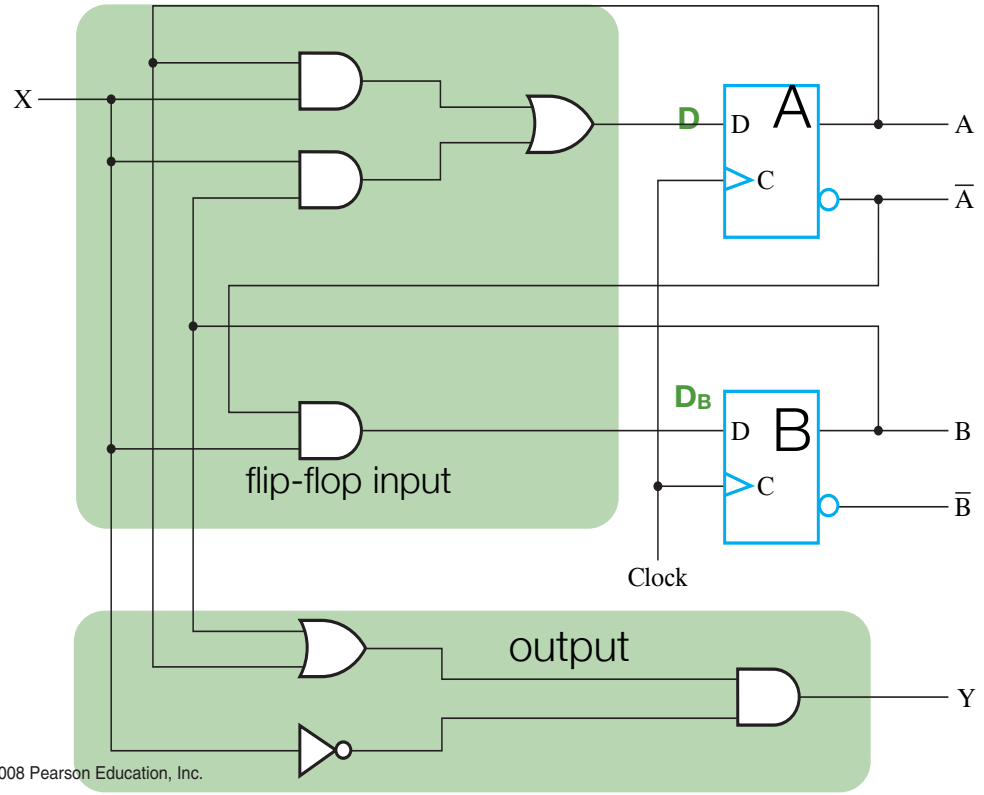
$A(t+1) = A(t)X(t) + X(t)B(t)$ [A = AX+XB]

$B(t+1) = \overline{A}(t)X(t)$ [B = $\overline{A}$X]

$Y(t) = (A(t) + B(t))\overline{X}(t)$ [Y = (A+B)$\overline{X}$]

At time t, next cycle's (t+1) FF's are set, but combinational output uses current (cycle t) FF values

Sequential circuit (schematic)



Example

t	A(t)	B(t)	X(t)	Y(t)
0	0	0	0	0
1	0	0	1	0
2	0	1	1	0
3	1	1	0	1
4	0	0	1	0
5	0	1	0	1
6	0	0	0	0
7	0	0	1	0
8	0	1	1	0
9	1	1	0	1

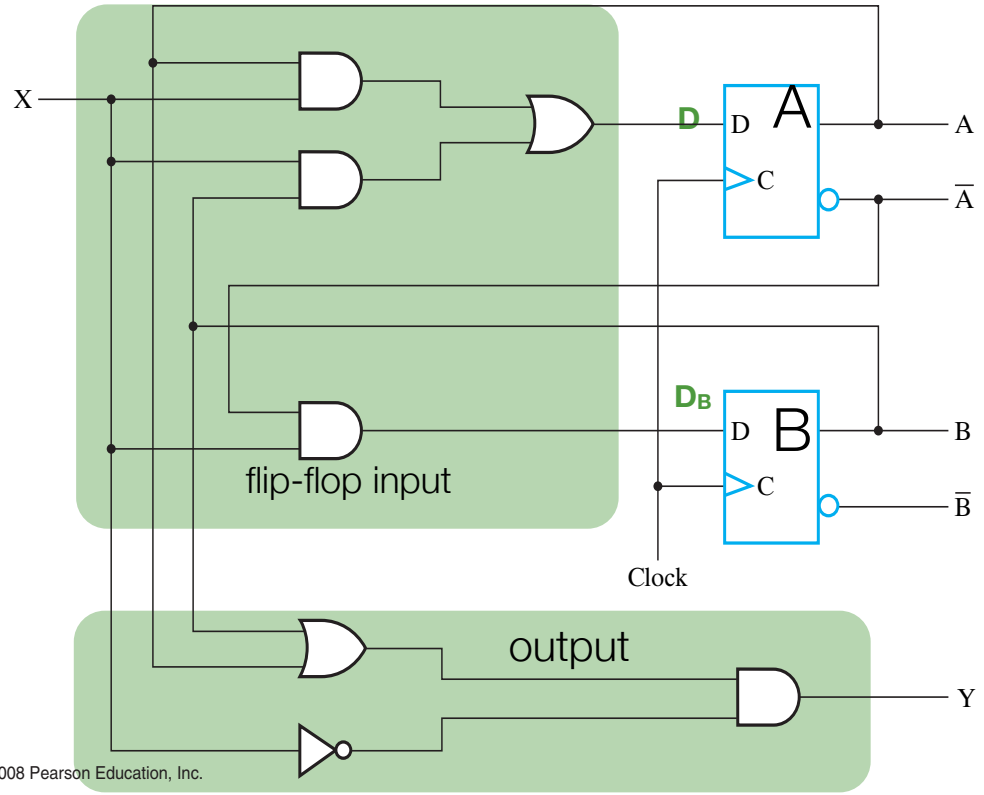
$A(t+1) = A(t)X(t) + X(t)B(t)$ [$A = AX+XB$]

$B(t+1) = \overline{A}(t)X(t)$ [$B = \overline{A}X$]

$Y(t) = (A(t) + B(t))\overline{X}(t)$ [$Y = (A+B)\overline{X}$]

Showing behavior over time - not a truth table!

Sequential circuit (schematic)



$$A(t+1) = A(t)X(t) + X(t)B(t) \quad [A = AX+XB]$$

$$B(t+1) = \overline{A}(t)X(t) \quad [B = \overline{A}X]$$

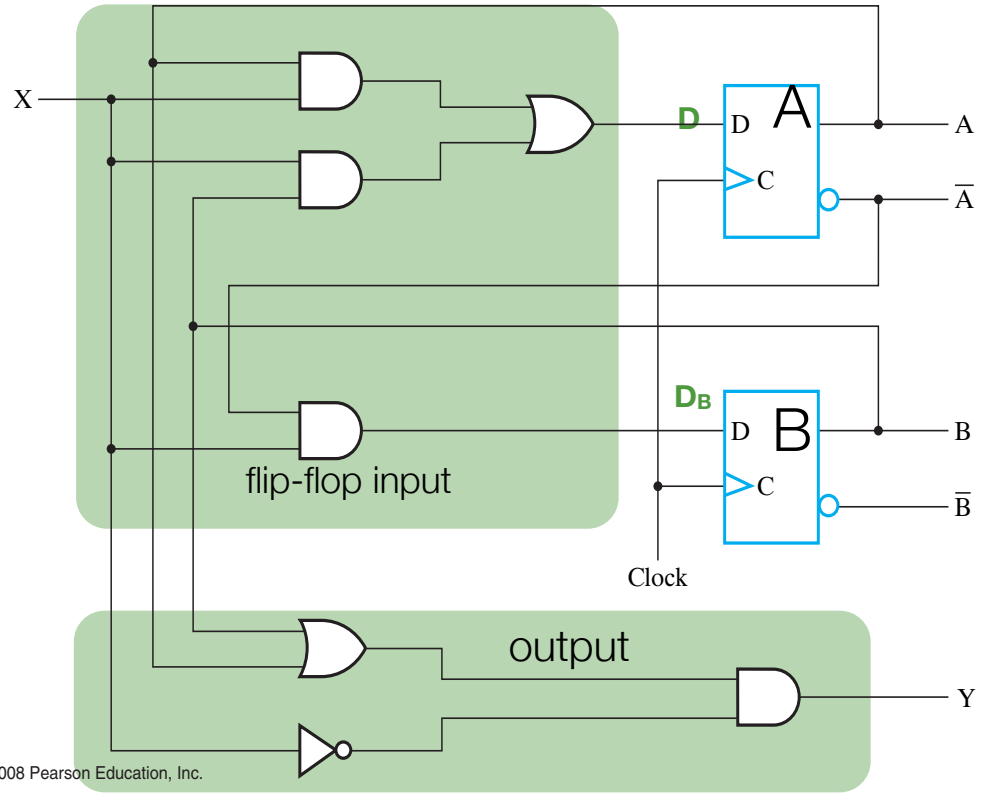
$$Y(t) = (A(t) + B(t))\overline{X}(t) \quad [Y = (A+B)\overline{X}]$$

Example

t	A(t)	B(t)	X(t)	Y(t)
0	0	0	0	0
1	0	0	1	0
2	0	1	1	0
3	1	1	0	1
4	0	0	1	0
5	0	1	0	1
6	0	0	0	0
7	0	0	1	0
8	0	1	1	0
9	1	1	0	1

Green: current clock cycle
Red: current values to be used as function parameters
Blue: values being determined

Sequential circuit (schematic)



$$A(t+1) = A(t)X(t) + X(t)B(t) \quad [A = AX+XB]$$

$$B(t+1) = \overline{A}(t)X(t) \quad [B = \overline{A}X]$$

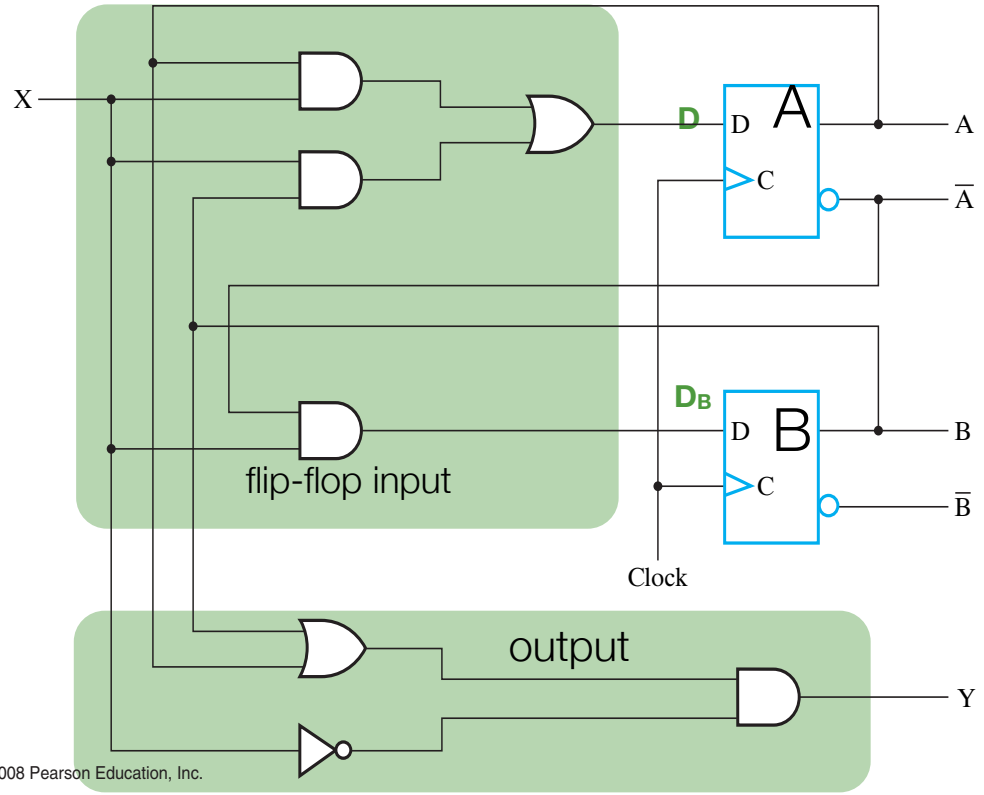
$$Y(t) = (A(t) + B(t))\overline{X}(t) \quad [Y = (A+B)\overline{X}]$$

Example

t	A(t)	B(t)	X(t)	Y(t)
0	0	0	0	0
1	0	0	1	0
2	0	1	1	0
3	1	1	0	1
4	0	0	1	0
5	0	1	0	1
6	0	0	0	0
7	0	0	1	0
8	0	1	1	0
9	1	1	0	1

Green: current clock cycle
Red: current values to be used as function parameters
Blue: values being determined

Sequential circuit (schematic)



Example

t	A(t)	B(t)	X(t)	Y(t)
0	0	0	0	0
1	0	0	1	0
2	0	1	1	0
3	1	1	0	1
4	0	0	1	0
5	0	1	0	1
6	0	0	0	0
7	0	0	1	0
8	0	1	1	0
9	1	1	0	1

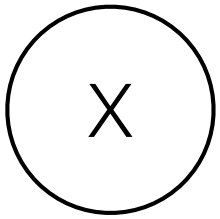
Green: current clock cycle
Red: current values to be used as function parameters
Blue: values being determined

$$A(t+1) = A(t)X(t) + X(t)B(t) \quad [A = AX+XB]$$
$$B(t+1) = \overline{A}(t)X(t) \quad [B = \overline{A}X]$$
$$Y(t) = (A(t) + B(t))\overline{X}(t) \quad [Y = (A+B)\overline{X}]$$

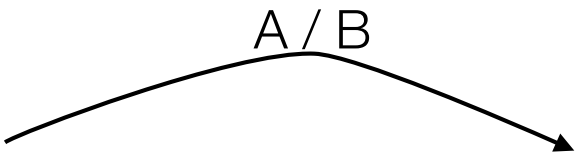
State Machines

State Machines

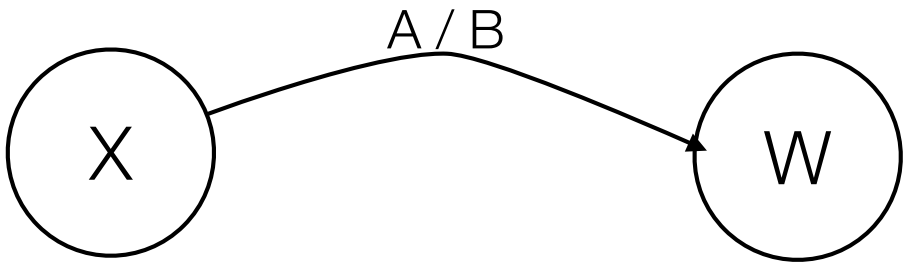
- We study the Mealy form (other form is Moore)



State with label “X”



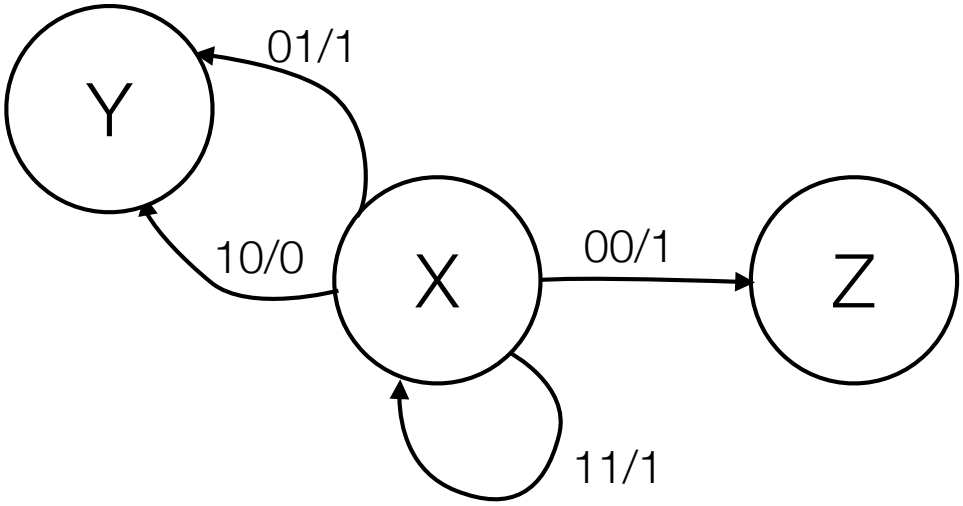
Transition with input “A”, outputs “B”



Time t: in state X, get input A, output B

Time t+1: in state W

Example: state X with 2 inputs, 1 output



Example State Machine

- 1 input, 1 output
- Let X = # of 1’s input so far, Y = # of 0’s input so far (incl current input).
- Output 1 whenever X-Y (including time t input) = 0 (mod 3)

Example State Machine

- 1 input, 1 output
- Let X = # of 1's input so far, Y = # of 0's input so far (incl current input).
- Output 1 whenever X-Y (including time t input) = 0 (mod 3)

- Pseudocode:

```
Int X=0; int Y=0; bool j;
While (1==1){
    Input j;
    If (j==1){
        X = X+1;
    }
    Else Y=Y+1;
    If (X-Y)%3 ==0{
        Output 1;
    }
    Else Output 0;
}
```

Over time, X and Y might grow very large, e.g., after 1,000,000 iterations, X+Y=1,000,000

Example State Machine

- 1 input, 1 output
- Let X = # of 1's input so far, Y = # of 0's input so far (incl current input).
- Output 1 whenever X-Y (including time t input) = 0 (mod 3)

- Pseudocode:

```
Int X=0; int Y=0; bool j;
While (1==1){
      Input j;
      If (j==1){
            X = X+1;
      }
      Else Y=Y+1;
      If (X-Y)%3 ==0{
            Output 1;
      }
      Else Output 0;
}
```

- Reduced State Pseudocode:

```
Int D=0; bool j;
While (1==1){
      Input j;
      If (j==1){
            D = D+1;
            If (D==3){
                  D=0;
            }
      }
      Else D=D-1;
      If (D==-1){
            D=2;
      }
      If D ==0{
            Output 1;
      }
      Else Output 0;
}
```

Just have variable D that keeps track of (X-Y) mod 3

Example State Machine

- 1 input, 1 output
- Let X = # of 1's input so far, Y = # of 0's input so far (incl current input).
- Output 1 whenever X-Y (including time t input) = 0 (mod 3)

- Pseudocode:

```
Int X=0; int Y=0; bool j;
While (1==1){
      Input j;
      If (j==1){
            X = X+1;
      }
      Else Y=Y+1;
      If (X-Y)%3 ==0{
            Output 1;
      }
      Else Output 0;
}
```

- Reduced State Pseudocode:

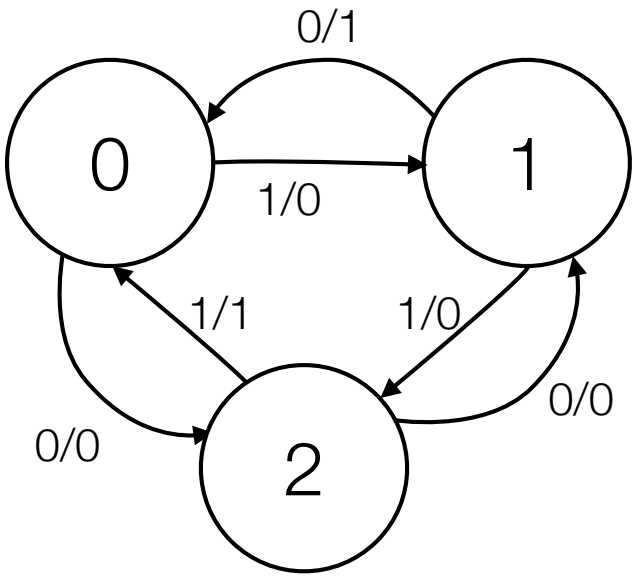
```
Int D=0; bool j;
While (1==1){
      Input j;
      If (j==1){
            D = D+1;
            If (D==3){
                  D=0;
            }
      }
      Else D=D-1;
      If (D==-1){
            D=2;
      }
      If D ==0{
            Output 1;
      }
      Else Output 0;
}
```

Just have variable D that keeps track of (X-Y) mod 3

D could be implemented using 2 bits (takes values 0,1,2)

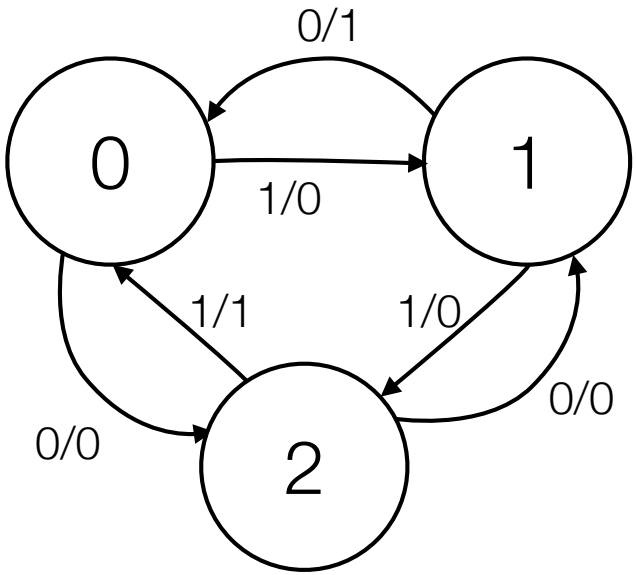
Example State Machine

- 1 input, 1 output
- Let X = # of 1's input so far, Y = # of 0's input so far (incl current input).
- Output 1 whenever X-Y (including time t input) = 0 (mod 3)



"From" State label is value of X-Y mod 3 not including current input.
"To" State is the value including current input

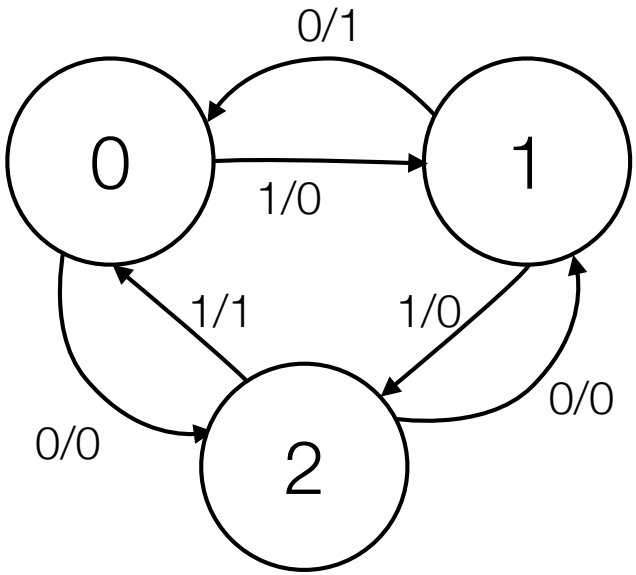
Example of the “X-Y = 0 mod 3”



t	X(t)	Y(t)	In	X(t+1)	Y(t+1)
0	0	0	0	0	1
1	0	1	1	1	1
2	1	1	1	2	1
3	2	1	0	2	2
4	2	2	1	3	2
5	3	2	1	4	2
6	4	2	0	4	3
7	4	3	1	5	3
8	5	3	0	5	4

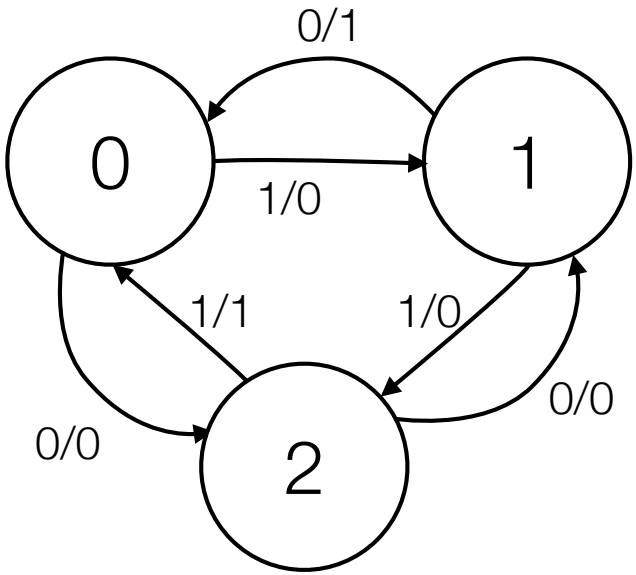
Some arbitrarily chosen input sequence

Example of the “X-Y = 0 mod 3”



t	X(t)	Y(t)	In	X(t+1)	Y(t+1)
0	0	0	0	0	1
1	0	1	1	1	1
2	1	1	1	2	1
3	2	1	0	2	2
4	2	2	1	3	2
5	3	2	1	4	2
6	4	2	0	4	3
7	4	3	1	5	3
8	5	3	0	5	4

Example of the “X-Y = 0 mod 3”

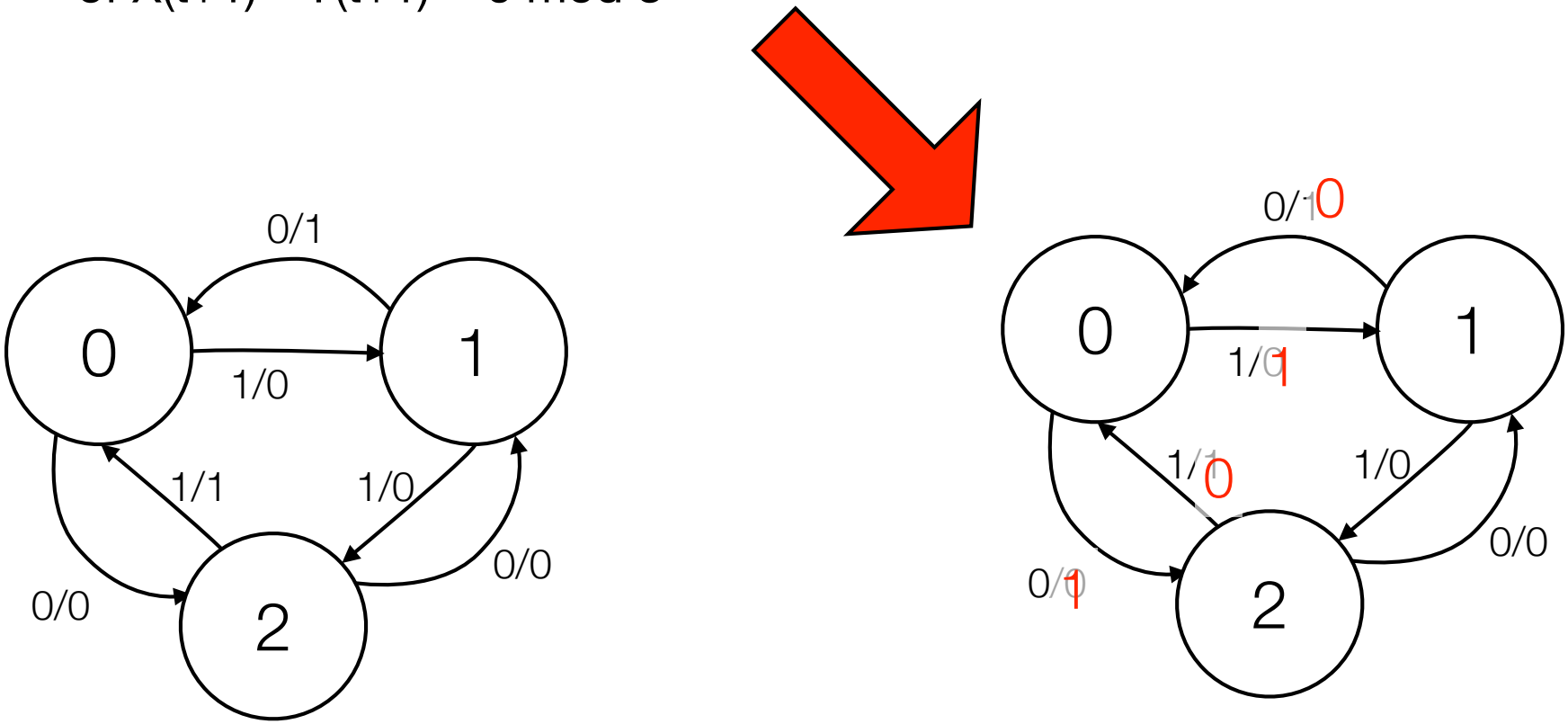


t	X(t)	Y(t)	In(t)	X-Y mod 3 (incl. In(t))	Out(t)	X(t+1)	Y(t+1)
0	0	0	0	2	0	0	1
1	0	1	1	0 →	1	1	1
2	1	1	1	1	0	2	1
3	2	1	0	0 →	1	2	2
4	2	2	1	1	0	3	2
5	3	2	1	2	0	4	2
6	4	2	0	1	0	4	3
7	4	3	1	2	0	5	3
8	5	3	0	1	0	5	4

As constructed: Out(t) = 1 when X(t+1) - Y(t+1) = 0 mod 3

Aside: alternate design for Out(t)

- Can modify state machine so that Out(t) = 1 when X(t) - Y(t) = 0 mod 3 instead of X(t+1) - Y(t+1) = 0 mod 3



Instead of output = 1 when entering the “0” state, output = 1 when leaving the 0 state

Now, outputs 1 when X-Y=0 mod 3, excluding current input

Example State Machine

- 1 input, 1 output
- Let X = # of 1's input so far, Y = # of 0's input so far (incl current input).
- Output 1 whenever X-Y (including time t input) = 0 (mod 3)

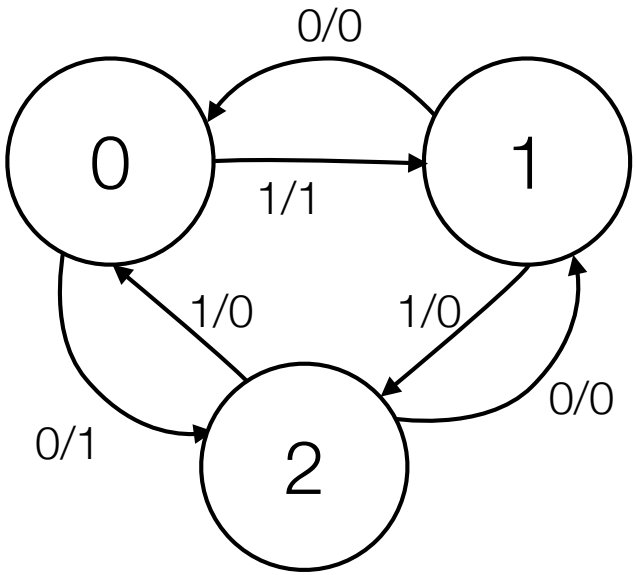
- Pseudocode:

```
Int X=0; int Y=0; bool j;
While (1==1){
    Input j;
    If (j==1){
        X = X+1;
    }
    Else Y=Y+1;
    If (X-Y)%3 ==0{
        Output 1;
    }
    Else Output 0;
}
```

- Reduced State Pseudocode:

```
Int D=0; bool j;
While (1==1){
    Input j;
    If (j==1){
        D = D+1;
        If (D==3){
            D=0;
        }
    }
    Else D=D-1;
    If (D==-1){
        D=2;
    }
    If D ==0{
        Output 1;
    }
    Else Output 0;
}
```

Example of the alternate “X-Y = 0 mod 3”

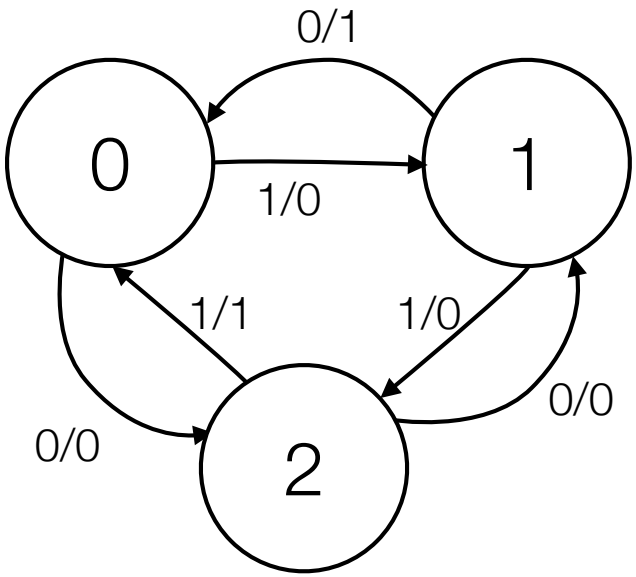


t	X(t)	Y(t)	In(t)	X(t+1)	Y(t+1)	X-Y mod 3 (time t)	Out(t)
0	0	0	0	0	1	0	1
1	0	1	1	1	1	2	0
2	1	1	1	2	1	0	1
3	2	1	0	2	2	1	0
4	2	2	1	3	2	0	1
5	3	2	1	4	2	1	0
6	4	2	0	4	3	2	0
7	4	3	1	5	3	1	0
8	5	3	0	5	4	2	0

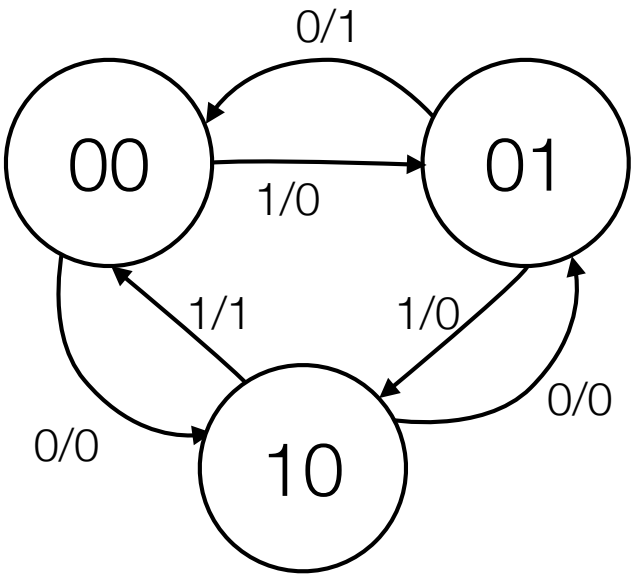
As constructed: Out(t) = 1 when X(t+1) - Y(t+1) = 0 mod 3

Back to Original Example State Machine

- 1 input, 1 output
- Let X = # of 1's input so far, Y = # of 0's input so far.
- Output 1 whenever X-Y (including time t input) = 0 (mod 3)



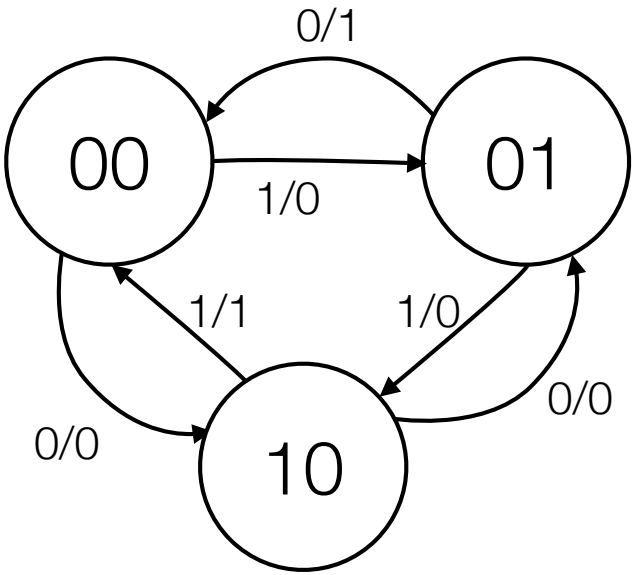
State label is current value of X-Y mod 3



Binary Labeling

From State Machine to Sequential Circuit

- Flip-Flop state represents current “state” of the State Machine
  - 1 FF per bit in the state representation



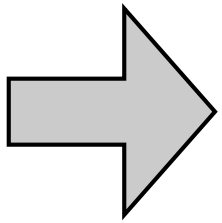
Let FF A store the high bit, B the low bit

A_{cur}	B_{cur}	In	A_{next}	B_{next}	Out
0	0	0	1	0	0
0	0	1	0	1	0
0	1	0	0	0	1
0	1	1	1	0	0
1	0	0	0	1	0
1	0	1	0	0	1
1	1	0	X	X	X
1	1	1	X	X	X

Design with D Flip-Flops

- D FF's are easy: we input the value to the FF that we want it set to

A_{cur}	B_{cur}	In	A_{next}	B_{next}	Out
0	0	0	1	0	0
0	0	1	0	1	0
0	1	0	0	0	1
0	1	1	1	0	0
1	0	0	0	1	0
1	0	1	0	0	1
1	1	0	X	X	X
1	1	1	X	X	X



A_{cur}	B_{cur}	In	A_{next}	D_A	B_{next}	D_B	Out
0	0	0	1	1	0	0	0
0	0	1	0	0	1	1	0
0	1	0	0	0	0	0	1
0	1	1	1	1	0	0	0
1	0	0	0	0	1	1	0
1	0	1	0	0	0	0	1
1	1	0	X	X	X	X	X
1	1	1	X	X	X	X	X

New columns indicate what values feed into D FF (mimic "next" values for that FF)

Design with D Flip-Flops Cont'd

- Build K-Maps, get equations for output and FF input vals (in terms of inputs and previous F vals)

A_{cur}	B_{cur}	In	A_{next}	D_A	B_{next}	D_B	Out
0	0	0	1	1	0	0	0
0	0	1	0	0	1	1	0
0	1	0	0	0	0	0	1
0	1	1	1	1	0	0	0
1	0	0	0	0	1	1	0
1	0	1	0	0	0	0	1
1	1	0	X	X	X	X	X
1	1	1	X	X	X	X	X

D_A:

A_{cur} \ In	1	0	1	0
0	0	X	X	
	B_{cur}			

D_B:

A_{cur} \ In	0	1	0	0
1	0	X	X	
	B_{cur}			

Out:

A_{cur} \ In	0	0	0	1
0	1	X	X	
	B_{cur}			

$$D_A = \bar{A}\bar{B}\bar{I} + BI$$

$$D_B = A\bar{I} + \bar{A}\bar{B}I$$

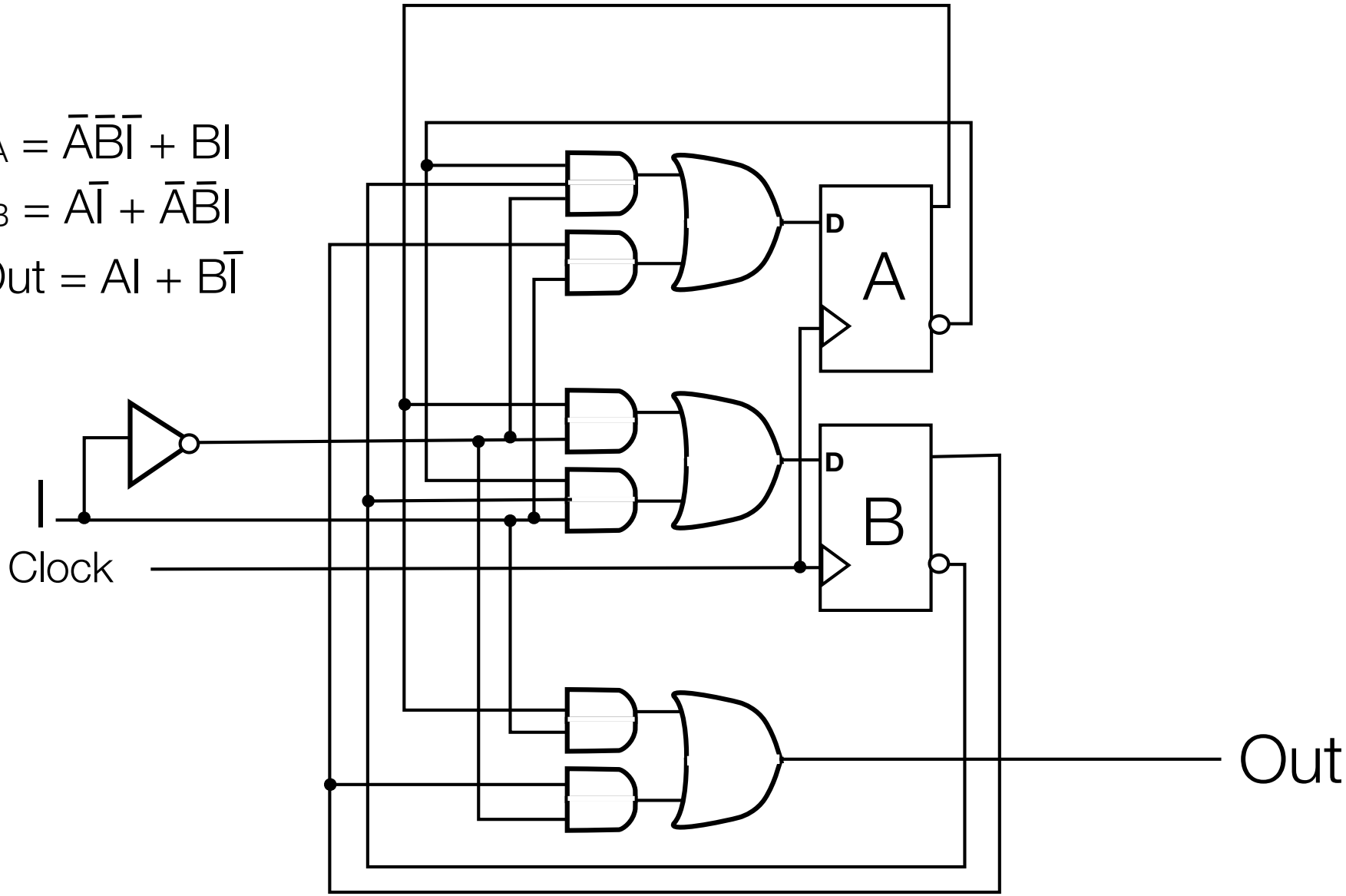
$$Out = AI + B\bar{I}$$

Design with D FF's cont'd

$D_A = \bar{A}\bar{B}\bar{I} + BI$

$D_B = A\bar{I} + \bar{A}\bar{B}I$

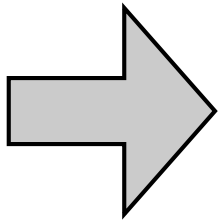
$Out = AI + B\bar{I}$



Design with T Flip Flops

- Value fed into T FF is 0 if FF should maintain value, 1 if it should flop

A_{cur}	B_{cur}	In	A_{next}	B_{next}	Out
0	0	0	1	0	0
0	0	1	0	1	0
0	1	0	0	0	1
0	1	1	1	0	0
1	0	0	0	1	0
1	0	1	0	0	1
1	1	0	X	X	X
1	1	1	X	X	X



A_{cur}	B_{cur}	In	A_{next}	T_A	B_{next}	T_B	Out
0	0	0	1	1	0		0
0	0	1	0	0	1		0
0	1	0	0	0	0		1
0	1	1	1	1	0		0
1	0	0	0	1	1		0
1	0	1	0	1	0		1
1	1	0	X	X	X		X
1	1	1	X	X	X		X

T_A:

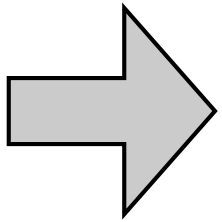
A_{cur} \ In \ B_{cur}	1	0	1	0
1	1	X	X	

$$T_A = A + BI + \bar{B}\bar{I}$$

Design with T Flip Flops

- Value fed into T FF is 0 if FF should maintain value, 1 if it should flop

A_{cur}	B_{cur}	In	A_{next}	B_{next}	Out
0	0	0	1	0	0
0	0	1	0	1	0
0	1	0	0	0	1
0	1	1	1	0	0
1	0	0	0	1	0
1	0	1	0	0	1
1	1	0	X	X	X
1	1	1	X	X	X



A_{cur}	B_{cur}	In	A_{next}	T_A	B_{next}	T_B	Out
0	0	0	1	1	0	0	0
0	0	1	0	0	1	1	0
0	1	0	0	0	0	1	1
0	1	1	1	1	0	1	0
1	0	0	0	1	1	1	0
1	0	1	0	1	0	0	1
1	1	0	X	X	X	X	X
1	1	1	X	X	X	X	X

T_B:

A_{cur} \ In	0	1	1	1
	1	0	X	X

B_{cur}

$$T_B = B + \bar{A}I + A\bar{I}$$

Design with JK Flip-Flops

- Note: to change A_{cur} to the correct A_{next} value, two possible input pairs can be fed into the J,K inputs of a JK Flip-Flop

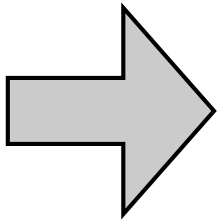
J	K	Q(t+1)
0	0	Q(t)
0	1	0
1	0	1
1	1	$\overline{Q(t)}$

A_{cur}	A_{next}	J,K
0	0	0,0 or 0,1 (0,X)
0	1	1,0 or 1,1 (1,X)
1	0	0,1 or 1,1 (X,1)
1	1	0,0 or 1,0 (X,0)

Design with JK Flip-Flop

A_{cur}	A_{next}	J,K
0	0	0,0 or 0,1 (0,X)
0	1	1,0 or 1,1 (1,X)
1	0	0,1 or 1,1 (X,1)
1	1	0,0 or 1,0 (X,0)

A_{cur}	B_{cur}	In	A_{next}	B_{next}	Out
0	0	0	1	0	0
0	0	1	0	1	0
0	1	0	0	0	1
0	1	1	1	0	0
1	0	0	0	1	0
1	0	1	0	0	1
1	1	0	X	X	X
1	1	1	X	X	X

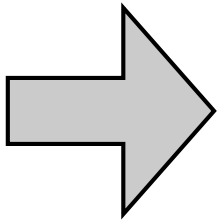


A_{cur}	B_{cur}	In	A_{next}	J_A	K_A	B_{next}	J_B	K_B	Out
0	0	0	1	1	X	0			0
0	0	1	0	0	X	1			0
0	1	0	0	0	X	0			1
0	1	1	1	1	X	0			0
1	0	0	0	X	1	1			0
1	0	1	0	X	1	0			1
1	1	0	X	X	X	X			X
1	1	1	X	X	X	X			X

Design with JK Flip-Flop

A_{cur}	A_{next}	J,K
0	0	0,0 or 0,1 (0,X)
0	1	1,0 or 1,1 (1,X)
1	0	0,1 or 1,1 (X,1)
1	1	0,0 or 1,0 (X,0)

A_{cur}	B_{cur}	In	A_{next}	B_{next}	Out
0	0	0	1	0	0
0	0	1	0	1	0
0	1	0	0	0	1
0	1	1	1	0	0
1	0	0	0	1	0
1	0	1	0	0	1
1	1	0	X	X	X
1	1	1	X	X	X



A_{cur}	B_{cur}	In	A_{next}	J_A	K_A	B_{next}	J_B	K_B	Out
0	0	0	1	1	X	0	0	X	0
0	0	1	0	0	X	1	1	X	0
0	1	0	0	0	X	0	X	1	1
0	1	1	1	1	X	0	X	1	0
1	0	0	0	X	1	1	1	X	0
1	0	1	0	X	1	0	0	X	1
1	1	0	X	X	X	X	X	X	X
1	1	1	X	X	X	X	X	X	X

Design with JK Flip-Flop

A_{cur}	B_{cur}	In	A_{next}	J_A	K_A	B_{next}	J_B	K_B	Out
0	0	0	1	1	X	0	0	X	0
0	0	1	0	0	X	1	1	X	0
0	1	0	0	0	X	0	X	1	1
0	1	1	1	1	X	0	X	1	0
1	0	0	0	X	1	1	1	X	0
1	0	1	0	X	1	0	0	X	↯
1	1	0	X	X	X	X	X	X	X
1	1	1	X	X	X	X	X	X	X

J_A:

A_{cur} \ In	1	0	1	0
	X	X	X	X

B_{cur}

K_A:

A_{cur} \ In	X	X	X	X
	1	1	X	X

B_{cur}

J_B:

A_{cur} \ In	0	1	X	X
	1	0	X	X

B_{cur}

K_B:

A_{cur} \ In	X	X	1	1
	X	X	X	X

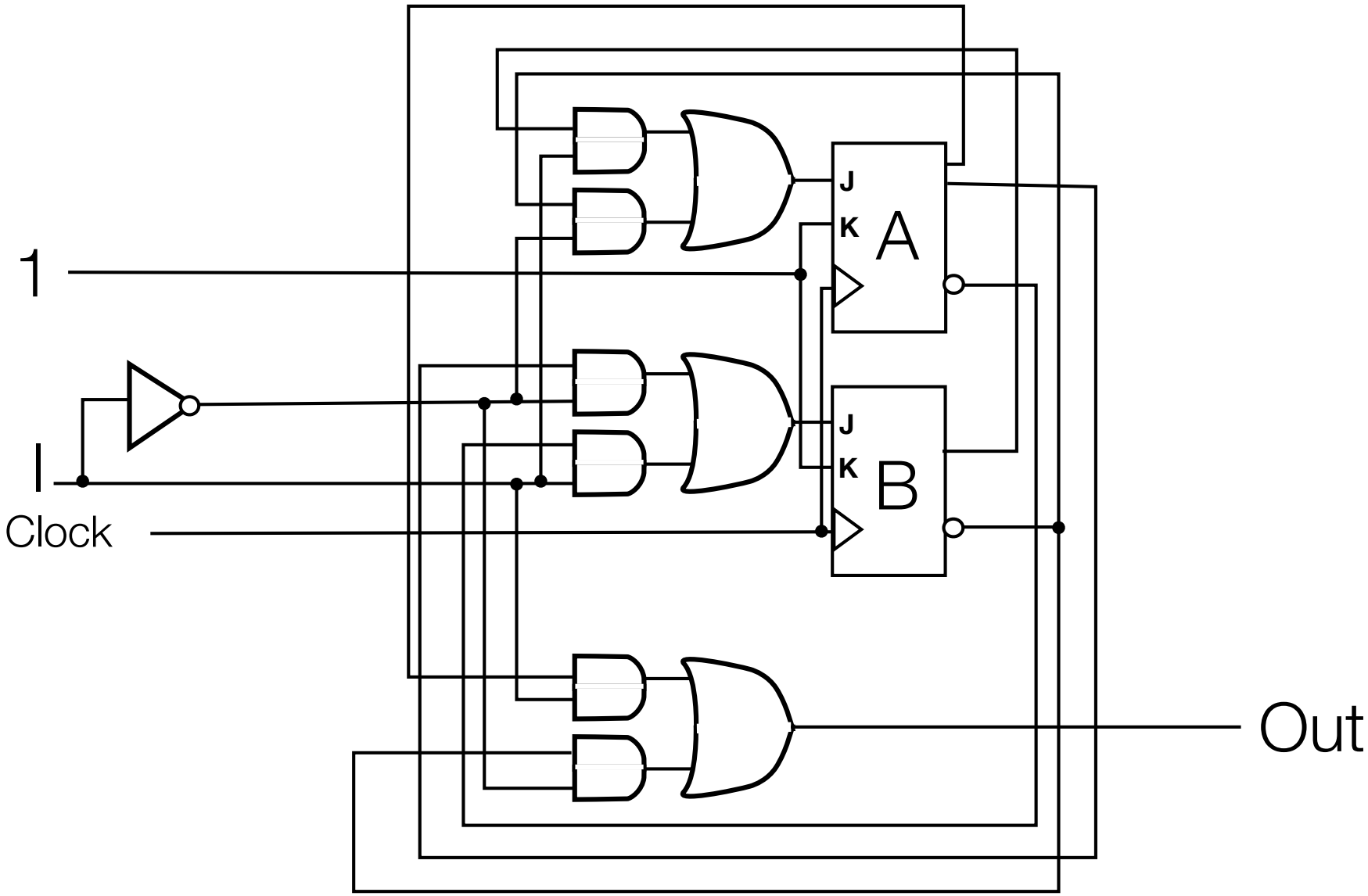
B_{cur}

$J_A = BI + \bar{B}\bar{I}$ $\quad B \oplus \bar{I}$ $\qquad K_A = 1$

$J_B = \bar{A}I + A\bar{I}$ $\quad A \oplus I$ $\qquad K_B = 1$

Design with JK FF's cont'd

$J_A = BI + \overline{B}\,\overline{I}$ $K_A = 1$ $J_B = \overline{A}I + A\overline{I}$ $K_B = 1$



2nd Example

Another Example: Message Generator

- Circuit's purpose is to generate a message.
- Each clock cycle, 1-bit input determines 1-bit output
  - When input is held fixed at 0: message is 011000000…
  - When input is held fixed at 1: message is 001111111…
  - Message restarts whenever current input differs from previous

Example

time	Input	Output
0	1	0
1	1	0
2	1	1
3	1	1
4	1	1
5	0	0
6	0	1
7	1	0
8	1	0
9	0	0
10	0	1
11	0	1
12	0	0
13	0	0

Figuring out state machine

- Q: What information needs to be retained?

- 0 input: message is 011000000…
- 1 input : message is 001111111…

Figuring out state machine

- Q: What information needs to be retained?
- A: 2 pieces of info
  - what previous input was
  - how far along we are in the message sequence

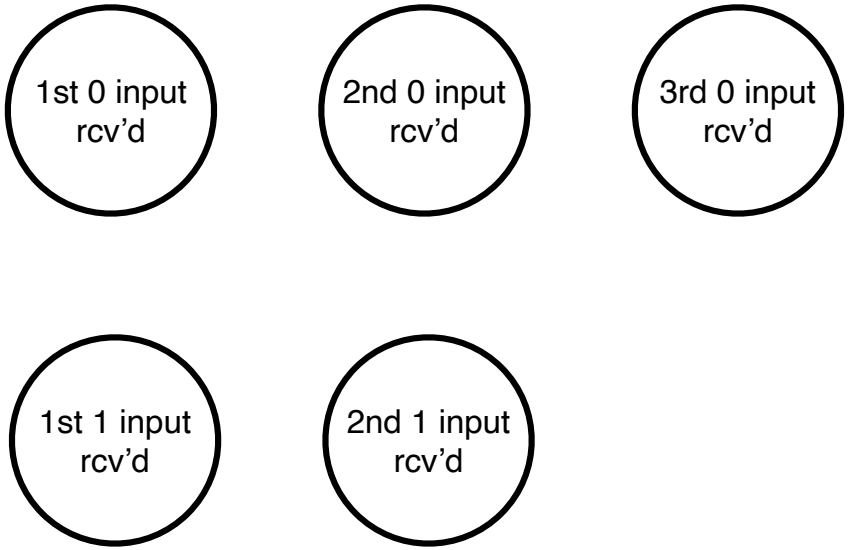
- 0 input: message is 011000000…
- 1 input : message is 001111111…

Figuring out state machine

- Q: What information needs to be retained?
- A: 2 pieces of info
  - what previous input was
  - how far along we are in the message sequence
- Note we only need to track where we are in the sequence to the point where it remains fixed.

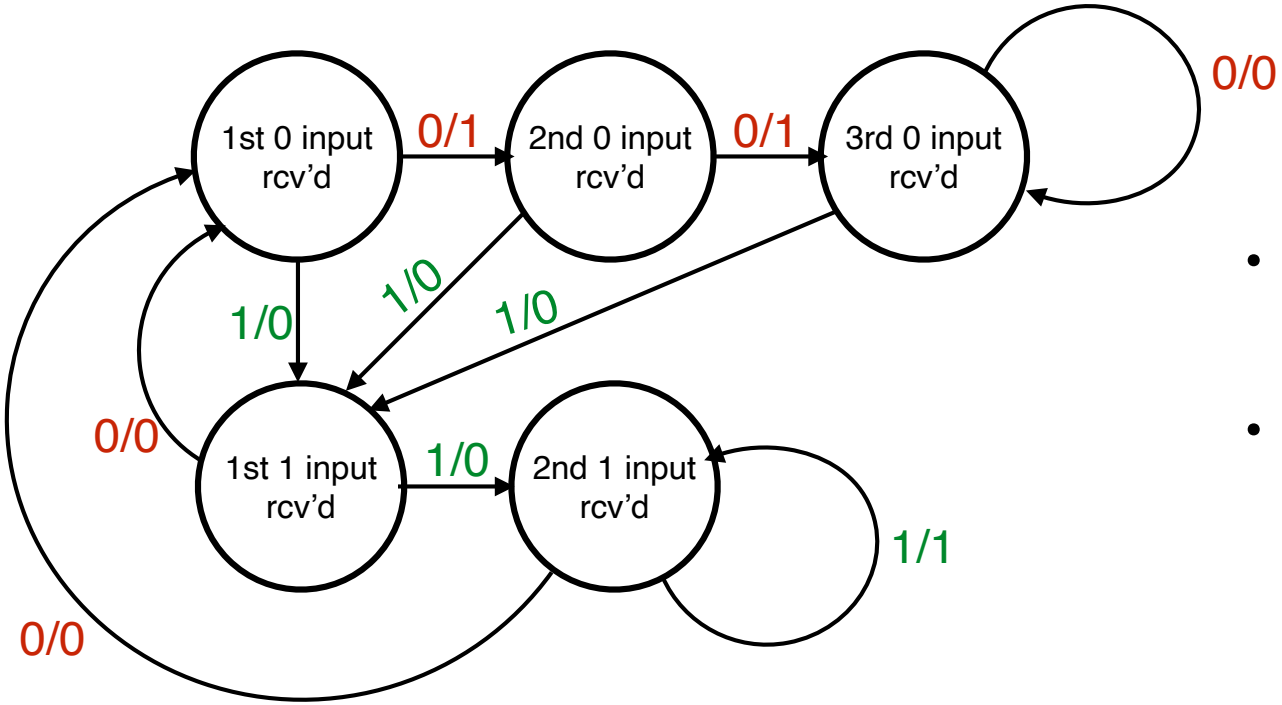
- 0 input: message is 011000000…
- 1 input : message is 001111111…

The States



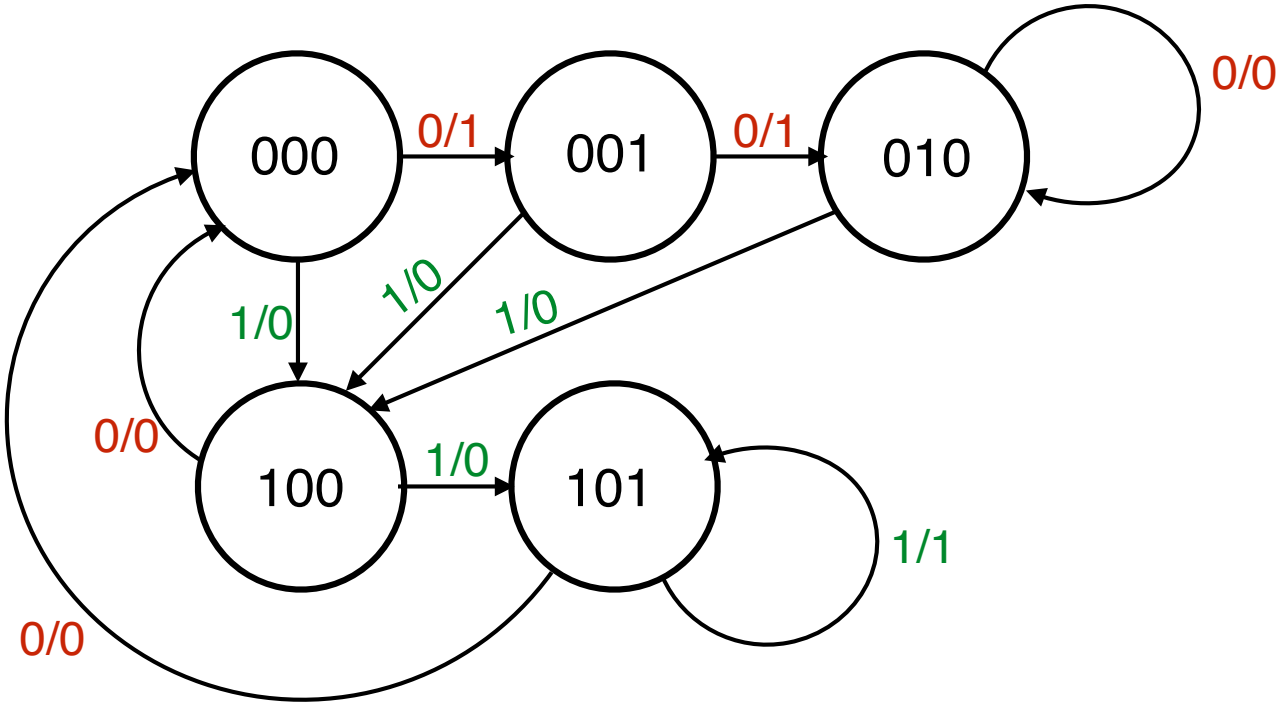
- 0 input: message is 011000000…
- 1 input : message is 001111111…

The States



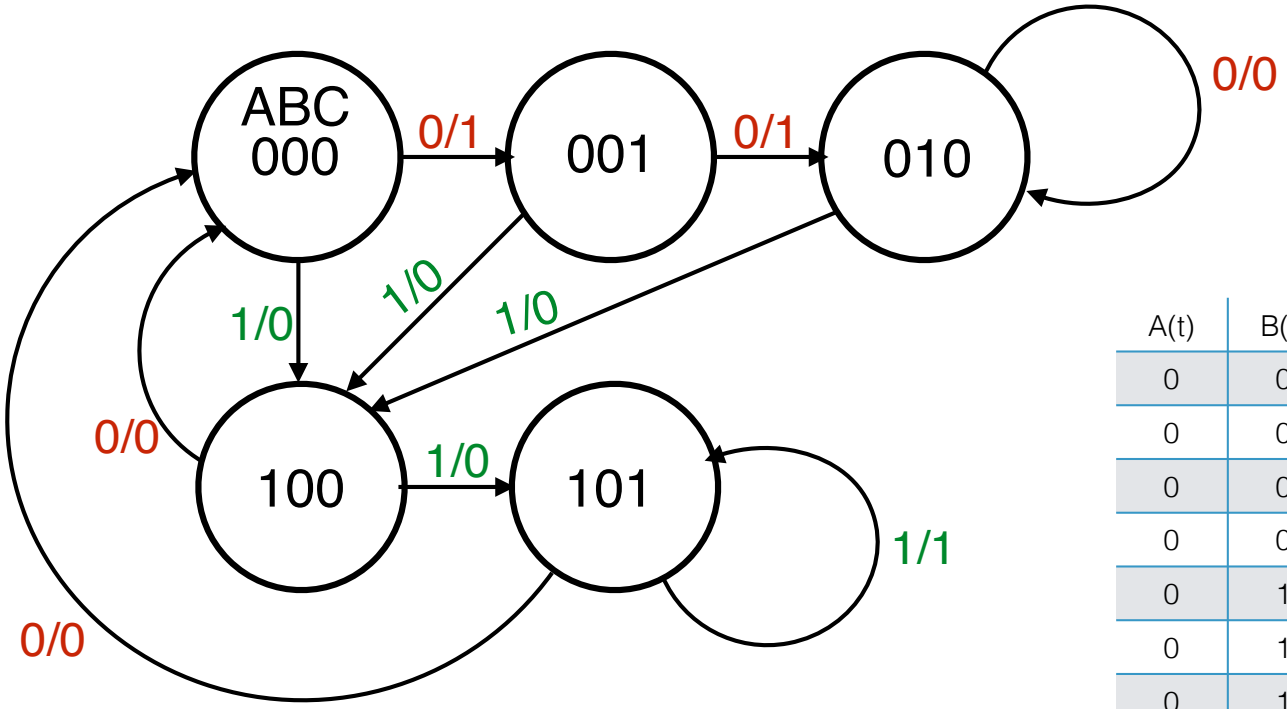
- 0 input: message is 011000000…
- 1 input : message is 001111111…

Number The States



- 5 states: each state needs a 3-bit labelling so that all states are unique
- Labelling can be arbitrary

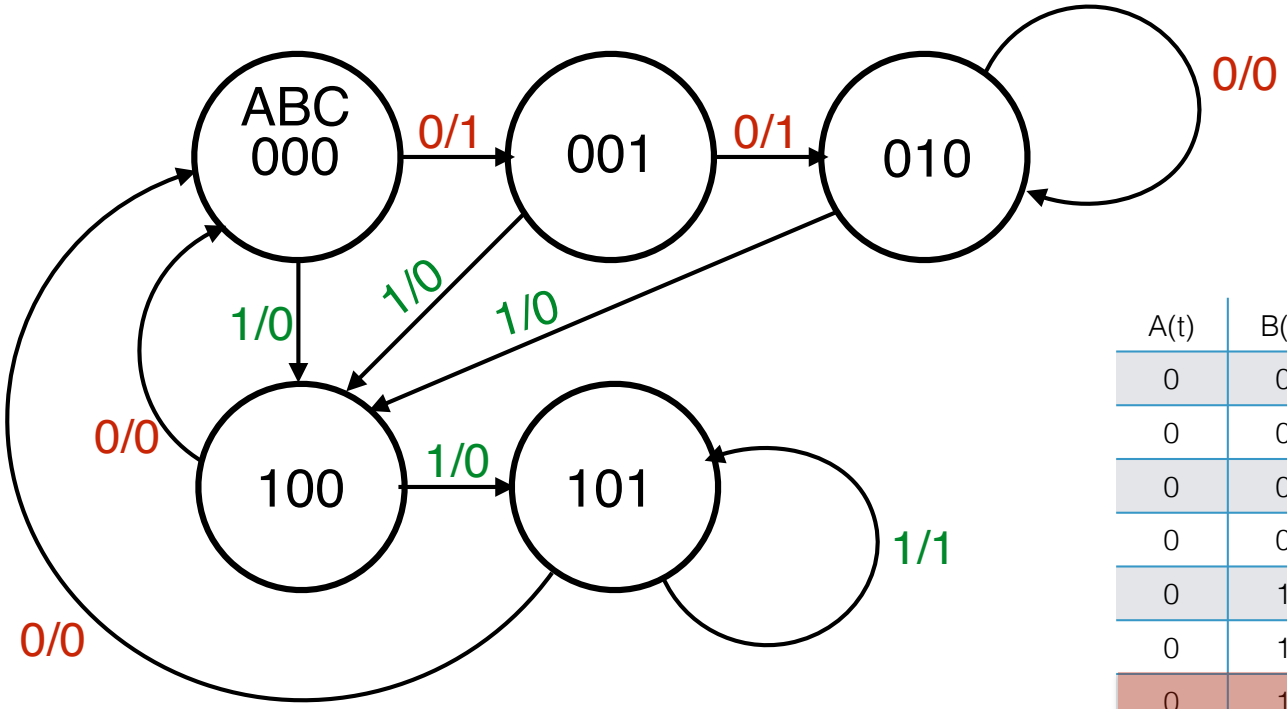
Convert State Machine to Table



Convert state machine to table form: consider all possible values of A(t),B(t),C(t), In(t)

A(t)	B(t)	C(t)	In(t)	Out(t)	A(t+1)	B(t+1)	C(t+1)
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				

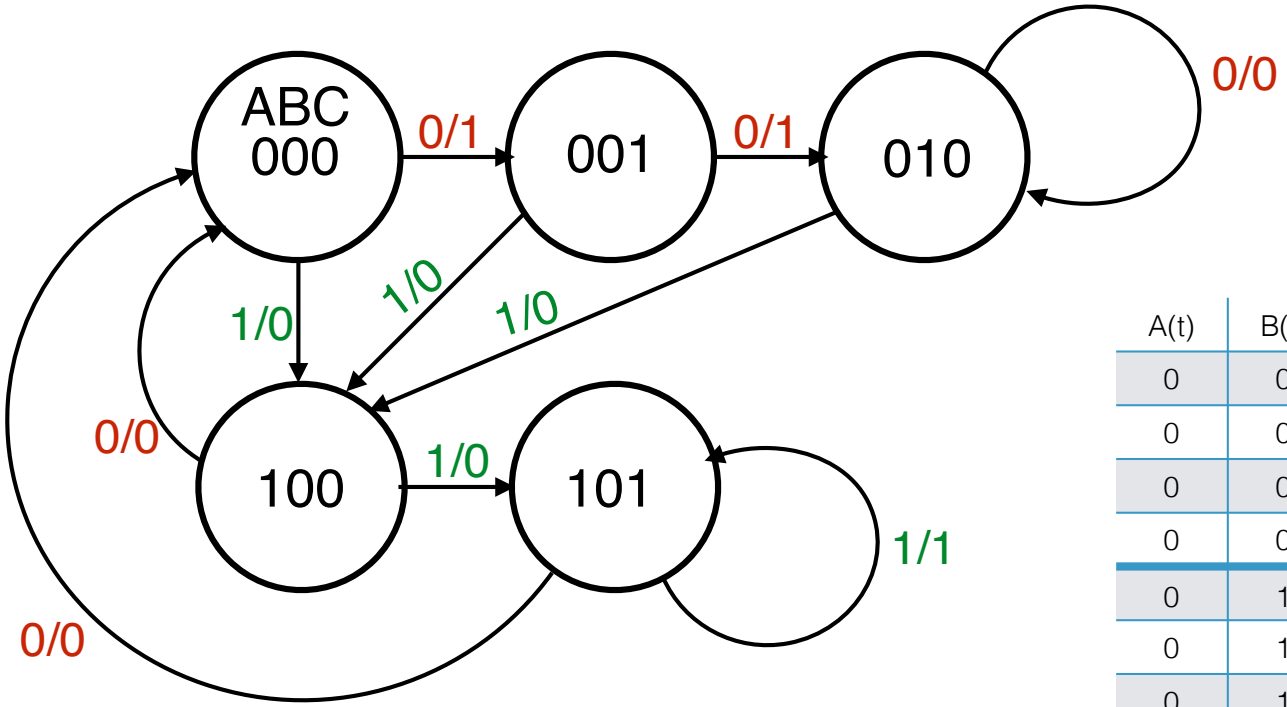
Don’t Cares for non-non-existent states



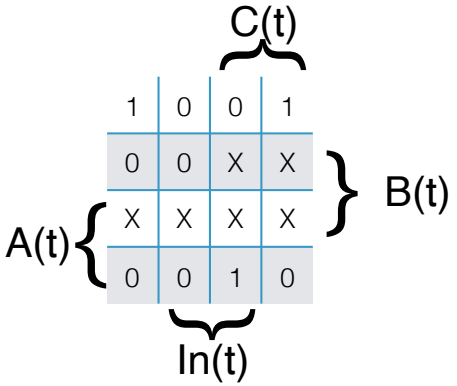
For table entries for states that don’t exist, can fill in with don’t-care conditions

A(t)	B(t)	C(t)	In(t)	Out(t)	A(t+1)	B(t+1)	C(t+1)
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	X	X	X	X
0	1	1	1	X	X	X	X
1	0	0	0				
1	0	0	1				
1	0	1	0				
1	0	1	1				
1	1	0	0	X	X	X	X
1	1	0	1	X	X	X	X
1	1	1	0	X	X	X	X
1	1	1	1	X	X	X	X

Determining Output



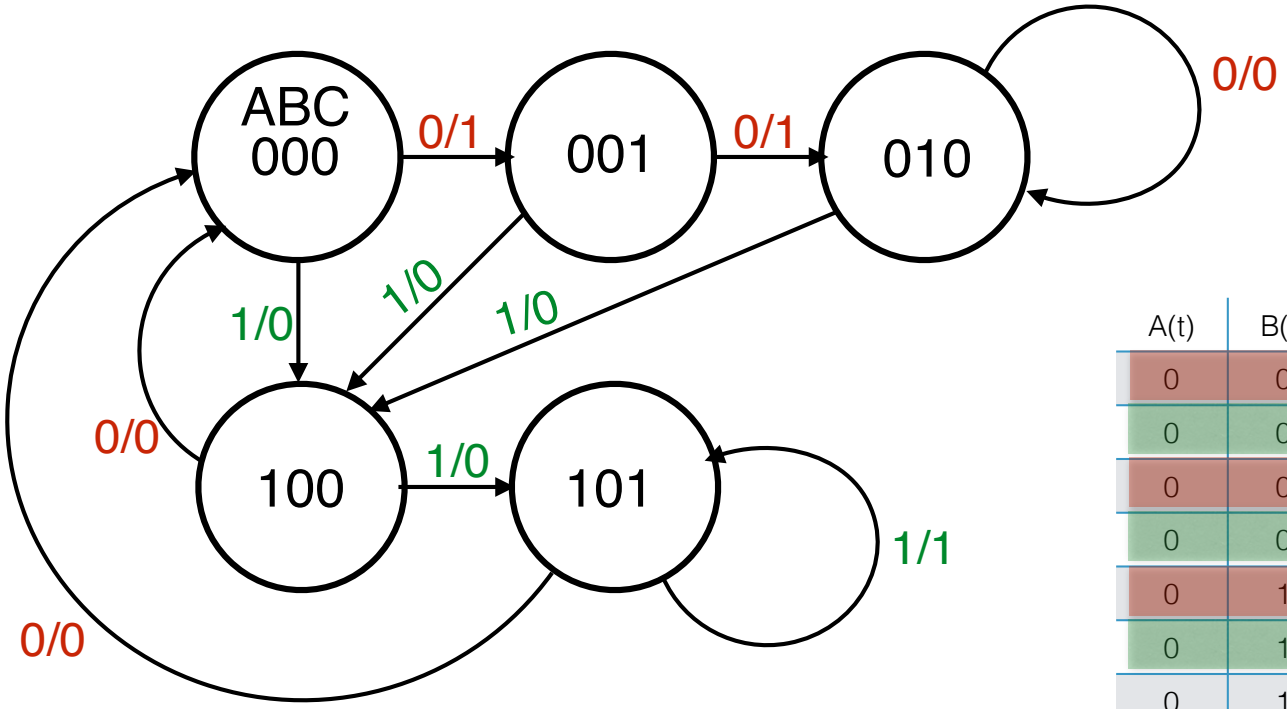
Out indicated on transitions: can already solve for output in terms of inputs via KMap



$$Out(t) = ACI + \bar{A}\bar{B}\bar{I}$$

A(t)	B(t)	C(t)	In(t)	Out(t)	A(t+1)	B(t+1)	C(t+1)
0	0	0	0	1			
0	0	0	1	0			
0	0	1	0	1			
0	0	1	1	0			
0	1	0	0	0			
0	1	0	1	0			
0	1	1	0	X	X	X	X
0	1	1	1	X	X	X	X
1	0	0	0	0			
1	0	0	1	0			
1	0	1	0	0			
1	0	1	1	1			
1	1	0	0	X	X	X	X
1	1	0	1	X	X	X	X
1	1	1	0	X	X	X	X
1	1	1	1	X	X	X	X

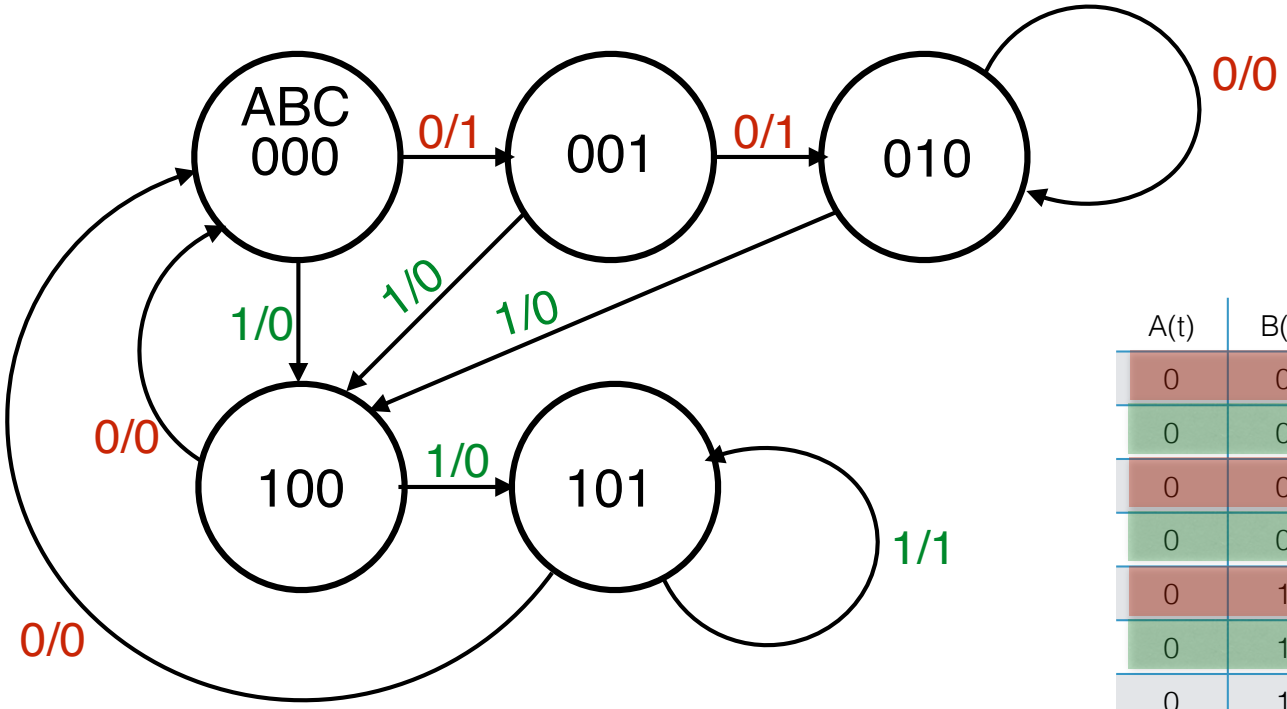
Enter in Next States



Can also determine next cycle's state (A(t+1), B(t+1), C(t+1)) based on current cycle's state (A(t), B(t), C(t)) and current input I(t)

A(t)	B(t)	C(t)	In(t)	Out(t)	A(t+1)	B(t+1)	C(t+1)
0	0	0	0	1	0	0	1
0	0	0	1	0	1	0	0
0	0	1	0	1	0	1	0
0	0	1	1	0	1	0	0
0	1	0	0	0	0	1	0
0	1	0	1	0	1	0	0
0	1	1	0	X	X	X	X
0	1	1	1	X	X	X	X
1	0	0	0	0	0	0	0
1	0	0	1	0	1	0	1
1	0	1	0	0	0	0	0
1	0	1	1	1	1	0	1
1	1	0	0	X	X	X	X
1	1	0	1	X	X	X	X
1	1	1	0	X	X	X	X
1	1	1	1	X	X	X	X

Which type of Flip-Flop to use?



Now must decide which type of F.F. we are using before going further.

Let's use JK Flip-Flops

A(t)	B(t)	C(t)	In(t)	Out(t)	A(t+1)	B(t+1)	C(t+1)
0	0	0	0	1	0	0	1
0	0	0	1	0	1	0	0
0	0	1	0	1	0	1	0
0	0	1	1	0	1	0	0
0	1	0	0	0	0	1	0
0	1	0	1	0	1	0	0
0	1	1	0	X	X	X	X
0	1	1	1	X	X	X	X
1	0	0	0	0	0	0	0
1	0	0	1	0	1	0	1
1	0	1	0	0	0	0	0
1	0	1	1	1	1	0	1
1	1	0	0	X	X	X	X
1	1	0	1	X	X	X	X
1	1	1	0	X	X	X	X
1	1	1	1	X	X	X	X

JK Flip Flops: J& K columns

A(t)	B(t)	C(t)	In(t)	Out(t)	A(t+1)	$J_A(t)$	$K_A(t)$	B(t+1)	$J_B(t)$	$K_B(t)$	C(t+1)	$J_C(t)$	$K_C(t)$
0	0	0	0	1	0			0			1		
0	0	0	1	0	1			0			0		
0	0	1	0	1	0			1			0		
0	0	1	1	0	1			0			0		
0	1	0	0	0	0			1			0		
0	1	0	1	0	1			0			0		
0	1	1	0	X	X			X			X		
0	1	1	1	X	X			X			X		
1	0	0	0	0	0			0			0		
1	0	0	1	0	1			0			1		
1	0	1	0	0	0			0			0		
1	0	1	1	1	1			0			1		
1	1	0	0	X	X			X			X		
1	1	0	1	X	X			X			X		
1	1	1	0	X	X			X			X		
1	1	1	1	X	X			X			X		

A_{cur}	A_{next}	J,K
0	0	0,0 or 0,1 (0,X)
0	1	1,0 or 1,1 (1,X)
1	0	0,1 or 1,1 (X,1)
1	1	0,0 or 1,0 (X,0)

Add Columns to determine what J and K should be based on current state, current input, and how a particular FF's stored value should change

JK Flip Flops: J& K columns

A(t)	B(t)	C(t)	In(t)	Out(t)	A(t+1)	$J_A(t)$	$K_A(t)$	B(t+1)	$J_B(t)$	$K_B(t)$	C(t+1)	$J_C(t)$	$K_C(t)$
0	0	0	0	1	0			0			1		
0	0	0	1	0	1			0			0		
0	0	1	0	1	0			1			0		
0	0	1	1	0	1			0			0		
0	1	0	0	0	0			1			0		
0	1	0	1	0	1			0			0		
0	1	1	0	X	X			X			X		
0	1	1	1	X	X			X			X		
1	0	0	0	0	0			0			0		
1	0	0	1	0	1			0			1		
1	0	1	0	0	0			0			0		
1	0	1	1	1	1			0			1		
1	1	0	0	X	X			X			X		
1	1	0	1	X	X			X			X		
1	1	1	0	X	X			X			X		
1	1	1	1	X	X			X			X		

A_{cur}	A_{next}	J,K
0	0	0,0 or 0,1 (0,X)
0	1	1,0 or 1,1 (1,X)
1	0	0,1 or 1,1 (X,1)
1	1	0,0 or 1,0 (X,0)

When in state 100, and input is 0, go to state 000

JK Flip Flops: J& K columns

A(t)	B(t)	C(t)	In(t)	Out(t)	A(t+1)	$J_A(t)$	$K_A(t)$	B(t+1)	$J_B(t)$	$K_B(t)$	C(t+1)	$J_C(t)$	$K_C(t)$
0	0	0	0	1	0			0			1		
0	0	0	1	0	1			0			0		
0	0	1	0	1	0			1			0		
0	0	1	1	0	1			0			0		
0	1	0	0	0	0			1			0		
0	1	0	1	0	1			0			0		
0	1	1	0	X	X			X			X		
0	1	1	1	X	X			X			X		
1	0	0	0	0	0			0			0		
1	0	0	1	0	1			0			1		
1	0	1	0	0	0			0			0		
1	0	1	1	1	1			0			1		
1	1	0	0	X	X			X			X		
1	1	0	1	X	X			X			X		
1	1	1	0	X	X			X			X		
1	1	1	1	X	X			X			X		

A_{cur}	A_{next}	J,K
0	0	0,0 or 0,1 (0,X)
0	1	1,0 or 1,1 (1,X)
1	0	0,1 or 1,1 (X,1)
1	1	0,0 or 1,0 (X,0)

When in state 100, and input is 0, go to state 000

i.e., when A(t)=1, B(t)=0, C(t)=0, I(t)=0 then A(t+1)=0, B(t+1)=0, C(t+1)=0

JK Flip Flops: J& K columns

A(t)	B(t)	C(t)	In(t)	Out(t)	A(t+1)	$J_A(t)$	$K_A(t)$	B(t+1)	$J_B(t)$	$K_B(t)$	C(t+1)	$J_C(t)$	$K_C(t)$
0	0	0	0	1	0			0			1		
0	0	0	1	0	1			0			0		
0	0	1	0	1	0			1			0		
0	0	1	1	0	1			0			0		
0	1	0	0	0	0			1			0		
0	1	0	1	0	1			0			0		
0	1	1	0	X	X			X			X		
0	1	1	1	X	X			X			X		
1	0	0	0	0	0			0			0		
1	0	0	1	0	1			0			1		
1	0	1	0	0	0			0			0		
1	0	1	1	1	1			0			1		
1	1	0	0	X	X			X			X		
1	1	0	1	X	X			X			X		
1	1	1	0	X	X			X			X		
1	1	1	1	X	X			X			X		

A_{cur}	A_{next}	J,K
0	0	0,0 or 0,1 (0,X)
0	1	1,0 or 1,1 (1,X)
1	0	0,1 or 1,1 (X,1)
1	1	0,0 or 1,0 (X,0)

Focus just on A for now:

i.e., when A(t)=1, B(t)=0, C(t)=0, I(t)=0 then A(t+1)=0

JK Flip Flops: J& K columns

A(t)	B(t)	C(t)	In(t)	Out(t)	A(t+1)	$J_A(t)$	$K_A(t)$	B(t+1)	$J_B(t)$	$K_B(t)$	C(t+1)	$J_C(t)$	$K_C(t)$
0	0	0	0	1	0			0			1		
0	0	0	1	0	1			0			0		
0	0	1	0	1	0			1			0		
0	0	1	1	0	1			0			0		
0	1	0	0	0	0			1			0		
0	1	0	1	0	1			0			0		
0	1	1	0	X	X			X			X		
0	1	1	1	X	X			X			X		
1	0	0	0	0	0	X	1	0			0		
1	0	0	1	0	1			0			1		
1	0	1	0	0	0			0			0		
1	0	1	1	1	1			0			1		
1	1	0	0	X	X			X			X		
1	1	0	1	X	X			X			X		
1	1	1	0	X	X			X			X		
1	1	1	1	X	X			X			X		

A_{cur}	A_{next}	J,K
0	0	0,0 or 0,1 (0,X)
0	1	1,0 or 1,1 (1,X)
1	0	0,1 or 1,1 (X,1)
1	1	0,0 or 1,0 (X,0)

i.e., when A(t)=1, B(t)=0, C(t)=0, I(t)=0 then A(t+1)=0

so when A(t)=1, B(t)=0, C(t)=0, I(t)=0 then $J_A(t)$=X, $K_A(t)$=1

JK Flip Flops: J& K columns

A(t)	B(t)	C(t)	In(t)	Out(t)	A(t+1)	$J_A(t)$	$K_A(t)$	B(t+1)	$J_B(t)$	$K_B(t)$	C(t+1)	$J_C(t)$	$K_C(t)$
0	0	0	0	1	0			0			1		
0	0	0	1	0	1			0			0		
0	0	1	0	1	0			1			0		
0	0	1	1	0	1			0			0		
0	1	0	0	0	0			1			0		
0	1	0	1	0	1			0			0		
0	1	1	0	X	X			X			X		
0	1	1	1	X	X			X			X		
1	0	0	0	0	0	X	1	0	0	X	0	0	X
1	0	0	1	0	1			0			1		
1	0	1	0	0	0			0			0		
1	0	1	1	1	1			0			1		
1	1	0	0	X	X			X			X		
1	1	0	1	X	X			X			X		
1	1	1	0	X	X			X			X		
1	1	1	1	X	X			X			X		

A_{cur}	A_{next}	J,K
0	0	0,0 or 0,1 (0,X)
0	1	1,0 or 1,1 (1,X)
1	0	0,1 or 1,1 (X,1)
1	1	0,0 or 1,0 (X,0)

Same idea for B & C

JK Flip Flops: J& K columns

A(t)	B(t)	C(t)	In(t)	Out(t)	A(t+1)	$J_A(t)$	$K_A(t)$	B(t+1)	$J_B(t)$	$K_B(t)$	C(t+1)	$J_C(t)$	$K_C(t)$
0	0	0	0	1	0	0	X	0	0	X	1	1	X
0	0	0	1	0	1	1	X	0	0	X	0	X	1
0	0	1	0	1	0	0	X	1	1	X	0	0	X
0	0	1	1	0	1	1	X	0	0	X	0	X	1
0	1	0	0	0	0	0	X	1	X	0	0	0	X
0	1	0	1	0	1	1	X	0	X	1	0	X	1
0	1	1	0	X	X	X	X	X	X	X	X	X	X
0	1	1	1	X	X	X	X	X	X	X	X	X	X
1	0	0	0	0	0	X	1	0	0	X	0	0	X
1	0	0	1	0	1	X	0	0	0	X	1	X	0
1	0	1	0	0	0	X	1	0	0	X	0	0	X
1	0	1	1	1	1	X	0	0	0	X	1	X	0
1	1	0	0	X	X	X	X	X	X	X	X	X	X
1	1	0	1	X	X	X	X	X	X	X	X	X	X
1	1	1	0	X	X	X	X	X	X	X	X	X	X
1	1	1	1	X	X	X	X	X	X	X	X	X	X

A_{cur}	A_{next}	J,K
0	0	0,0 or 0,1 (0,X)
0	1	1,0 or 1,1 (1,X)
1	0	0,1 or 1,1 (X,1)
1	1	0,0 or 1,0 (X,0)

Same idea for all other rows

JK Flip Flops: J& K columns

A(t)	B(t)	C(t)	In(t)	Out(t)	A(t+1)	$J_A(t)$	$K_A(t)$	B(t+1)	$J_B(t)$	$K_B(t)$	C(t+1)	$J_C(t)$	$K_C(t)$
0	0	0	0	1	0	0	X	0	0	X	1	1	X
0	0	0	1	0	1	1	X	0	0	X	0	X	1
0	0	1	0	1	0	0	X	1	1	X	0	0	X
0	0	1	1	0	1	1	X	0	0	X	0	X	1
0	1	0	0	0	0	0	X	1	X	0	0	0	X
0	1	0	1	0	1	1	X	0	X	1	0	X	1
0	1	1	0	X	X	X	X	X	X	X	X	X	X
0	1	1	1	X	X	X	X	X	X	X	X	X	X
1	0	0	0	0	0	X	1	0	0	X	0	0	X
1	0	0	1	0	1	X	0	0	0	X	1	X	0
1	0	1	0	0	0	X	1	0	0	X	0	0	X
1	0	1	1	1	1	X	0	0	0	X	1	X	0
1	1	0	0	X	X	X	X	X	X	X	X	X	X
1	1	0	1	X	X	X	X	X	X	X	X	X	X
1	1	1	0	X	X	X	X	X	X	X	X	X	X
1	1	1	1	X	X	X	X	X	X	X	X	X	X

Just need these columns to get circuitry that feeds into the J,K inputs of each of the Flip-Flops

JK Flip Flops: J& K columns

A(t)	B(t)	C(t)	In(t)	$J_A(t)$	$K_A(t)$	$J_B(t)$	$K_B(t)$	$J_C(t)$	$K_C(t)$
0	0	0	0	0	X	0	X	1	X
0	0	0	1	1	X	0	X	X	1
0	0	1	0	0	X	1	X	0	X
0	0	1	1	1	X	0	X	X	1
0	1	0	0	0	X	X	0	0	X
0	1	0	1	1	X	X	1	X	1
0	1	1	0	X	X	X	X	X	X
0	1	1	1	X	X	X	X	X	X
1	0	0	0	X	1	0	X	0	X
1	0	0	1	X	0	0	X	X	0
1	0	1	0	X	1	0	X	0	X
1	0	1	1	X	0	0	X	X	0
1	1	0	0	X	X	X	X	X	X
1	1	0	1	X	X	X	X	X	X
1	1	1	0	X	X	X	X	X	X
1	1	1	1	X	X	X	X	X	X

J_A:

A(t) \ In(t)	C(t)=0	C(t)=1	C(t)=1	C(t)=0
B(t)=0	0	1	X	X
B(t)=1	X	X	X	X
B(t)=1	X	X	X	X

$= In(t) = I$

K_A:

A(t) \ In(t)	C(t)=0	C(t)=1	C(t)=1	C(t)=0
B(t)=0	X	X	X	X
B(t)=1	X	X	X	X
B(t)=1	X	X	X	X
B(t)=1	1	0	0	1

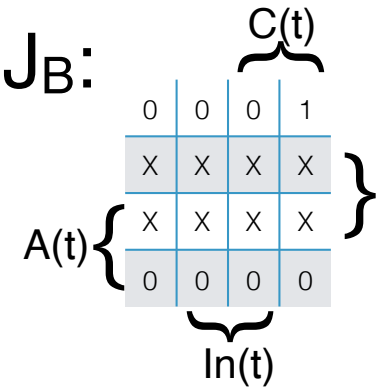
$= \overline{In(t)} = \bar{I}$

Just need these columns to get circuitry that feeds into the J,K inputs of each of the Flip-Flops

JK Flip Flops: J& K columns

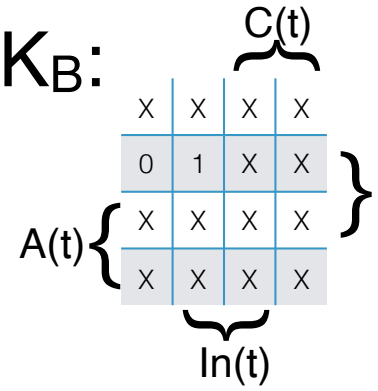
A(t)	B(t)	C(t)	In(t)	$J_A(t)$	$K_A(t)$	$J_B(t)$	$K_B(t)$	$J_C(t)$	$K_C(t)$
0	0	0	0	0	X	0	X	1	X
0	0	0	1	1	X	0	X	X	1
0	0	1	0	0	X	1	X	0	X
0	0	1	1	1	X	0	X	X	1
0	1	0	0	0	X	X	0	0	X
0	1	0	1	1	X	X	1	X	1
0	1	1	0	X	X	X	X	X	X
0	1	1	1	X	X	X	X	X	X
1	0	0	0	X	1	0	X	0	X
1	0	0	1	X	0	0	X	X	0
1	0	1	0	X	1	0	X	0	X
1	0	1	1	X	0	0	X	X	0
1	1	0	0	X	X	X	X	X	X
1	1	0	1	X	X	X	X	X	X
1	1	1	0	X	X	X	X	X	X
1	1	1	1	X	X	X	X	X	X

J_B:



$= \bar{A}C\bar{I}$

K_B:



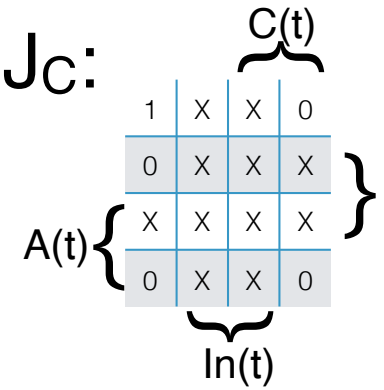
$= I$

Just need these columns to get circuitry that feeds into the J,K inputs of each of the Flip-Flops

JK Flip Flops: J& K columns

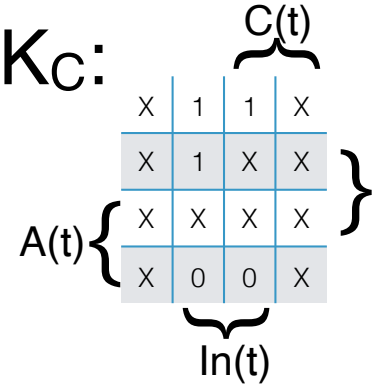
A(t)	B(t)	C(t)	In(t)	$J_A(t)$	$K_A(t)$	$J_B(t)$	$K_B(t)$	$J_C(t)$	$K_C(t)$
0	0	0	0	0	X	0	X	1	X
0	0	0	1	1	X	0	X	X	1
0	0	1	0	0	X	1	X	0	X
0	0	1	1	1	X	0	X	X	1
0	1	0	0	0	X	X	0	0	X
0	1	0	1	1	X	X	1	X	1
0	1	1	0	X	X	X	X	X	X
0	1	1	1	X	X	X	X	X	X
1	0	0	0	X	1	0	X	0	X
1	0	0	1	X	0	0	X	X	0
1	0	1	0	X	1	0	X	0	X
1	0	1	1	X	0	0	X	X	0
1	1	0	0	X	X	X	X	X	X
1	1	0	1	X	X	X	X	X	X
1	1	1	0	X	X	X	X	X	X
1	1	1	1	X	X	X	X	X	X

J_C:



$= \bar{A}\bar{B}\bar{C}$

K_C:

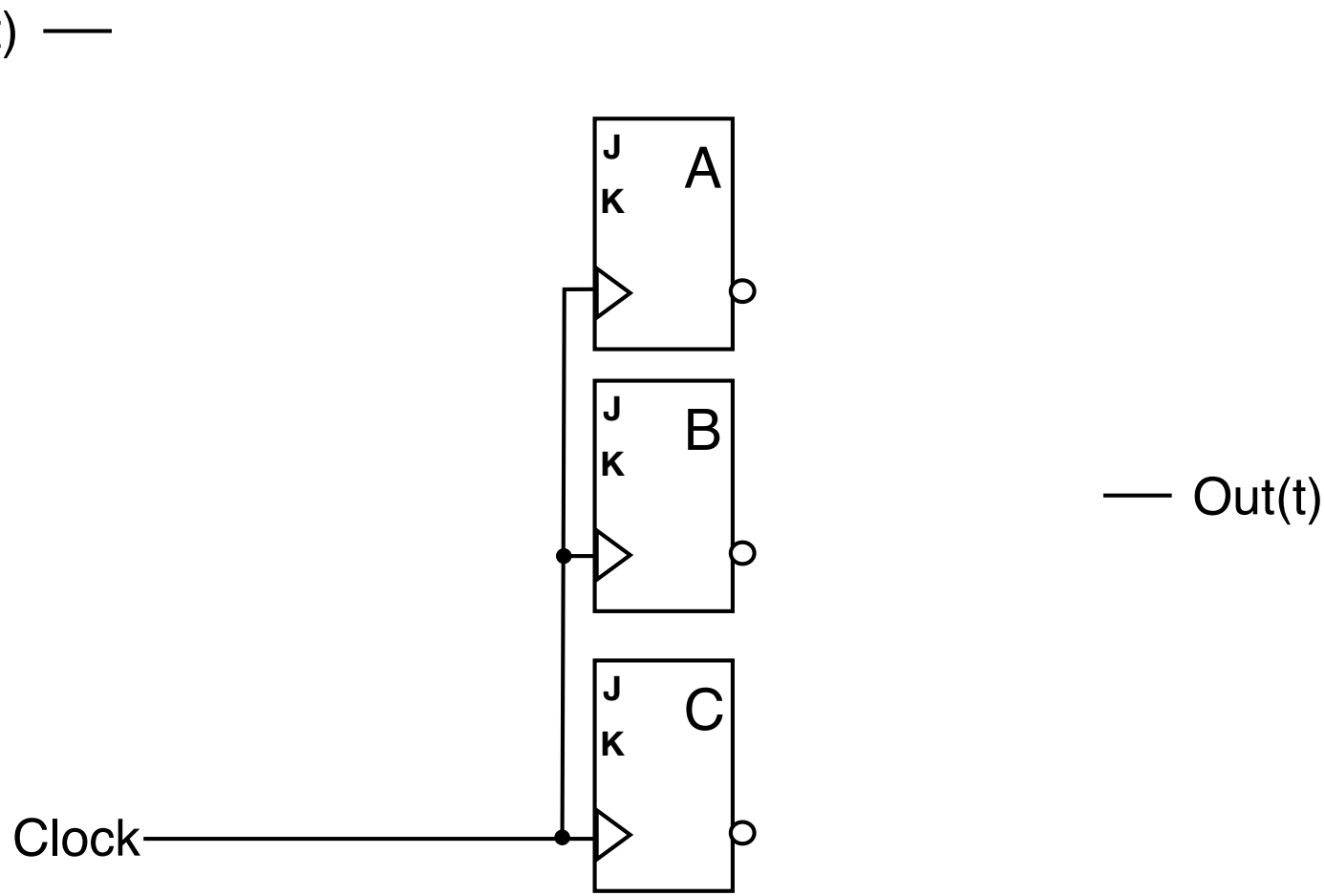


$= \bar{A}$

Just need these columns to get circuitry that feeds into the J,K inputs of each of the Flip-Flops

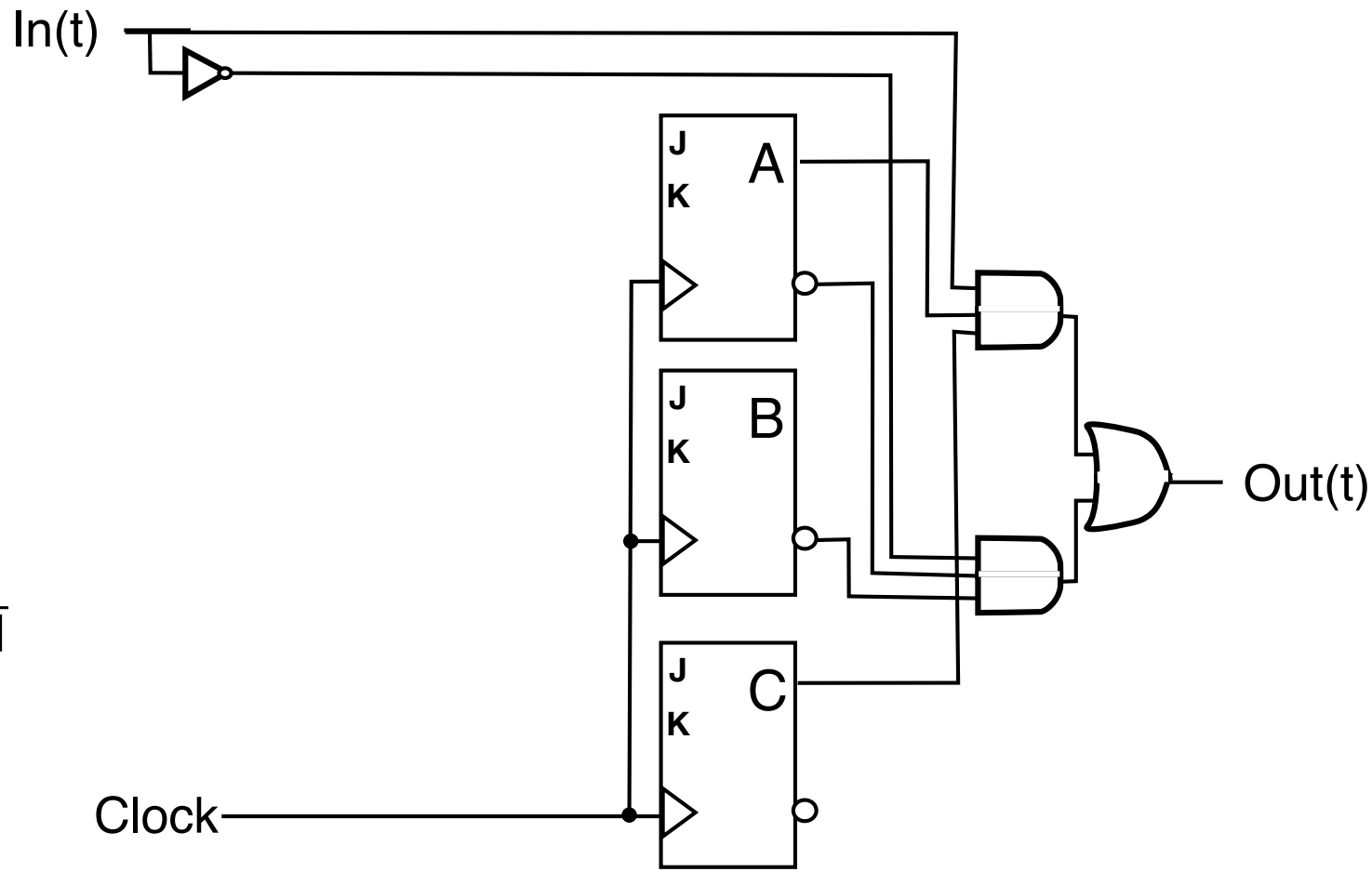
The Sequential Circuit

- $J_A = I$
- $K_A = \bar{I}$
- $J_B = \bar{A}C\bar{I}$
- $K_B = I$
- $J_C = \bar{A}\bar{B}\bar{C}$
- $K_C = \bar{A}$
- $Out = ACI + \bar{A}\bar{B}\bar{I}$



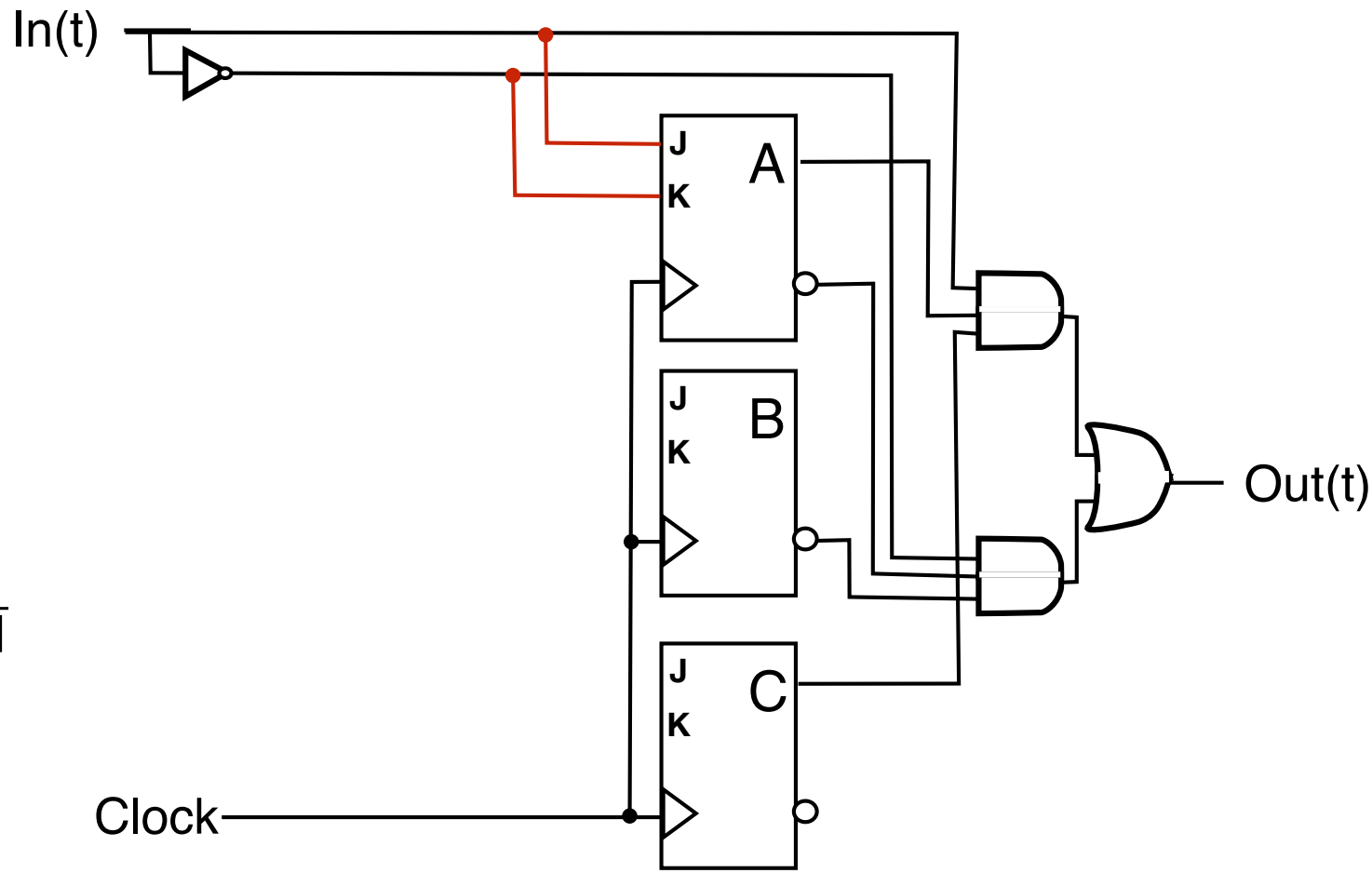
The Sequential Circuit

- $J_A = I$
- $K_A = \bar{I}$
- $J_B = \bar{A}C\bar{I}$
- $K_B = I$
- $J_C = \bar{A}\bar{B}\bar{C}$
- $K_C = \bar{A}$
- $Out = ACI + \bar{A}\bar{B}\bar{I}$



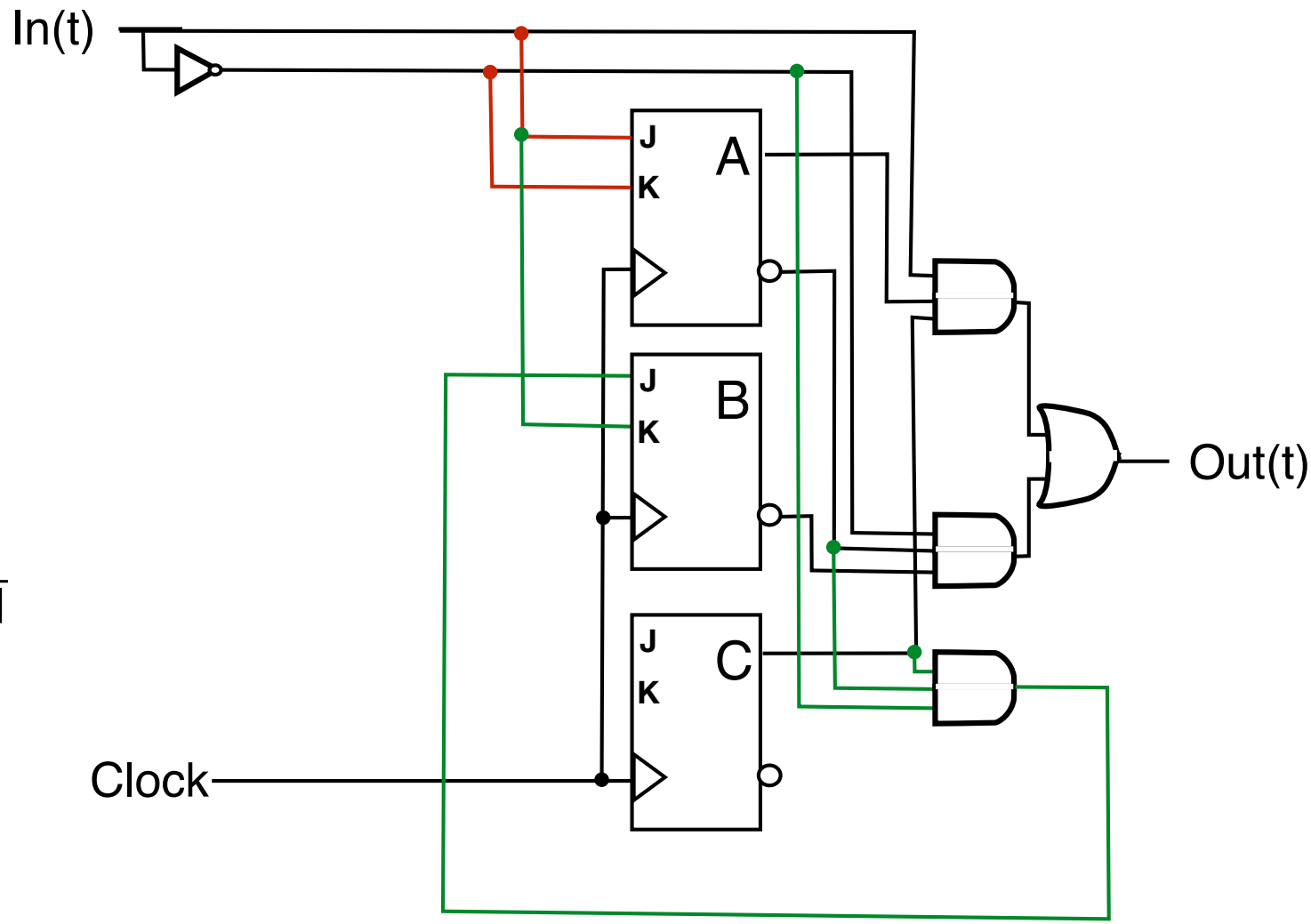
The Sequential Circuit

- $J_A = I$
- $K_A = \bar{I}$
- $J_B = \bar{A}C\bar{I}$
- $K_B = I$
- $J_C = \bar{A}\bar{B}\bar{C}$
- $K_C = \bar{A}$
- $Out = ACI + \bar{A}\bar{B}\bar{I}$



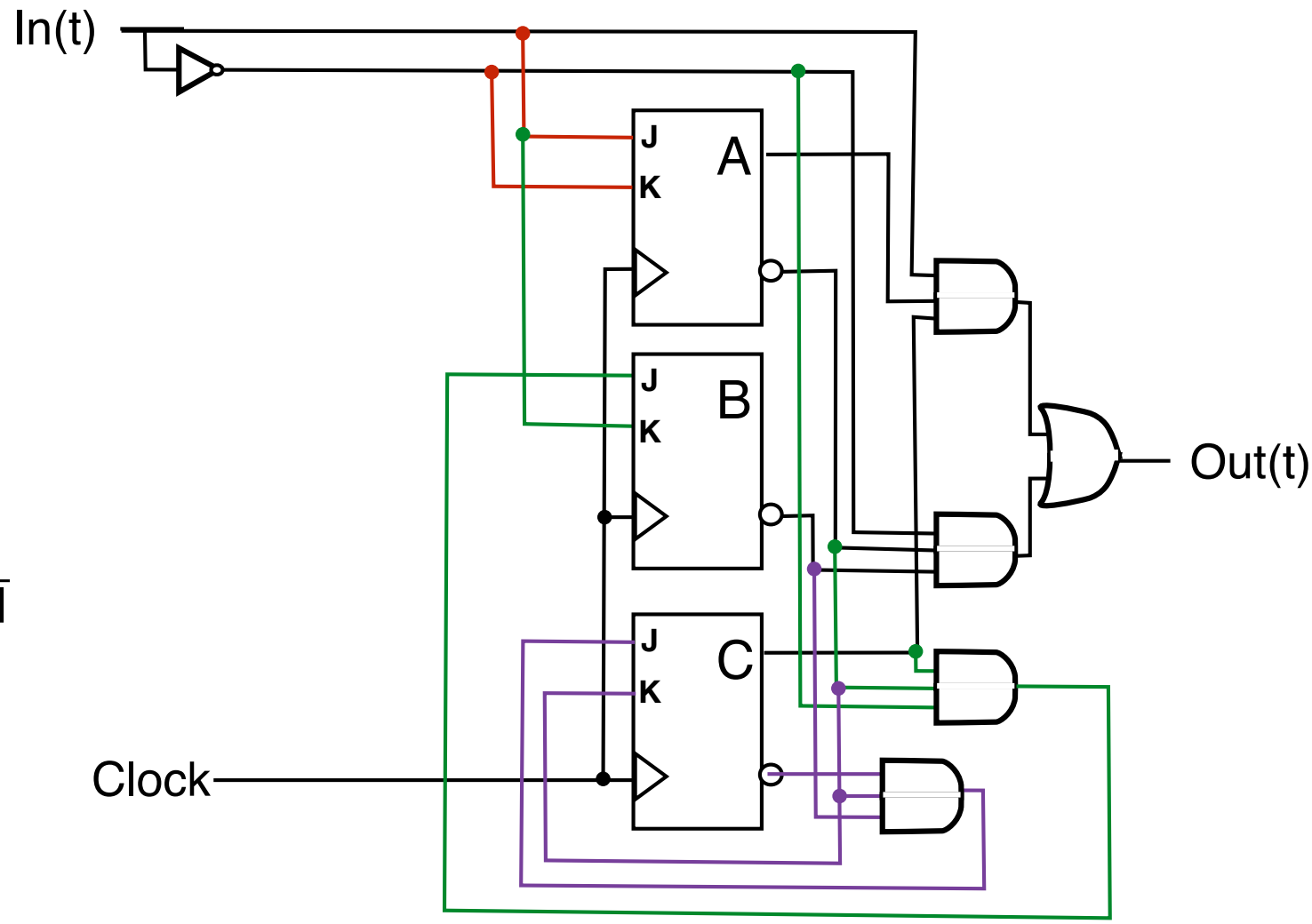
The Sequential Circuit

- $J_A = I$
- $K_A = \bar{I}$
- $J_B = \bar{A}C\bar{I}$
- $K_B = I$
- $J_C = \bar{A}\bar{B}\bar{C}$
- $K_C = \bar{A}$
- $Out = ACI + \bar{A}\bar{B}\bar{I}$



The Sequential Circuit

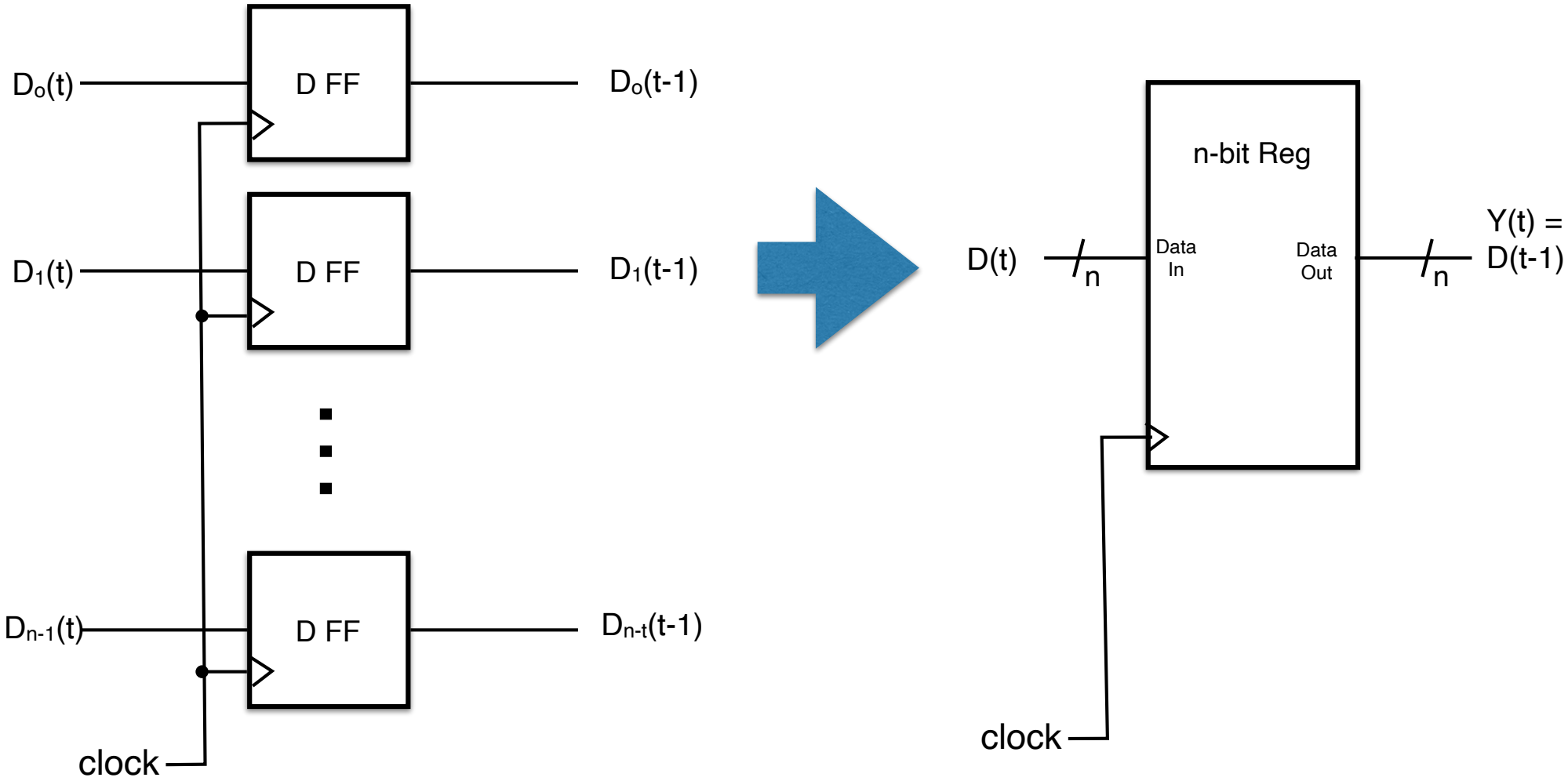
- $J_A = I$
- $K_A = \bar{I}$
- $J_B = \bar{A}C\bar{I}$
- $K_B = I$
- $J_C = \bar{A}\bar{B}\bar{C}$
- $K_C = \bar{A}$
- $Out = ACI + \bar{A}\bar{B}\bar{I}$



Registers

Registers

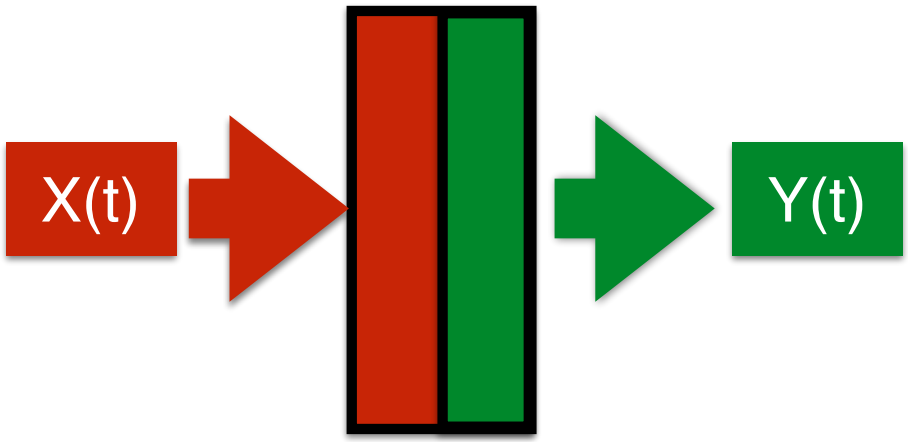
A flip-flop can store 1 bit. A register is a set of n flip flops that stores n bits in parallel.



Register Behavior

- Each clock cycle t, register "set" to an n-bit input value
  $X(t)=X_{n-1}(t)\ X_{n-2}(t)\ \ldots\ X_1(t)\ X_0(t)$
- Output
  - $Y(t) = Y_{n-1}(t)\ Y_{n-2}(t)\ \ldots\ Y_1(t)\ Y_0(t) = X_{n-1}(t\text{-}1)\ X_{n-2}(t\text{-}1)\ \ldots\ X_1(t\text{-}1)\ X_0(t\text{-}1) = X(t\text{-}1)$.
  - and of course, $Y(t+1) = X(t)$
- In other words, the output of a register is the input from the previous clock cycle

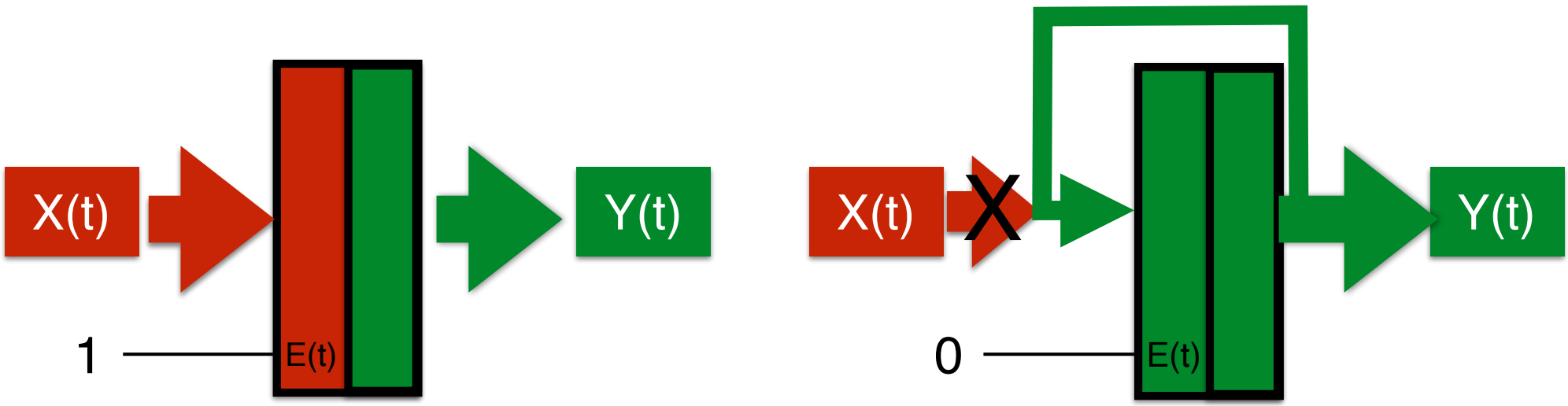
Snapshot of register at time t (t^{th} clock cycle)



Register w/ Write Enable

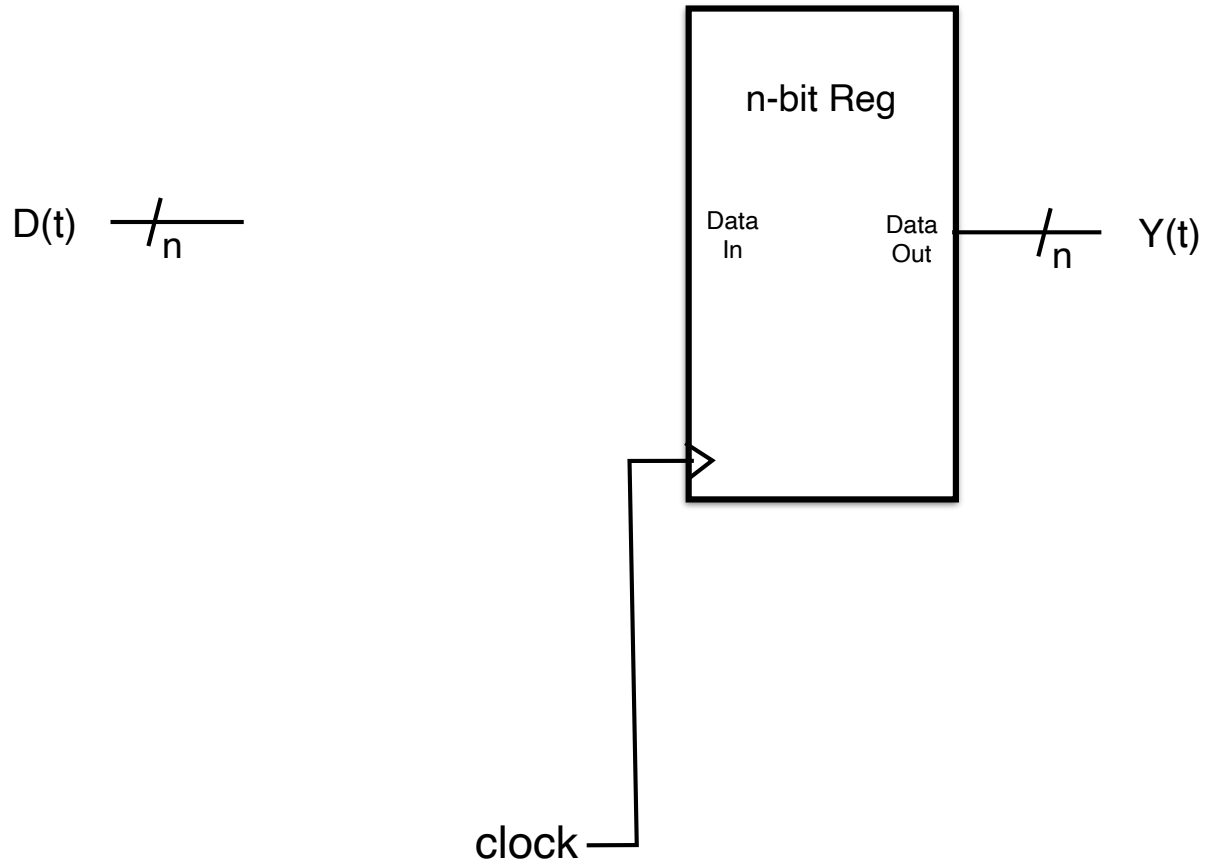
Register w/ Write Enable

- A register with an additional (1-bit) input signal, E(t)
  - E(t)=1: behave like register w/o enable, Y(t+1) = X(t) will be output next clock cycle
  - E(t)=0: hold value in register, ignore current input: Y(t+1) = Y(t)

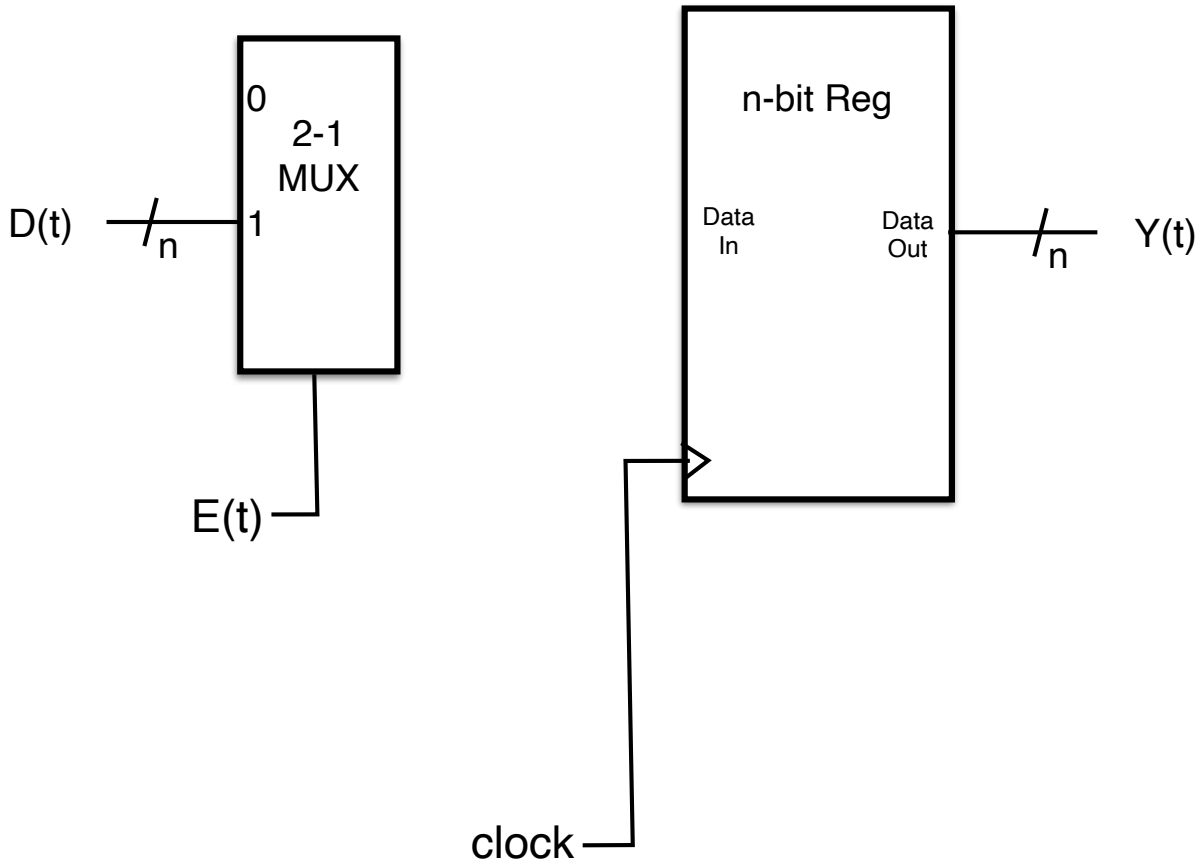


Note: When not enabled, the register value is not being “zero‘d“out!

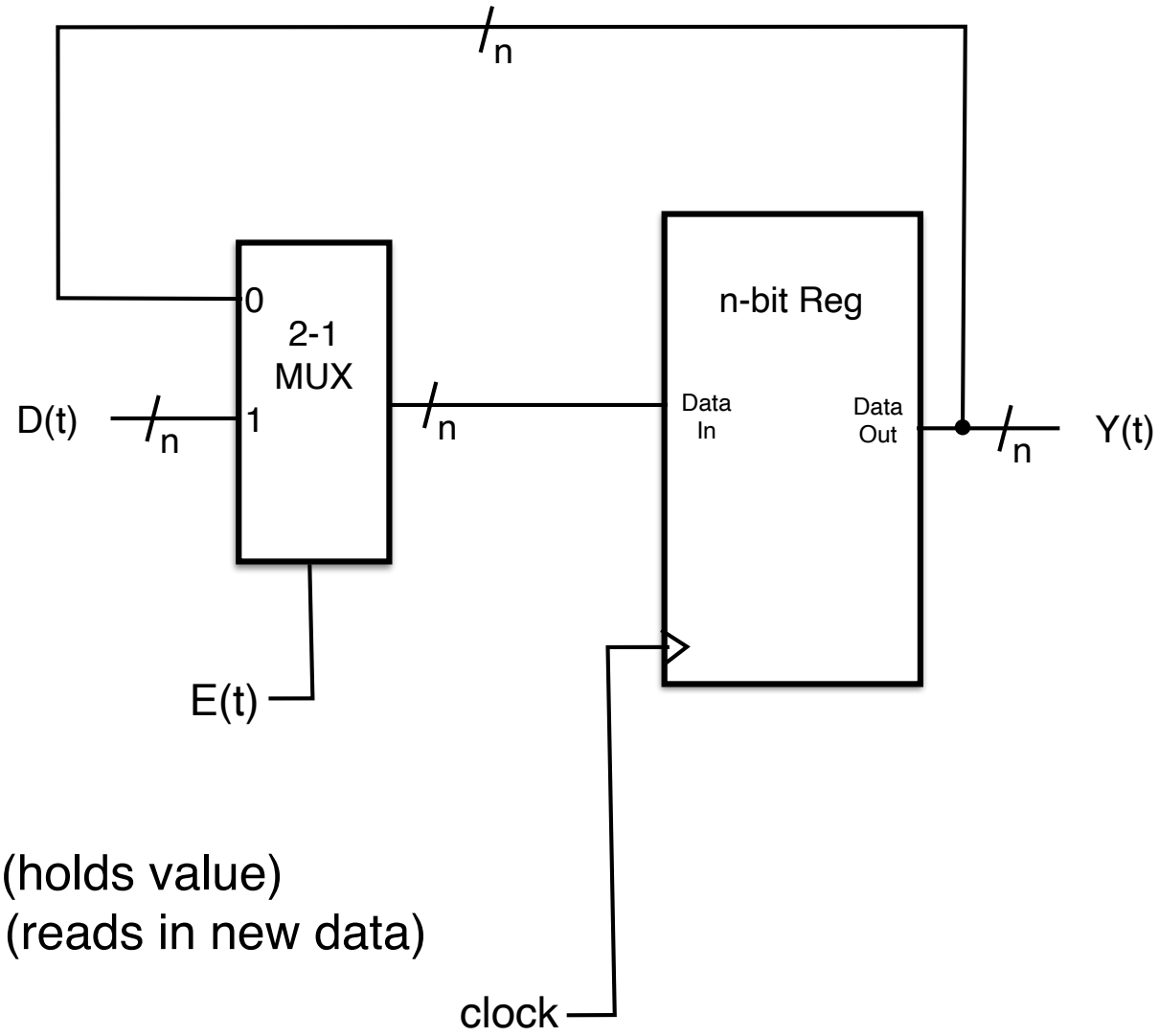
Building Register w/ Write Enable from Register



Building Register w/ Write Enable from Register



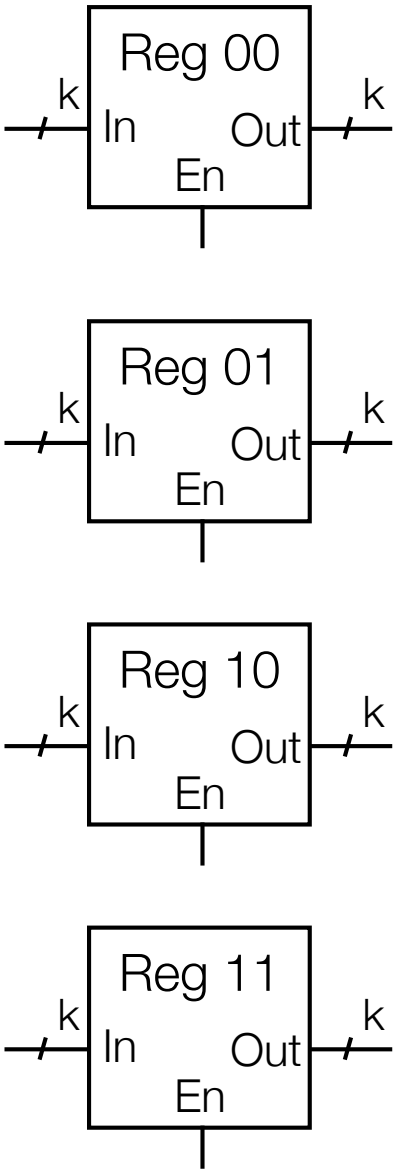
Building Register w/ Write Enable from Register

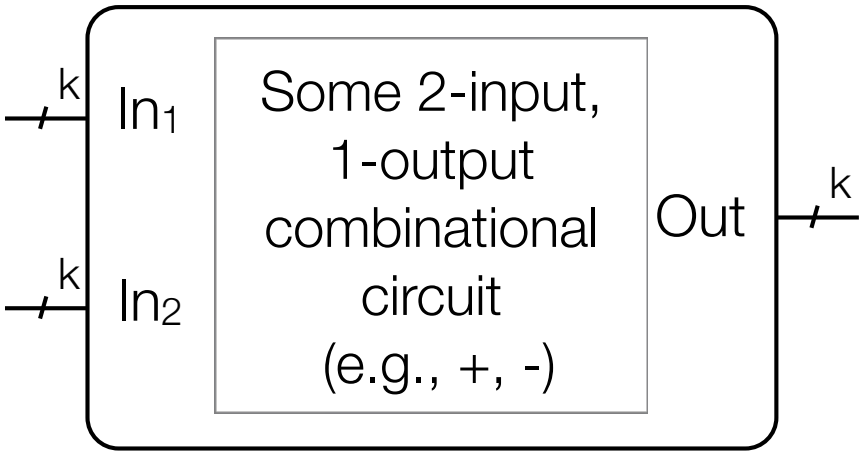


E(t)=0: Y(t+1) = Y(t) (holds value)
E(t)=1: Y(t+1) = D(t) (reads in new data)

Register Selection (using MUXs)

Register MUXing



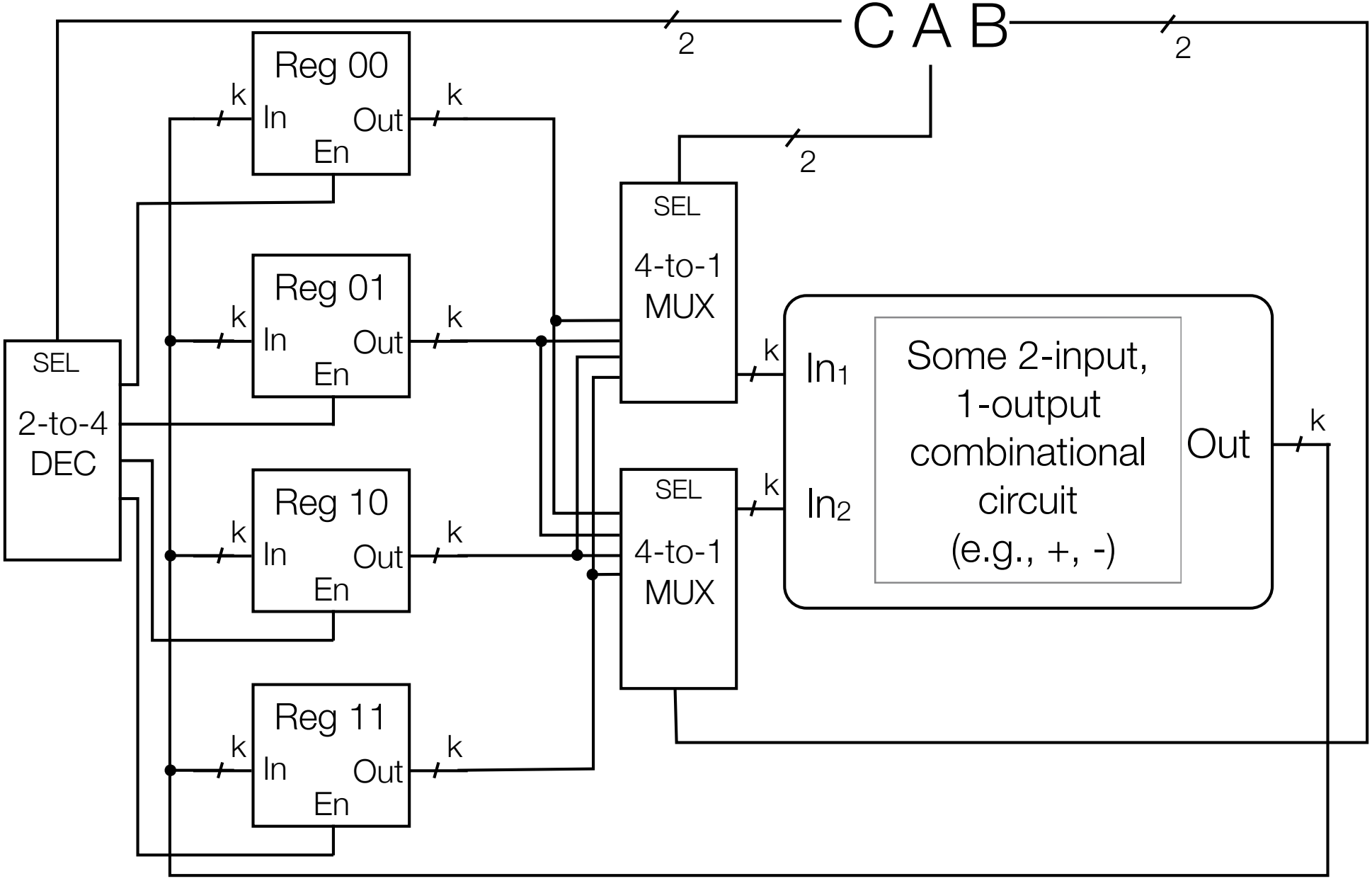


- Suppose have
  - 4 k-bit registers with Enable (for writing)
  - combinational logic to perform some operation OP
- Goal: Allow system to select registers
  - 2 registers selected for inputs (A & B)
  - 1 register for output (C)
  - At end of cycle: C = A OP B
- Note: A&B can be same register, C can also be same as A or B

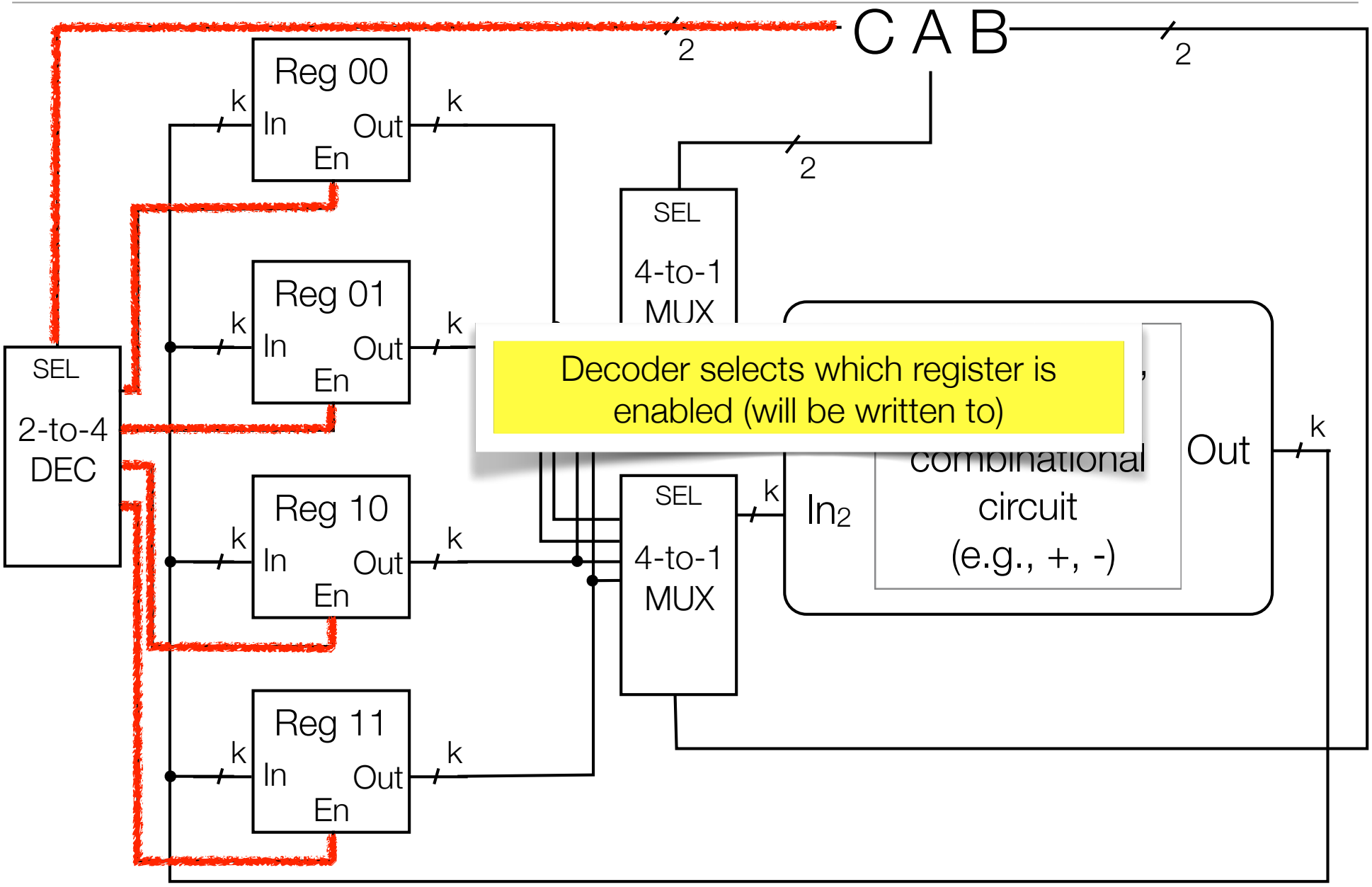
Register MUXing example

- e.g.,
  - Reg01 = Reg01 OP Reg10
  - Reg10 = Reg10 OP Reg10 (if OP were +, this would do Reg10 *= 2)
- OP selecting: be able to choose from different OPs, e.g.
  - Reg01 = Reg01 + Reg10
  - Reg10 = Reg01 * Reg00

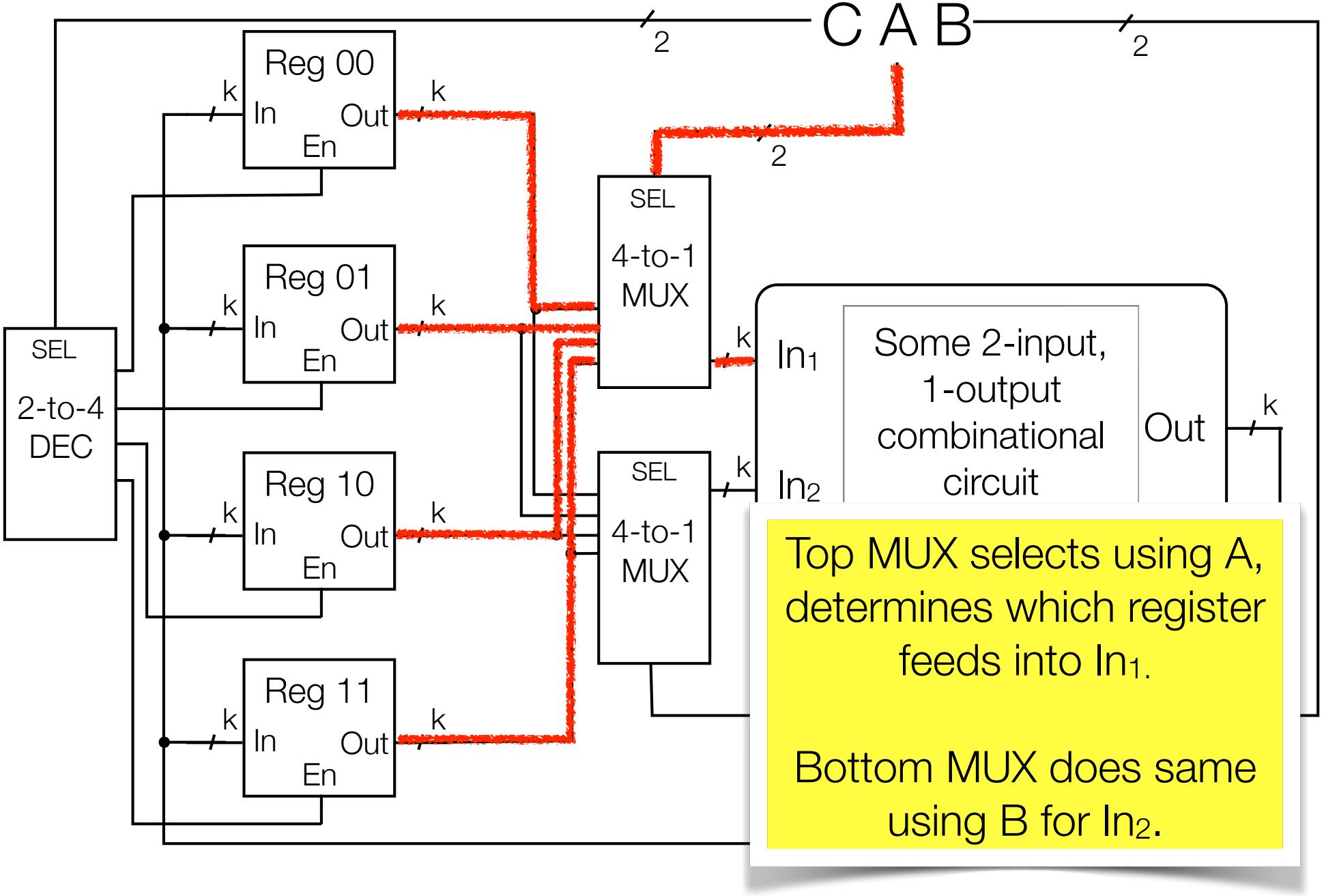
Register MUXing



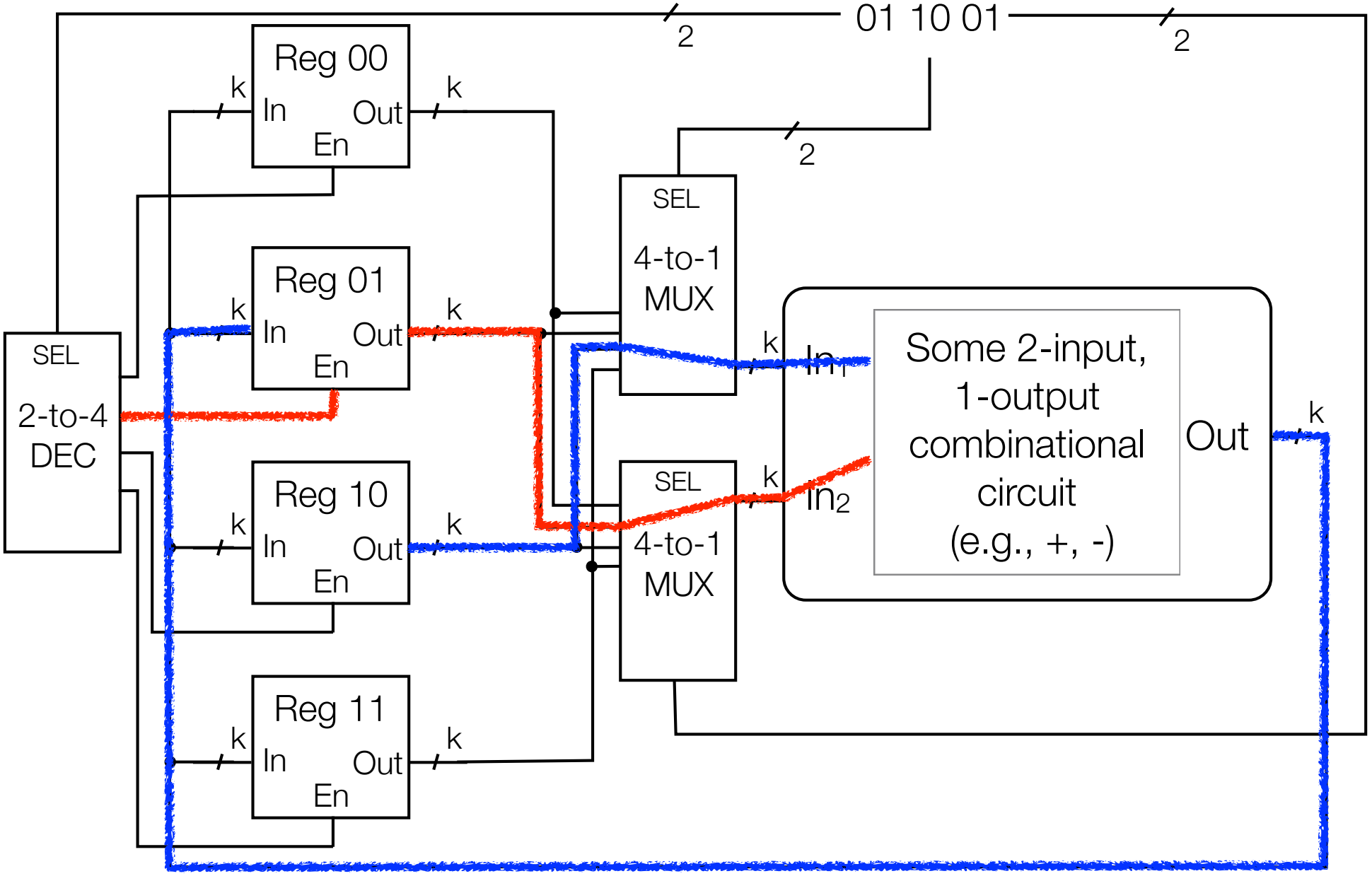
Register MUXing



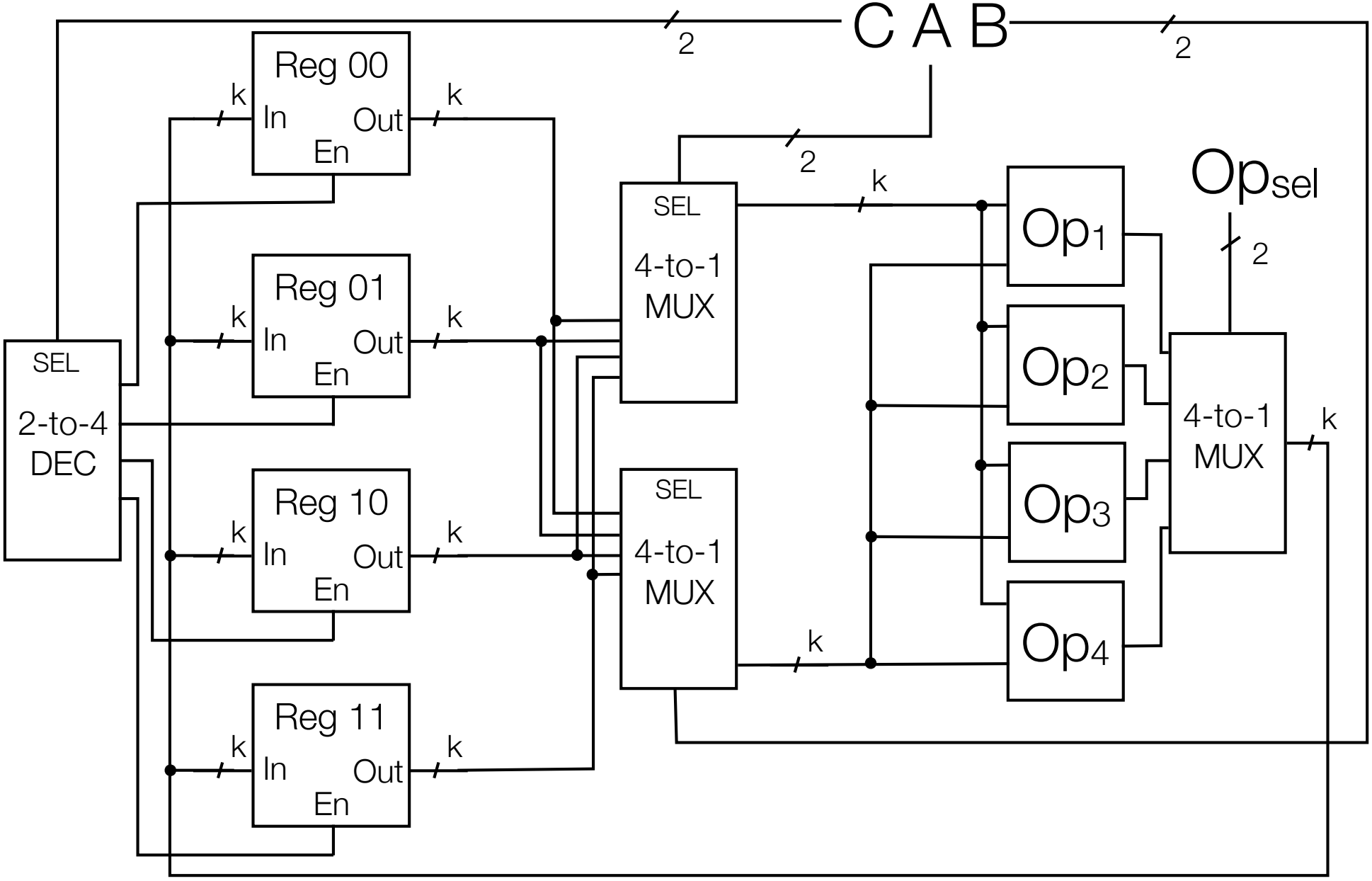
Register MUXing



Example: Reg01 = Reg10 OP Reg01



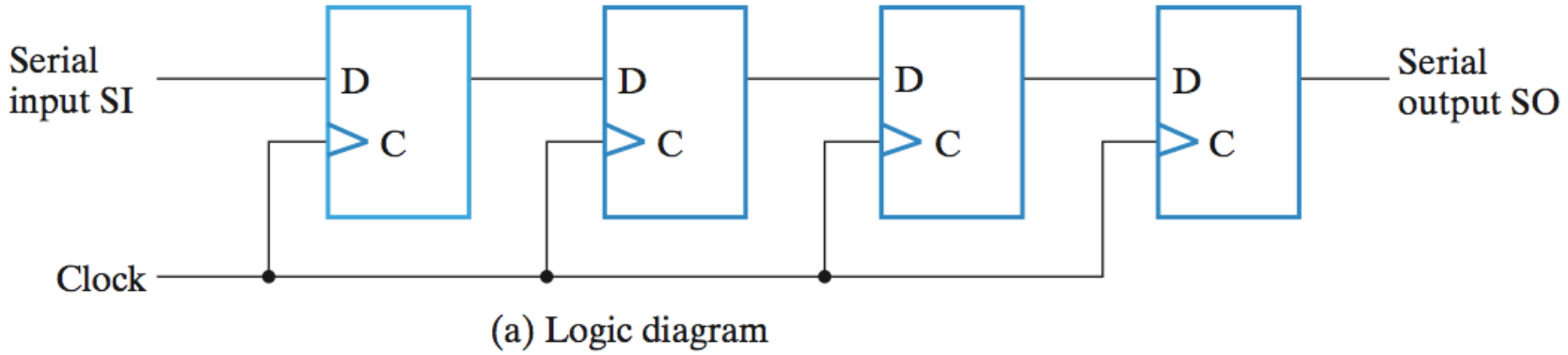
Register MUXing and OP MUXing



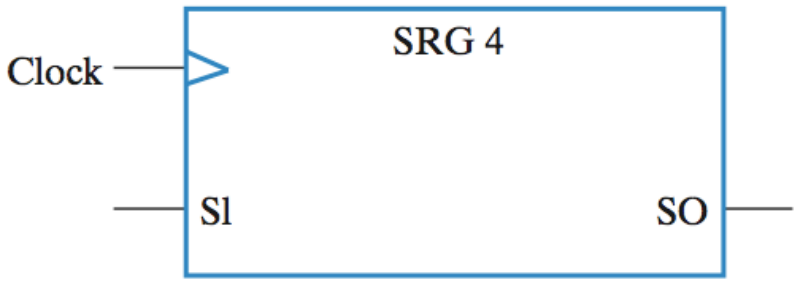
Specialized Registers

Shift register

A register capable of shifting bits laterally in one direction

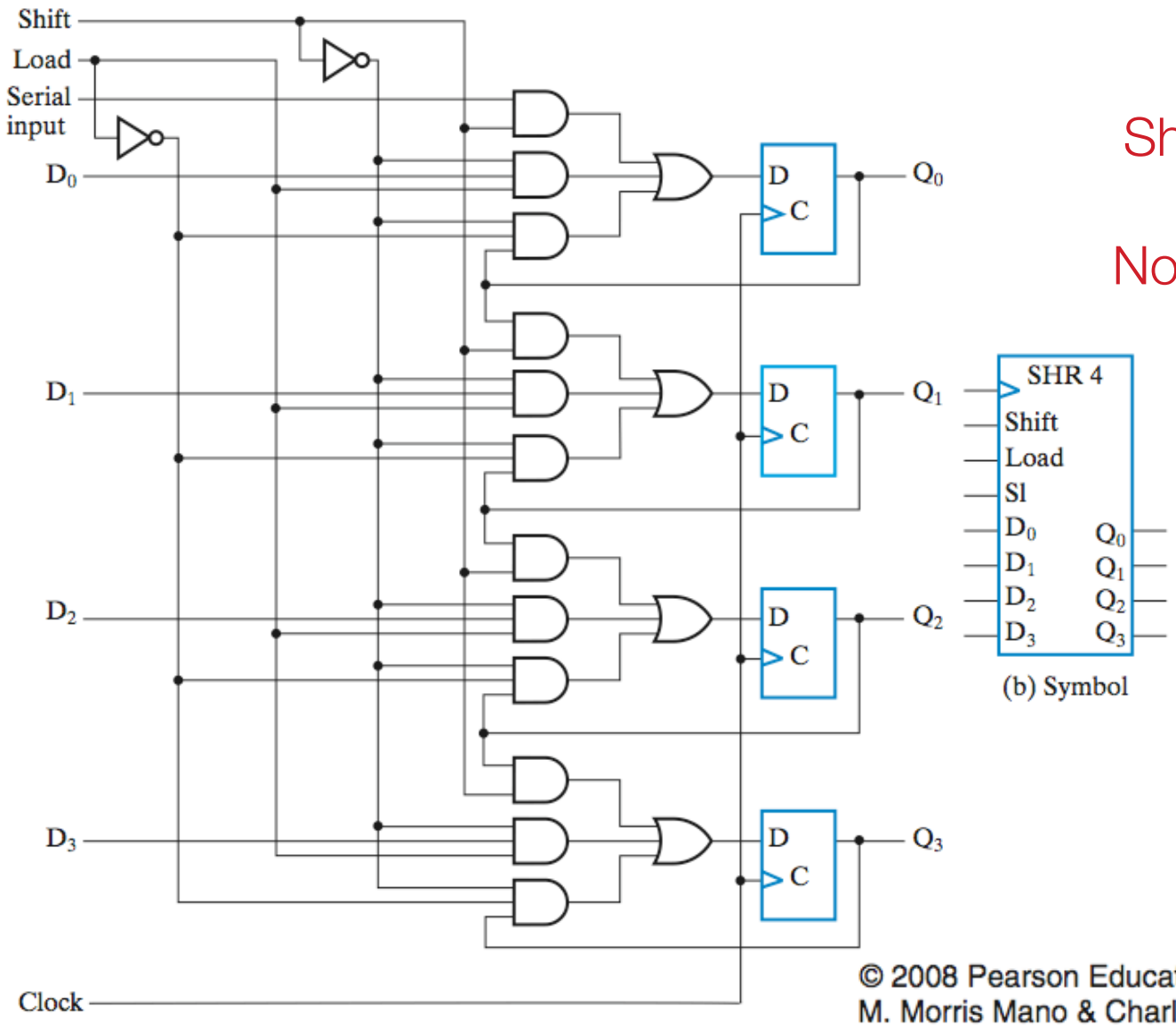


(a) Logic diagram



(b) Symbol

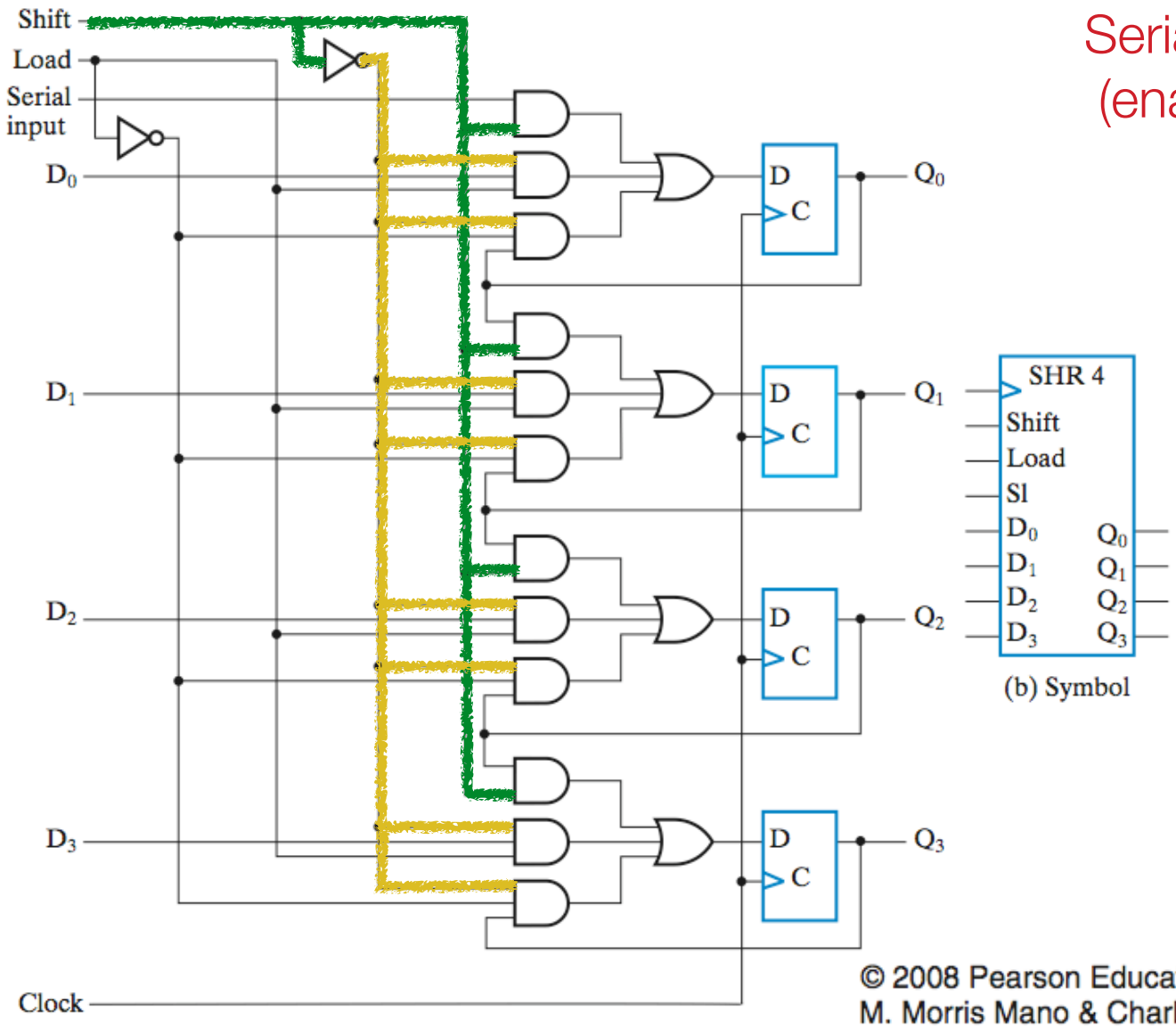
Shift register w. parallel load



(b) Symbol

Three modes:
Shift (w/ serial input)
Parallel input
No input (do nothing)

Shift register w. parallel load

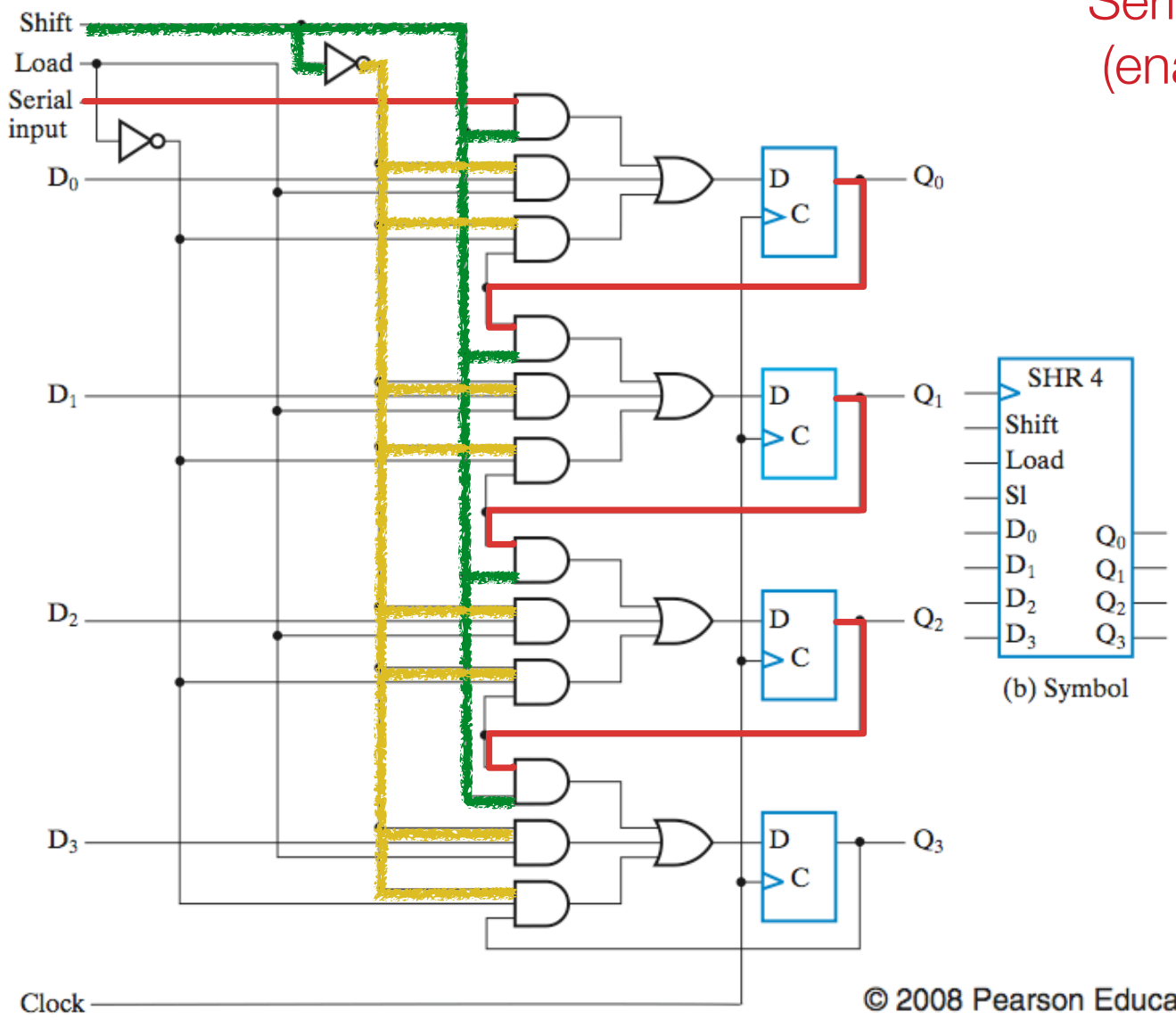


Serial input operation (enabled by Shift=1)

Enable Signal (1)

Disable Signal (0)

Shift register w. parallel load



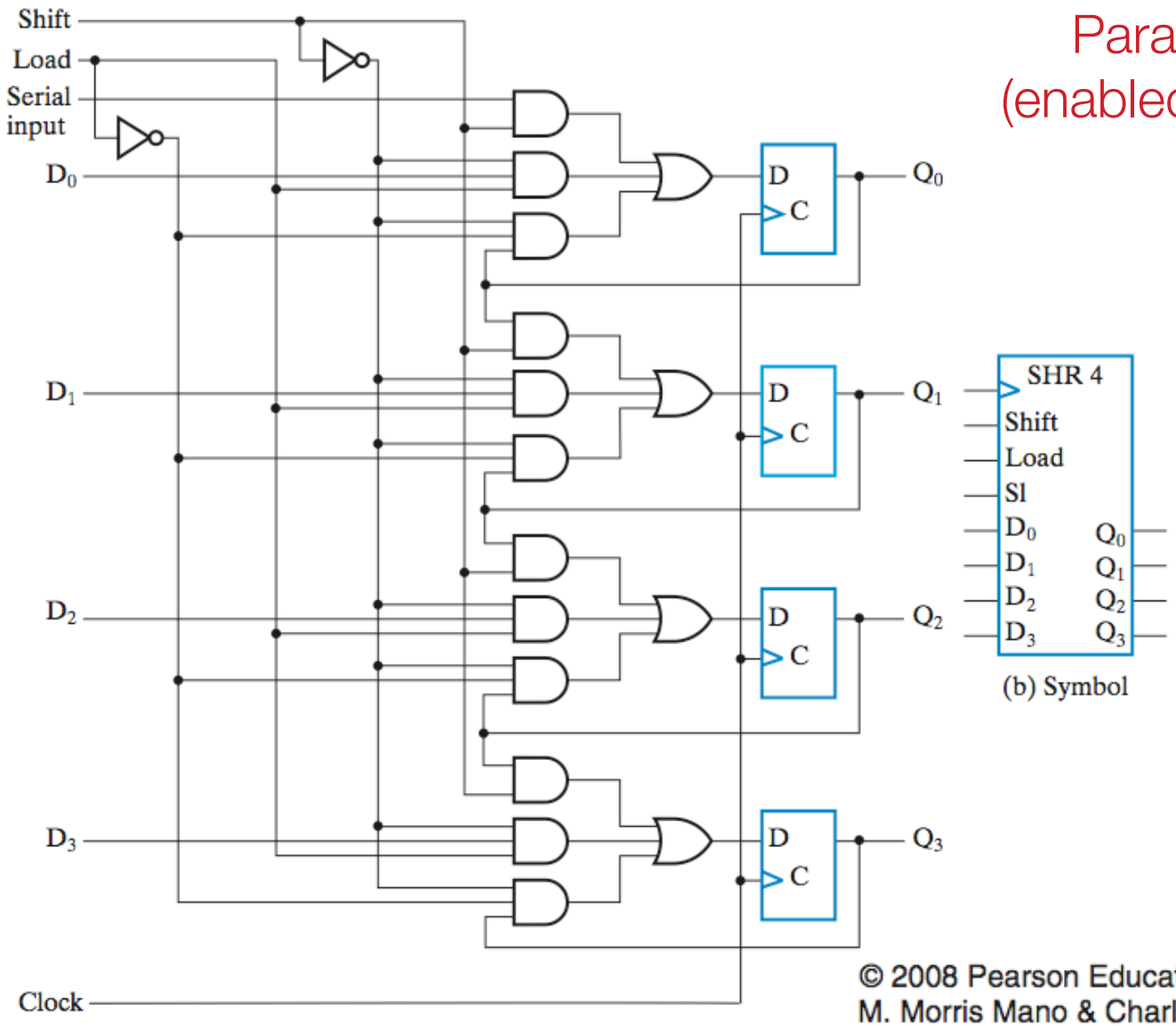
Serial input operation (enabled by Shift=1)

Enable Signal (1)

Disable Signal (0)

M. Morris Mano & Charles R. Kime
LOGIC AND COMPUTER DESIGN FUNDAMENTALS, 4e

Shift register w. parallel load

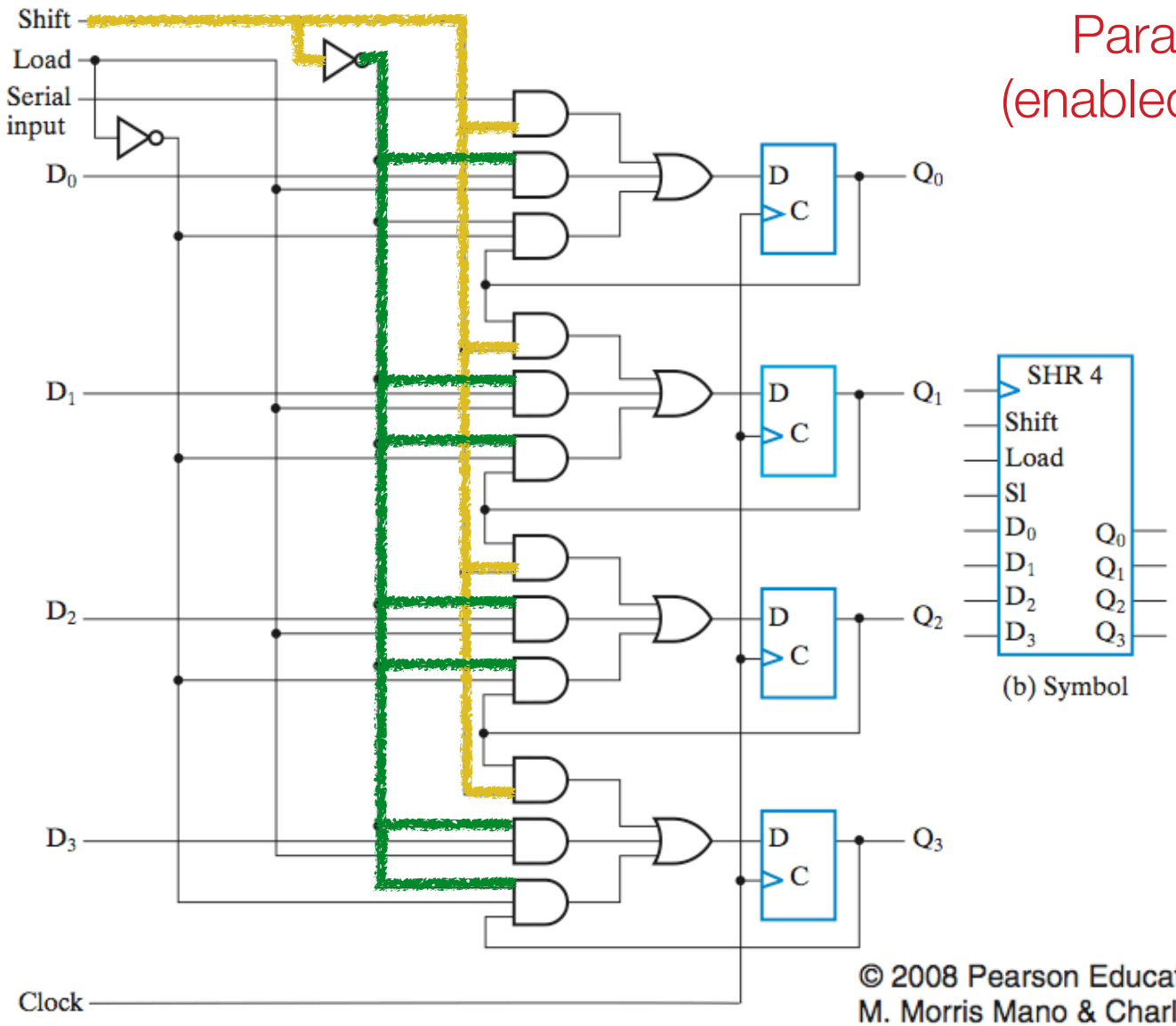


(b) Symbol

Parallel load operation
(enabled by Load AND $\overline{\text{Shift}}$)

M. Morris Mano & Charles R. Kime
LOGIC AND COMPUTER DESIGN FUNDAMENTALS, 4e

Shift register w. parallel load



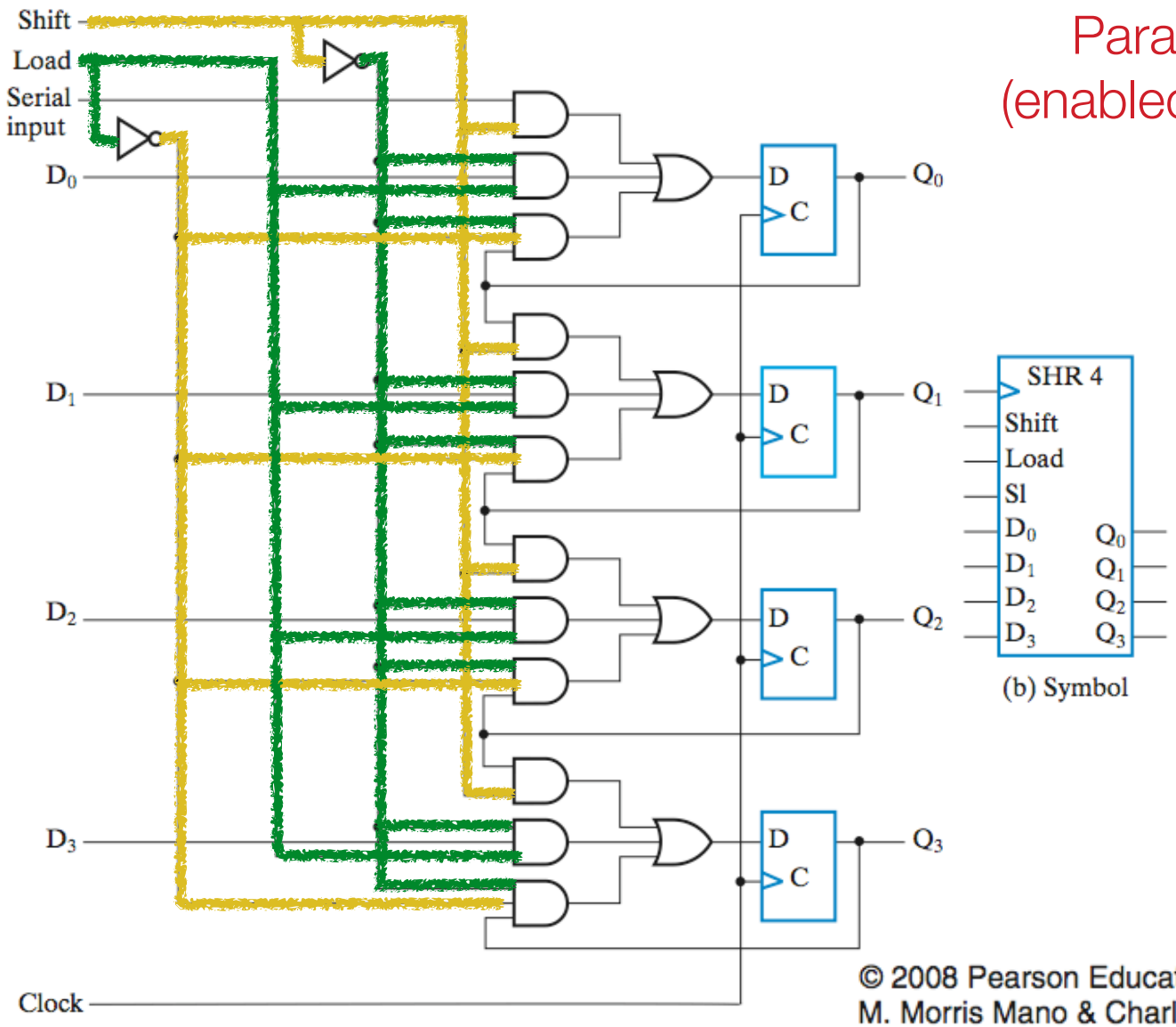
Parallel load operation
(enabled by Load AND $\overline{\text{Shift}}$)

Enable Signal (1)

Disable Signal (0)

M. Morris Mano & Charles R. Kime
LOGIC AND COMPUTER DESIGN FUNDAMENTALS, 4e

Shift register w. parallel load



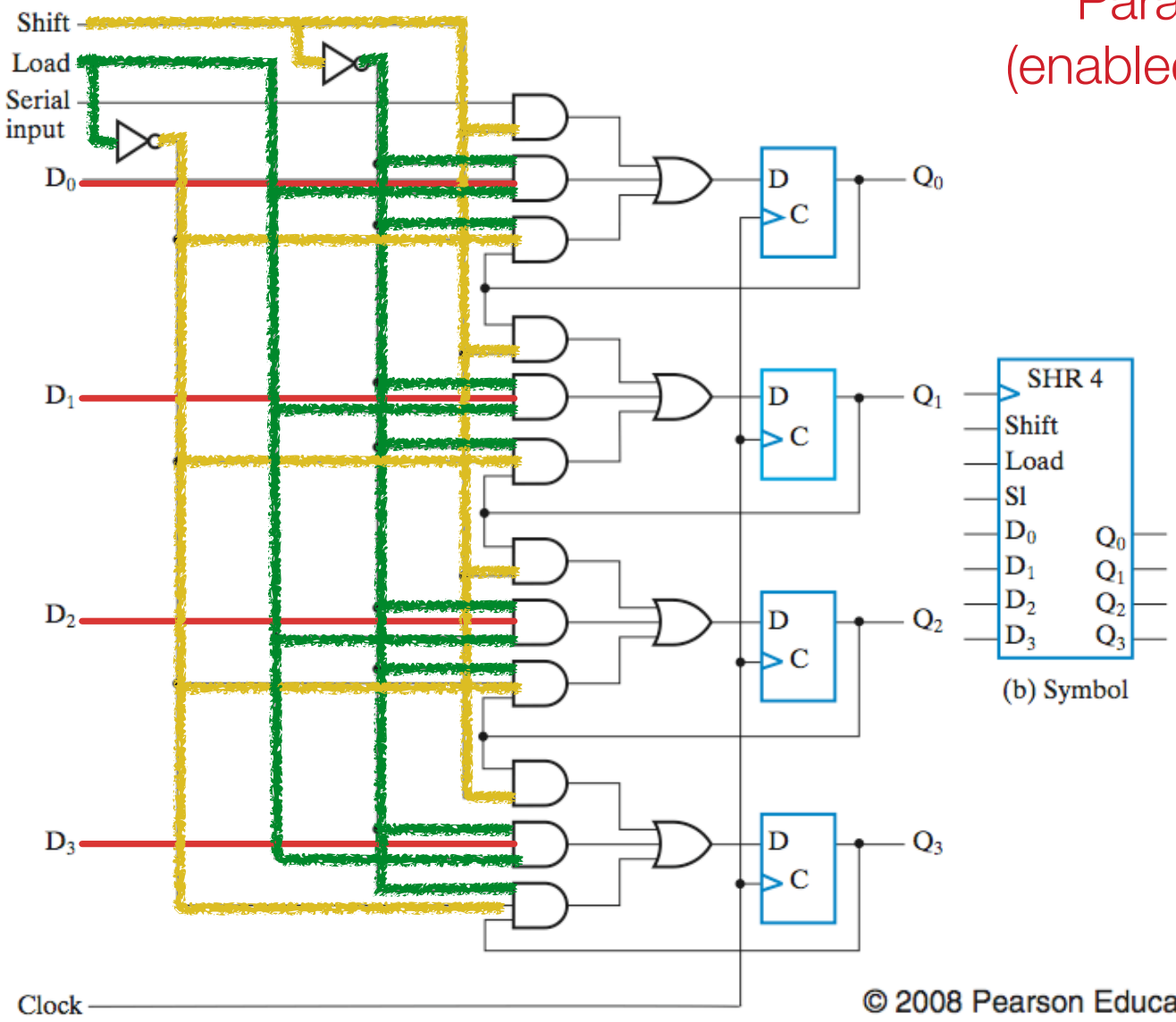
Parallel load operation
(enabled by Load AND $\overline{\text{Shift}}$)

Enable Signal (1)

Disable Signal (0)

(b) Symbol

Shift register w. parallel load



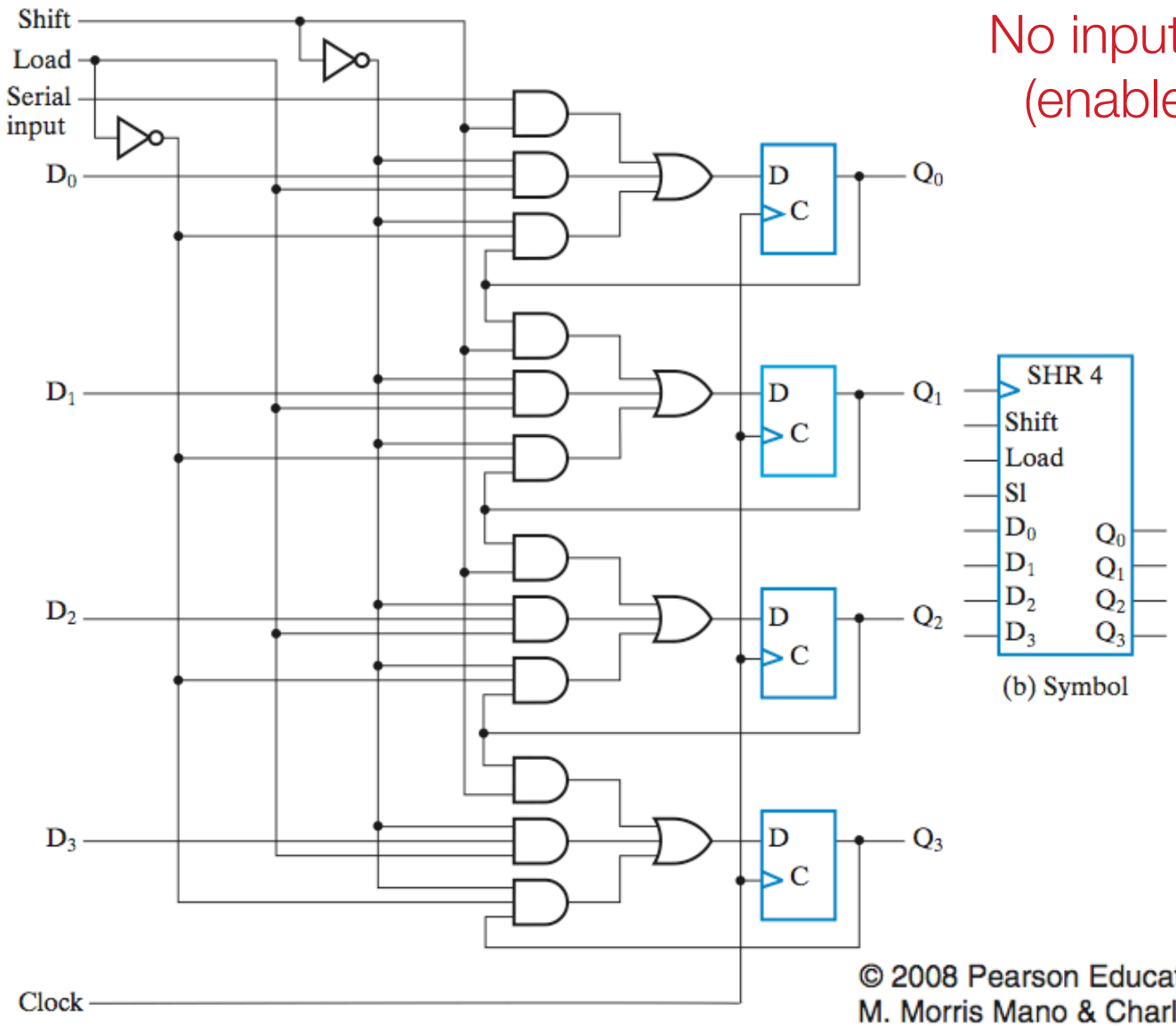
Parallel load operation
(enabled by Load AND $\overline{\text{Shift}}$)

Enable Signal (1)

Disable Signal (0)

M. Morris Mano & Charles R. Kime
LOGIC AND COMPUTER DESIGN FUNDAMENTALS, 4e

Shift register w. parallel load

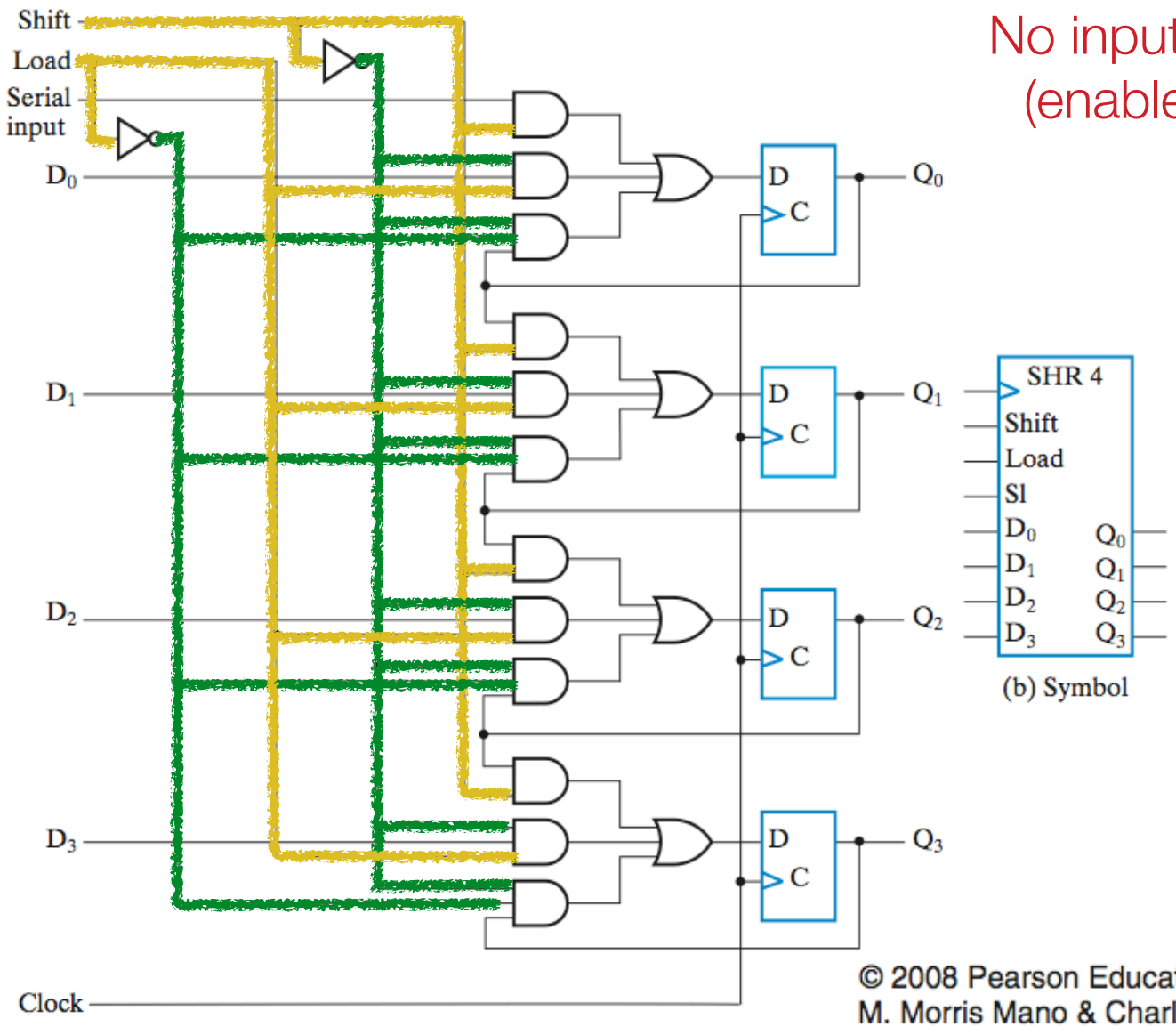


No input (Do nothing) operation
(enabled by $\overline{\text{Load}}$ AND $\overline{\text{Shift}}$)

(b) Symbol

M. Morris Mano & Charles R. Kime
LOGIC AND COMPUTER DESIGN FUNDAMENTALS, 4e

Shift register w. parallel load



No input (Do nothing) operation
(enabled by $\overline{\text{Load}}$ AND $\overline{\text{Shift}}$)

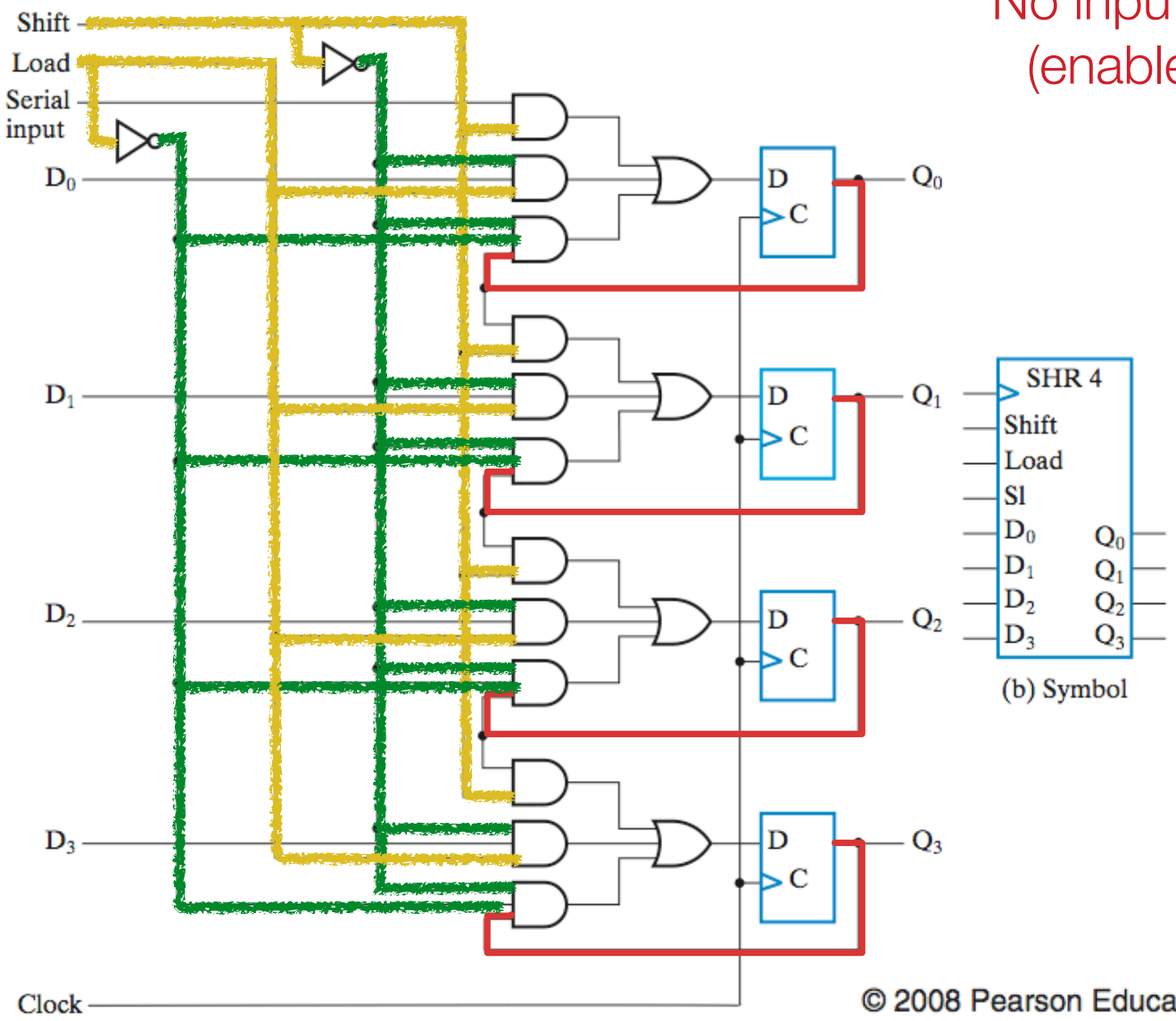
Enable Signal (1)

Disable Signal (0)

(b) Symbol

M. Morris Mano & Charles R. Kime
LOGIC AND COMPUTER DESIGN FUNDAMENTALS, 4e

Shift register w. parallel load



No input (Do nothing) operation
(enabled by $\overline{\text{Load}}$ AND $\overline{\text{Shift}}$)

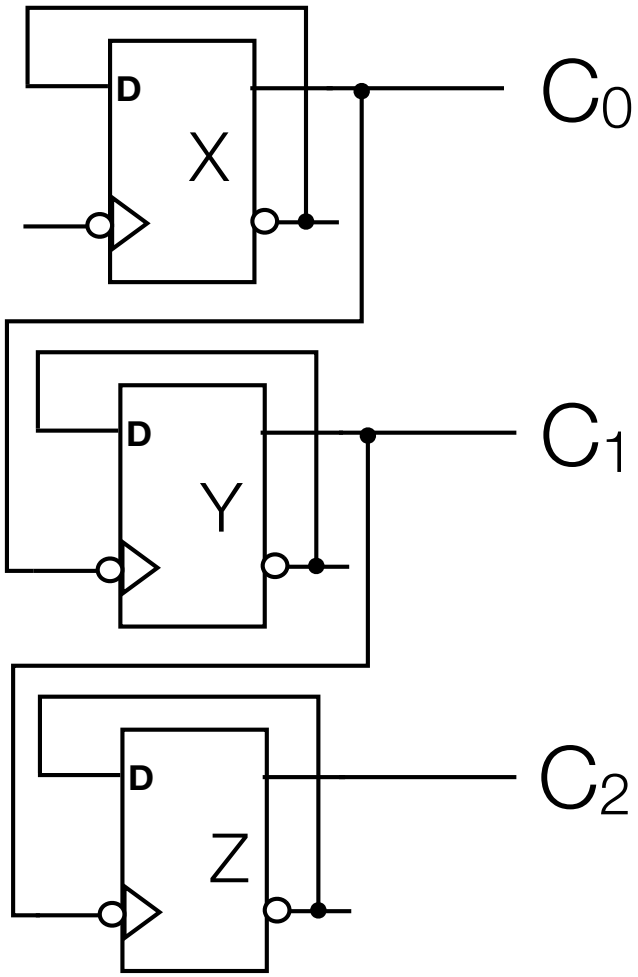
Enable Signal (1)

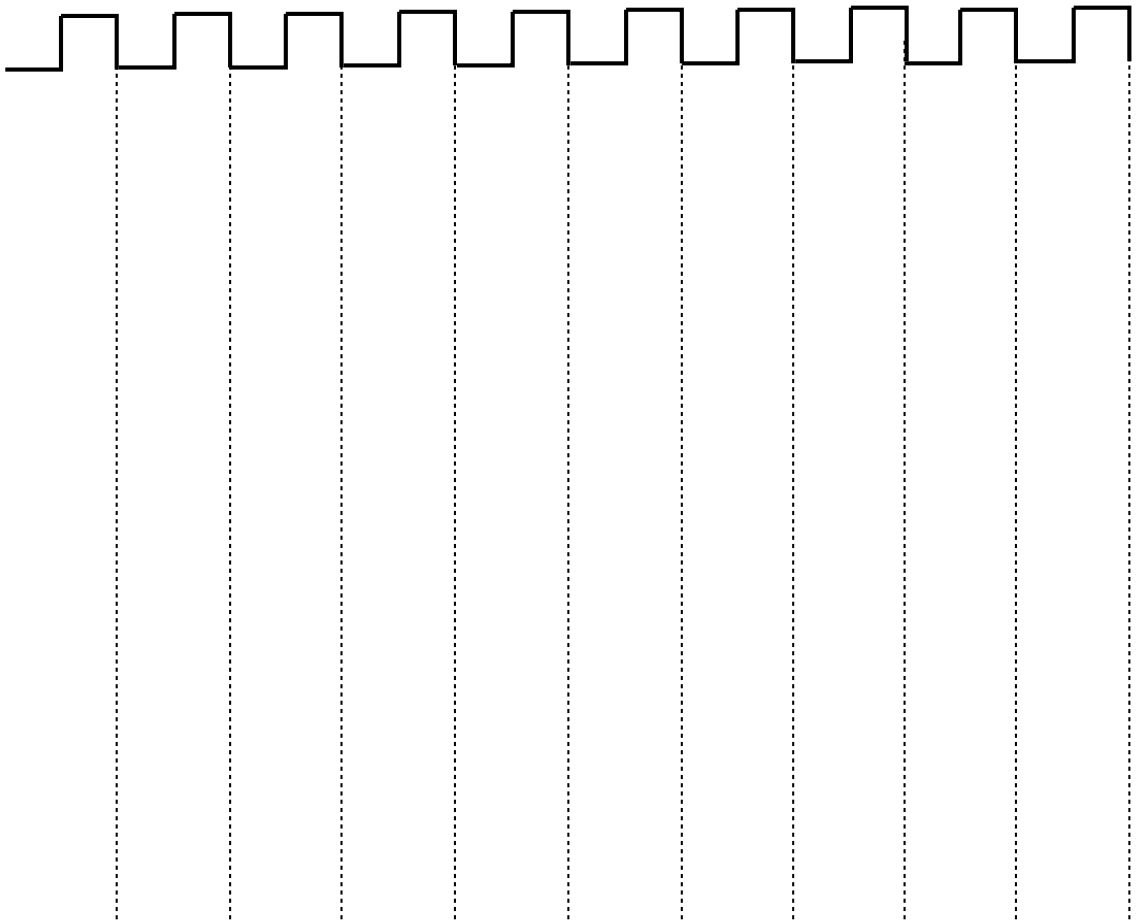
Disable Signal (0)

(b) Symbol

M. Morris Mano & Charles R. Kime
LOGIC AND COMPUTER DESIGN FUNDAMENTALS, 4e

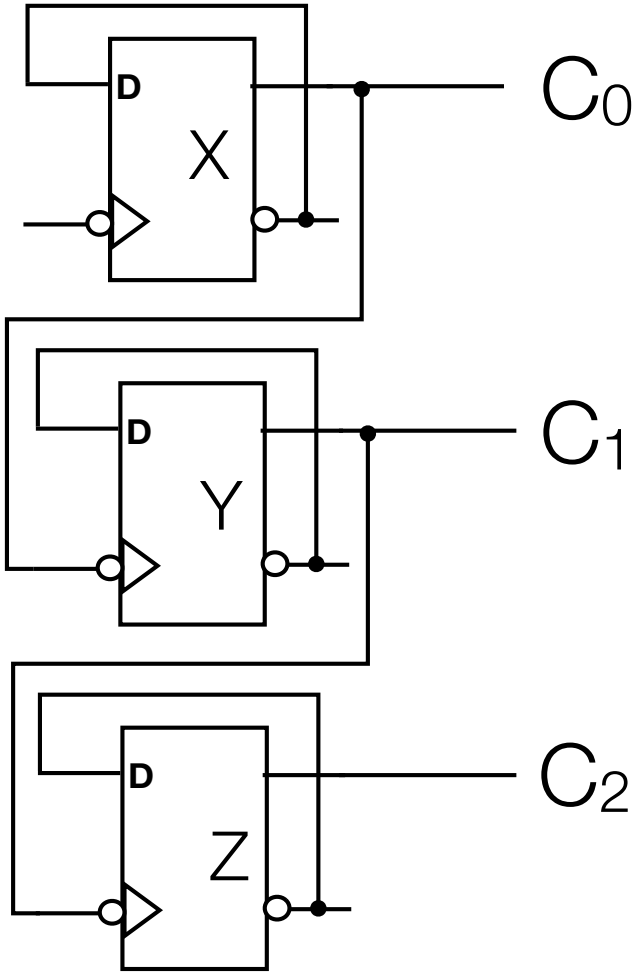
Ripple Counter

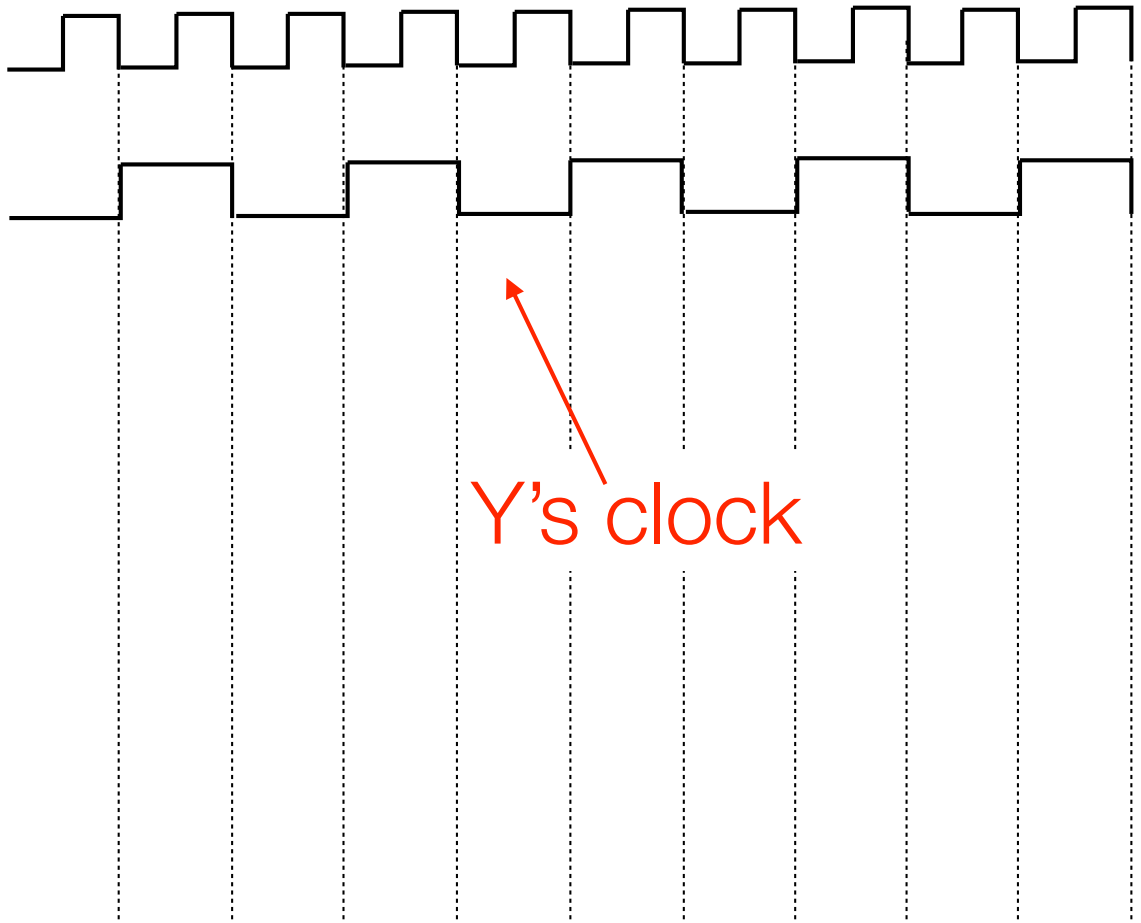




- Each flip-flop feeds Q into D: complements its value
- X's Q feeds into Y's clock, Y's Q feeds into Z's clock - why?

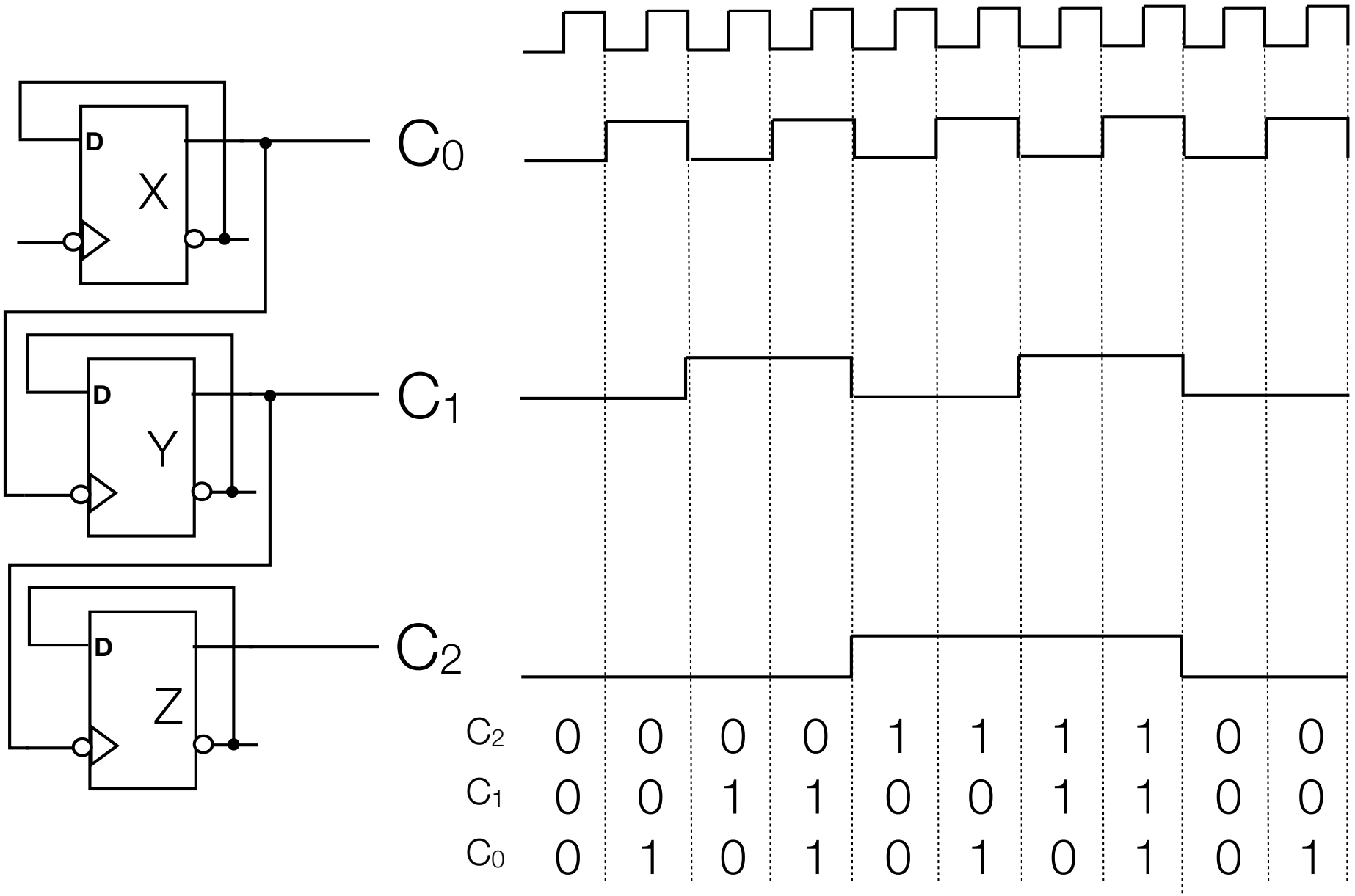
Ripple Counter



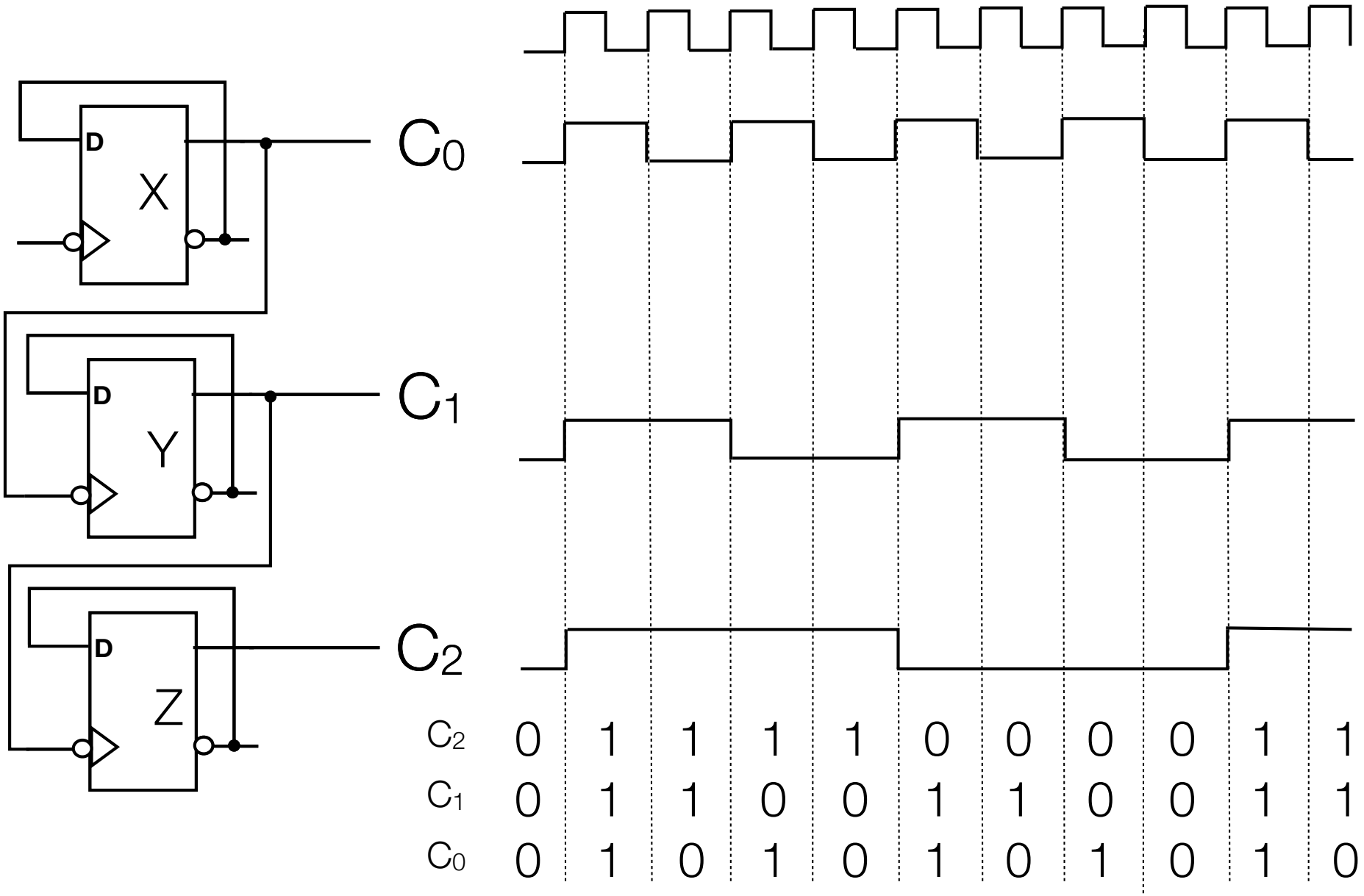


- Note: each FF is negative-edge triggered

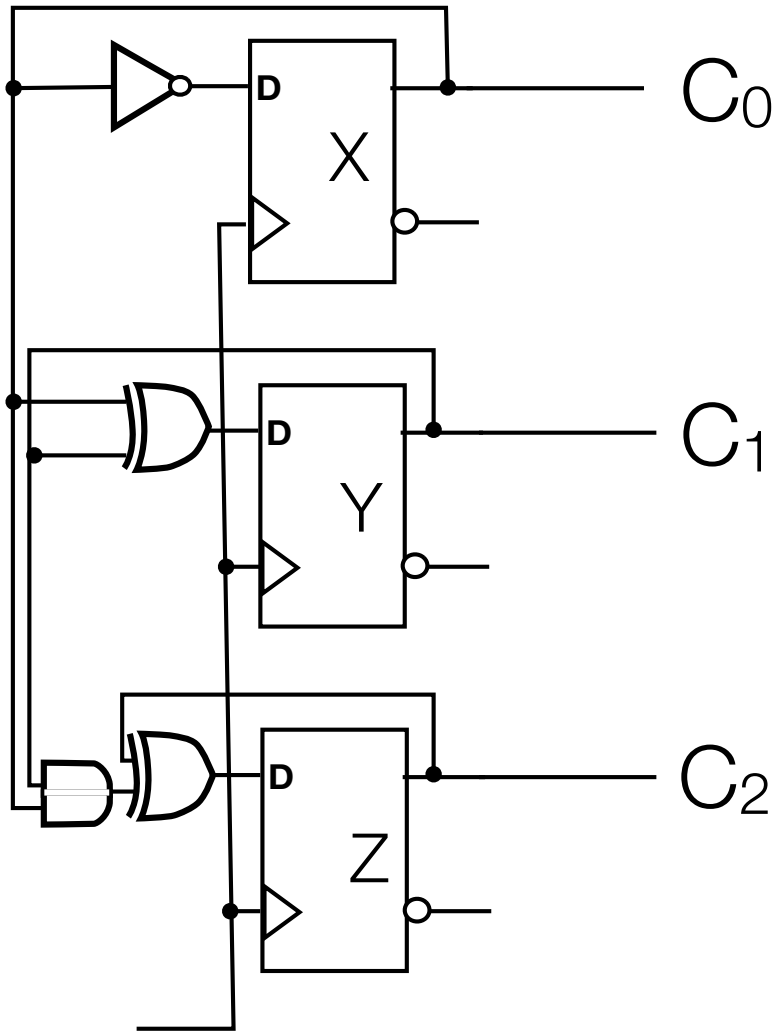
Ripple Counter



Ripple Counter with Positive Edge Triggering



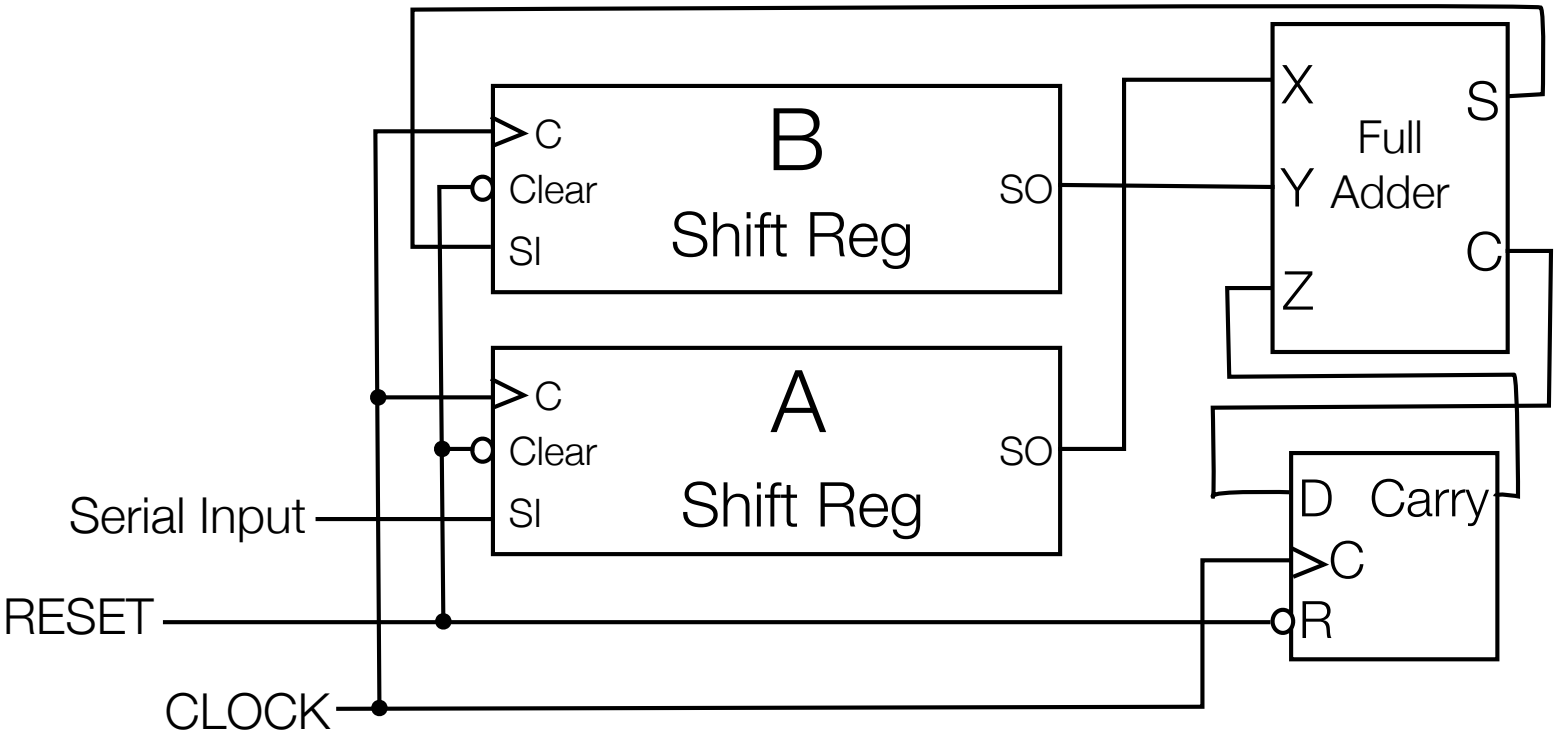
Serial Counter (counting upward)



C_2	0	0	0	0	1	1	1	1	0	0
C_1	0	0	1	1	0	0	1	1	0	0
C_0	0	1	0	1	0	1	0	1	0	1

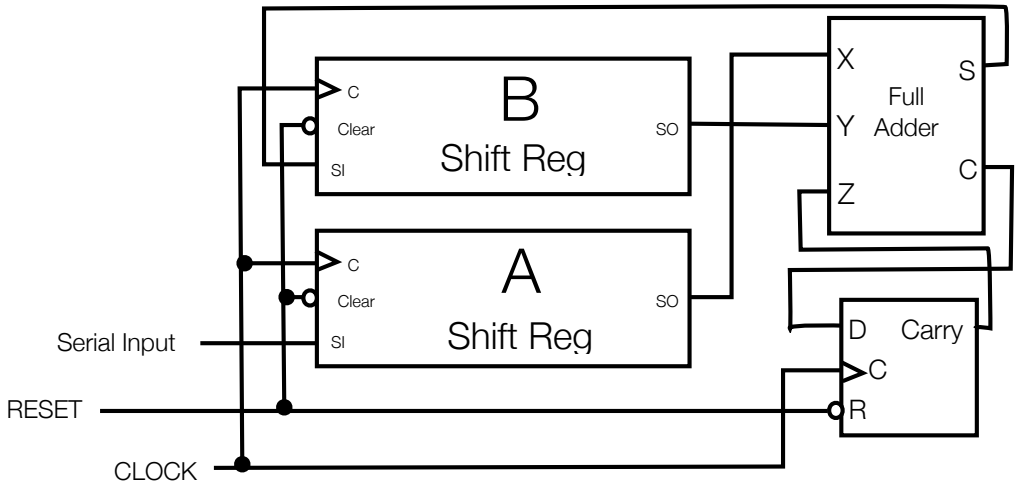
- Y(t+1) differs from Y(t) only when X(t)=1
- Z(t+1) differs from Z(t) when both X(t)=1 and Y(t)=1
- so Y changes when X (evaluates to TRUE), Z changes when XY (evaluates to TRUE)
- $X = \bar{X}$, $Y = Y \oplus X$, $Z = Z \oplus XY$
- for the jth bit (in a j+1 bit counter), $B_j = B_j \oplus Bj_{-1}B_{j-2}...B_1B_0$

Serial Adder Circuit



- New Value pushed into A register
- Result pushed into B register
- If Shift Reg holds N bits, SUM starts calculating during Nth cycle (0 is initial cycle)
- Done 2N cycles later (3N total cycles)

Serial Adder Circuit Example: 0101 + 0011 = 1000



To time 7, just rolling values into registers by adding with 0's (B output)

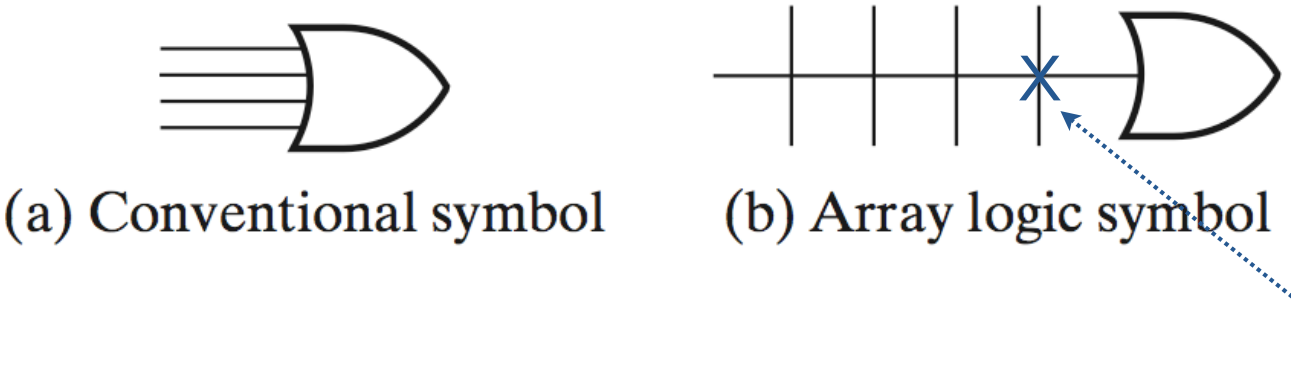
Adding starts time 8 with least significant bits

Time	In	A	B	S	C
0	1	"0000"	"0000"	0	0
1	0	"1000"	"0000"	0	0
2	1	"0100"	"0000"	0	0
3	0	"1010"	"0000"	0	0
4	1	"0101"	"0000"	1	0
5	1	"1010"	"1000"	0	0
6	0	"1101"	"0100"	1	0
7	0	"0110"	"1010"	0	0
8	X	"0011"	"0101"	0	1
9	X	"X001"	"0010"	0	1
10	X	"XX00"	"0001"	0	1
11	X	"XXX0"	"0000"	1	0
12	X	"XXXX"	"1000"	0	0

Programmable Logic Devices

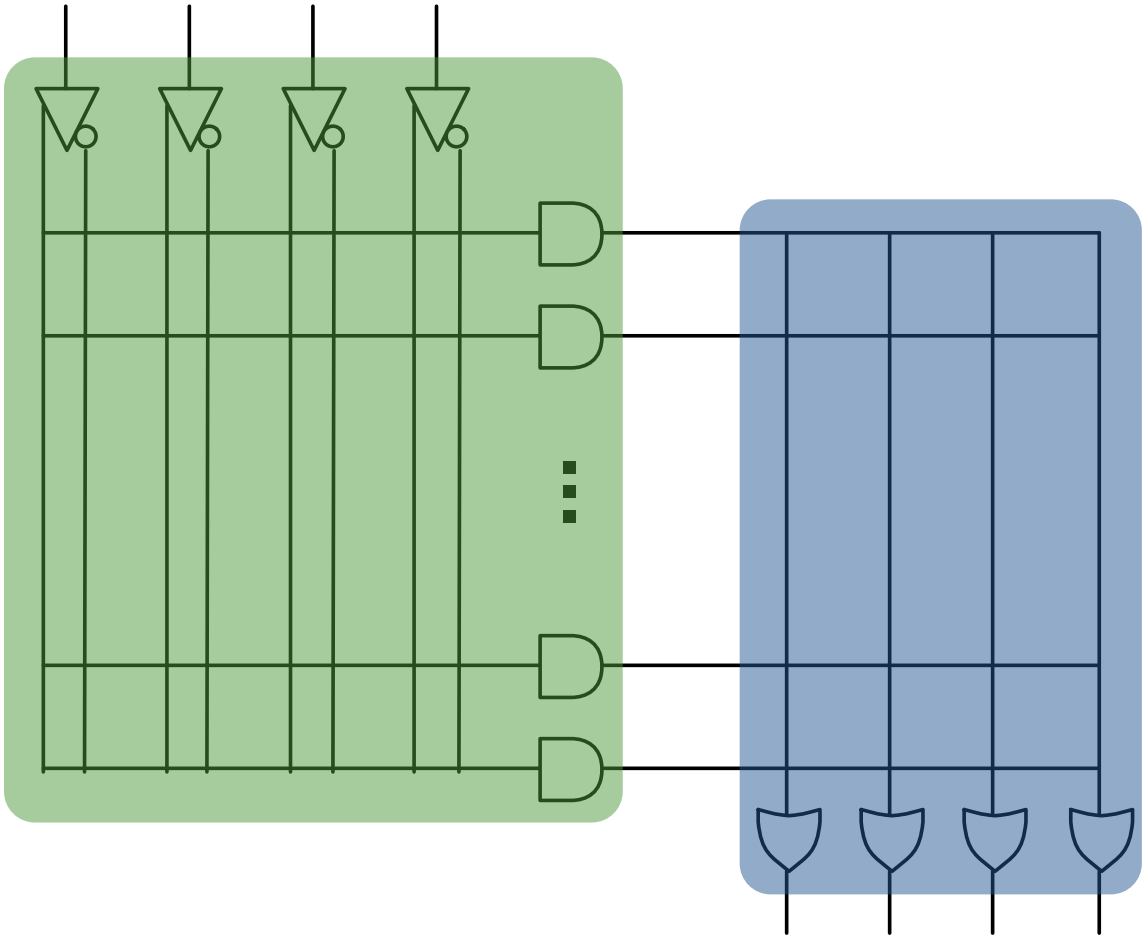
Programmable Logic Devices

- Programmable logic devices (PLDs)
  - Structured like memories
  - Used to implement combinational logic



  - "X" on array logic means wire connected to logic gate (e.g. above)
  - Connections can be either permanent (e.g., fuse, mask) or not (e.g., Flash)

General PLD architecture

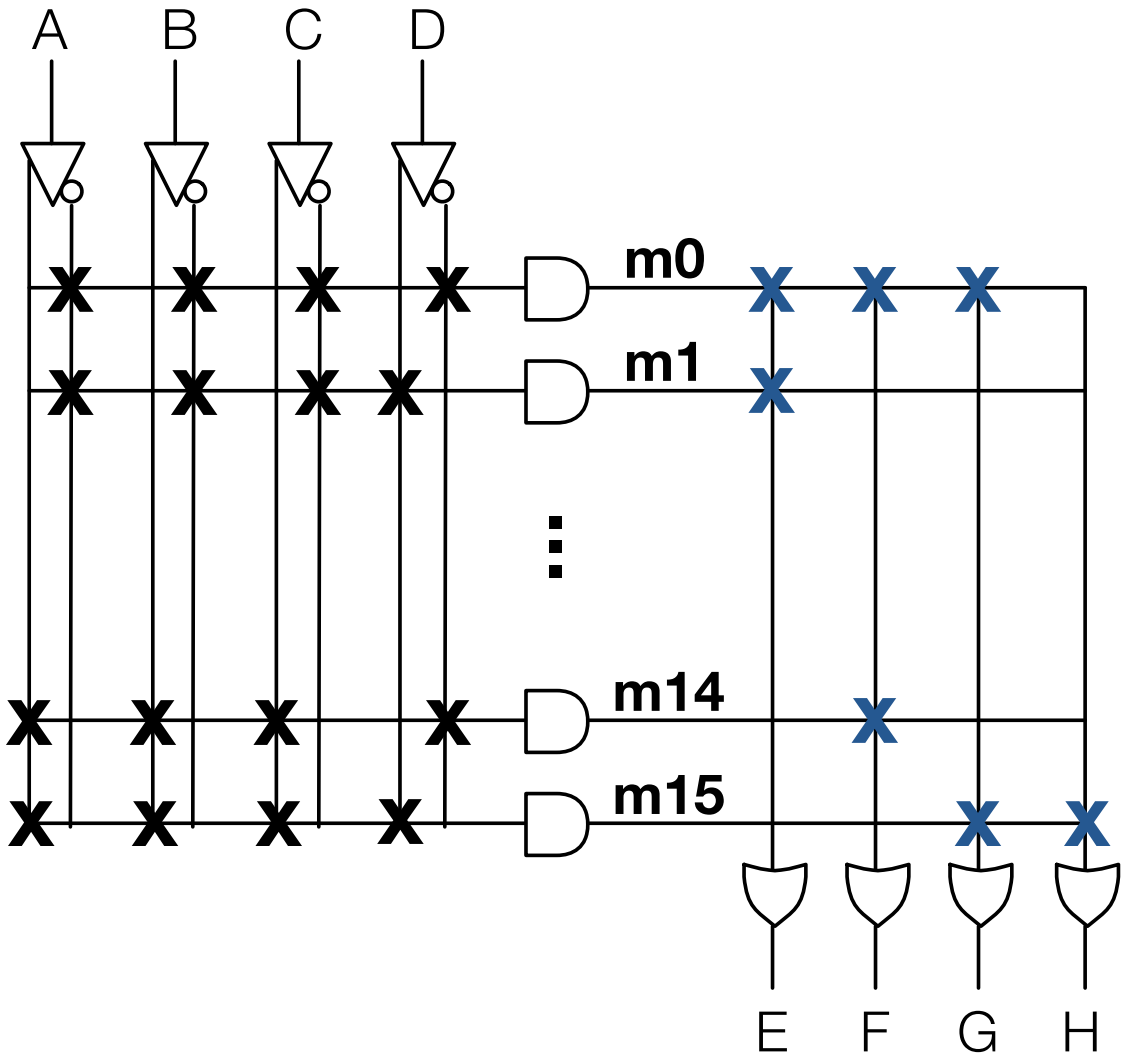


Fixed AND, programmable OR = Programmable ROM (PROM)

Programmable AND, fixed OR = Programmable Array Logic (PAL)

Programmable AND, programmable OR = Programmable Logic Array (PLA)

Programmable ROM (PROM)

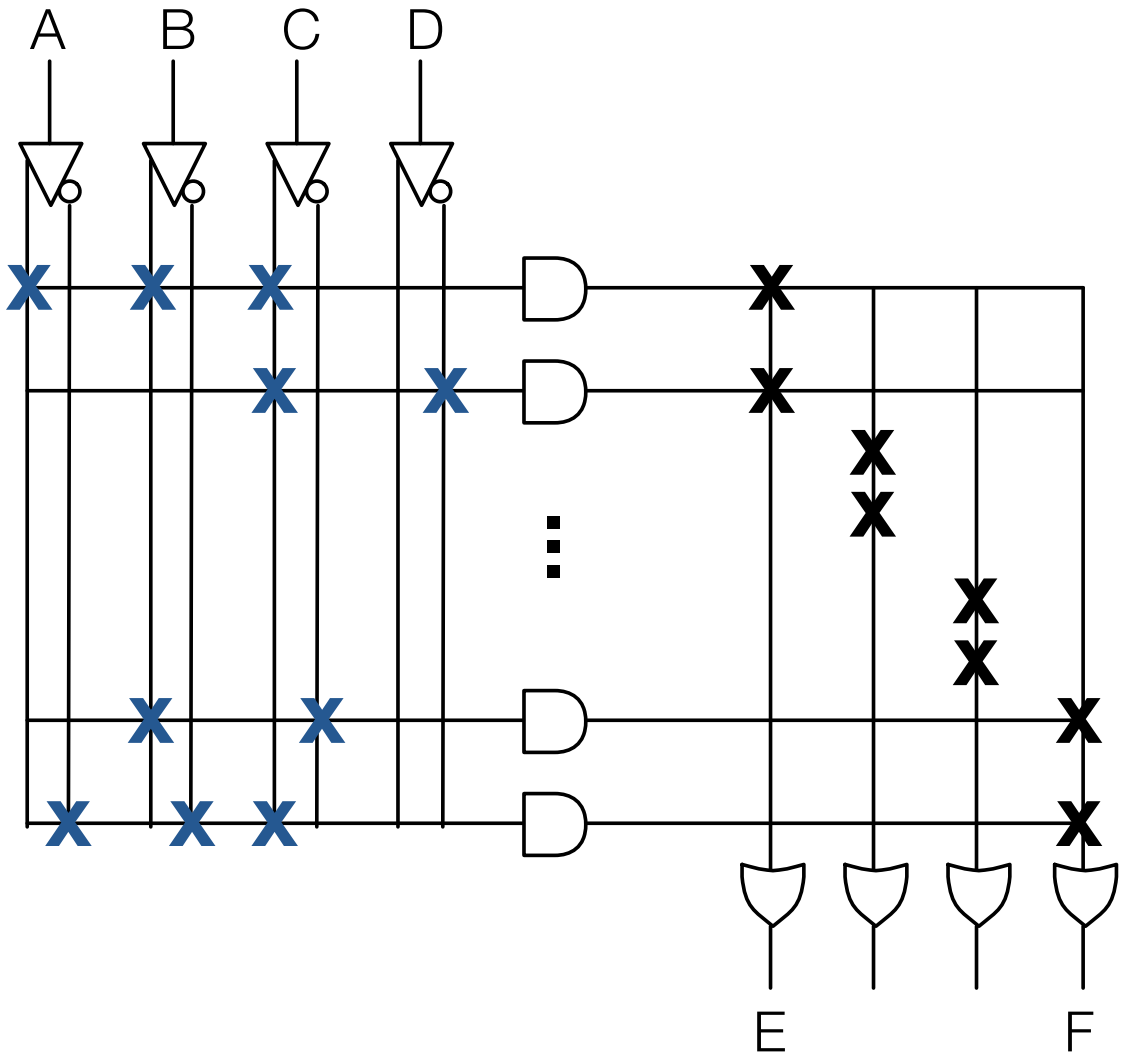


Summing of Minterms!!!

Fixed (**X**) AND, programmable (**X**) OR

e.g., E = m0 + m1 = $\overline{A}\overline{B}\overline{C}\overline{D} + \overline{A}\overline{B}\overline{C}D$

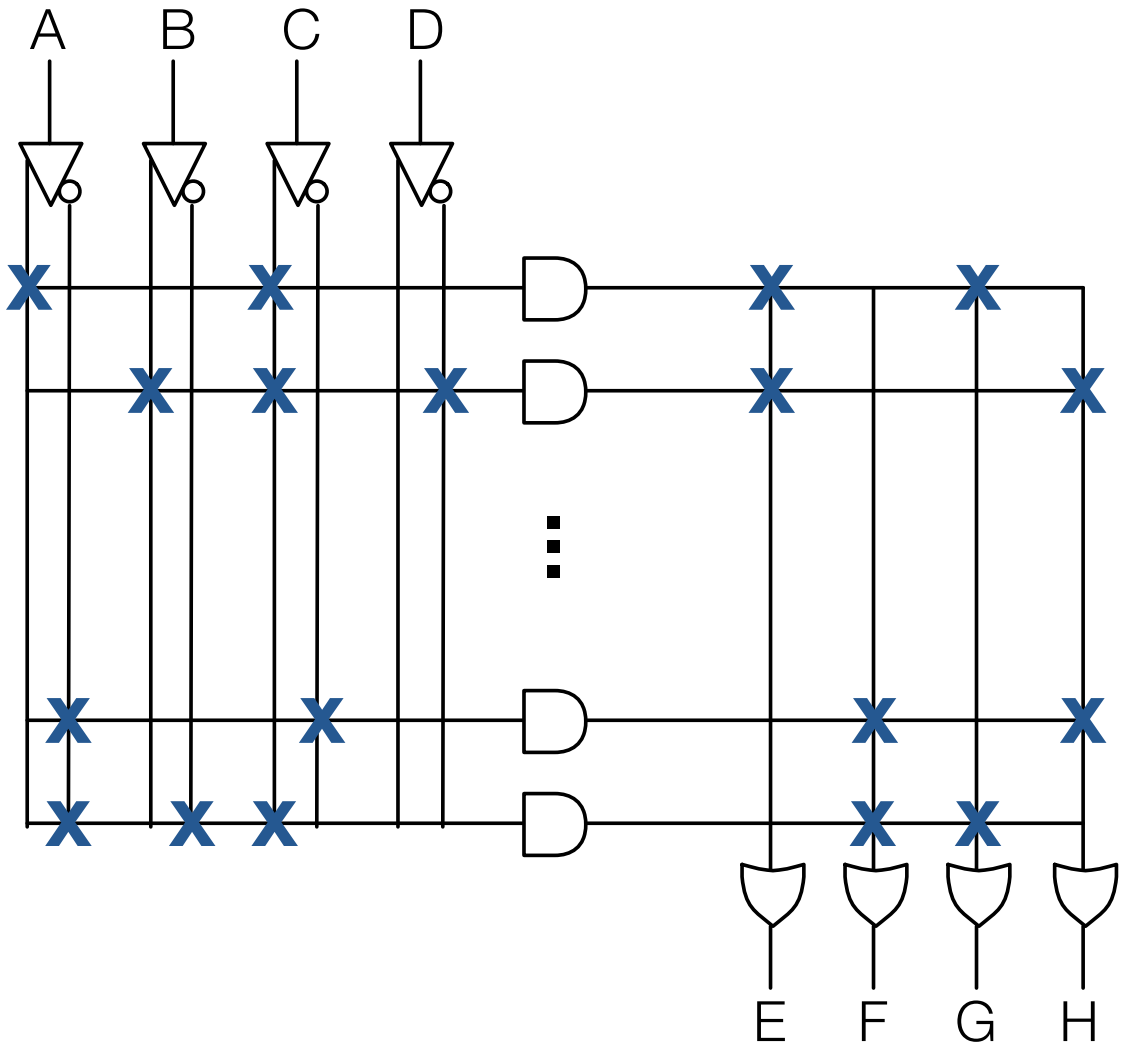
Programmable Array Logic (PAL)



Choose 2 product terms per function

Programmable (**X**) AND, fixed (**X**) OR

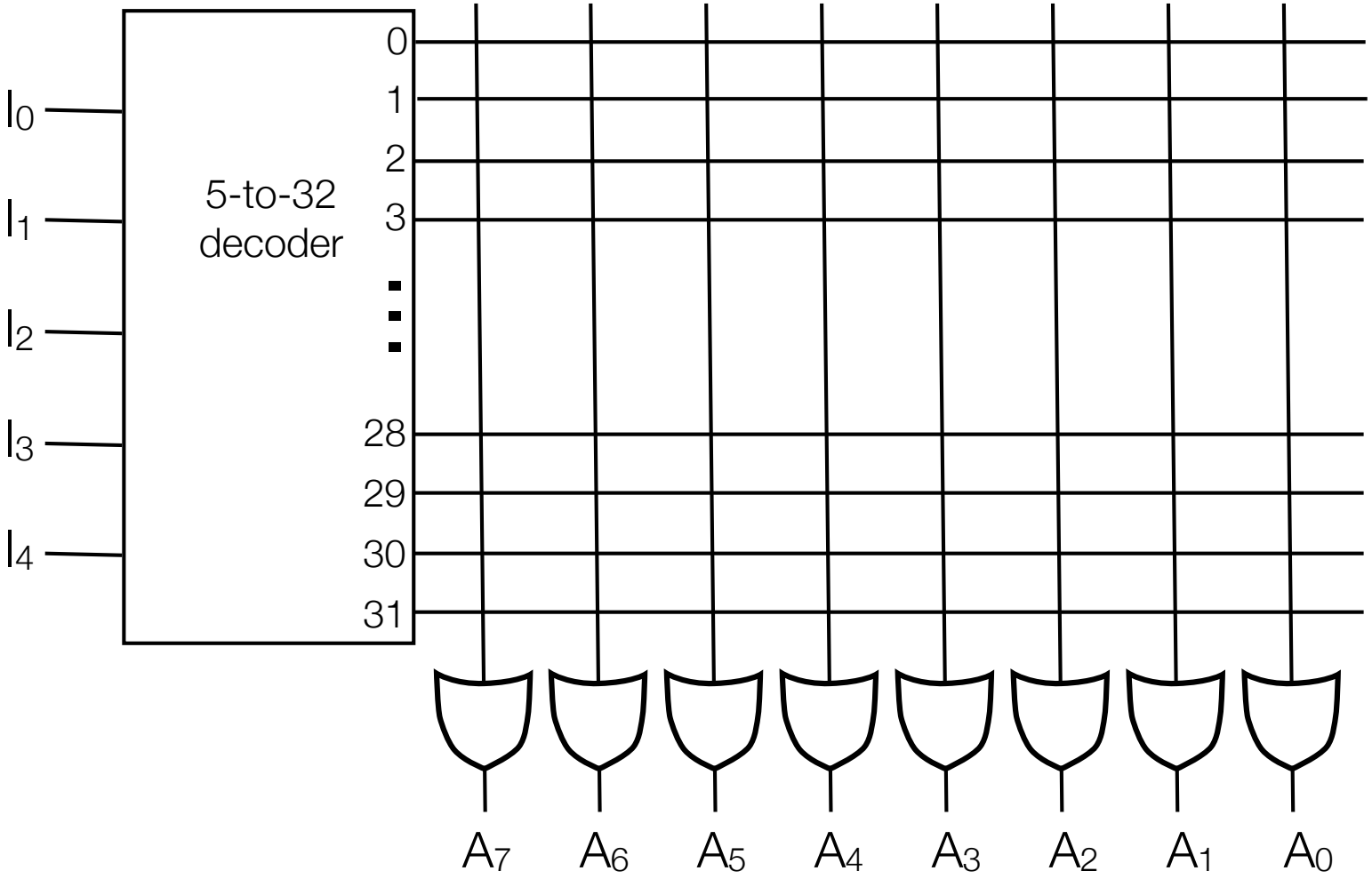
Programmable Logic Array (PLA)



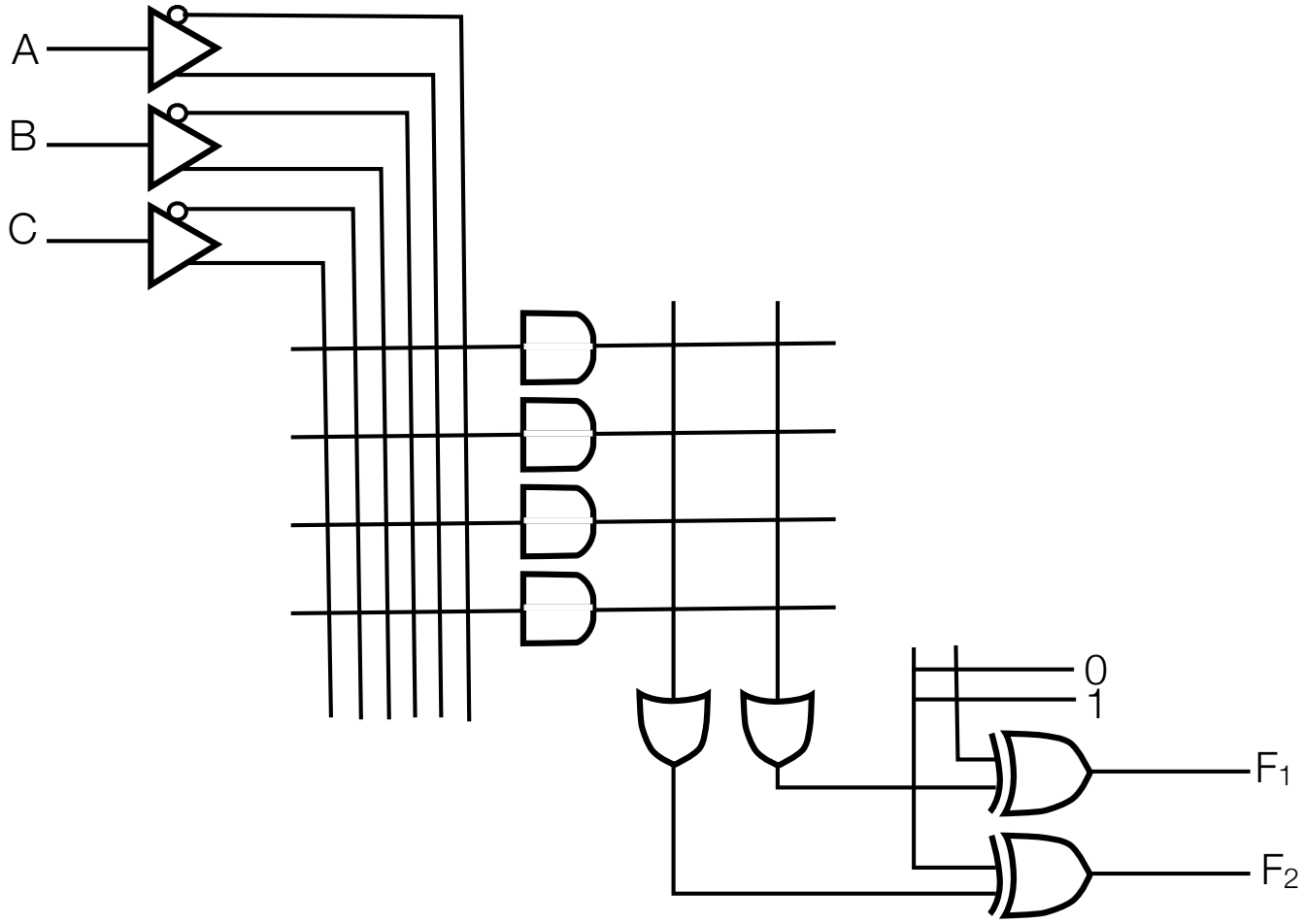
Pick product terms & pick how to combine!

Programmable (**X**) AND, programmable (**X**) OR

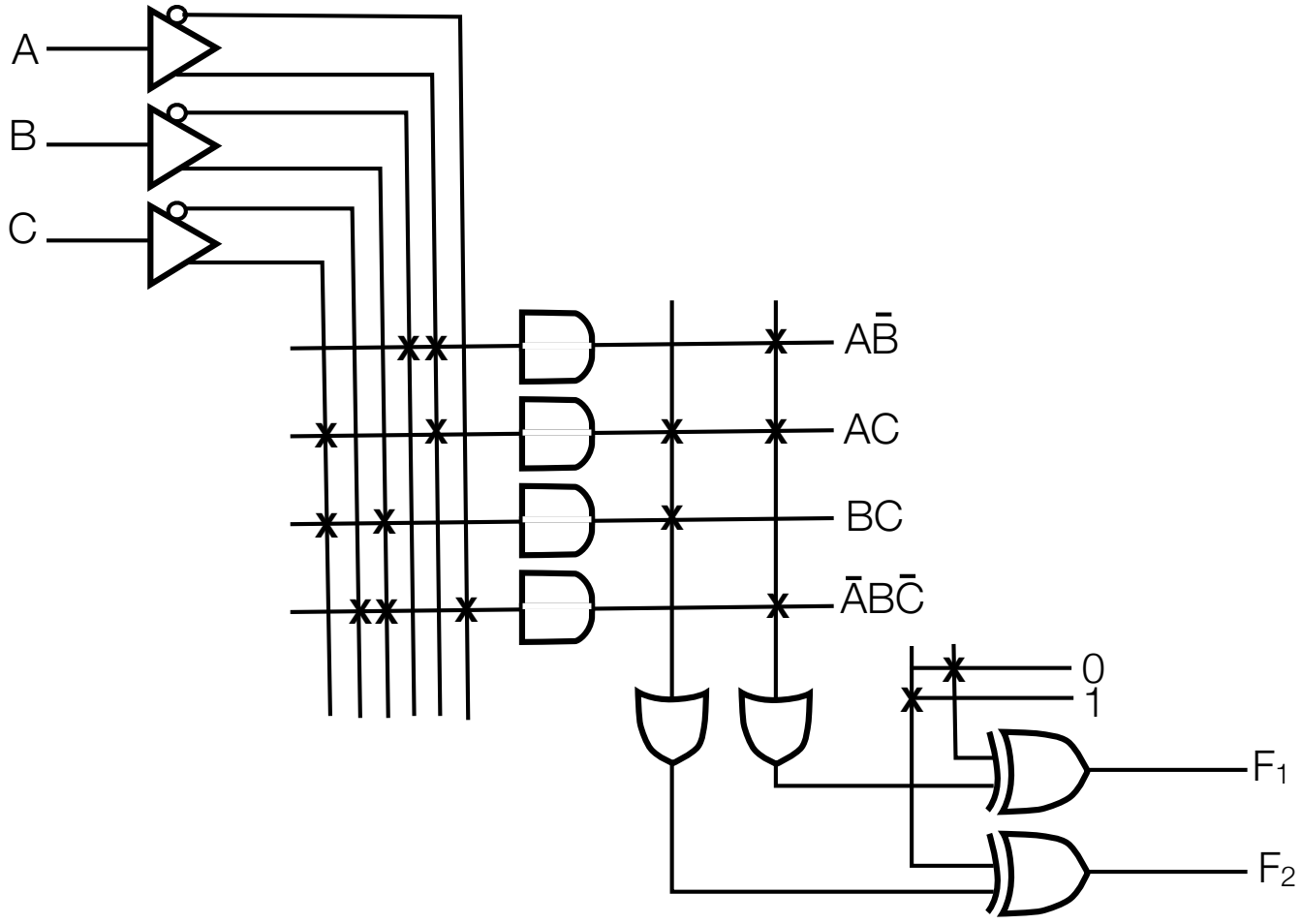
ROM


I0
I1
I2
I3
I4
5-to-32
decoder
0
1
2
3
28
29
30
31
A7
A6
A5
A4
A3
A2
A1
A0

PLA

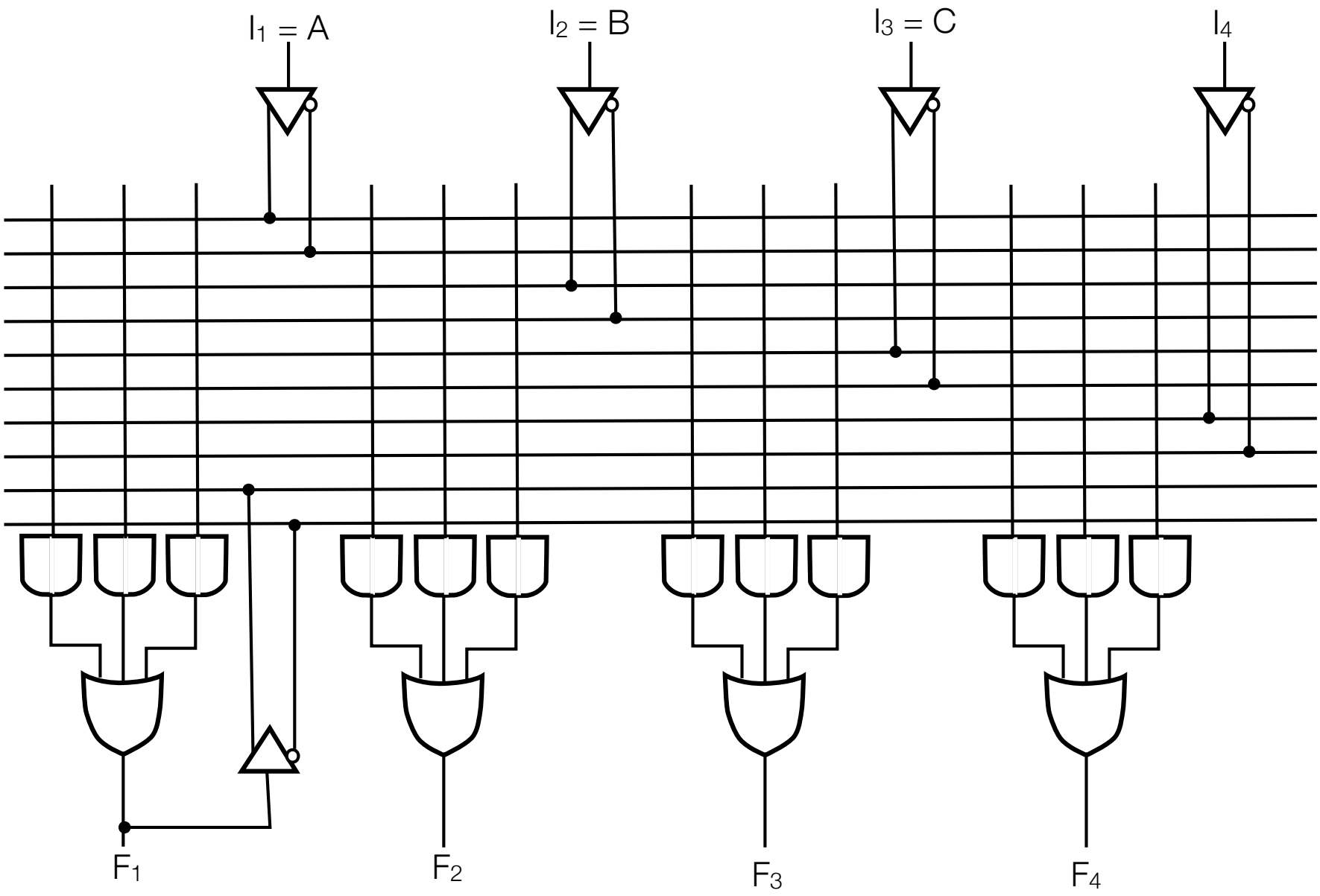


PLA Example

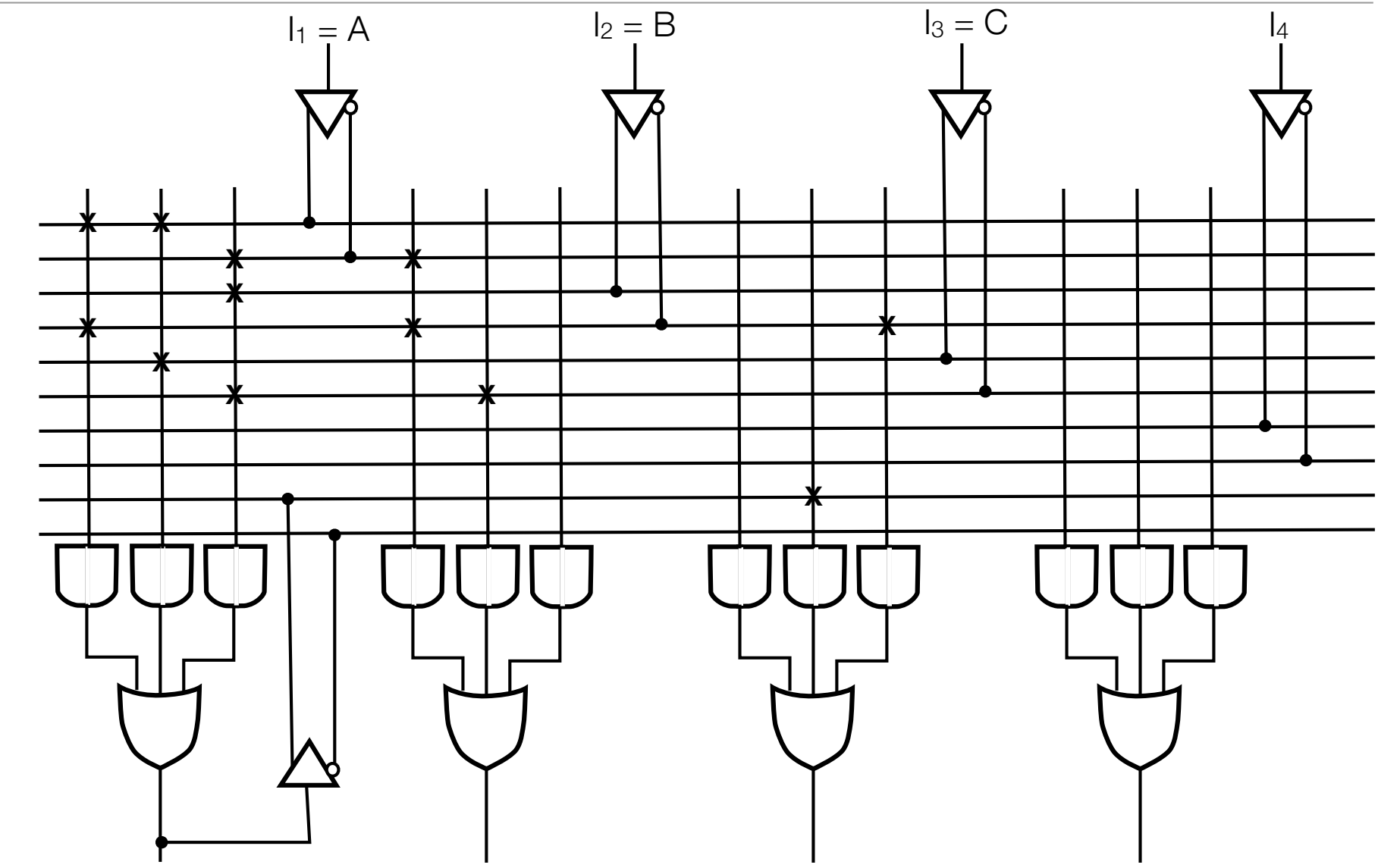


$F_1 = A\bar{B} + AC + \bar{A}B\bar{C}$ $\qquad$ $F_2 = (AC + BC) \oplus 1 = \overline{AC + BC}$

PAL



PAL example



$F_1 = A\bar{B} + AC + \bar{A}B\bar{C}$

$F_2 = \bar{A}\bar{B} + \bar{C}$

$F_3 = F_1 + \bar{B}$