Finite State Machine

Today's lecture

- **Goal: Build a sequential circuit from a state diagram**
  - Step 0: Problem specification
  - Step 1: Build the state diagram
  - Setp 2: Build the state table
  - Step 3: Build the sequential circuit using D flip-flops
- **Timing diagram**
- **Another example: Sequence recognizer**

If a combinational logic circuit is an implementation of a Boolean function, then a sequential logic circuit can be considered an implementation of a finite state machine.

Step 0: Problem Specification

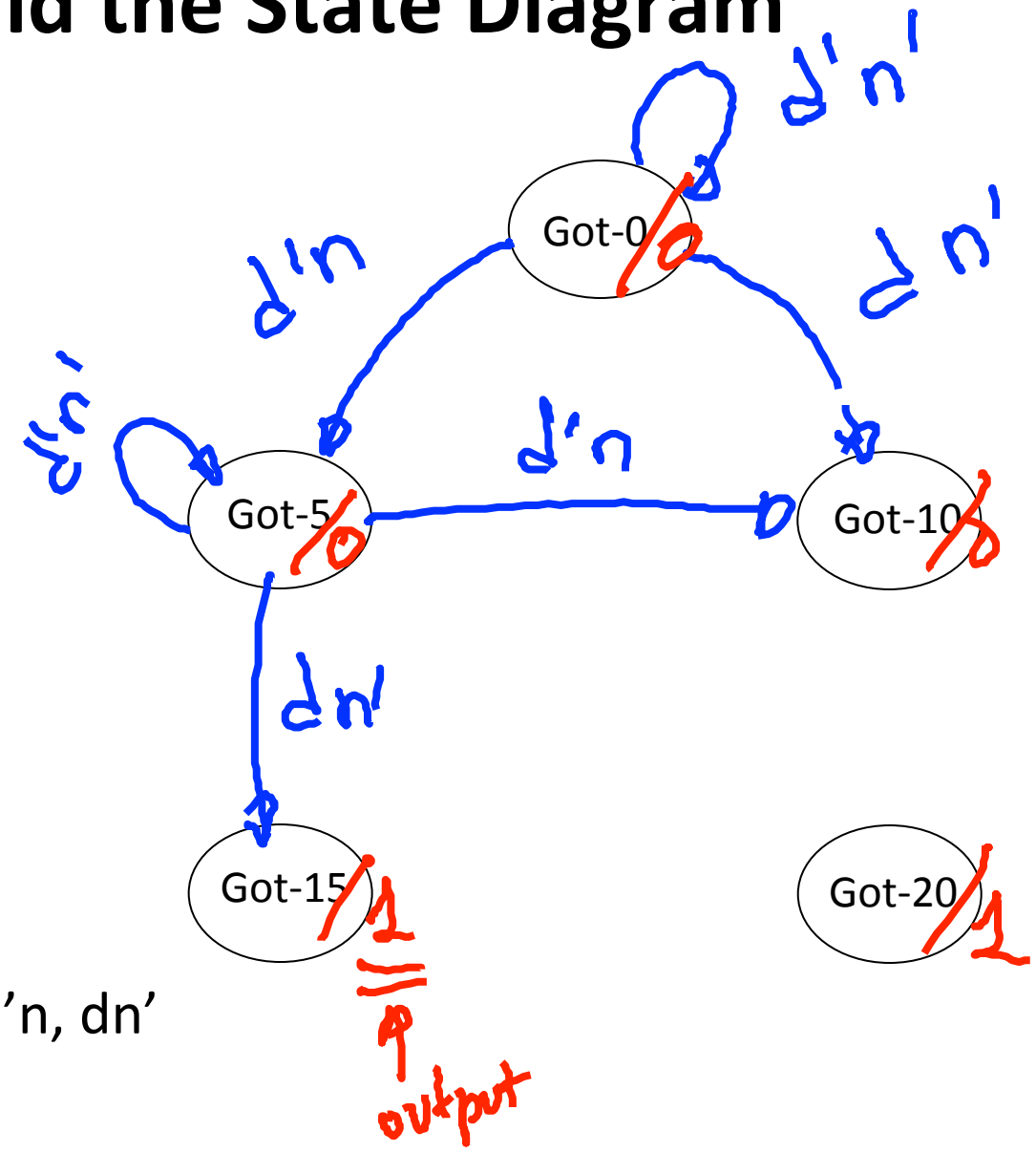
- We have a candy machine that dispenses candies that cost 15-cents
  - Accepts
    - nickels (5-cents)
    - dimes (10-cents)
  - Dispenses a candy if the balance is ≥ 15-cents
  - When the customer overpays
    - the machine does not return change, but
    - keeps the balance for future transactions

Step 1: Build the State Diagram

- **Inputs**
  - nickel : 5 cents
  - dime : 10 cents
- **Ouputs**
  - candy = C { 0 No candy; 1 Candy
- **State identification**
  - A Got 0 : Balance = 0
  - B Got 5 : Balance = 5
  - C Got 10 : Balance = 10
  - D Got 15 : Balance = 15
  - E Got 20 : Balance = 20

d n	
0 0	No coin
0 1	nickel
1 0	dime
1 1	not allow
C	
0	no candy
1	candy

Step 1: Build the State Diagram



Inputs: d'n', d'n, dn'

Step 1: Build the State Diagram

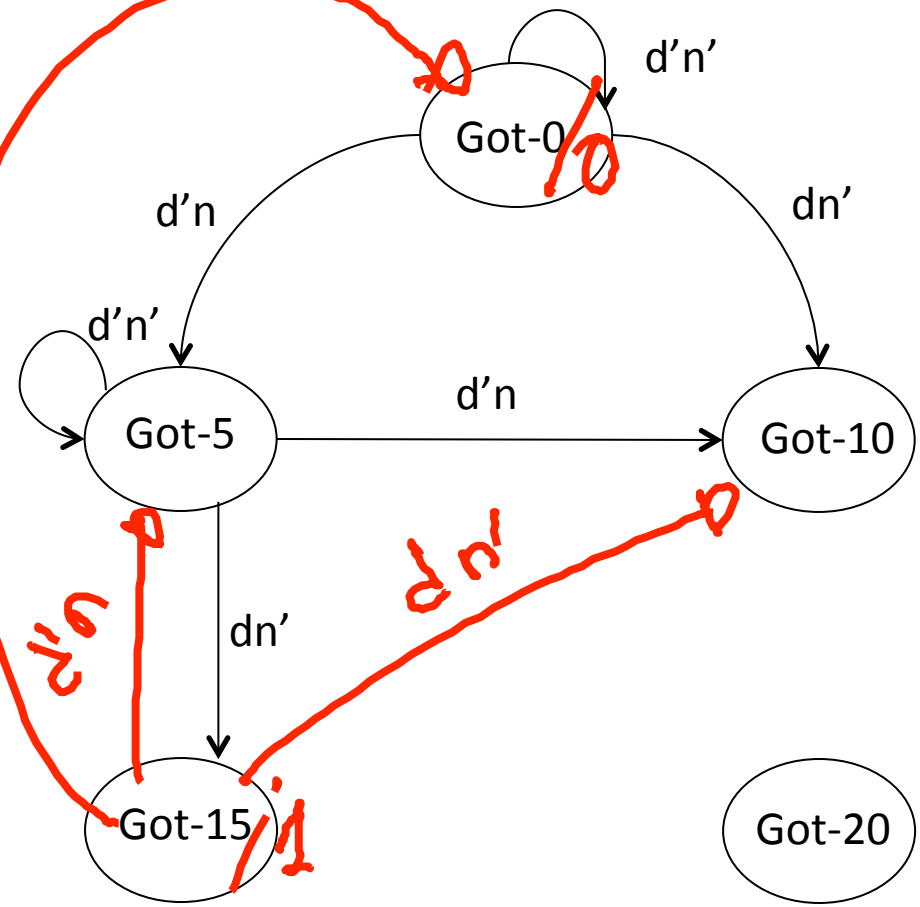
What are the transitions for state Got-15?

A: d'n': Got0; d'n: Got5; dn': Got10

B: d'n': Got15; d'n: Got5; dn': Got10

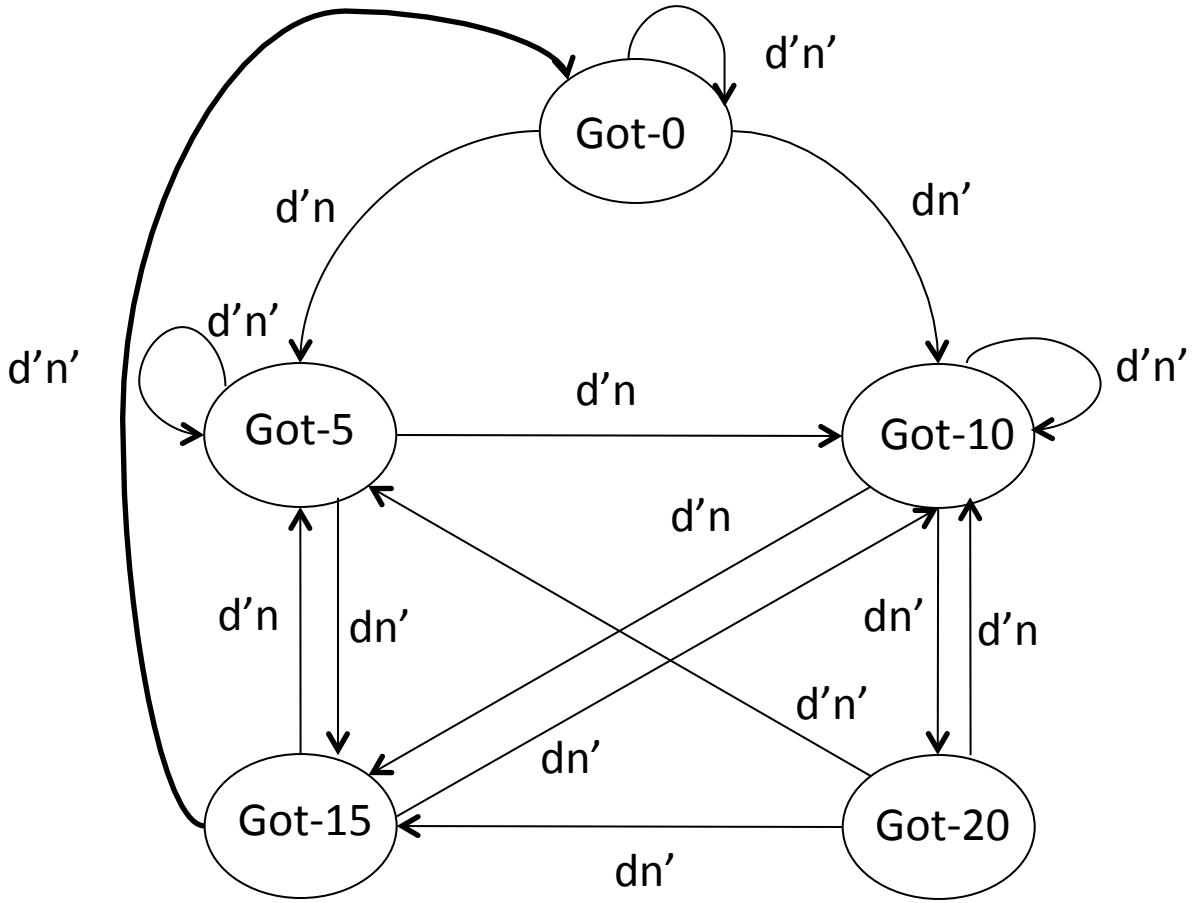
C: d'n': Got0; d'n: Got0; dn': Got0

D: d'n': Got0; d'n: Got20; dn': Got20



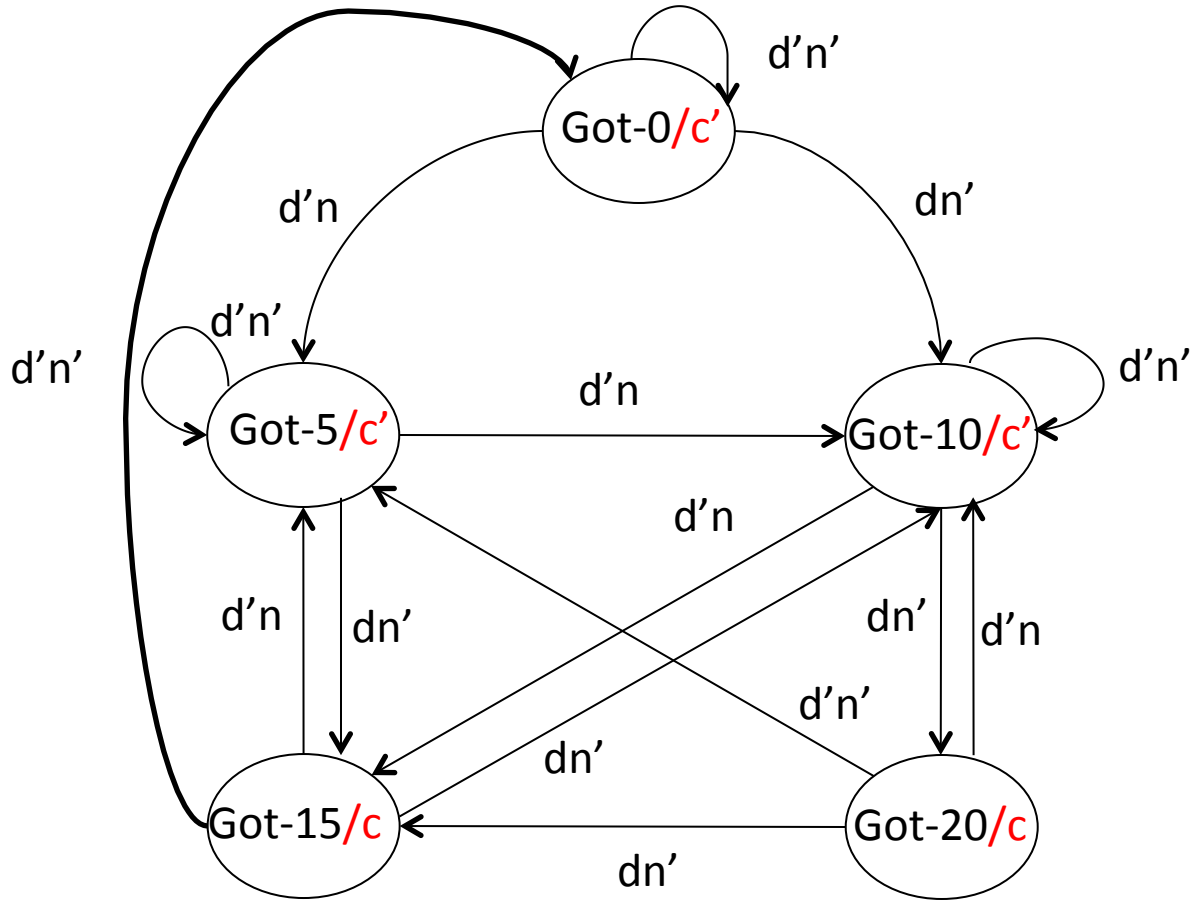
Inputs: d'n', d'n, dn'

Step 1: Build the State Diagram



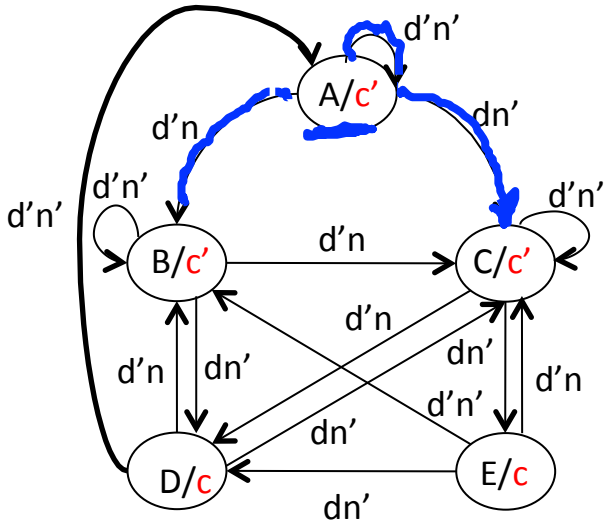
Inputs: d'n', d'n, dn'

Step 1: Build the State Diagram



Output: c or c'

Step 2: Build a State Table

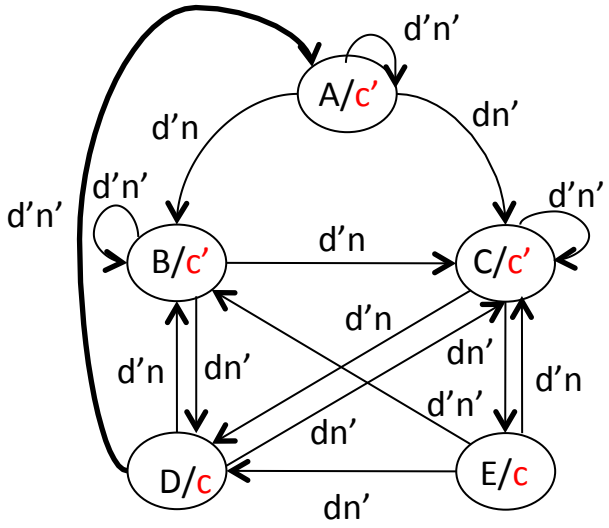


First 3 entries of the table:

A: A,D,E
B: B,C,D
C: A, B,C
D: A,B,E

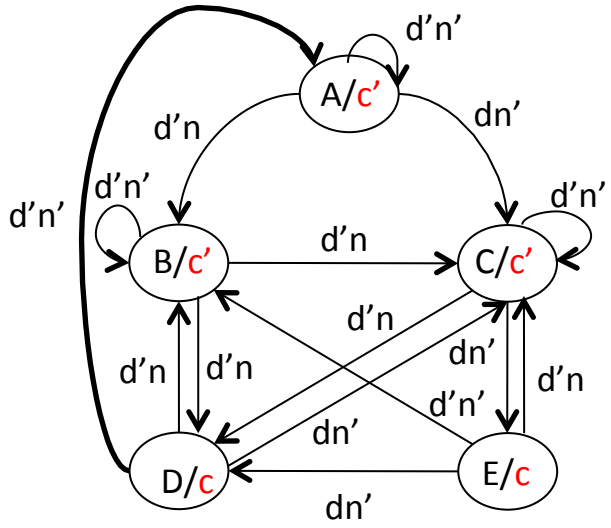
Current State	Input	Next State
A	d'n'	A
A	d'n	B
A	dn'	C
B	d'n'	
B	d'n	
B	dn'	
C	d'n'	
C	d'n	
C	dn'	
D	d'n'	
D	d'n	
D	dn'	
E	d'n'	
E	d'n	
E	dn'	

Step 2: Build a State Table



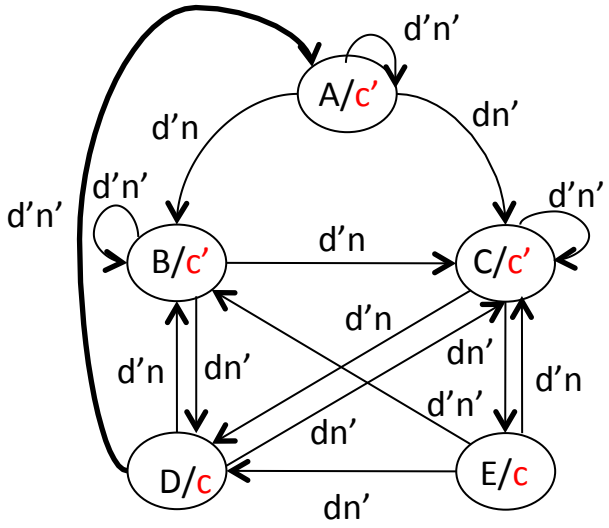
Current State	Input	Next State
A	d'n'	A
A	d'n	B
A	dn'	C
B	d'n'	B
B	d'n	C
B	dn'	D
C	d'n'	C
C	d'n	D
C	dn'	E
D	d'n'	A
D	d'n	B
D	dn'	C
E	d'n'	B
E	d'n	C
E	dn'	D

Step 2: Build a State Table



Output	Current State	Input	Next State
c'	A	d'n'	A
	A	d'n	B
	A	dn'	C
c'	B	d'n'	B
	B	d'n	C
	B	dn'	D
c'	C	d'n'	C
	C	d'n	D
	C	dn'	E
c	D	d'n'	A
	D	d'n	B
	D	dn'	C
c	E	d'n'	B
	E	d'n	C
	E	dn'	D

Step 2: Build a State Table



Why do we need a sequential logic to build this circuit?

Output	Current State	Input	Next State
	A	d'n'	A
	A	d'n	B
	A	dn'	C
	B	d'n'	B
	B	d'n	C
	B	dn'	D
	C	d'n'	C
	C	d'n	D
	C	dn'	E
	D	d'n'	A
	D	d'n	B
	D	dn'	C
	E	d'n'	B
	E	d'n	C
	E	dn'	D

Step 2: Build Sequential Circuit

Current State	Input	Next State
A	d'n'	A
A	d'n	B
A	dn'	C
B	d'n'	B
B	d'n	C
B	dn'	D
C	d'n'	C
C	d'n	D
C	dn'	E
D	d'n'	A
D	d'n	B
D	dn'	C
E	d'n'	B
E	d'n	C
E	dn'	D

State Encoding:

5 states: How many bits?

Option 1: 3 bits

Option 2: 1 bit per state ✓

A: 10000
B: 01000
C: 00100

One hot encoding

Step 2: Build Sequential Circuit

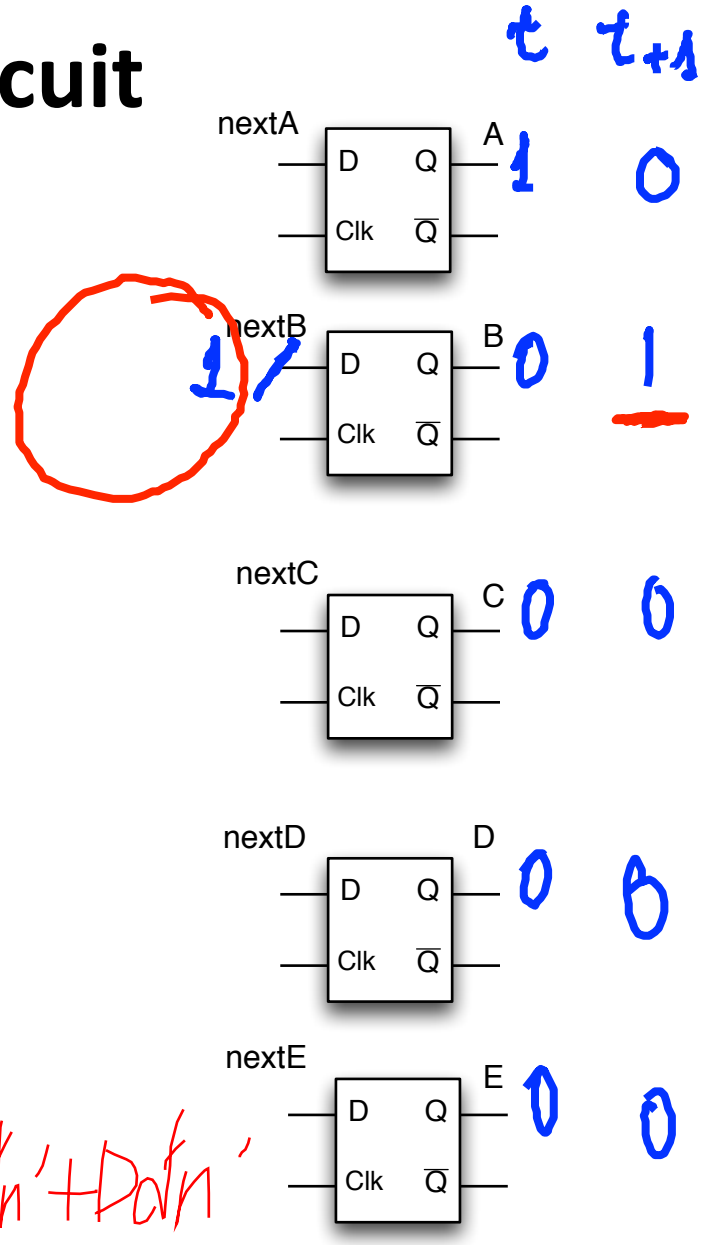
Current State	Input	Next State
A	d'n'	A
A	d'n	B
A	dn'	C
B	d'n'	B
B	d'n	C
B	dn'	D
C	d'n'	C
C	d'n	D
C	dn'	E
D	d'n'	A
D	d'n	B
D	dn'	C
E	d'n'	B
E	d'n	C
E	dn'	D

One hot encoding

State A = 10000
State B = 01000
State C = 00100
State D = 00010
State E = 00001

D flip-flop

Current State	Next State
0	0
1	1



nextA = Aa'n' + Da'n'

Step 2: Build Sequential Circuit

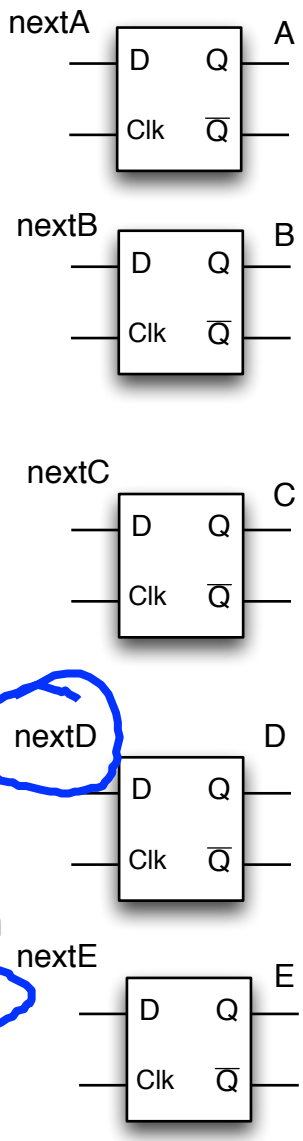
Current State	Input	Next State
A	d'n'	A
A	d'n	B
A	dn'	C
B	d'n'	B
B	d'n	C
B	dn'	D
C	d'n'	C
C	d'n	D
C	dn'	E
D	d'n'	A
D	d'n	B
D	dn'	C
E	d'n'	B
E	d'n	C
E	dn'	D

One hot encoding

State A = 10000
State B = 01000
State C = 00100
State D = 00010
State E = 00001

next D?

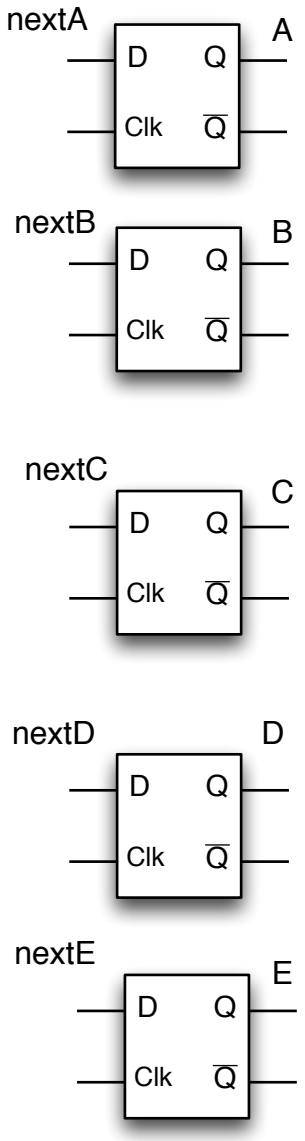
A: nextD = Ad'n'+ Dd'n'
B: nextD= Ad'n + Bd'n' + Dd'n
C: nextD = Bdn' + Cd'n + Edn'
D: nextD =1



nextD = Ed'n + Bdn' + Cd'n

Step 2: Build Sequential Circuit

Current State	Input	Next State
A	d'n'	A
A	d'n	B
A	dn'	C
B	d'n'	B
B	d'n	C
B	dn'	D
C	d'n'	C
C	d'n	D
C	dn'	E
D	d'n'	A
D	d'n	B
D	dn'	C
E	d'n'	B
E	d'n	C
E	dn'	D



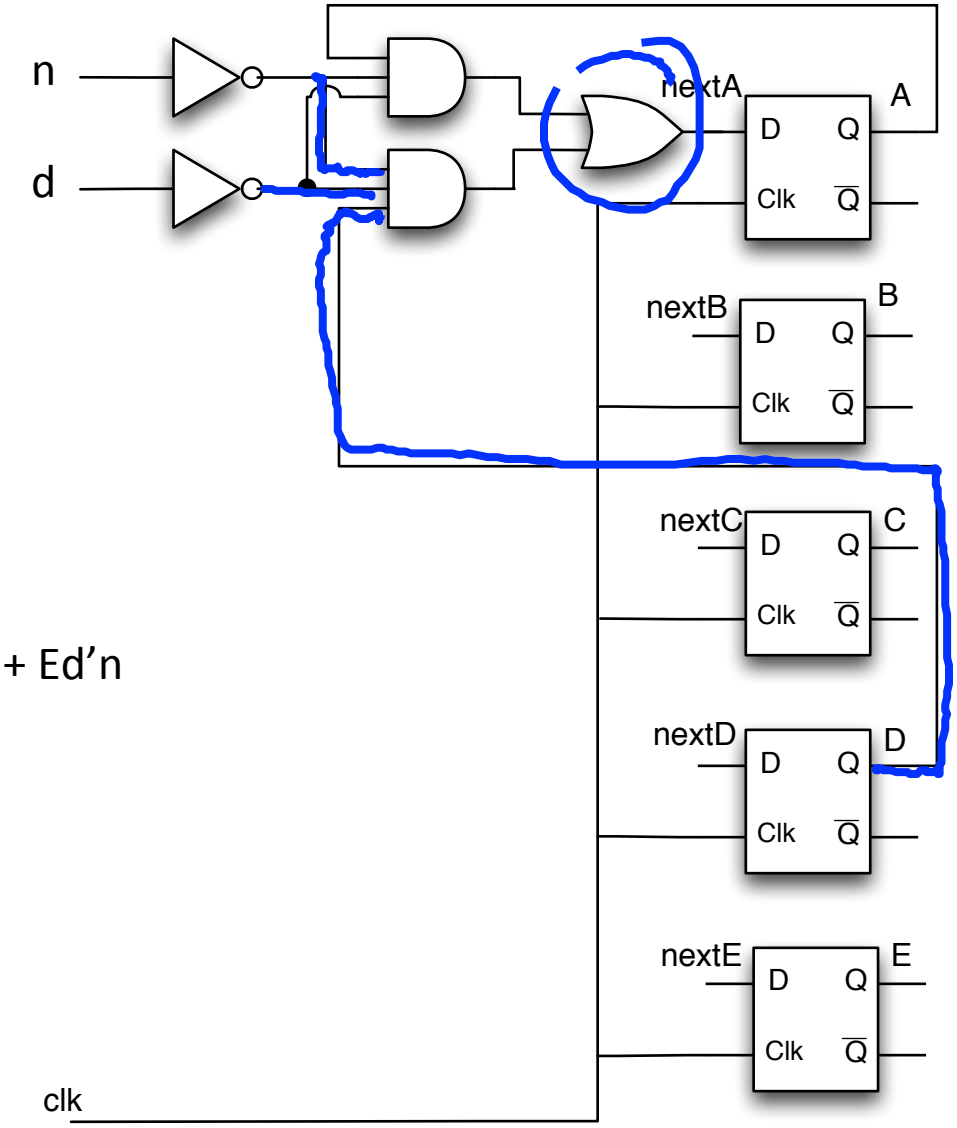
Step 2: Build a State Table

Current State	Input	Next State
A	d'n'	A
A	d'n	B
A	dn'	C
B	d'n'	B
B	d'n	C
B	dn'	D
C	d'n'	C
C	d'n	D
C	dn'	E
D	d'n'	A
D	d'n	B
D	dn'	C
E	d'n'	B
E	d'n	C
E	dn'	D

nextA = Ad'n'+ Dd'n'
nextB = Ad'n + Bd'n' + Dd'n + Edn'
nextC = And' + Bd'n + Cd'n' + Ddn' + Ed'n
nextD = Bdn' + Cd'n + Edn'
nextE = Cdn'

Output: Candy = D + E

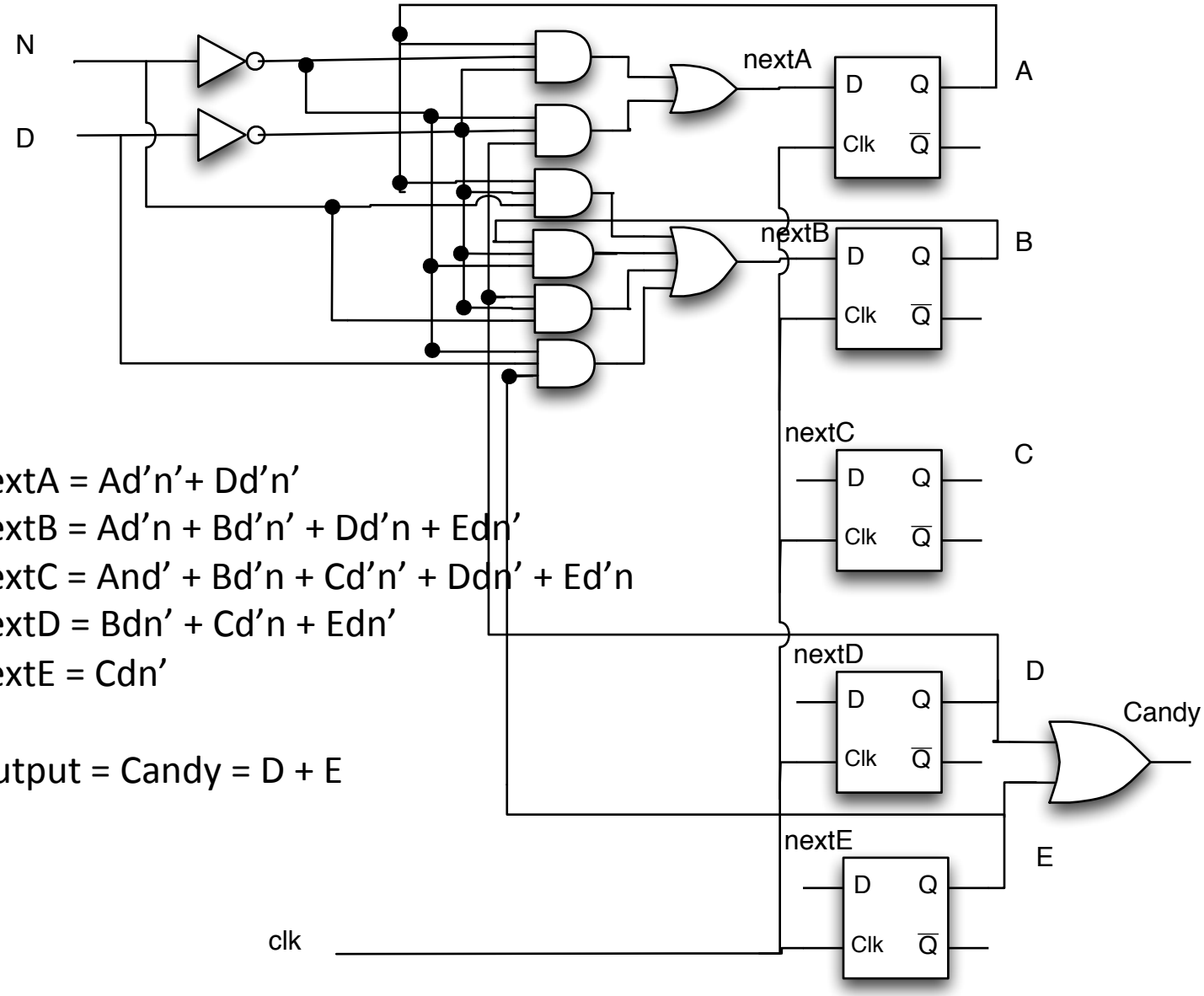
Step 3: Sequential circuit with D flip-flops



nextA = Ad'n'+ Dd'n'
nextB = Ad'n + Bd'n' + Dd'n + Edn'
nextC = And' + Bd'n + Cd'n' + Ddn' + Ed'n
nextD = Bdn' + Cd'n + Edn'
nextE = Cdn'

Output: c = D + E

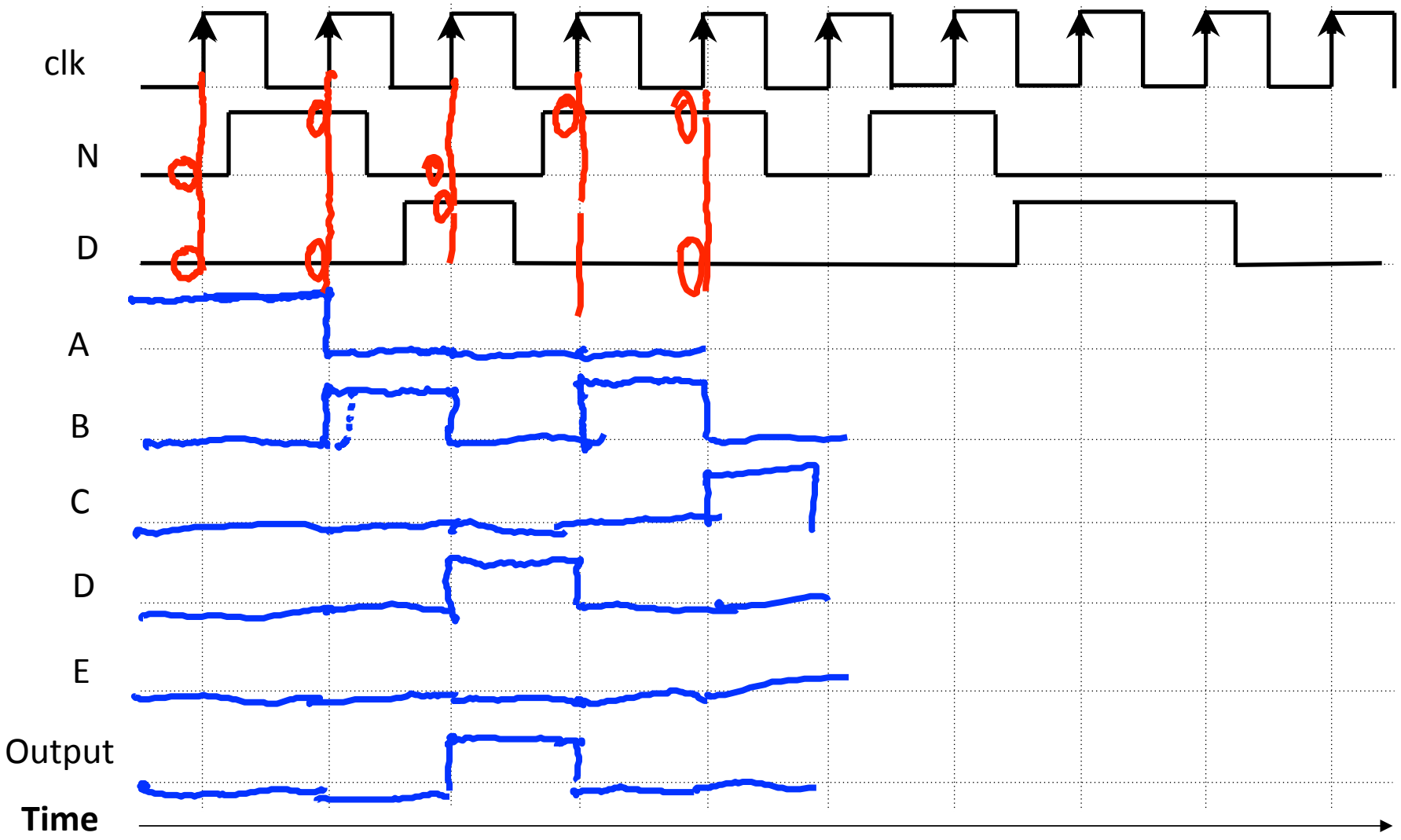
Step 3: Sequential circuit with D flip-flops



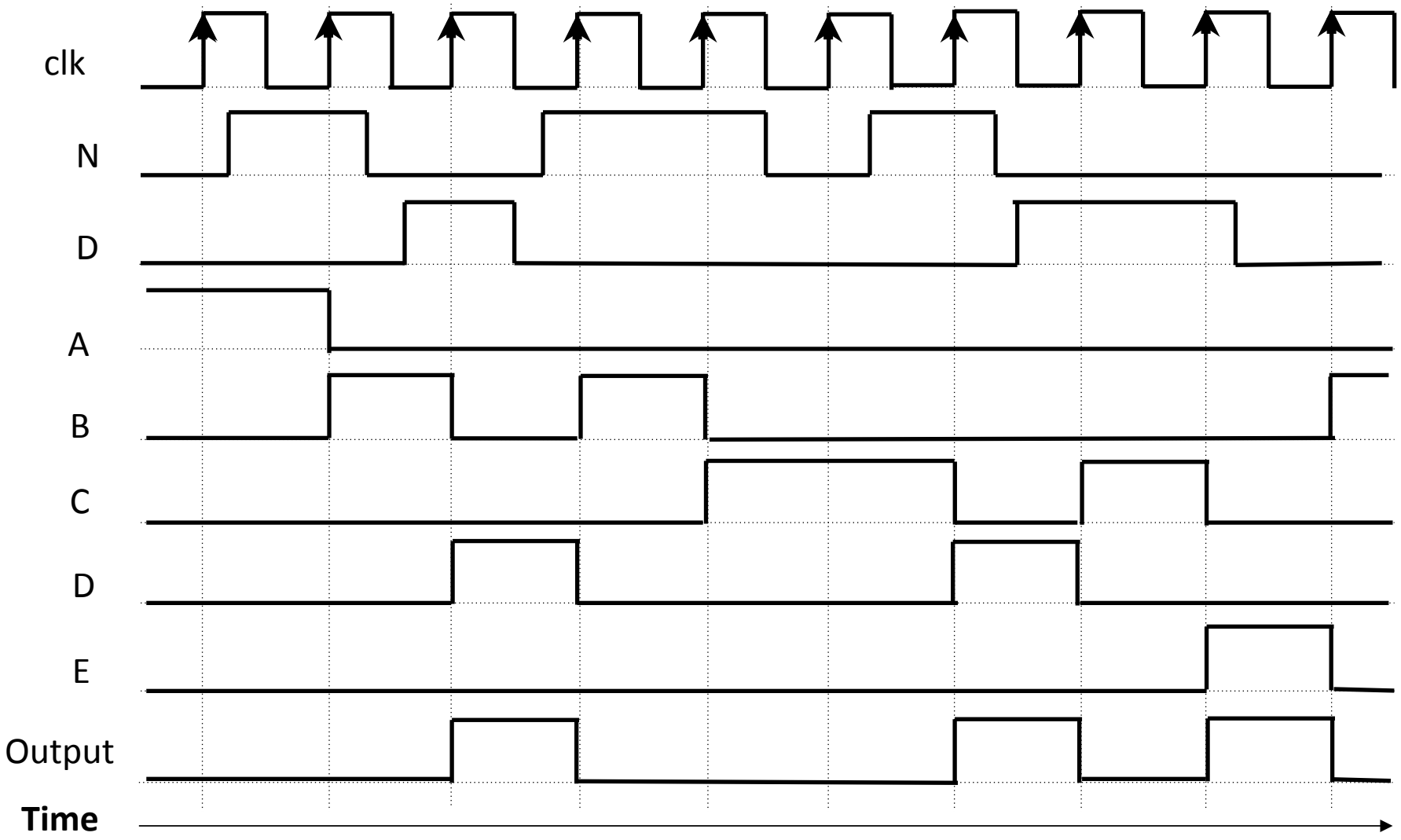
nextA = Ad'n'+ Dd'n'
nextB = Ad'n + Bd'n' + Dd'n + Edn'
nextC = And' + Bd'n + Cd'n' + Ddn' + Ed'n
nextD = Bdn' + Cd'n + Edn'
nextE = Cdn'

Output = Candy = D + E

Timing Diagram



Timing Diagram



Another example: Sequence recognizer

- A sequence recognizer is a special kind of sequential circuit that looks for a special bit pattern in some input.
- The recognizer circuit has one input, X.
- There is one output, Z, which is 1 when the desired pattern is found.

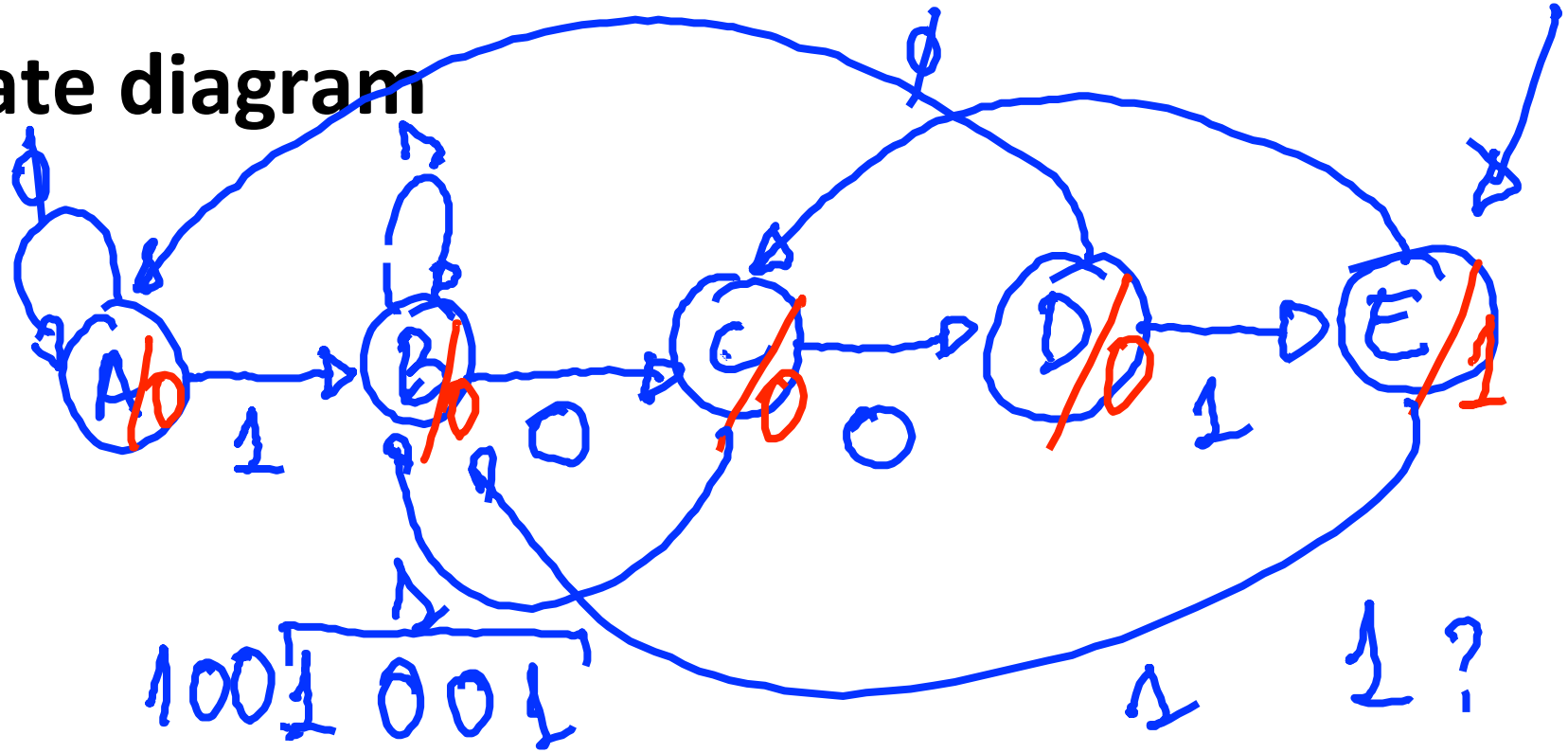
- Our example will detect the bit pattern "1001":

Inputs:	1 1 1 0 0 1 1 0 1 0 0 1 0 0 1 1 0 …
Outputs:	0 0 0 0 0 0 1 0 0 0 0 0 1 0 0 1 0 …

Here, one input and one output bit appear every clock cycle.

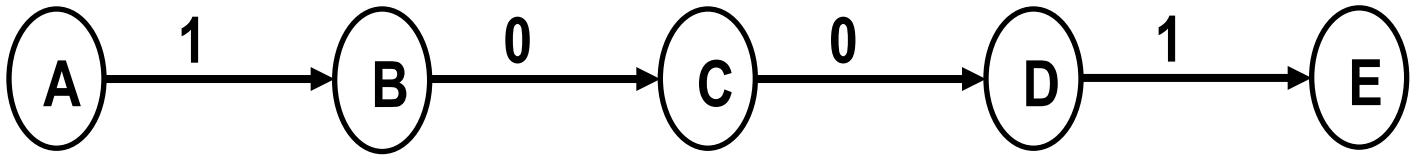
- This requires a sequential circuit because the circuit has to "remember" the inputs from previous clock cycles, in order to determine whether or not a match was found.

State diagram



State	Meaning
A	None of the desired pattern (1001) has been input yet.
B	We've already seen the first bit (1) of the desired pattern.
C	We've already seen the first two bits (10) of the desired pattern.
D	We've already seen the first three bits (100) of the desired pattern.
E	We've seen the pattern (1001)

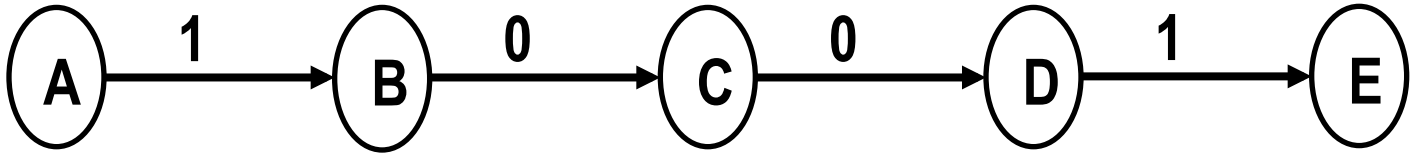
State diagram



State	Meaning
A	None of the desired pattern (1001) has been input yet.
B	We've already seen the first bit (1) of the desired pattern.
C	We've already seen the first two bits (10) of the desired pattern.
D	We've already seen the first three bits (100) of the desired pattern.
E	We've seen the pattern (1001)

State diagram

- We need *two* outgoing arrows for each node, to account for the possibilities of X=0 and X=1.



Inputs: 1 1 1 0 0 1 1 0 1 0 0 1 0 0 1 1 0 …

Outputs: 0 0 0 0 0 0 1 0 0 0 0 0 1 0 0 1 0 …

Transitions for E:

A: x=0-> A; x=1 -> B

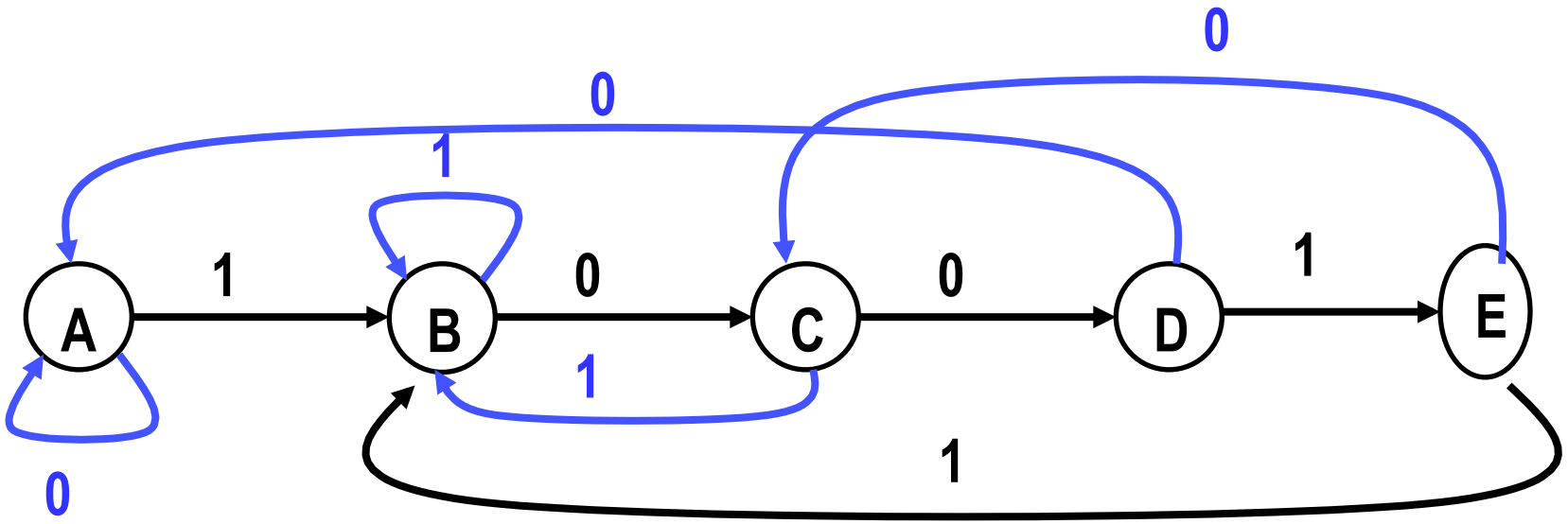
B: x=0-> C; x=1 -> B

C: x=0-> B; x=1 -> A

D: x=0-> D; x=1 -> B

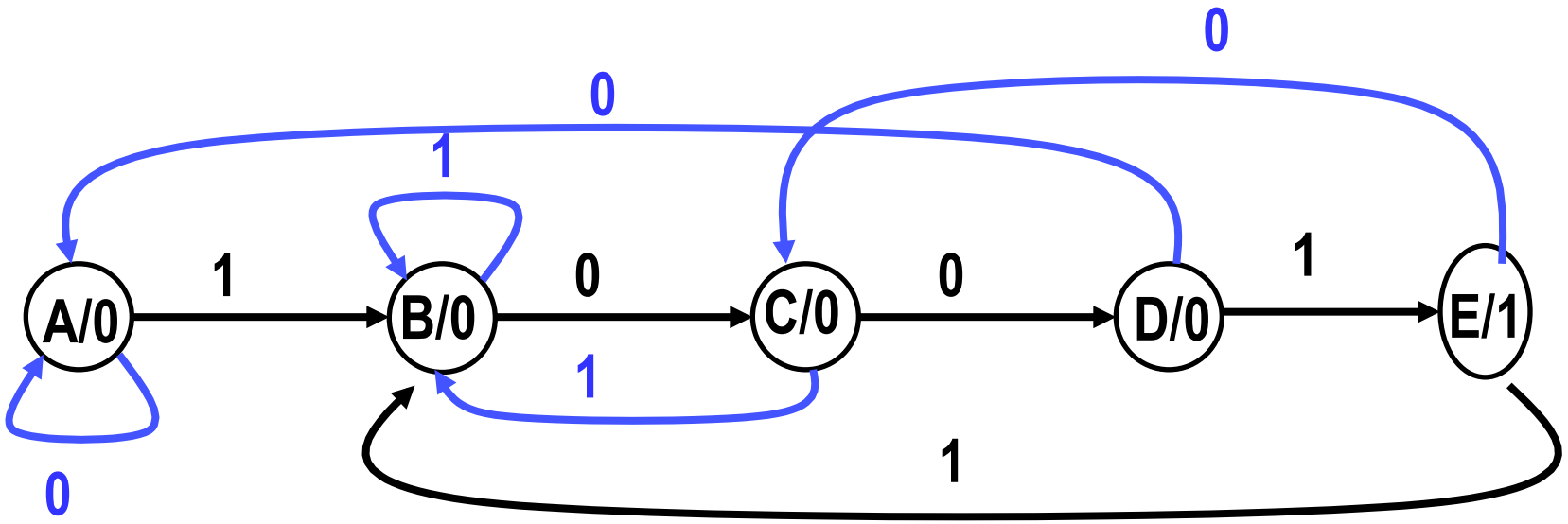
State diagram

- We need *two* outgoing arrows for each node, to account for the possibilities of X=0 and X=1.

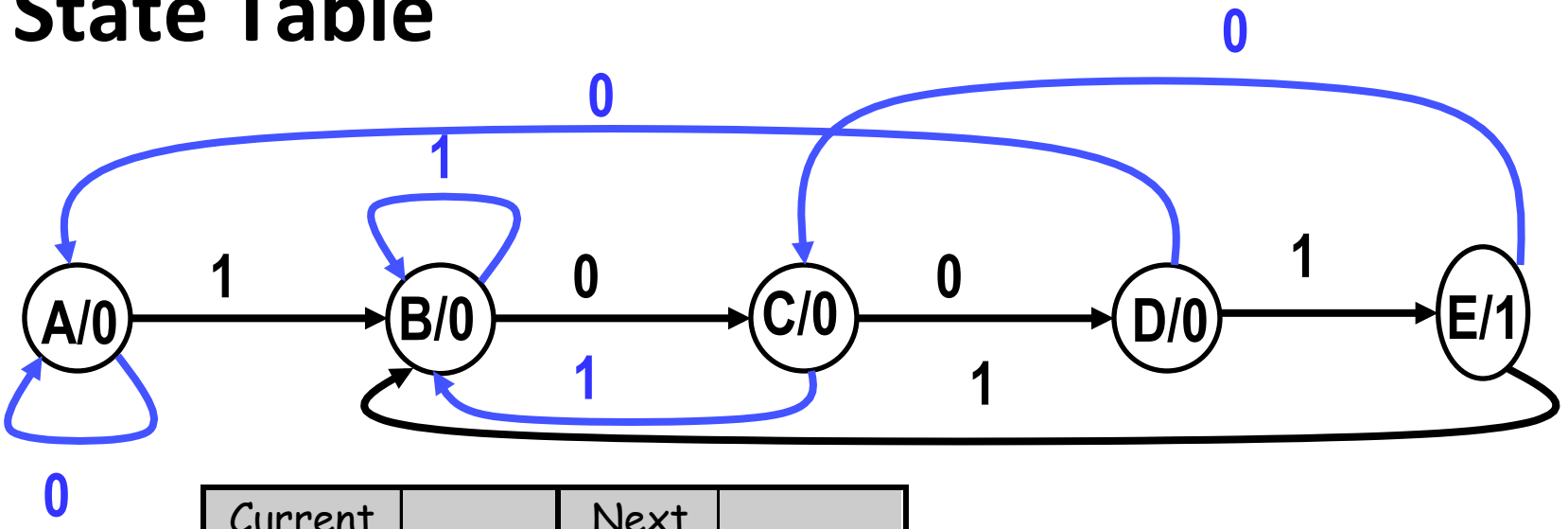


State diagram

- Need to determine the output

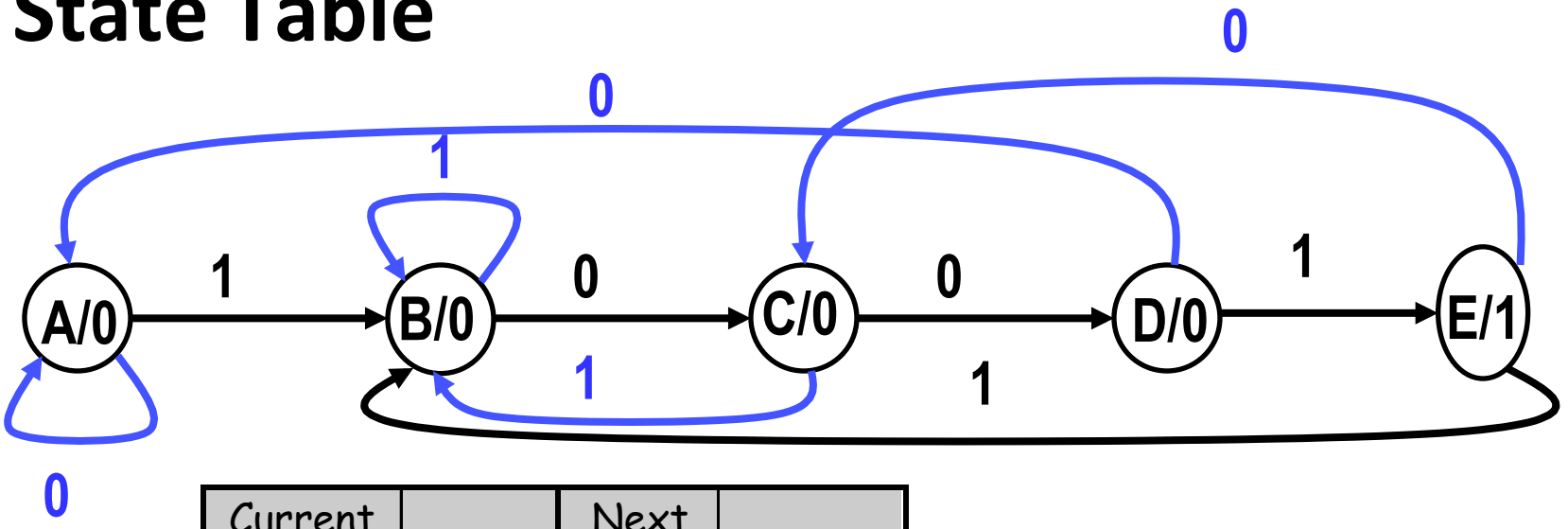


State Table



Current State	Input	Next State	Output
A	0	A	0
A	1	B	0
B	0	C	0
B	1	B	0
C	0	D	0
C	1	B	0
D	0	A	0
D	1	E	1
E	0	C	0
E	1	B	0

State Table



Current State	Input	Next State	Output
A	0	A	0
A	1	B	0
B	0	C	0
B	1	B	0
C	0	D	0
C	1	B	0
D	0	A	0
D	1	E	1
E	0	C	0
E	1	B	0

Anext = Ax' + Dx'
Bnext = Ax + Bx + Cx + Ex
Cnext = Bx' + Ex'
Dnext = Cx'
Enext = Dx

Output = E