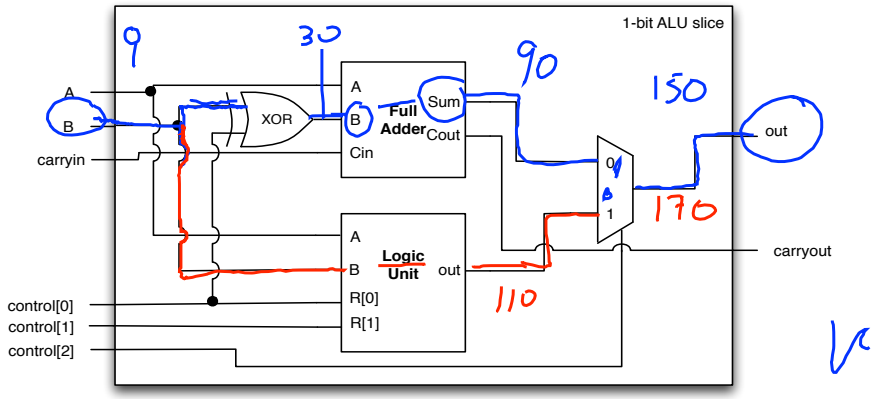
Computing components from ALU1



What is the worst case propagation delay from B to out?
(consider both arithmetic and logic operations)

A: 120ps
B: 150ps
C: 160ps
D: 170ps
E: 190ps

XOR gate

In	Out	Delay
A,B	out	30ps

Full Adder

In	Out	Delay
A,B	Sum	60ps
Cin	Sum	30ps
A,B	Cout	90ps
Cin	Cout	60ps

Logic Unit

In	Out	Delay
A,B	out	110ps
R	out	10ps

2-to-1 Multiplexor

In	Out	Delay
A,B	out	60ps
R	out	80ps

Sequential Logic & Finite State Machines

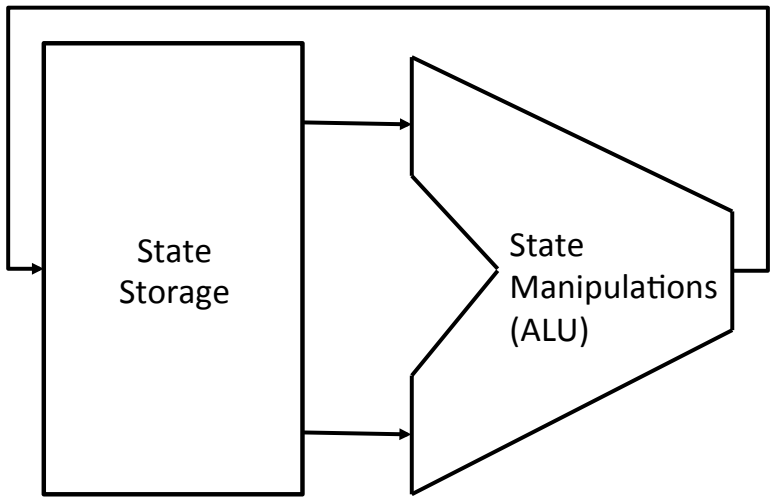
Today's lecture

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

Sequential Logic

- A circuit whose **output depends not only on the present value** of its input signals **but on the sequence of past inputs**
  - The input history.

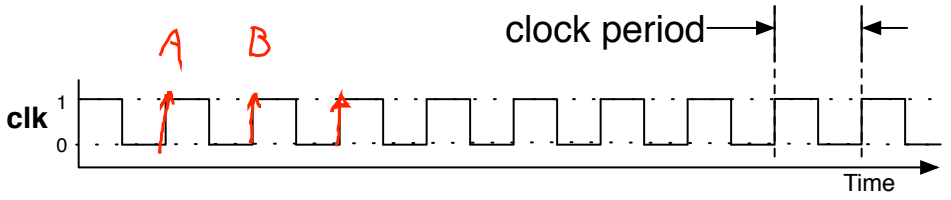
- That is, **sequential logic** has state (memory) while **combinational logic** does not.



Synchronous Design

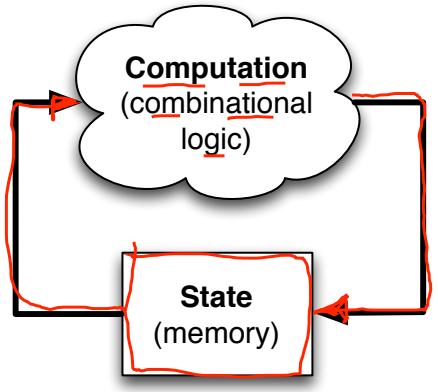
- *The easiest (and most common) way to build computers*
- All state elements get updated at the same time
  - Using a clock signal

- Clock signal
  - A square wave with a constant period

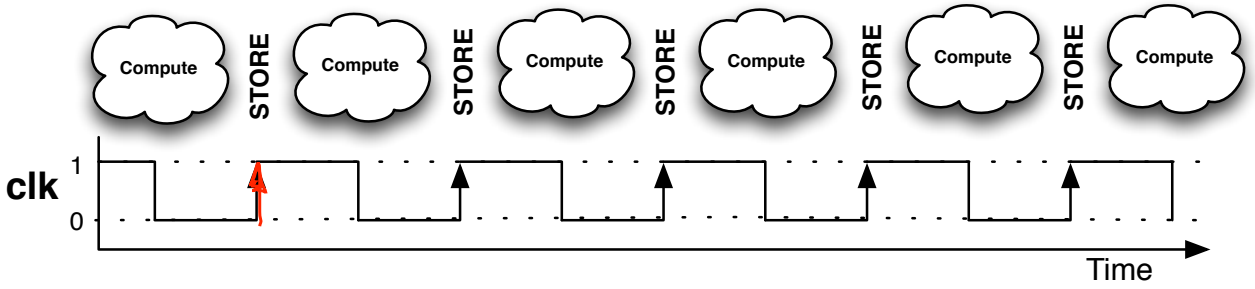


- We always update state at the same point in wave
  - E.g., the rising edge

Synchronous Design, cont.

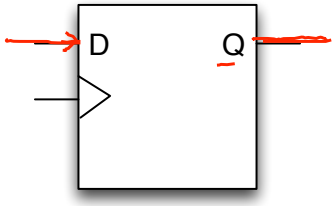


- Alternate between computation and updating state.



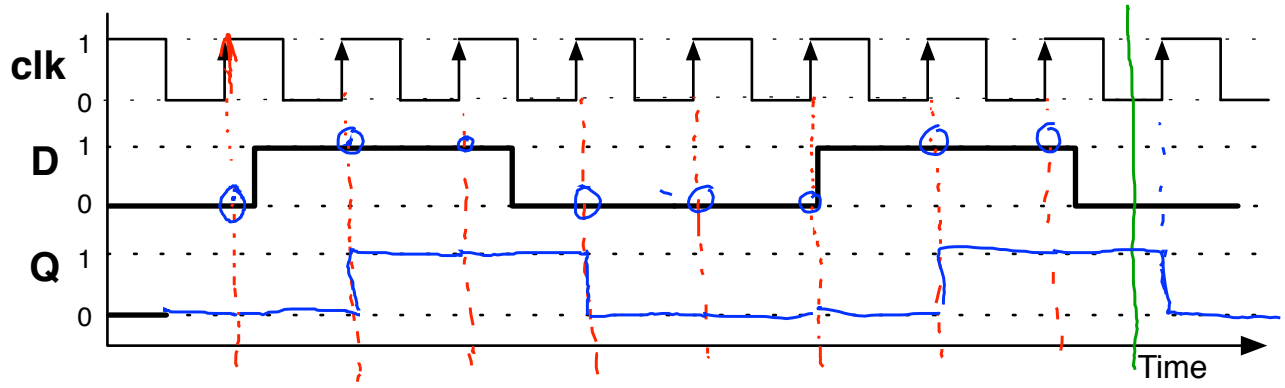
The state element that we really want…

- The D flip flop
  - Holds 1 bit of state
    - Output as Q.
  - Inputs
    - Copies D input into state on rising edge of clock.



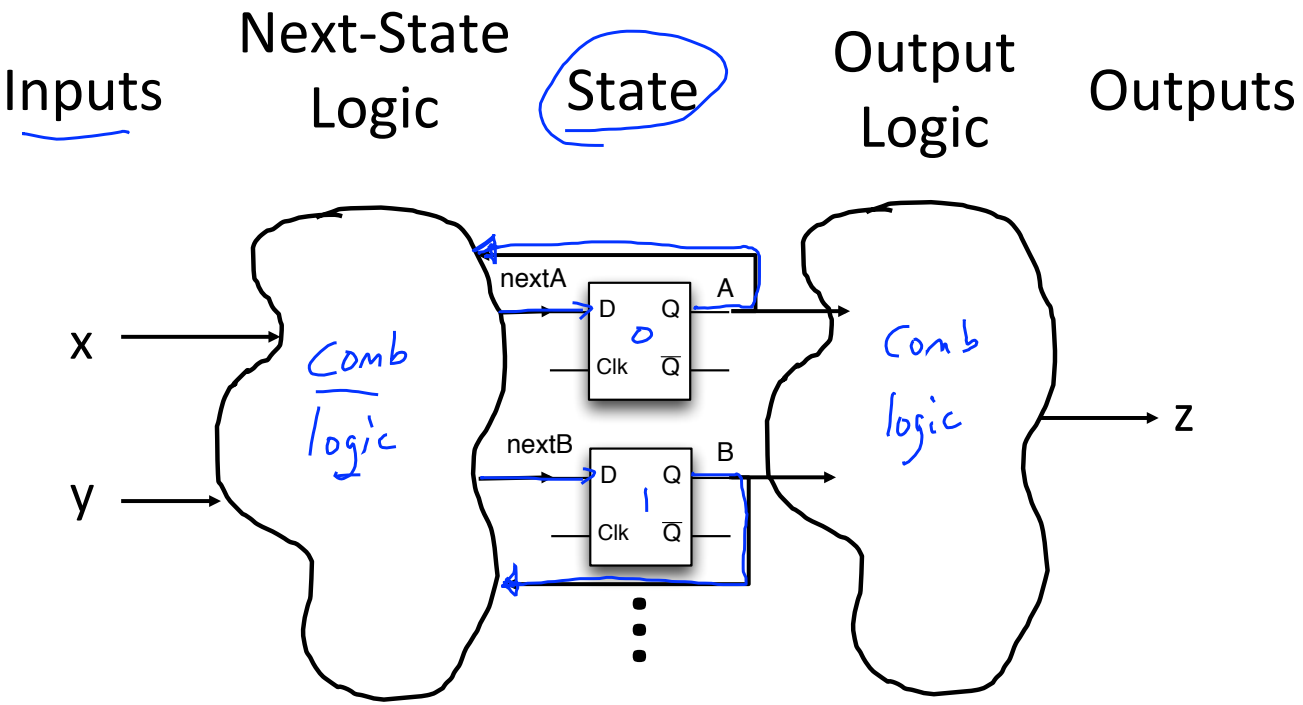
Q output

0
1



a) 0
b) 1

Finite State Machine (Moore 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

d n	
10	dime (10)
01	nickel (5)
00	nothing (0)

- Ouputs

c	
0	no candy for you.
1	dispense candy

- State identification

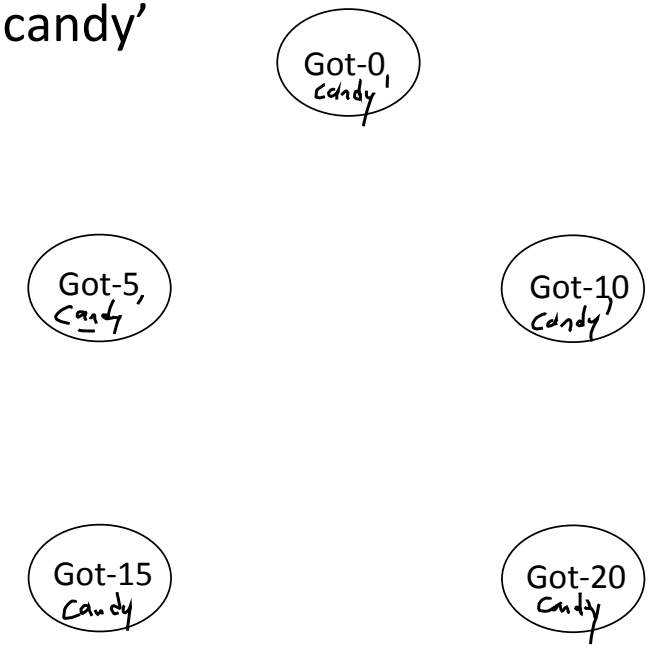
balance of 0 cents
" " 5 "

balance of 10 cents
" " 15 cents
balance of 20 cents

5 states.

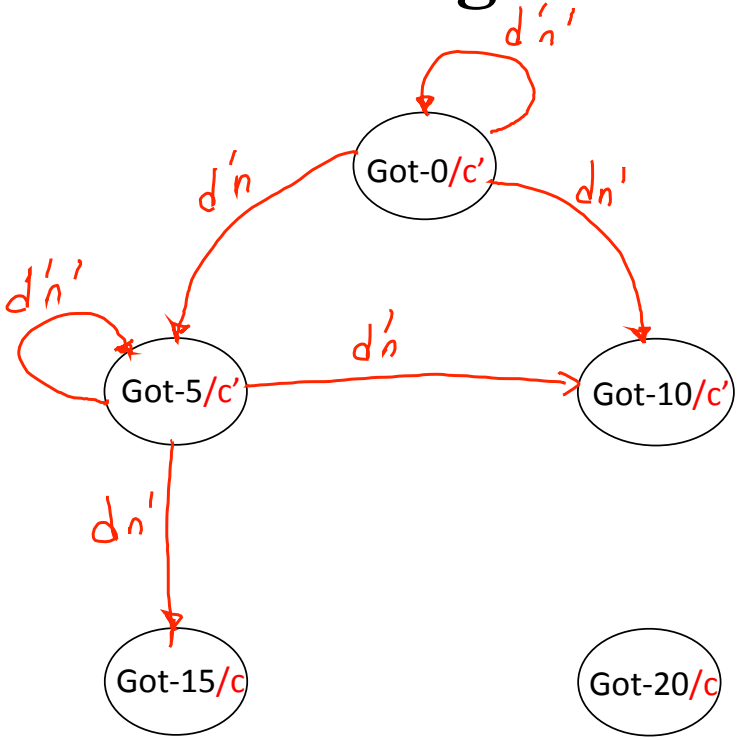
Step 1: Build the State Diagram

Outputs: candy or candy’



a) candy
b) candy'

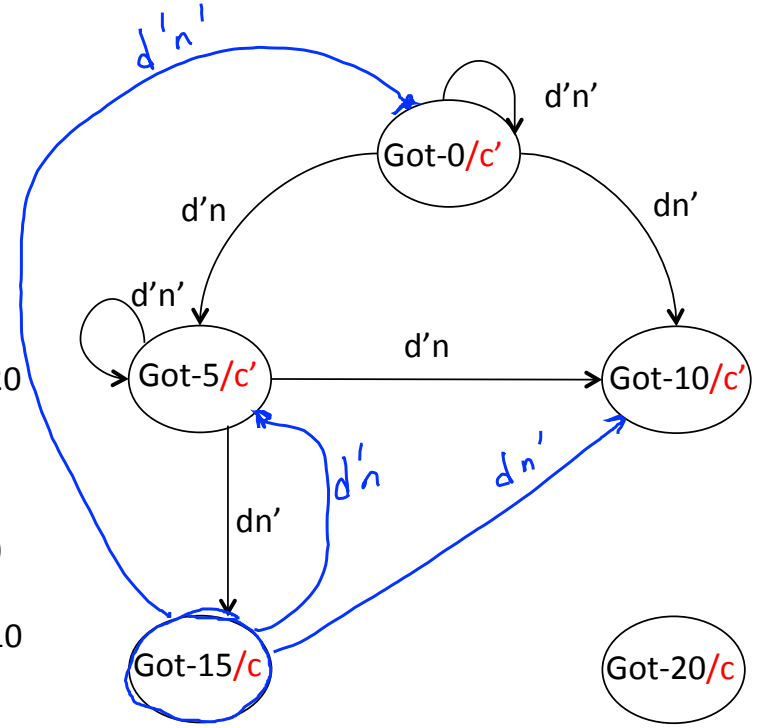
Step 1: Build the State Diagram



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

∅ 5 10

Step 1: Build the State Diagram



What are the transitions for state Got-15?

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

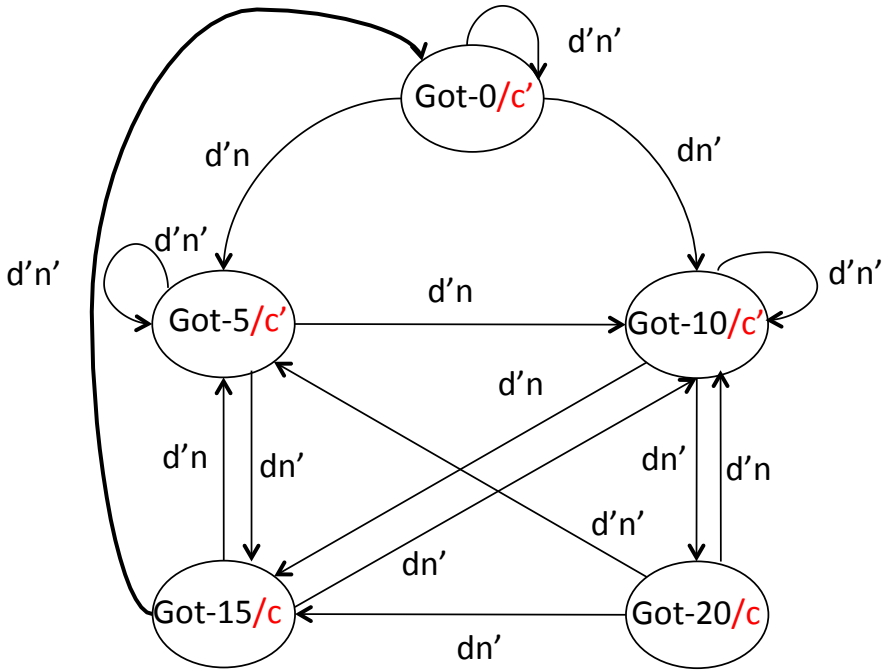
B: d'n': Got0; d'n: Got0; dn': Got0

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

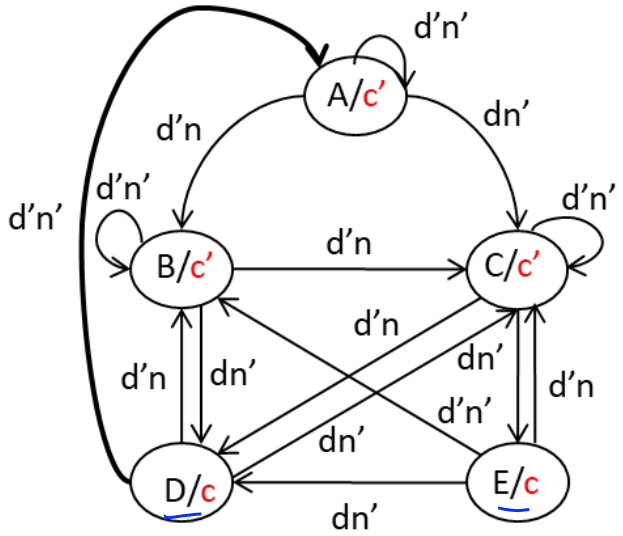
D: d'n': Got15; d'n: Got5; dn': Got10

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

Final State Diagram



Step 2: Build a State Table



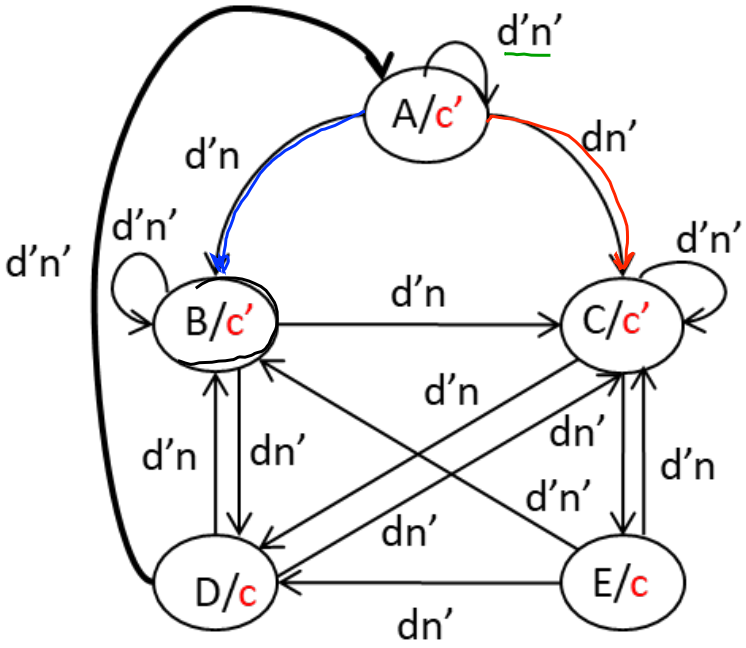
Current State	Output **candy**
A	0
B	0
C	0
D	1
E	1

15
20

Candy = D + E;

In a 'Moore machine', the outputs are a function only of the current state.

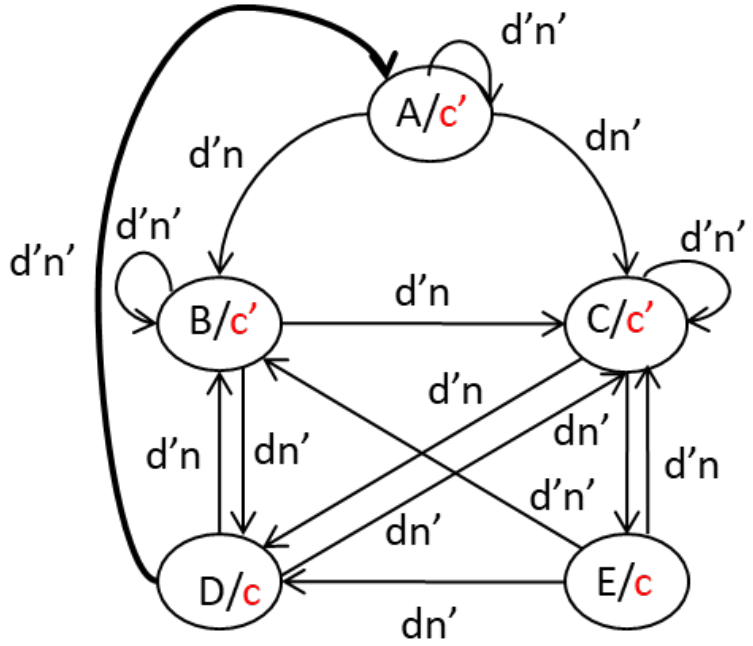
Step 2: Build a Next-State Table



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	d'n	
C	dn'	
D	d'n'	
D	d'n	
D	dn'	
E	d'n'	
E	d'n	
E	dn'	

	A	B	C	D	E
A	A	B	B	B	C
B	B	C	C	D	D
C	C	D	E	A	E

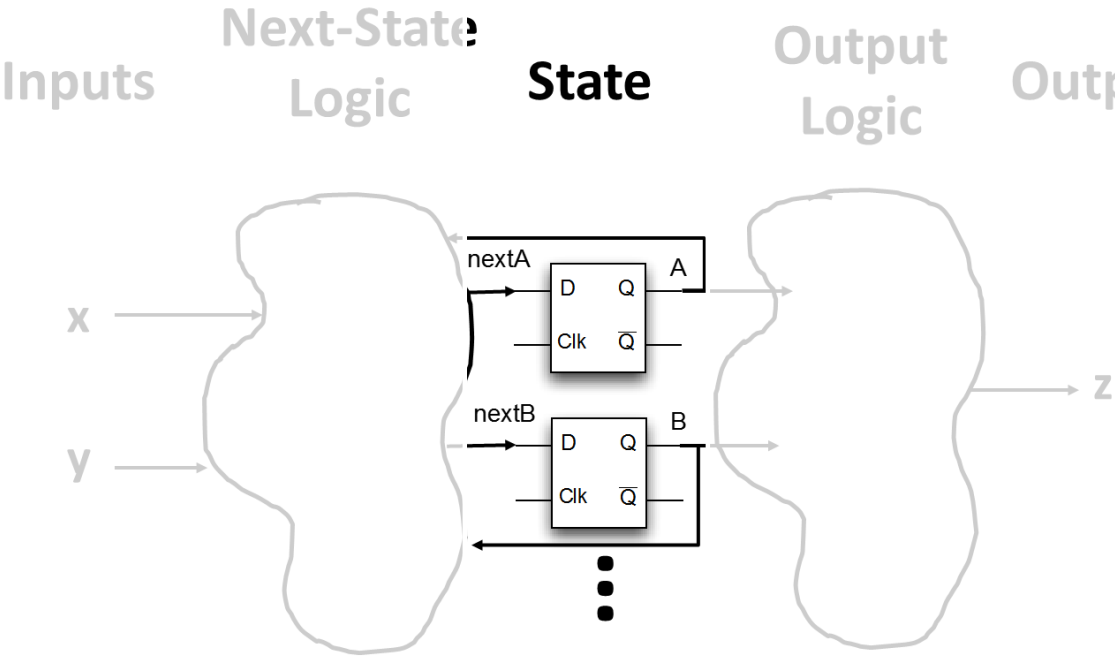
Step 2: Build a Next-State Table



Why do we need sequential logic to build this 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 3: Build Sequential Circuit



Step 3: Build Sequential Circuit (State)

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

5 flip flops

State Encoding:

minimum

5 states: How many bits?

$2^2 = 4$

$2^3 = 8$

a) 2
b) 3 ✓
c) 4
d) 5
e) 6

Step 3: Build Sequential Circuit (State)

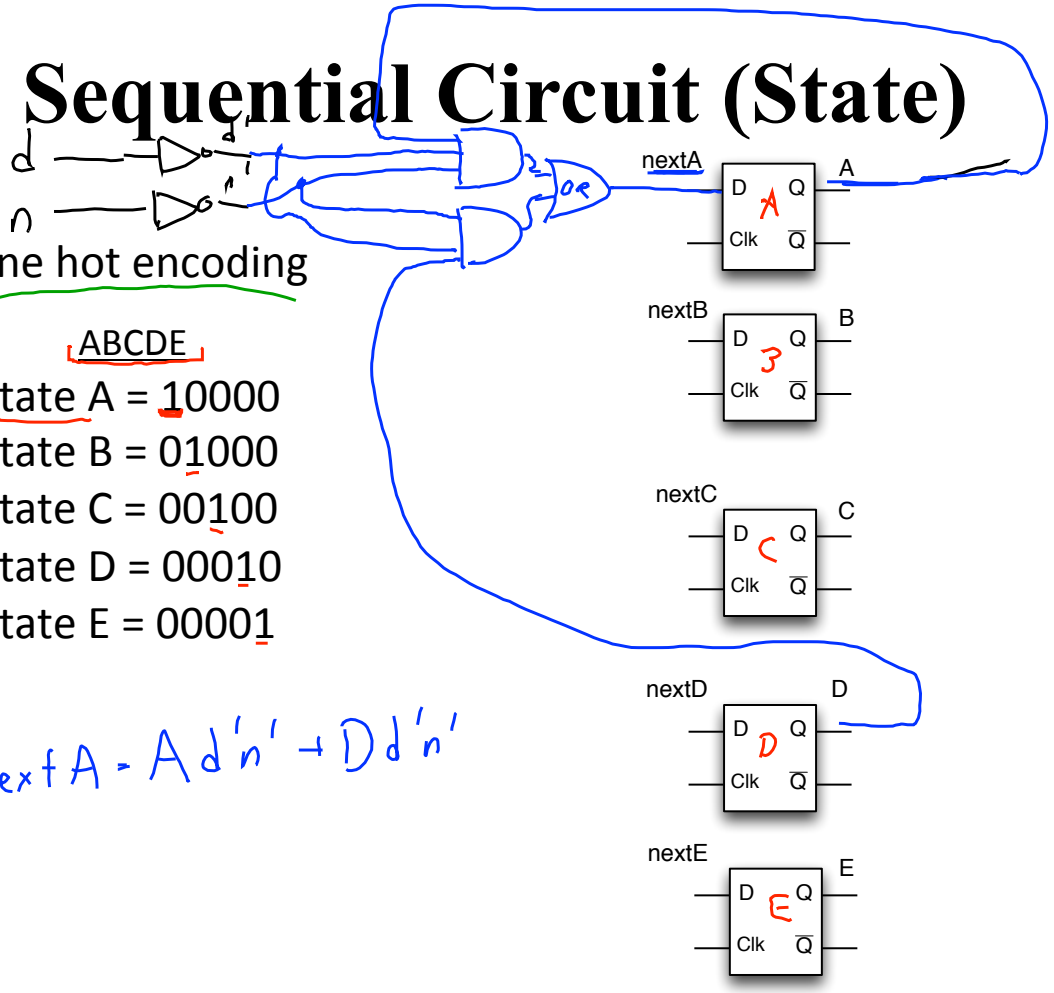
Current State	Input	Next State
A	d'n'	A
A	d'n	B
A	dn'	C
B	d'n'	B
B	d'n	C
B	dn'	D
C	d'n'	C
C	d'n	D
C	dn'	E
D	d'n'	A
D	d'n	B
D	dn'	C
E	d'n'	B
E	d'n	C
E	dn'	D

One hot encoding

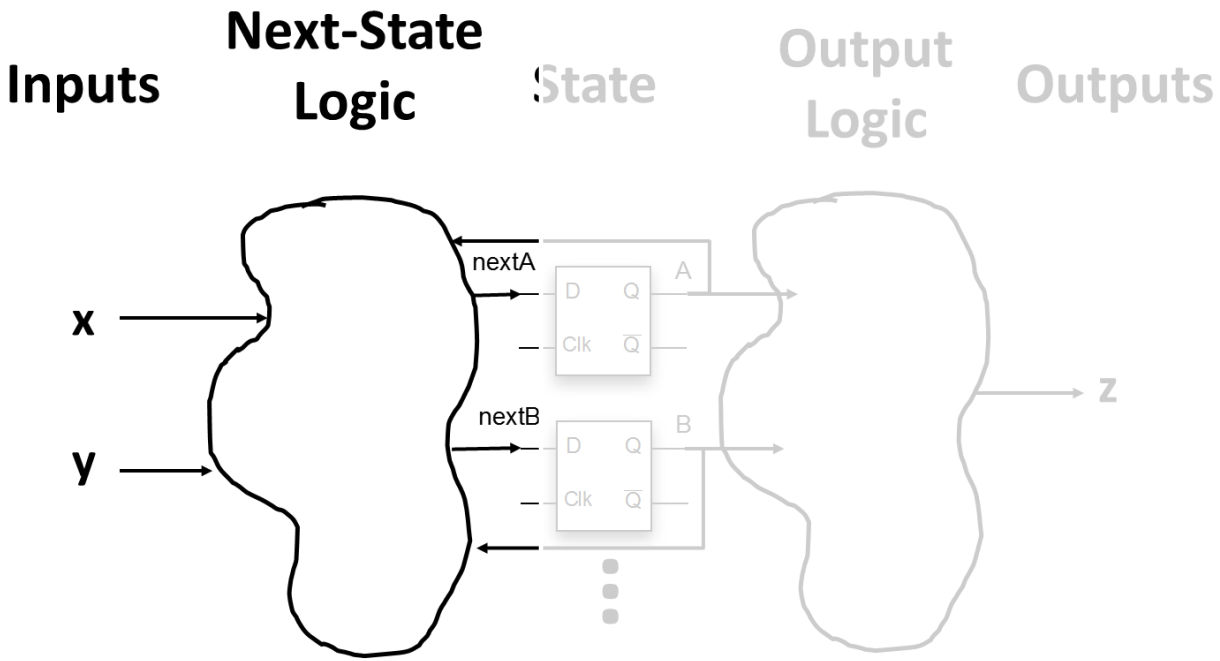
ABCDE

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

nextA = Ad'n' + Dd'n'



Step 3: Build Sequential Circuit

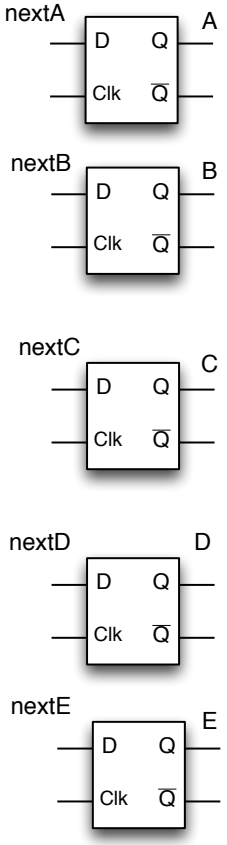


Step 3: Build Sequential Circuit (Next-State Logic)

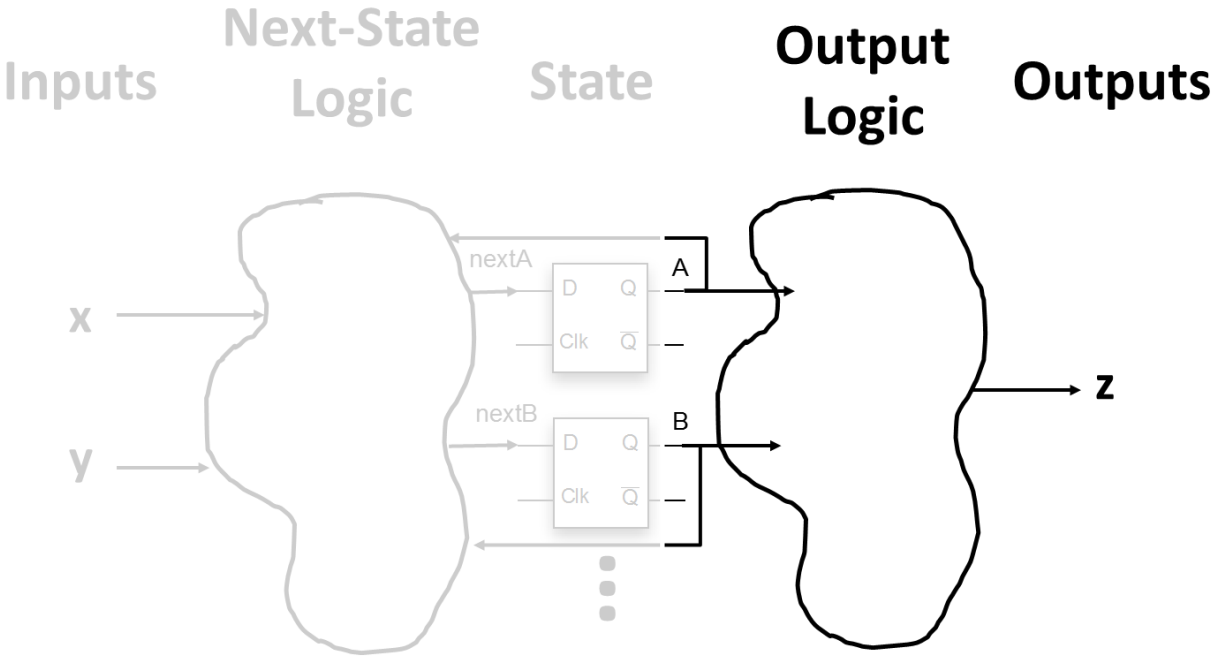
Current State	Input	Next State
A	d'n'	A
A	d'n	B
A	dn'	C
B	d'n'	B
B	d'n	C
B	dn'	D
C	d'n'	C
C	d'n	D
C	dn'	E
D	d'n'	A
D	d'n	B
D	dn'	C
E	d'n'	B
E	d'n	C
E	dn'	D

next D?

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



Step 3: Build Sequential Circuit



Step 3: Build Sequential Circuit (output logic)

Current State	Input	Next State	Output
A	d'n'	A	0
A	d'n	B	0
A	dn'	C	0
B	d'n'	B	0
B	d'n	C	0
B	dn'	D	0
C	d'n'	C	0
C	d'n	D	0
C	dn'	E	0
D	d'n'	A	1
D	d'n	B	1
D	dn'	C	1
E	d'n'	B	1
E	d'n	C	1
E	dn'	D	1

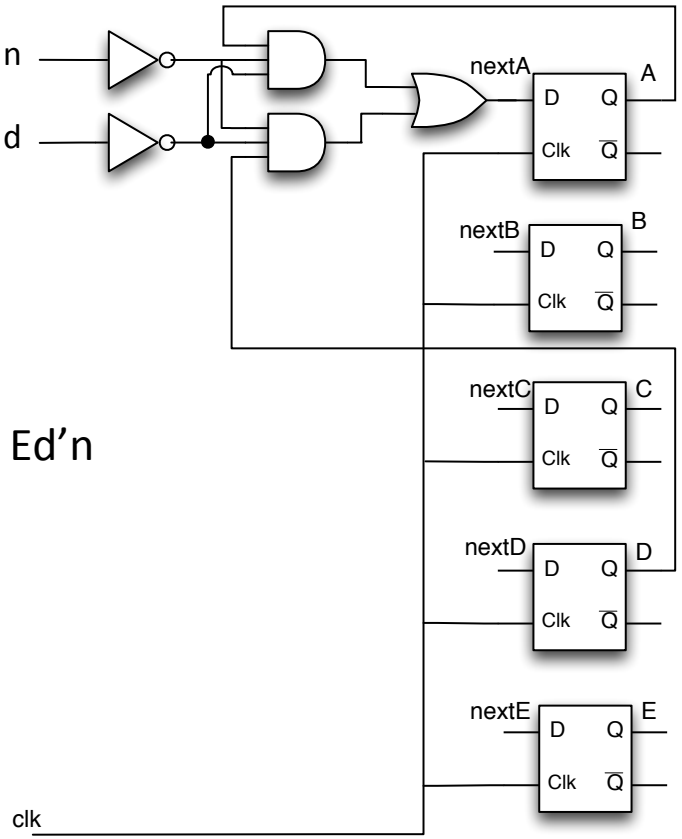
nextA = Ad'n'+ Dd'n'
nextB = Ad'n + Bd'n' + Dd'n + Edn'
nextC = And' + Bd'n + Cd'n' + Ddn' + Ed'n
nextD = Bdn' + Cd'n + Edn'
nextE = Cdn'

Output: Candy = D + E

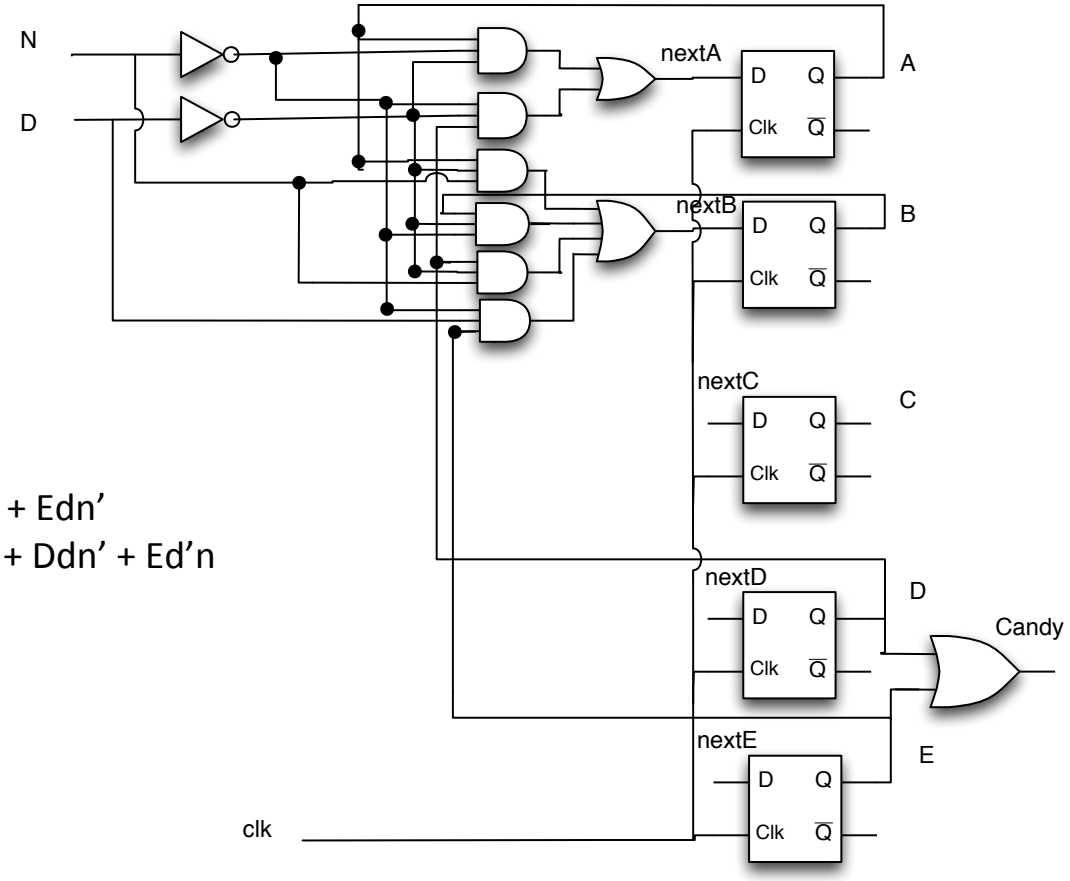
Step 3:Build Sequential Circuit (output logic)

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



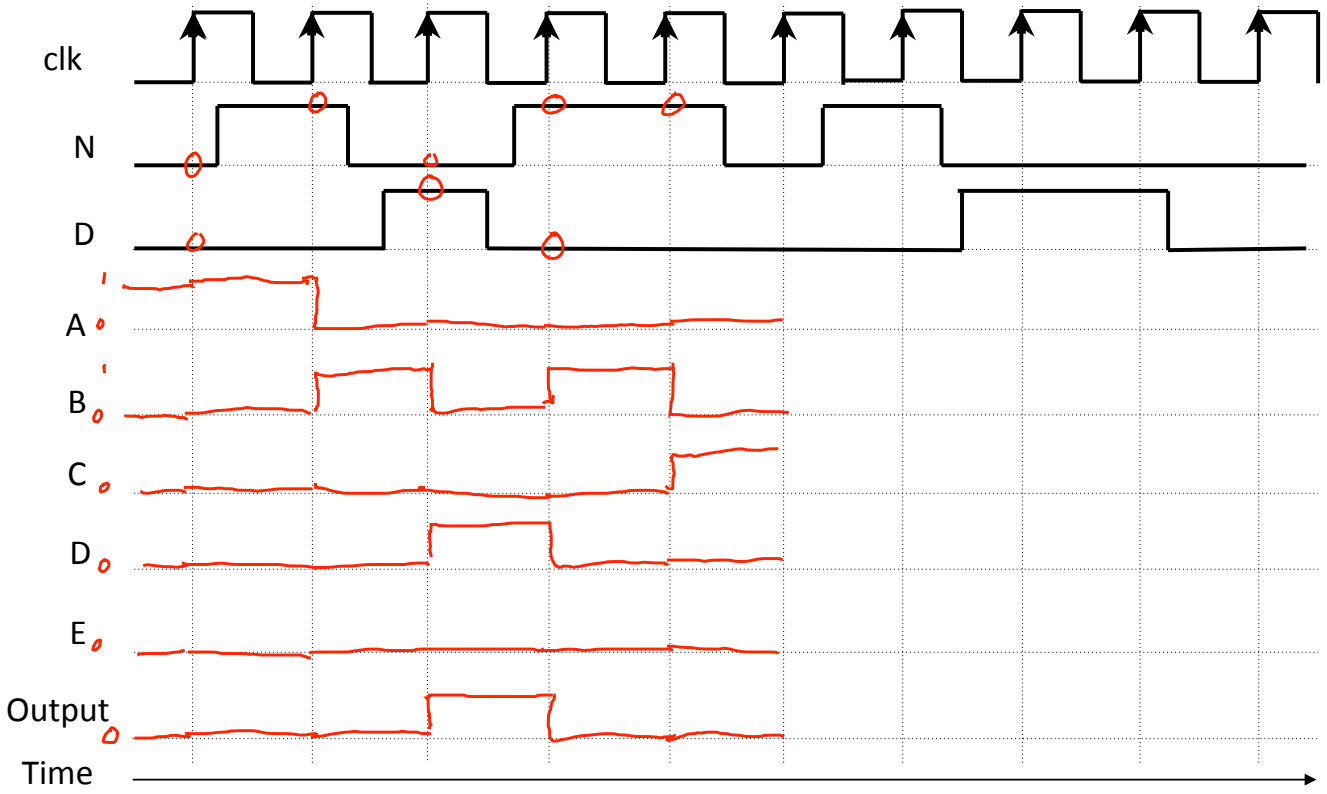
Step 3: Build Sequential Circuit ("finished")



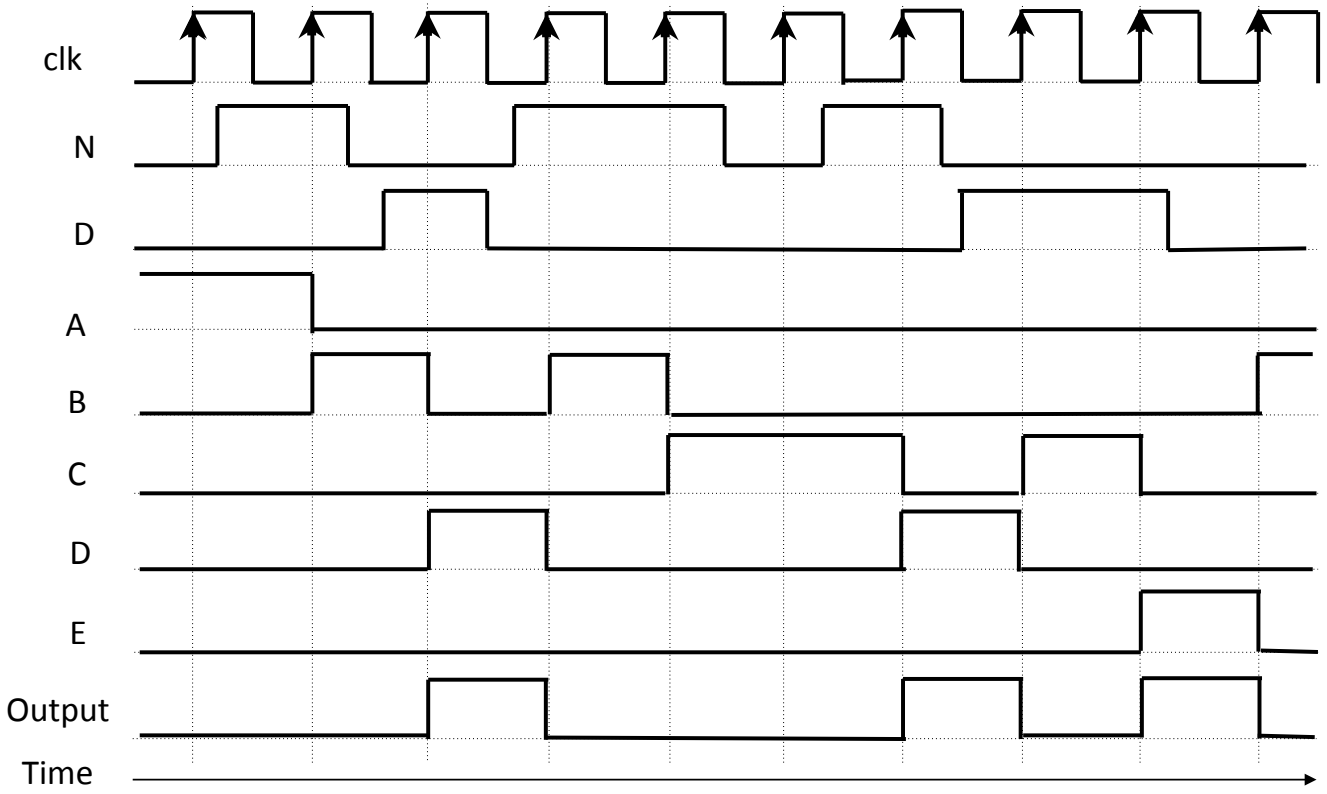
nextA = Ad'n'+ Dd'n'
nextB = Ad'n + Bd'n' + Dd'n + Edn'
nextC = And' + Bd'n + Cd'n' + Ddn' + Ed'n
nextD = Bdn' + Cd'n + Edn'
nextE = Cdn'

Output = Candy = D + E

Timing Diagram (update state on clock edges)

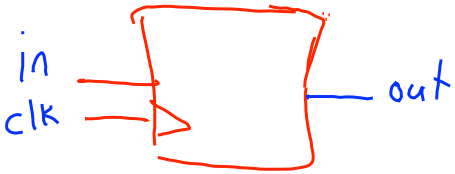


Timing Diagram



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.



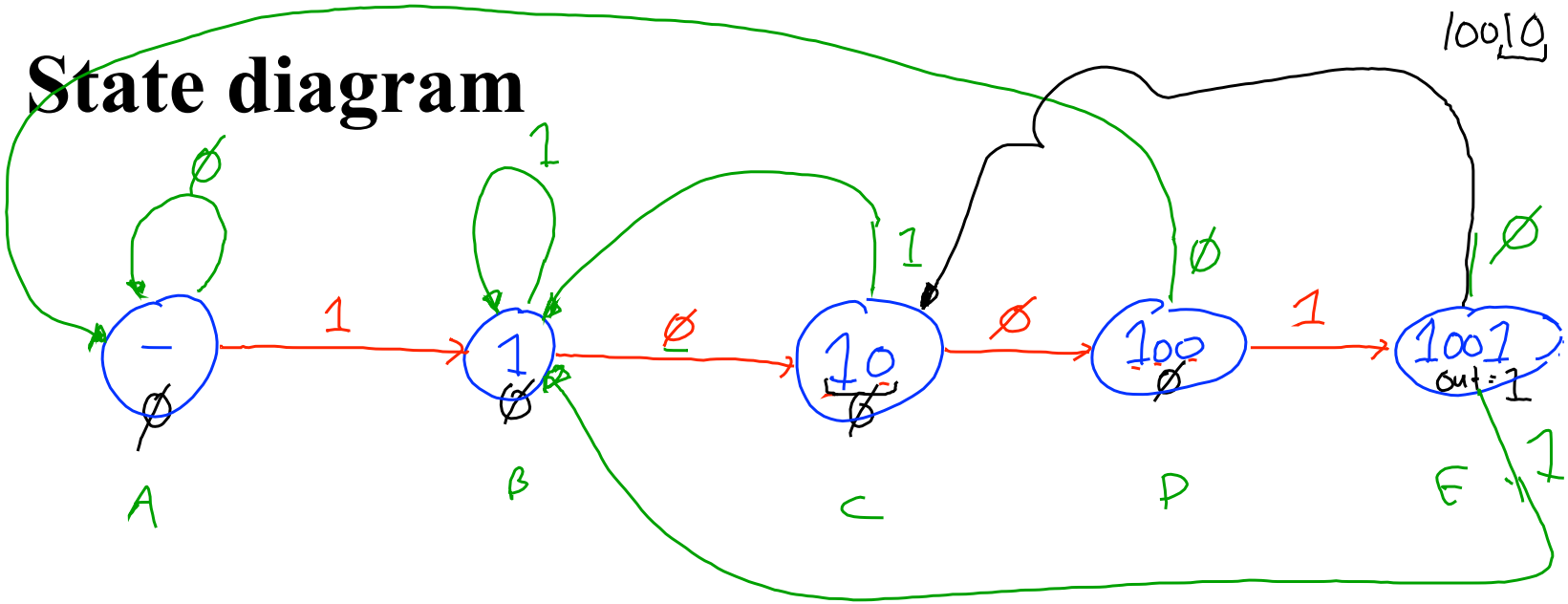
- Our example will detect the bit pattern "1001":

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.

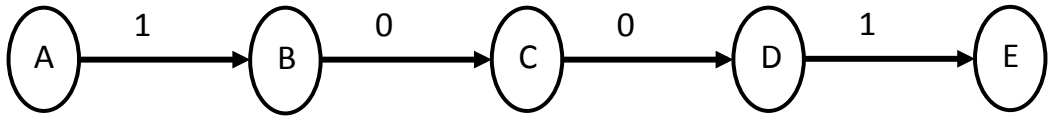
- This requires a sequential circuit because the circuit has to "remember" the inputs from previous clock cycles, in order to determine whether or not a match was found.

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

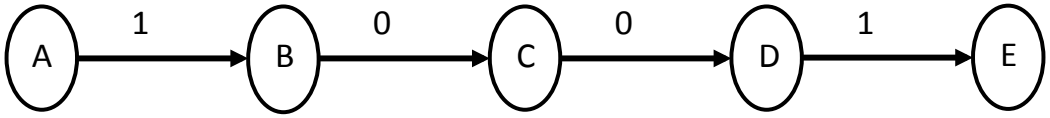


State	Meaning
A	None of the desired pattern (1001) has been input yet.
B	We've already seen the first bit (1) of the desired pattern.
C	We've already seen the first two bits (10) of the desired pattern.
D	We've already seen the first three bits (100) of the desired pattern.
E	We've seen the pattern (1001)

State diagram

i>clicker

- 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

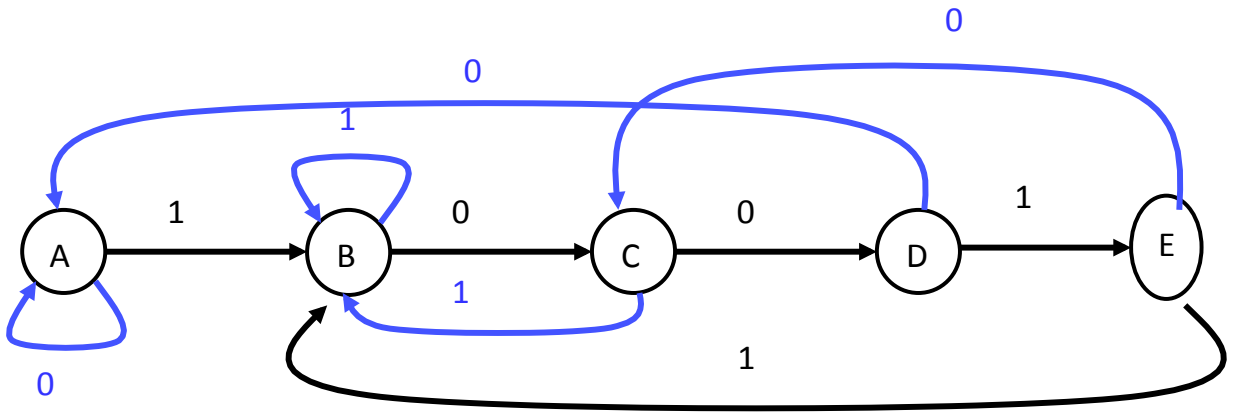
B: x=0-> C; x=1 -> B

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

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

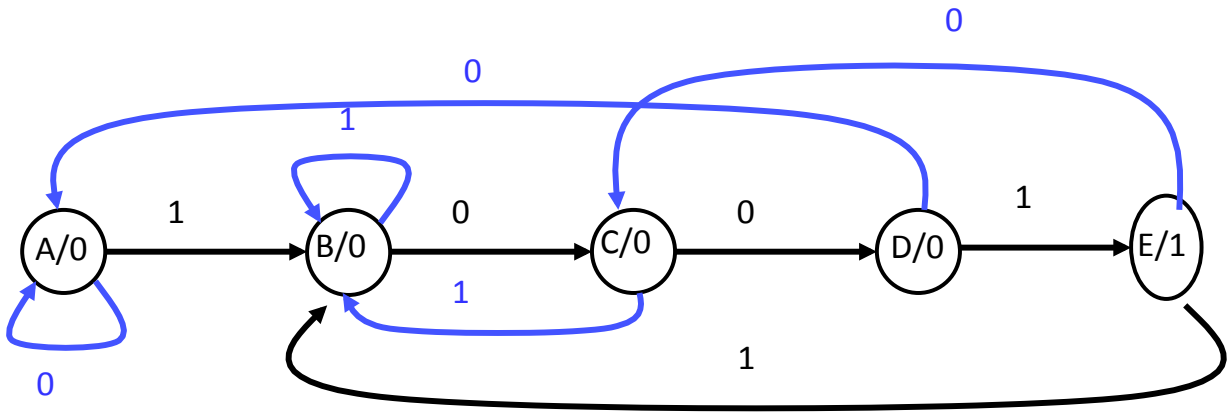
State diagram

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

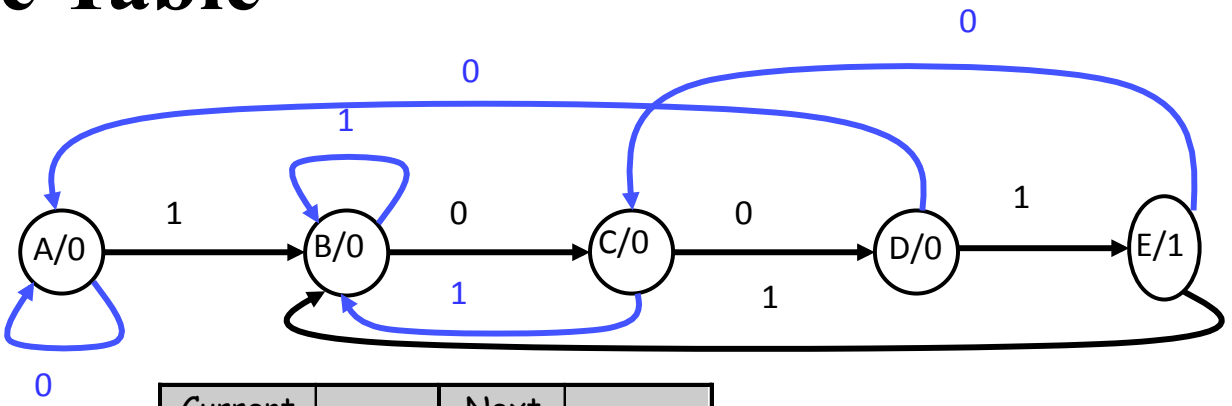


State diagram

- Need to determine the output

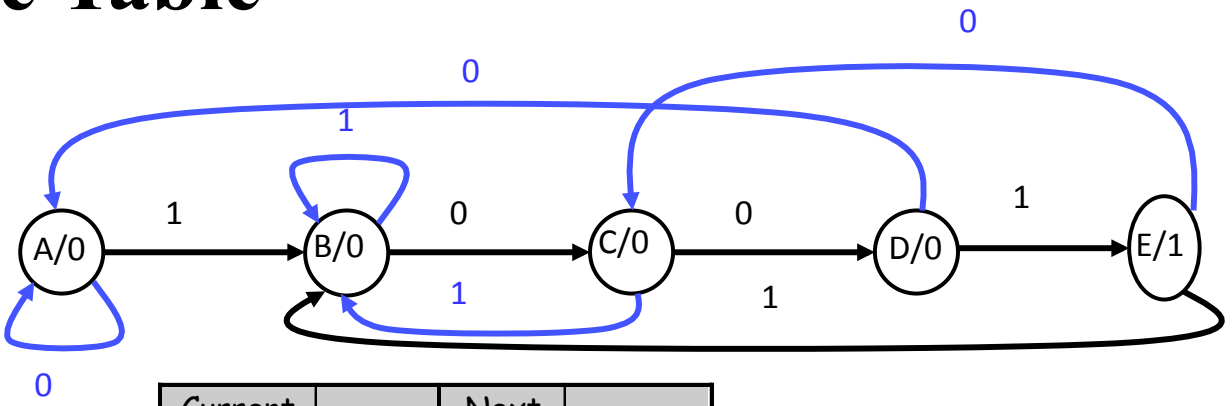


State Table



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