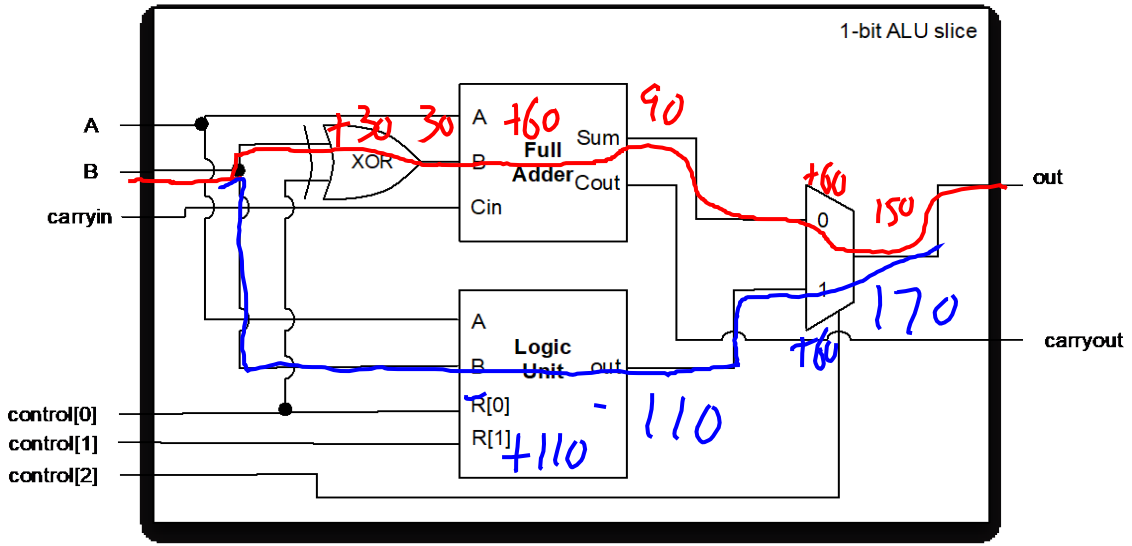
Computing components from ALU1



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
0,1	out	60ps
S	out	80ps

Exam 2 content starts today

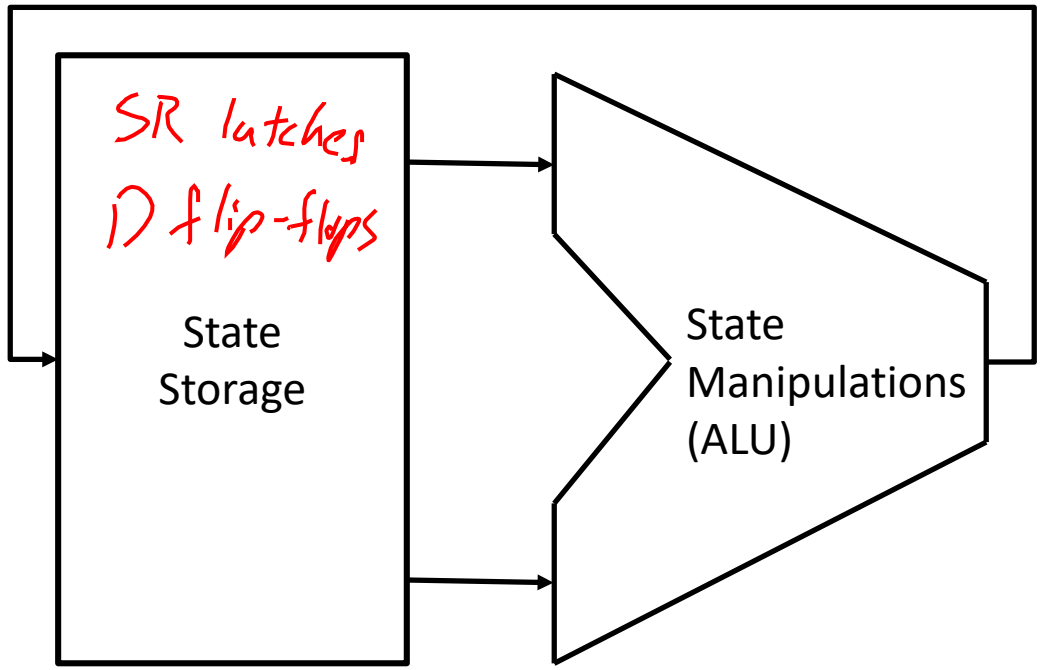
Finite State Machines

Happy job fairing

Honor's section lecture & HW post tonight

State – the central concept of computing

How do we generate control signals for circuits? Finite State Machines



Today's lecture

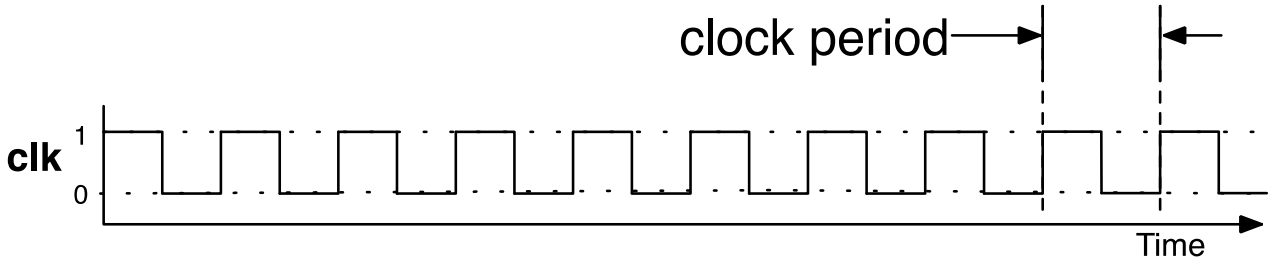
- Goal: Build a sequential circuit from a state diagram
  - Step 0: Problem specification
  - Step 1: Build the state diagram
  - Setp 2: Build the state table
  - Step 3: Build the sequential circuit using D flip-flops
- Timing diagram
- Another example: Sequence recognizer

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.

Synchronous Design

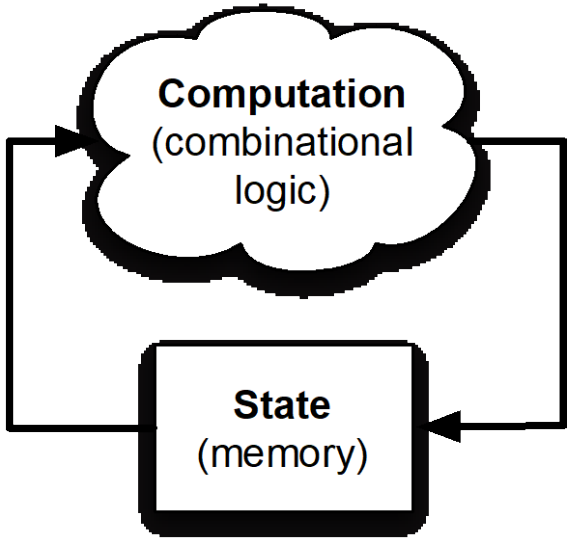
- *The easiest (and most common) way to build computers*
- All state elements get updated at the same time
  - Using a clock signal

- Clock signal
  - A square wave with a constant period

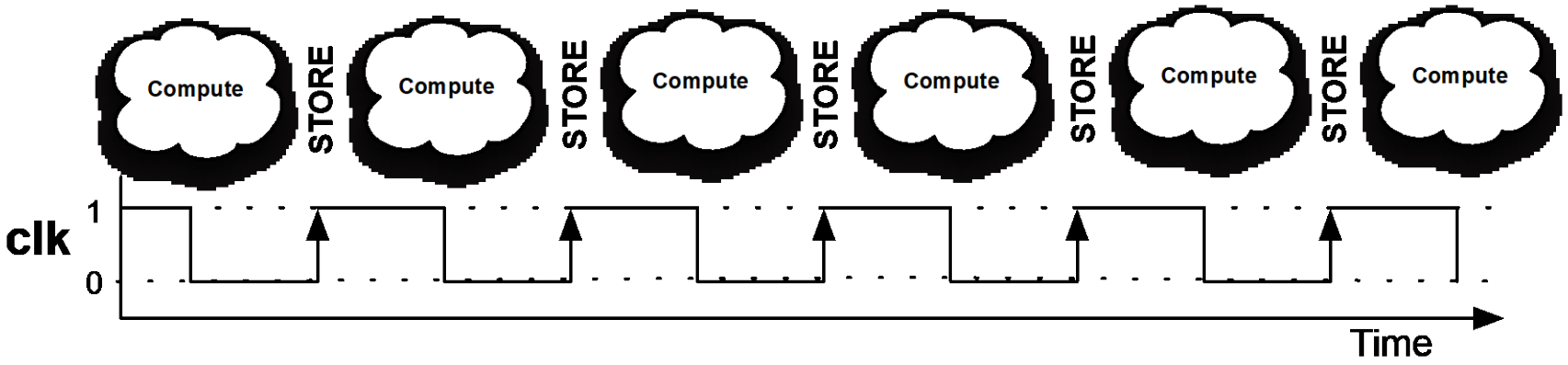


- We always update state at the same point in wave
  - E.g., the rising edge

Synchronous Design, cont.

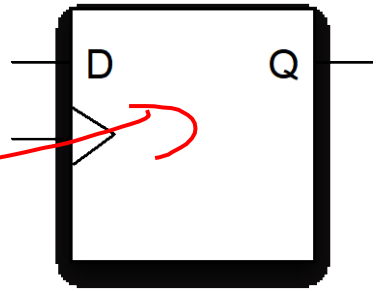


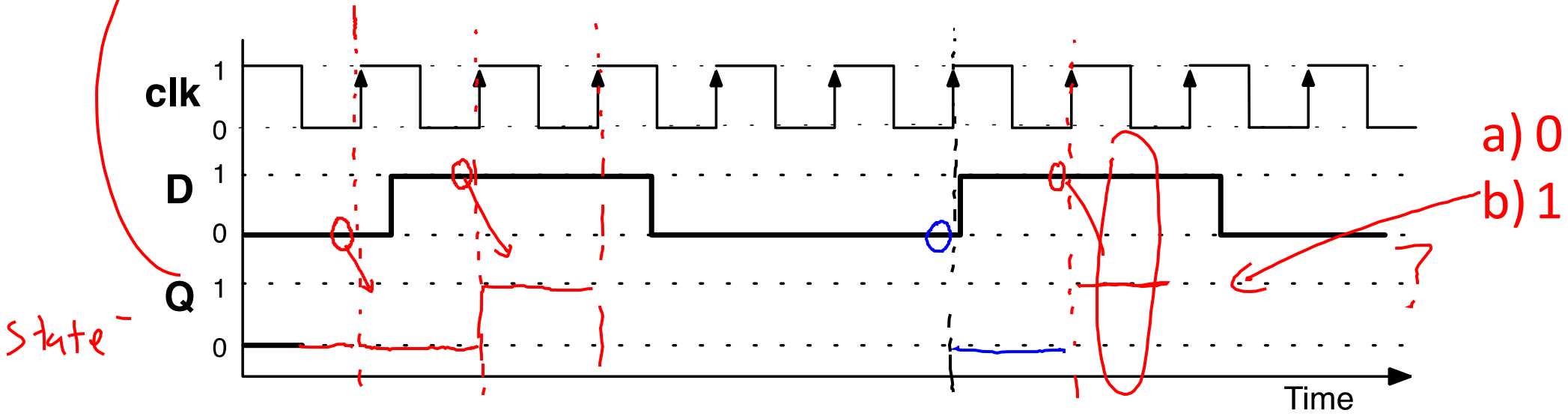
- Alternate between computation and updating state.

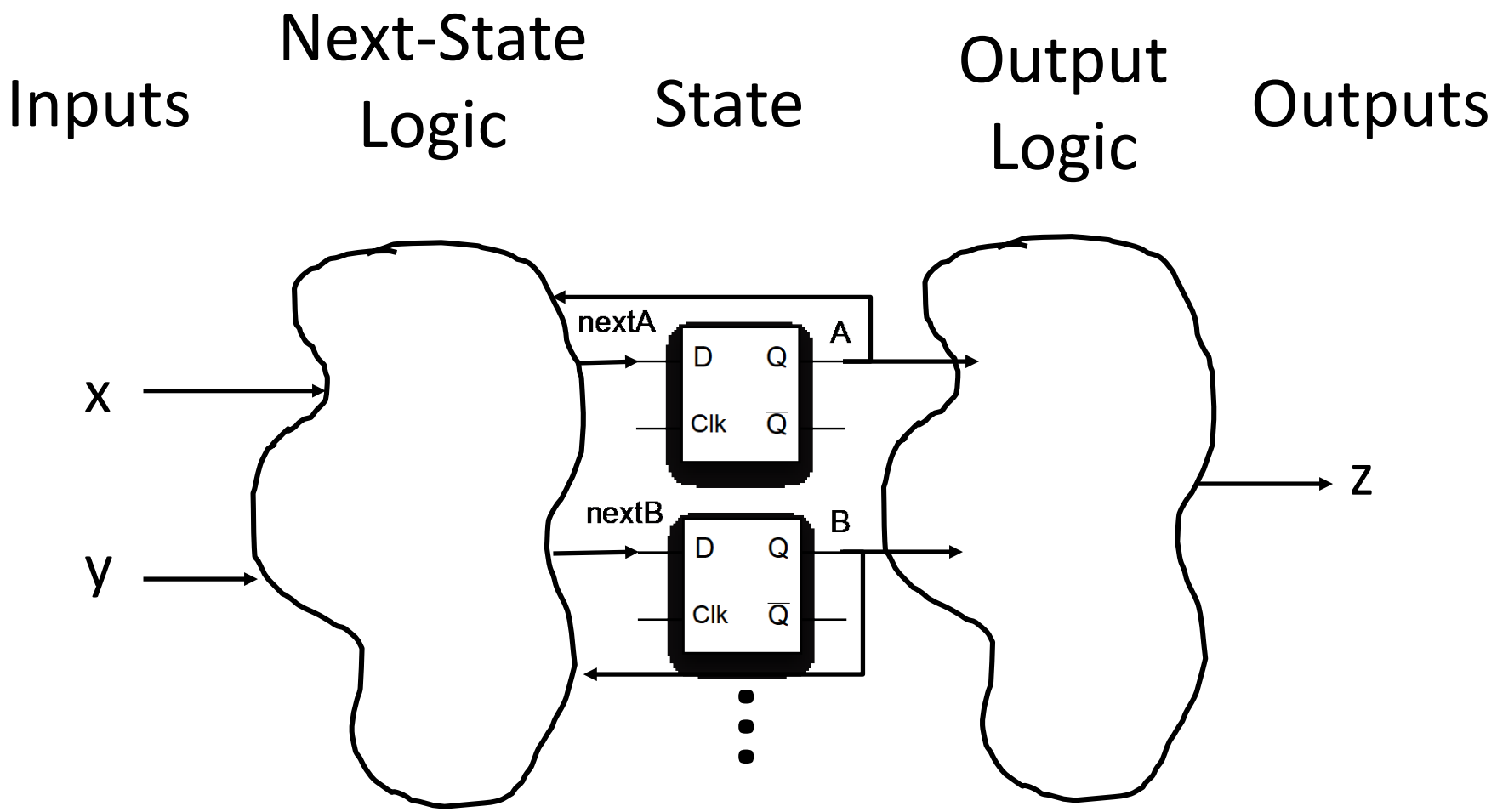


The state element that we really want…

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






Inputs
Next-State Logic
State
Output Logic
Outputs
x
y
nextA
D
Q
Clk
Q
A
nextB
D
Q
Clk
Q
B
z

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

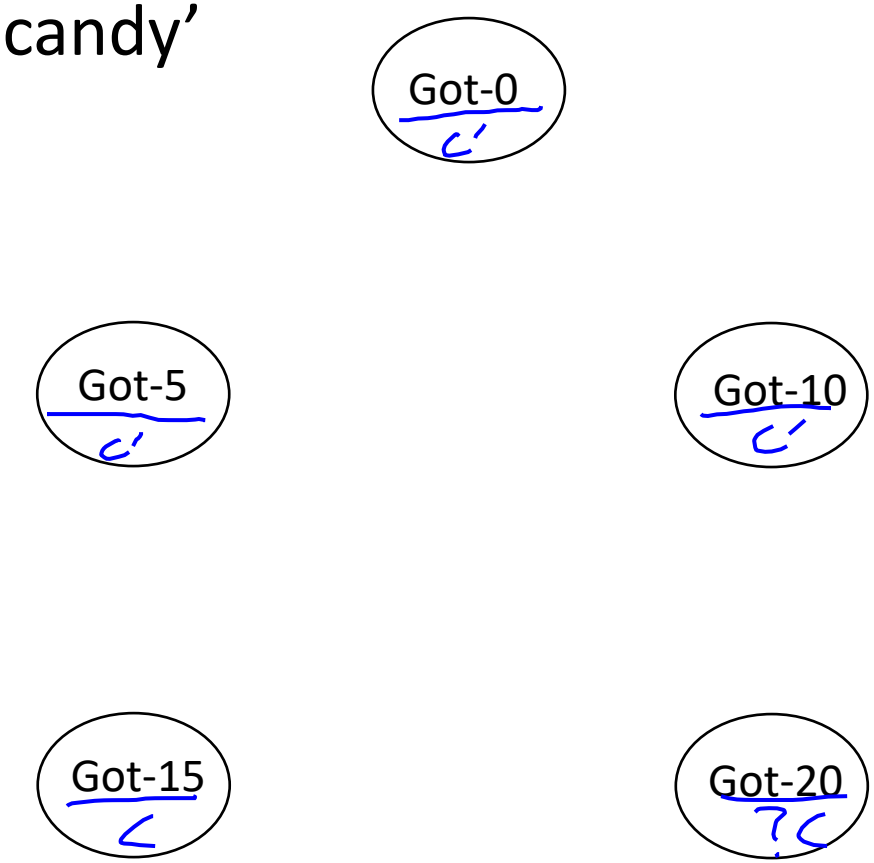
- Inputs
  - $d = 1$ iff dime inserted
  - $n = 1$ iff nickel inserted
- Ouputs
  - $c = 1$ iff dispense candy
- State identification
  - 0 ¢
  - 5 ¢
  - 10 ¢
  - 15 ¢
  - 20 ¢

d n	
0 0	no money inserted
0 1	nickel "
1 0	dime "
1 1	not possible

c	
0	no candy
1	candy

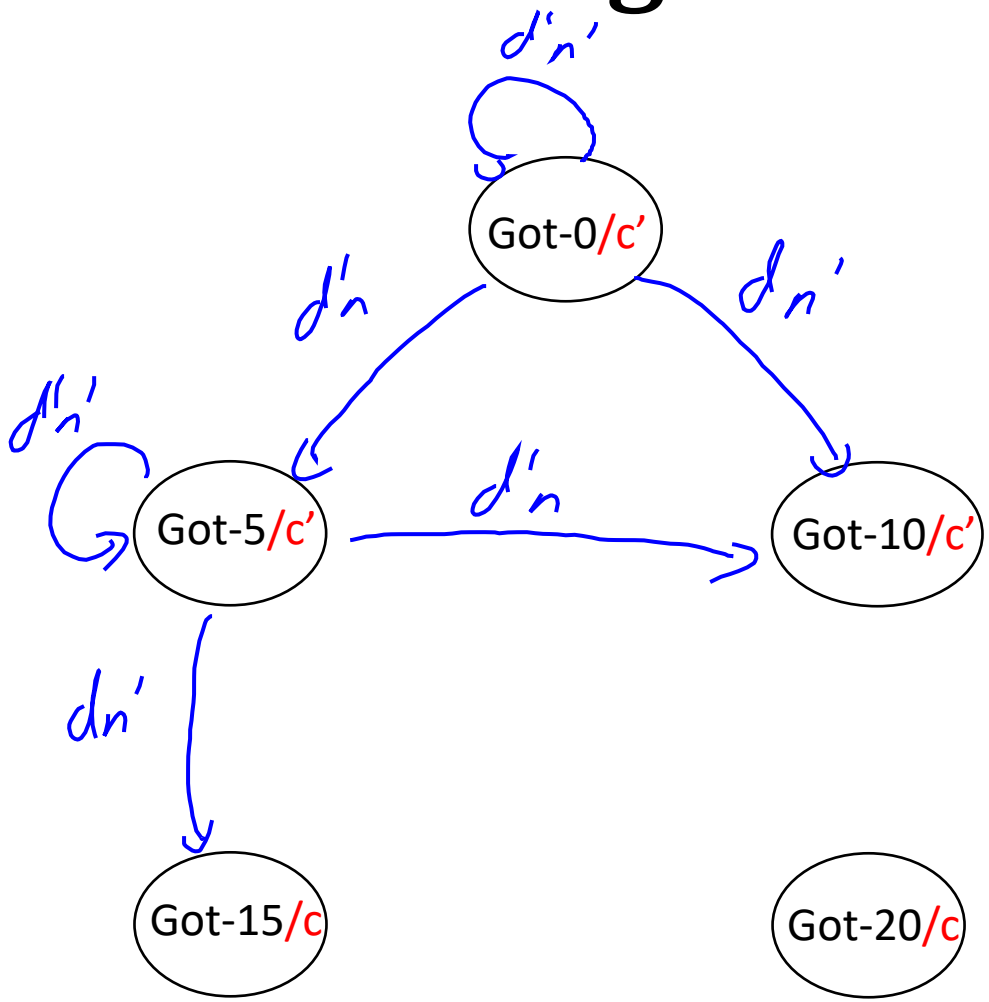
Step 1: Build the State Diagram

Outputs: candy or candy’



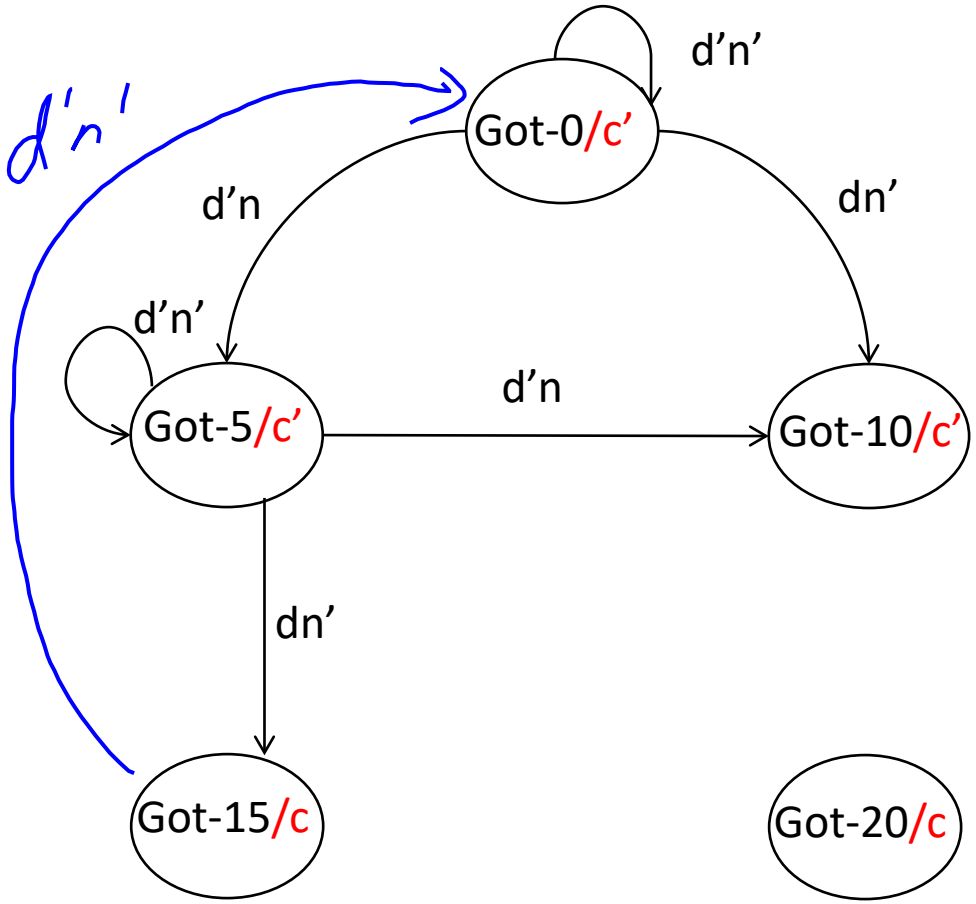
a) candy
b) candy'

Step 1: Build the State Diagram



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

Step 1: Build the State Diagram



What are the transitions for state Got-15?

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

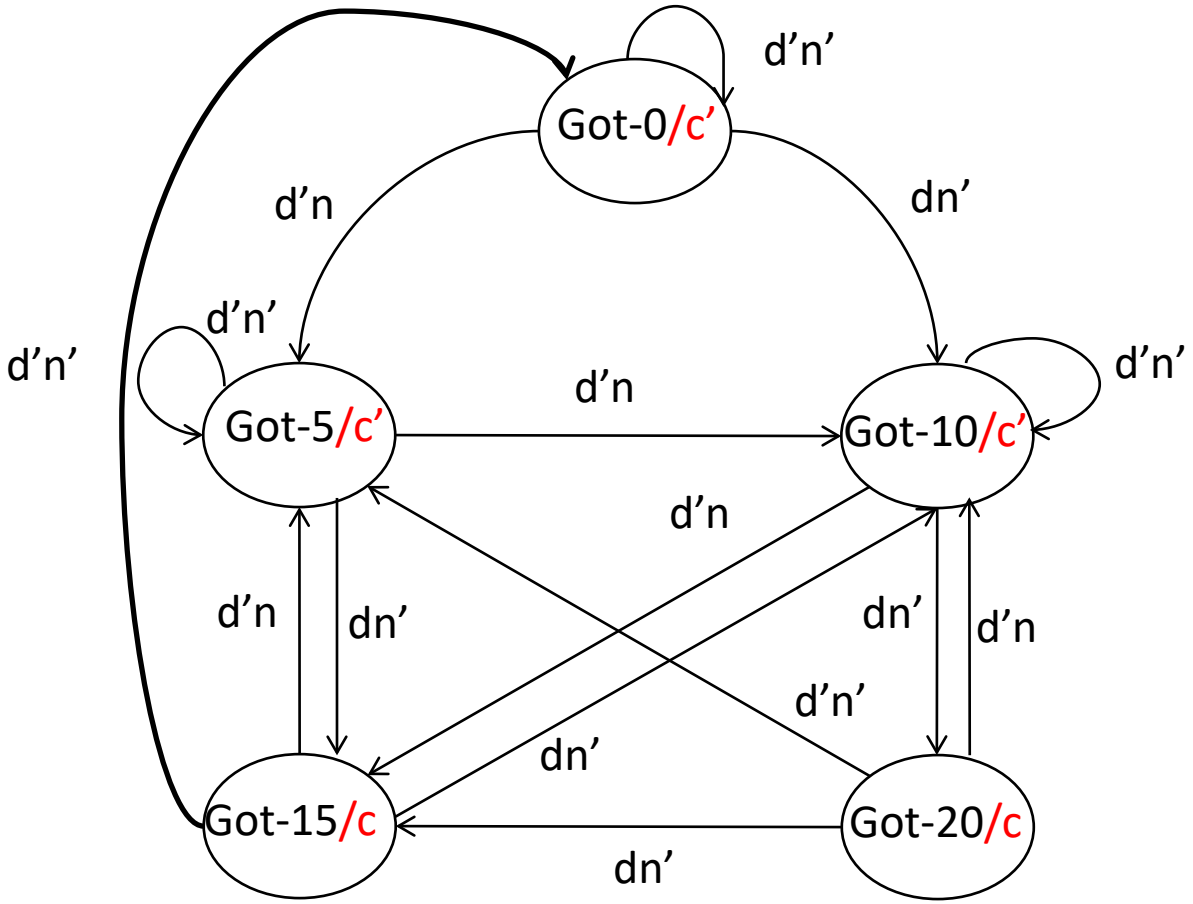
B: d'n': Got0; d'n: Got0; dn': Got0

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

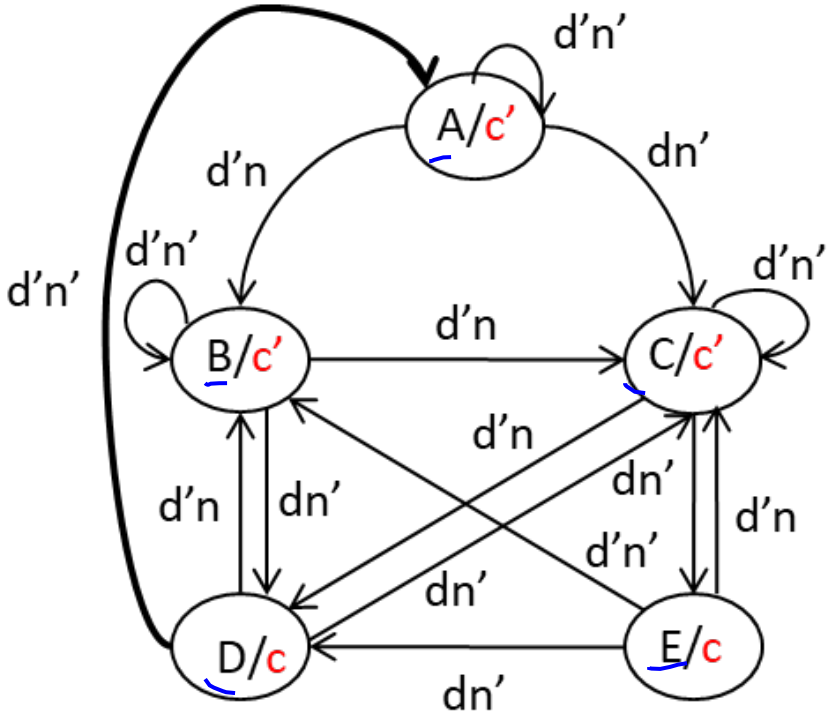
D: d'n': Got15; d'n: Got5; dn': Got10

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

Final State Diagram



Step 2: Build a State Table

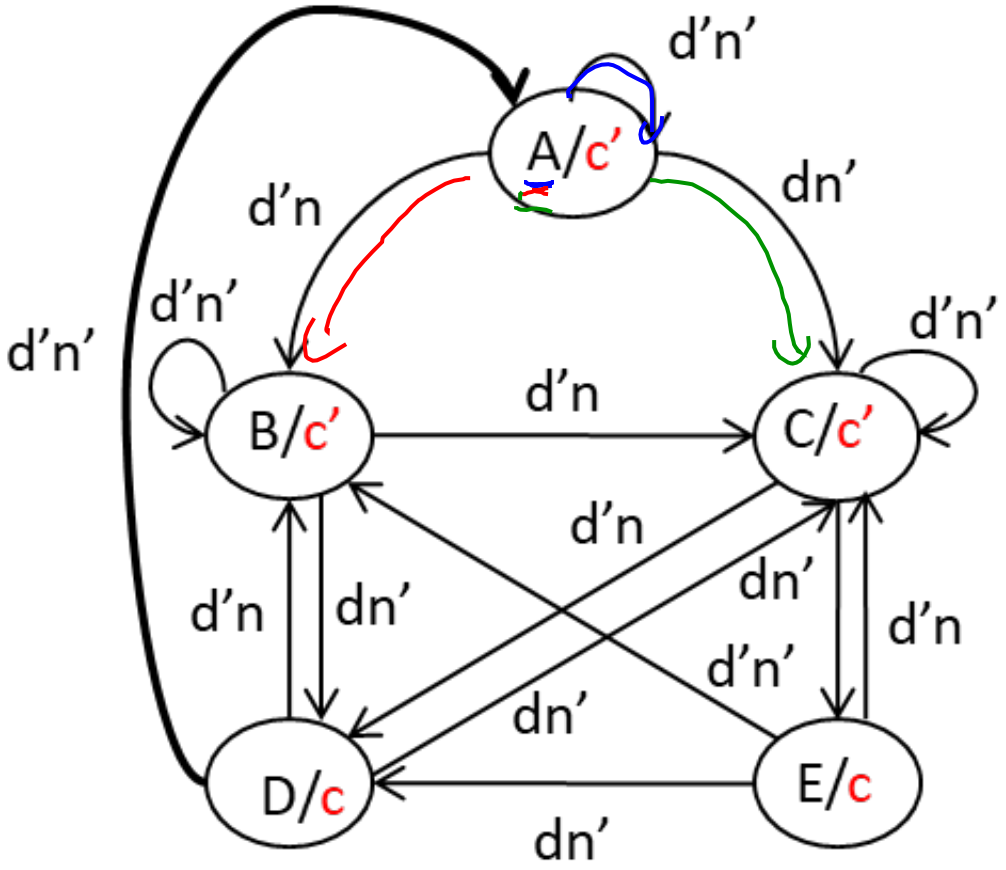


Current State	Output **candy**
A	0
B	0
C	0
D	1
E	1

candy = D + E

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

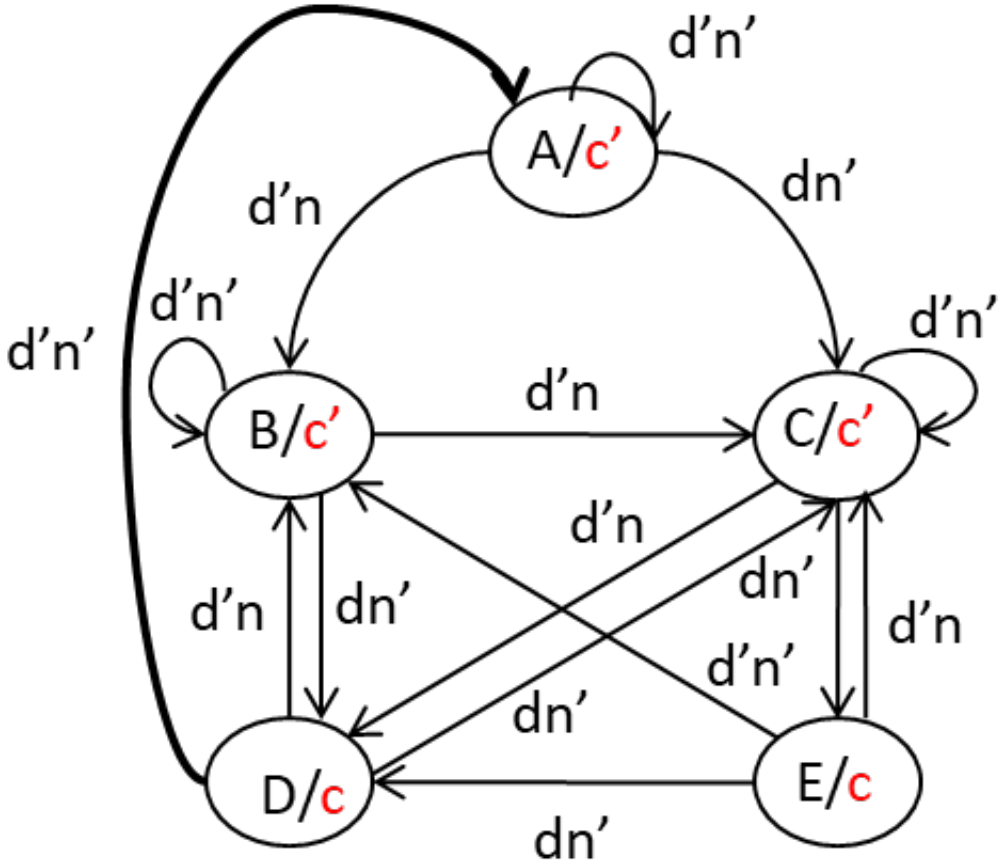
Step 2: Build a Next-State Table



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

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

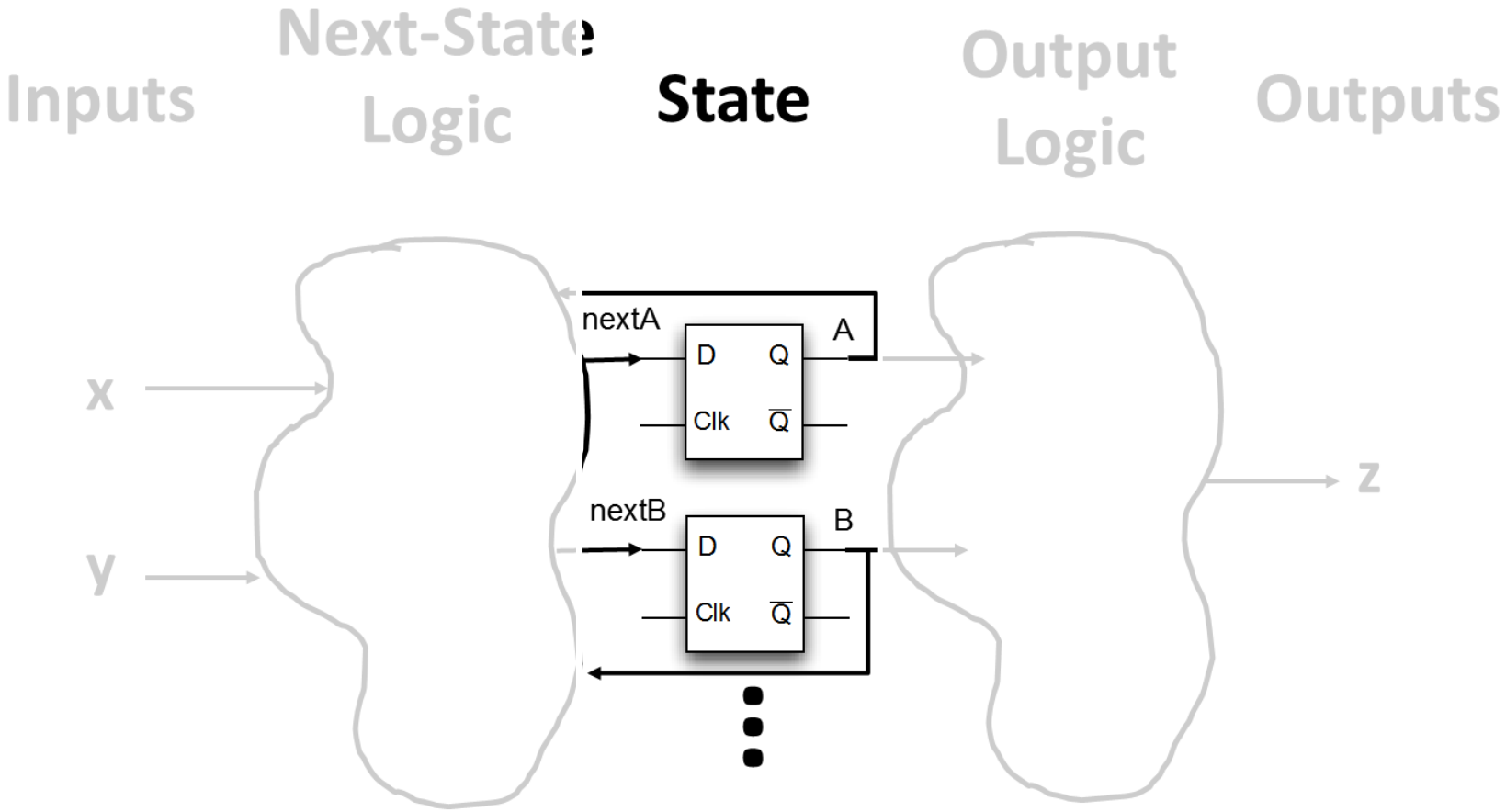
Step 2: Build a Next-State Table



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
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

Why do we need sequential logic to build this circuit?

Step 3: Build Sequential Circuit



Step 3: 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?

a) 2
b) 3 – minimal (efficient)
c) 4
d) 5 – one hot (easy)
e) 6

Step 3: 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

ABCDE

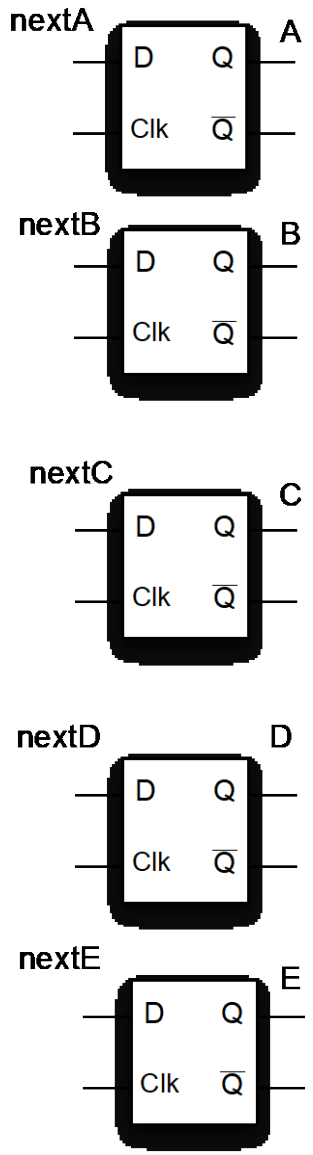
State A = 10000

State B = 01000

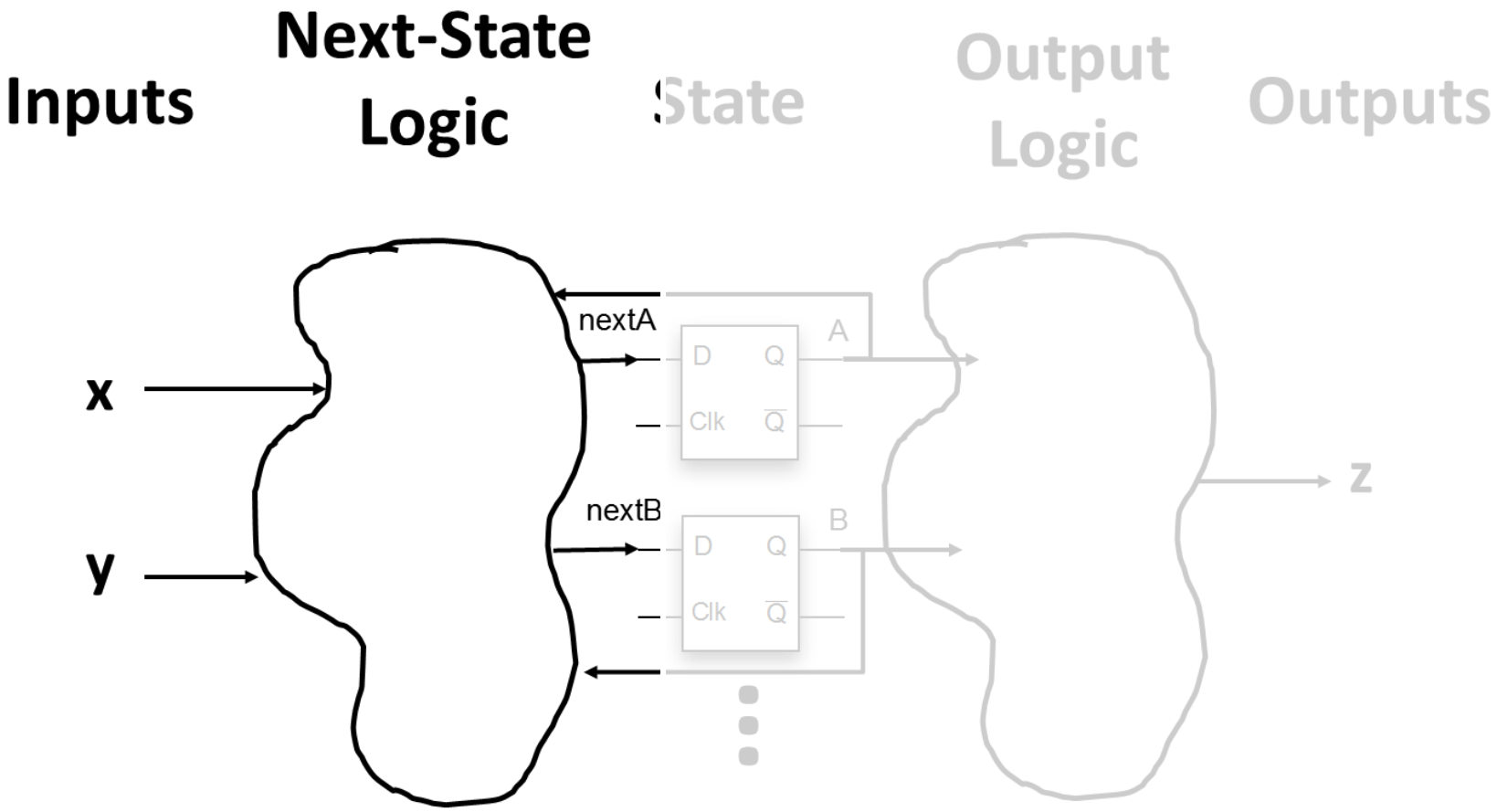
State C = 00100

State D = 00010

State E = 00001



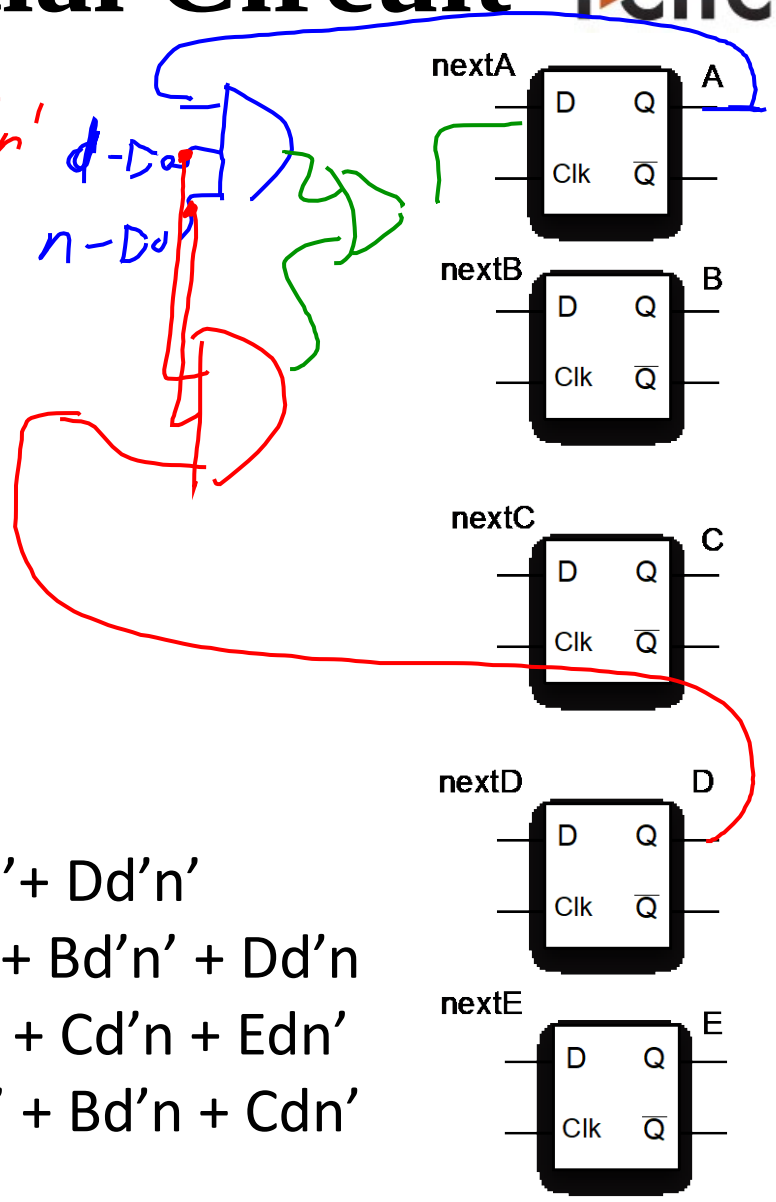
Step 3: Build Sequential Circuit State



Step 3: 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

nextA = Ad'n' + Dd'n'



next D?

A: nextD = Ad'n'+ Dd'n'

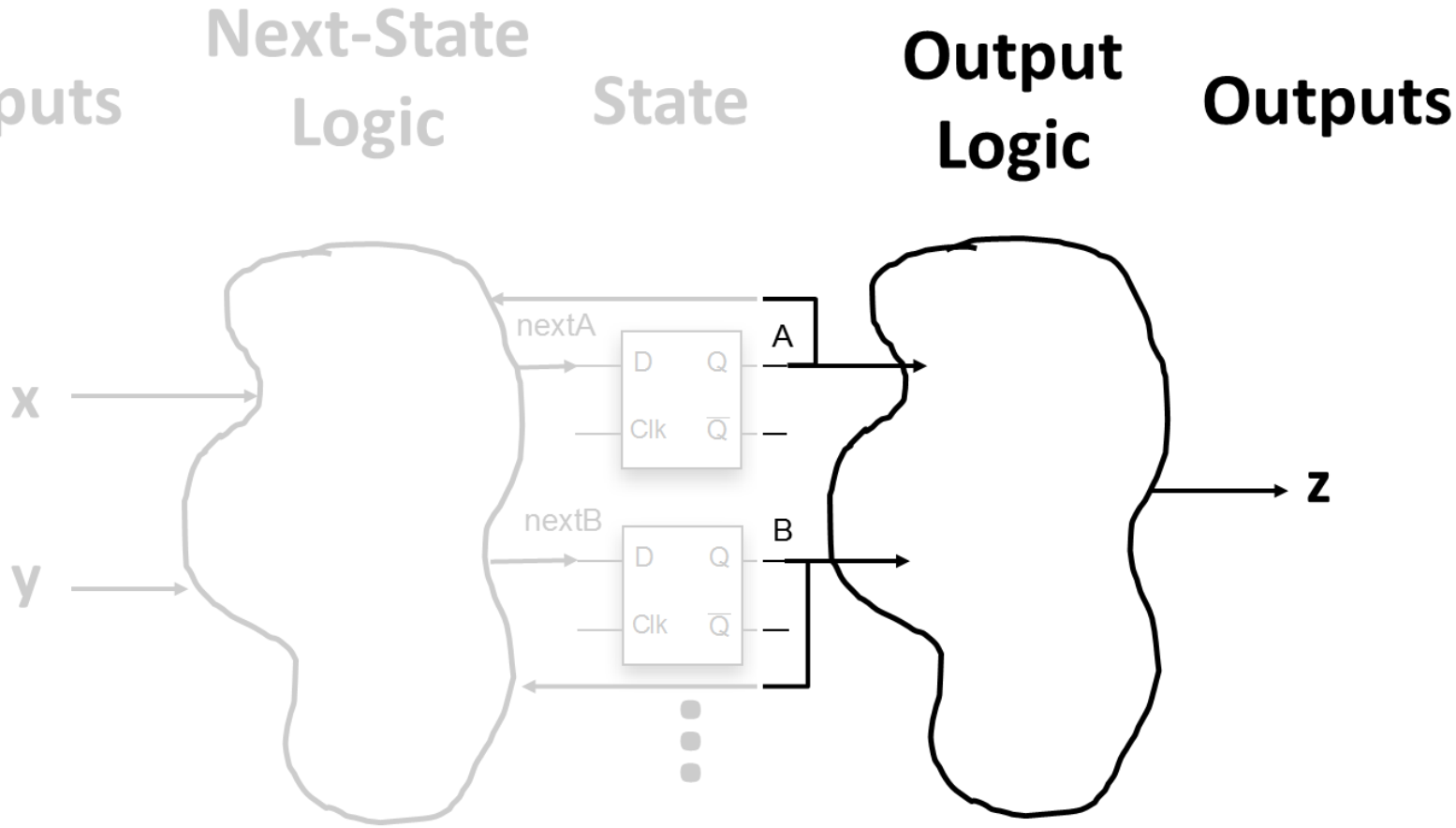
B: nextD= Ad'n + Bd'n' + Dd'n

C: nextD = Bdn' + Cd'n + Edn'

D: nextD= Ad'n' + Bd'n + Cdn'

E: nextD =1

Step 3: Build Sequential Circuit State



Step 3: Build Sequential Circuit

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’

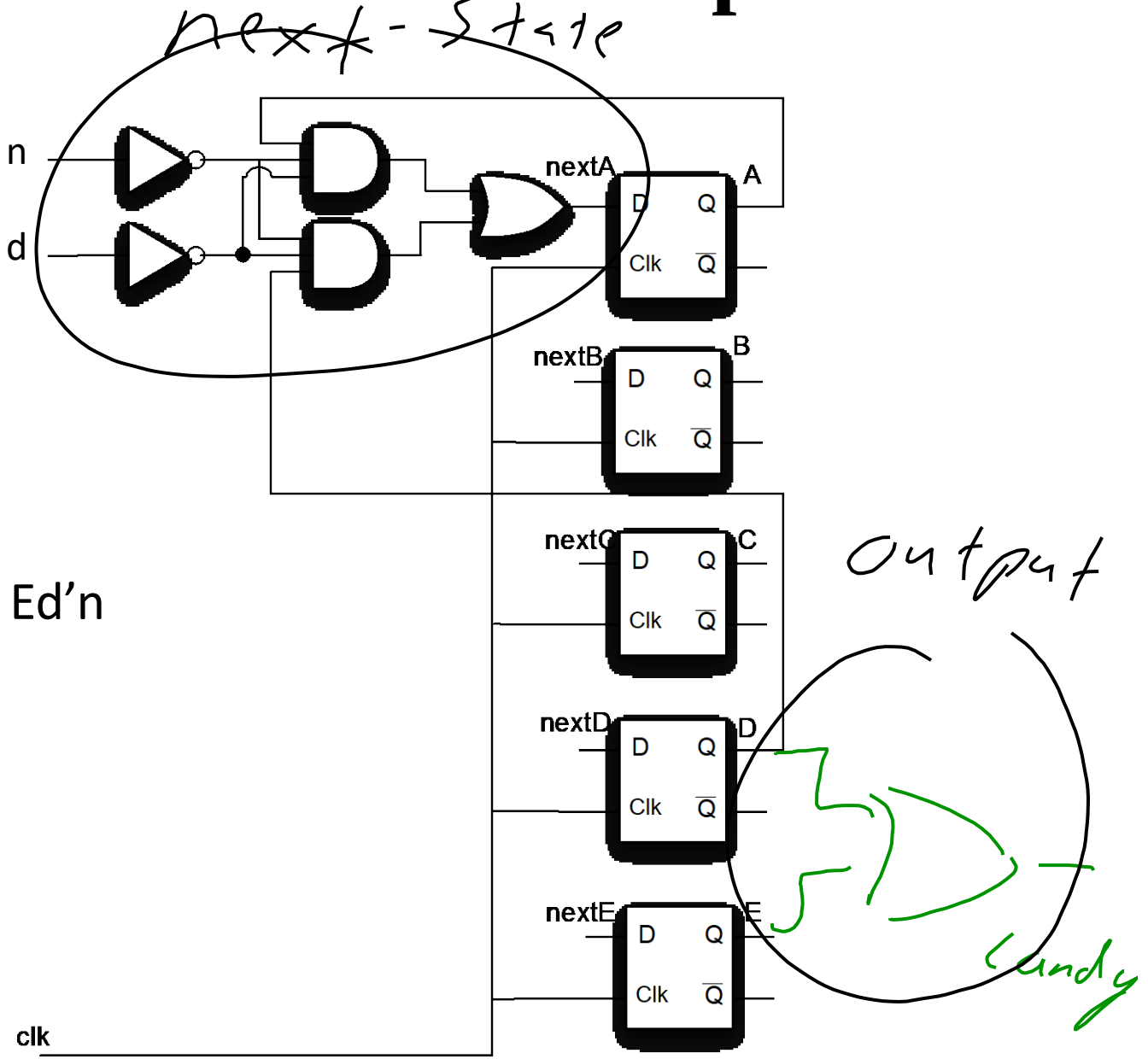
nextC = And’ + Bd’n + Cd’n’ + Ddn’ + Ed’n

nextD = Bdn’ + Cd’n + Edn’

nextE = Cdn’

Output: Candy = D + E

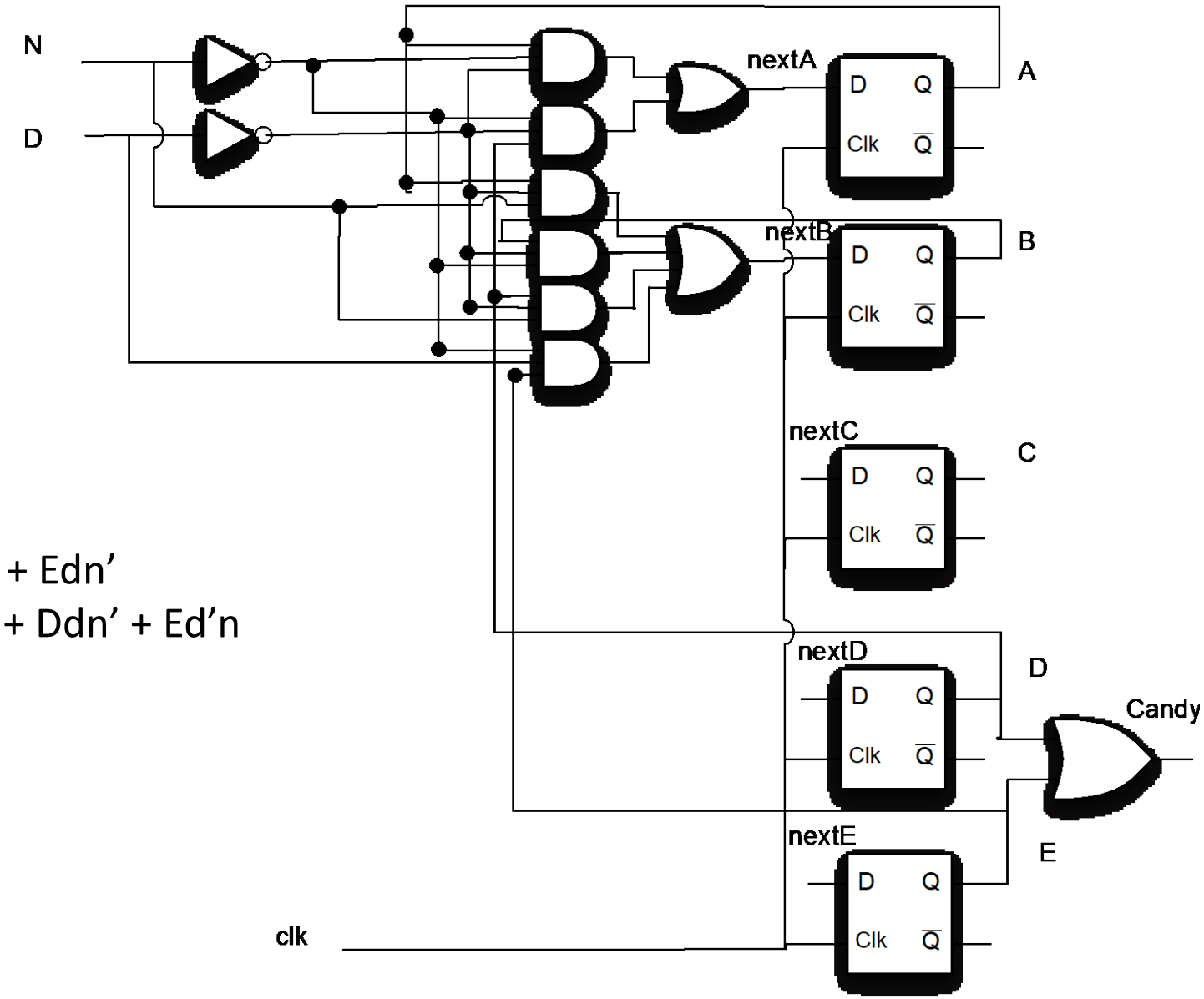
Step 3: Sequential circuit with D flip-flops



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

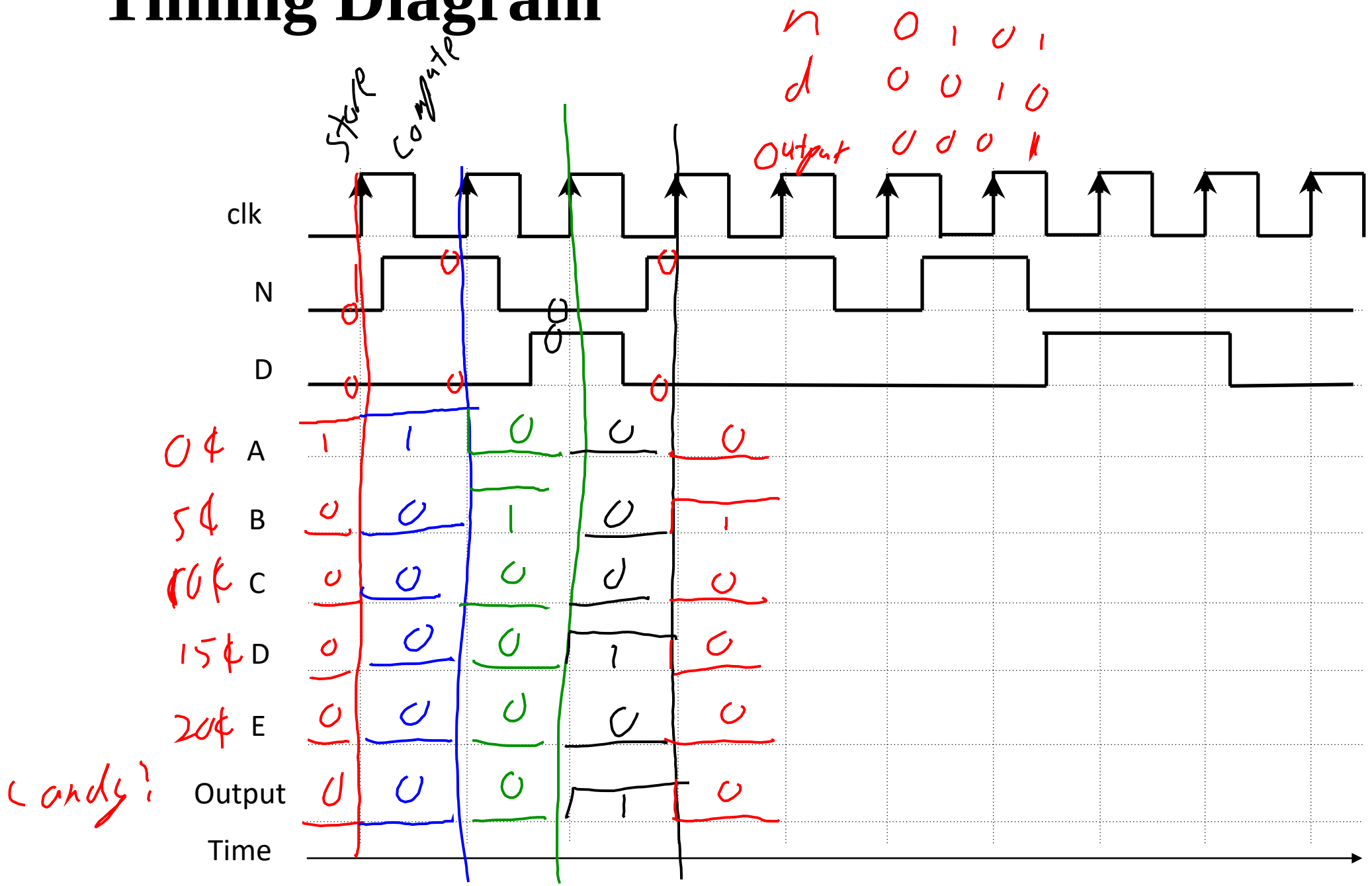
Step 3: Sequential circuit with D flip-flops



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