JOINT INSTITUTE
交大密西根学院

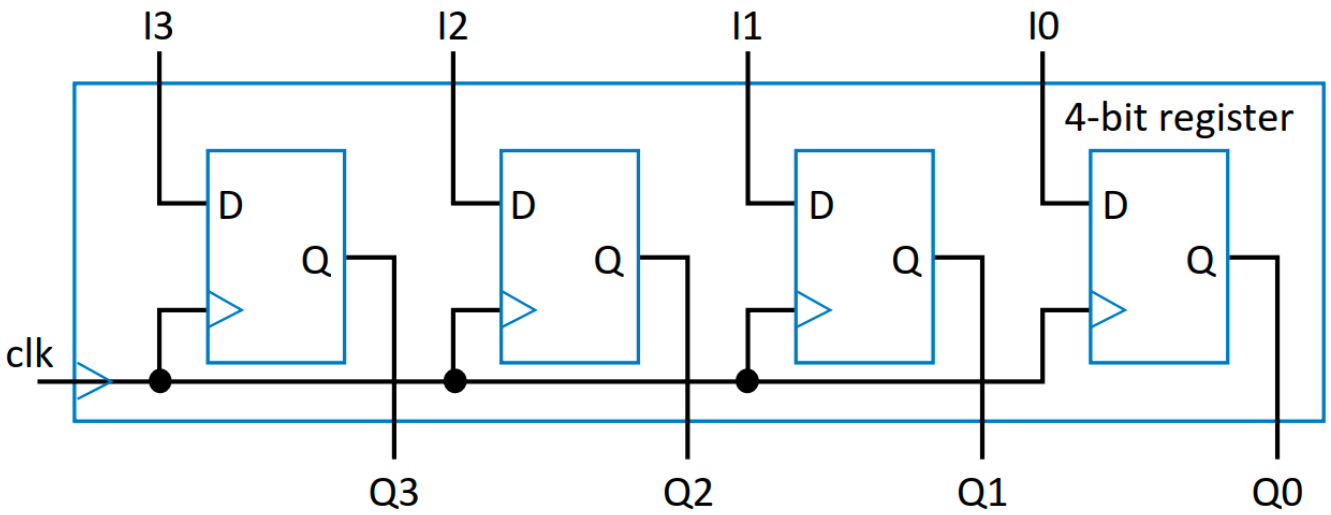
ECE2700J SU24 RC4

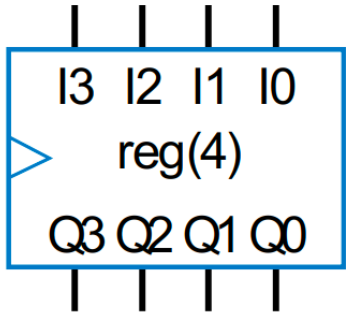
Register, Shifter & FSM

Wenyue Li
7/8/2024

Register & Shifter

Basic register

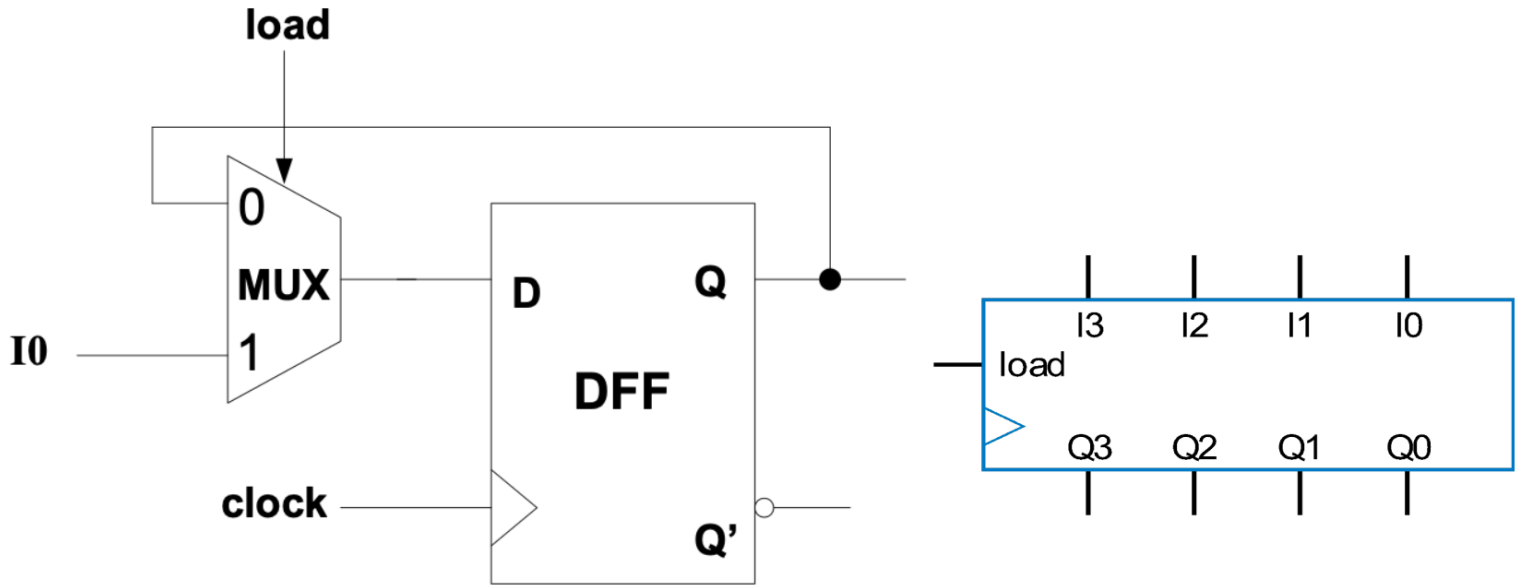




- Can store data, very common in datapath
- Basic register: Loaded every cycle
- Basic Register & DFF

Register with parallel load

- Add 2x1 mux to each flip-flop
- Register's load input selects mux input to pass
  - Either existing flip-flop value, or new value to load



Synchronous active high Load

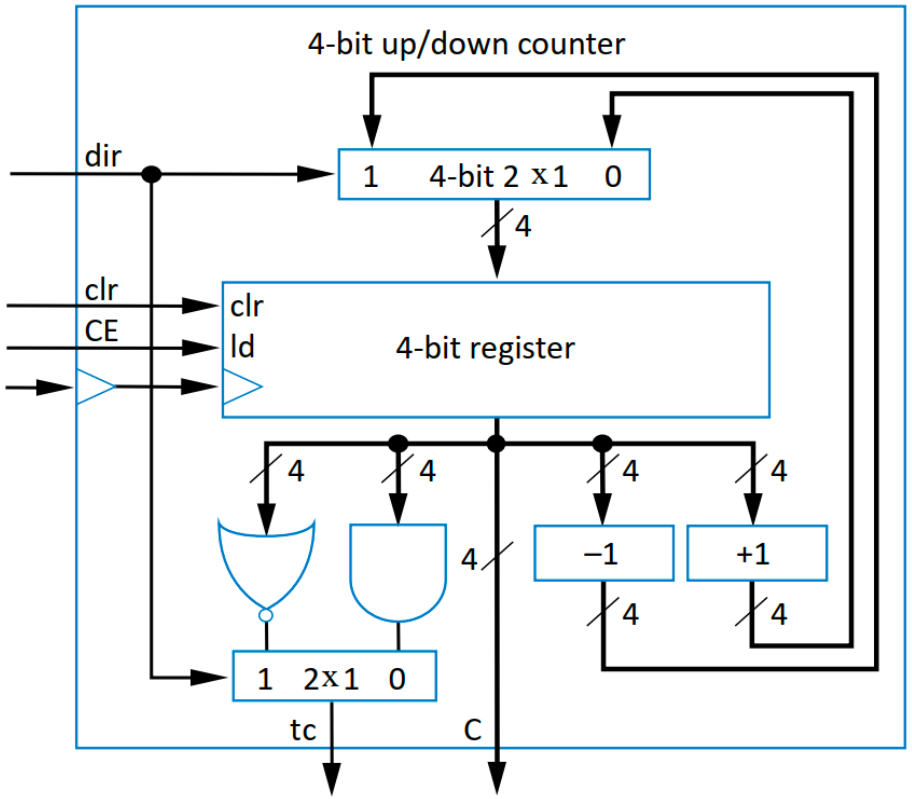
Register with parallel load

Exercise: use a 4-bit register with parallel load to make a 4-bit binary counter. There is one control input load. When load is 1, the counter counts, otherwise the data remains. (Hint: you can use incrementer)

Application of register with parallel load

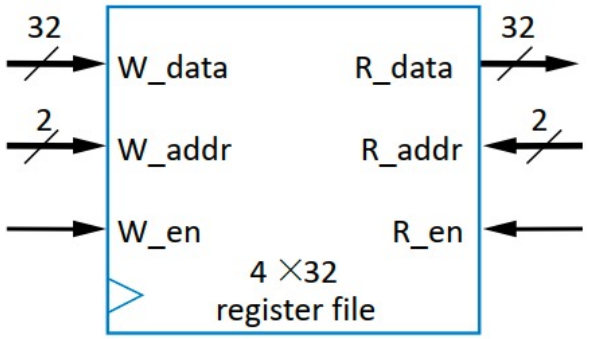
Up/Down-Counter

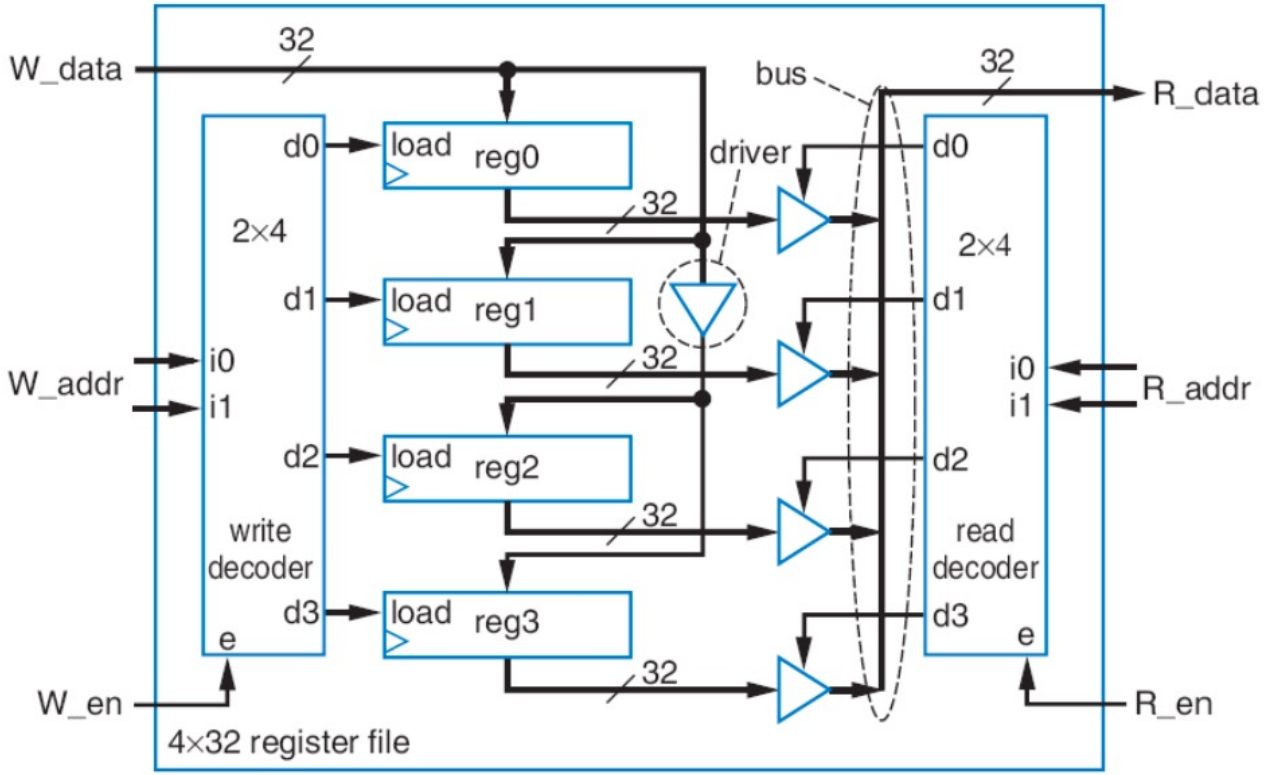
- Can count either up or down
  - Includes both incrementer and decrementer
  - Use dir input to select, using 2x1 MUX, dir=0 means count up
  - Likewise, dir selects appropriate terminal count value

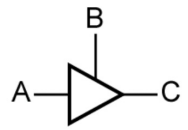


Register & Shifter

Register file





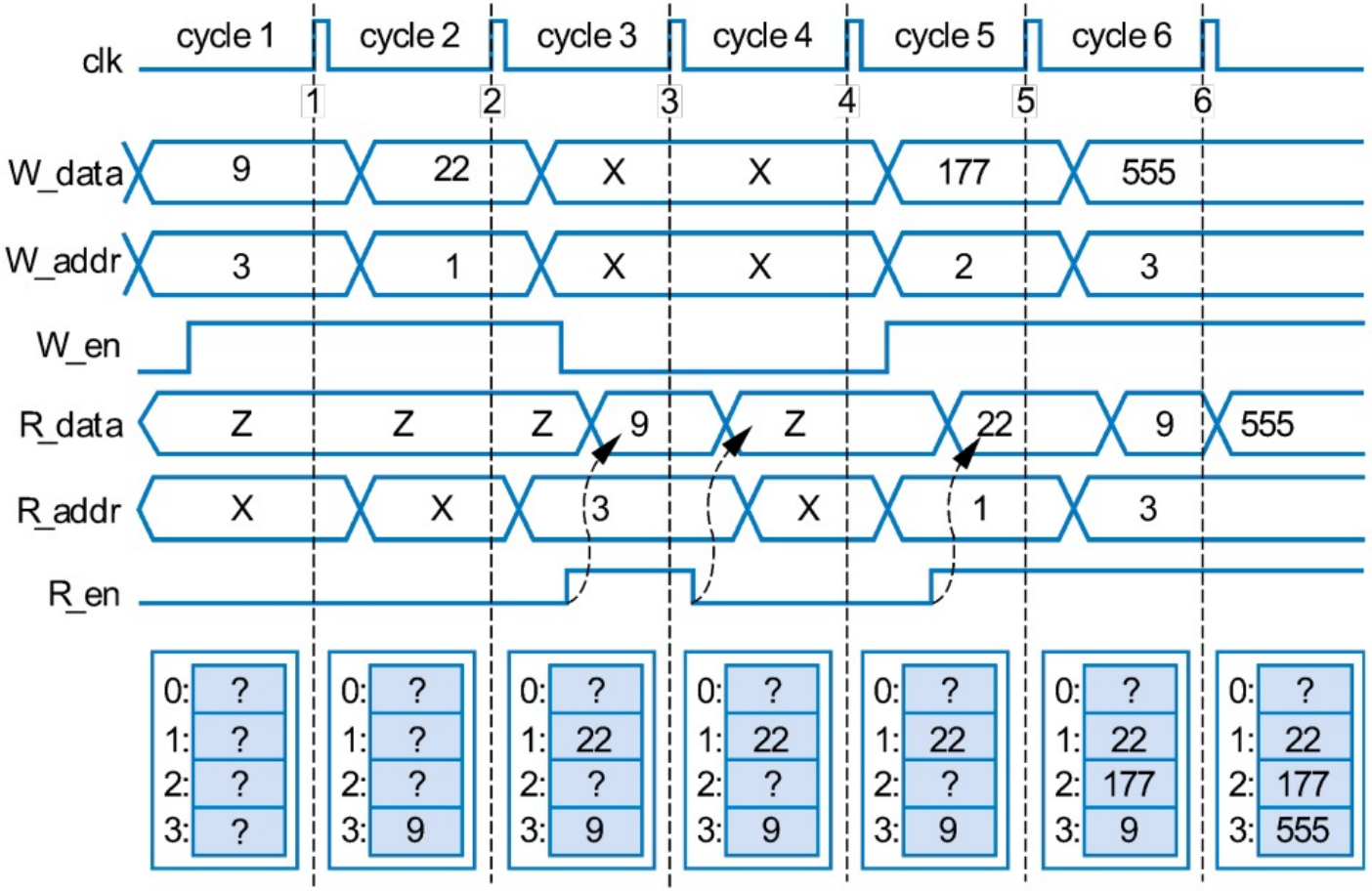


B	A	C
0	0	Z
0	1	Z
1	0	0
1	1	1

- A buffer is a one directional transmission logic device
  - somewhat like a NOT gate without complementing the binary value
  - amplify the driving capability of a signal
  - insert delay
  - Protect input from output

Register & Shifter

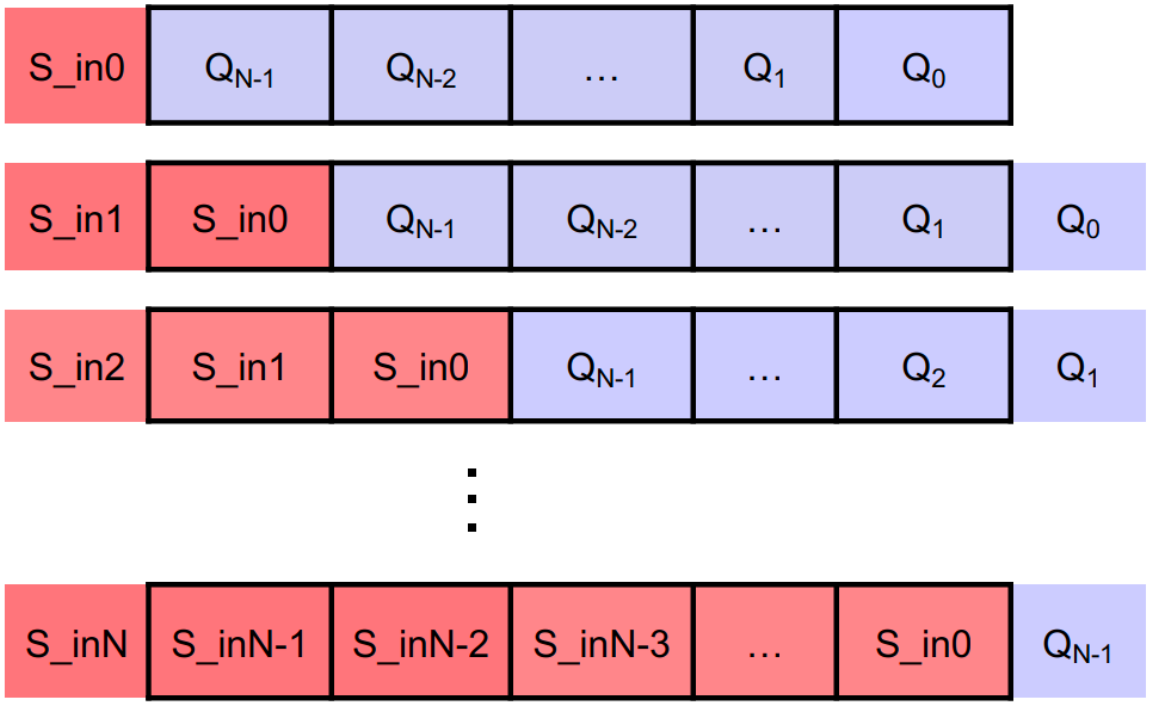
Application of register file



Register & Shifter

Shifter register

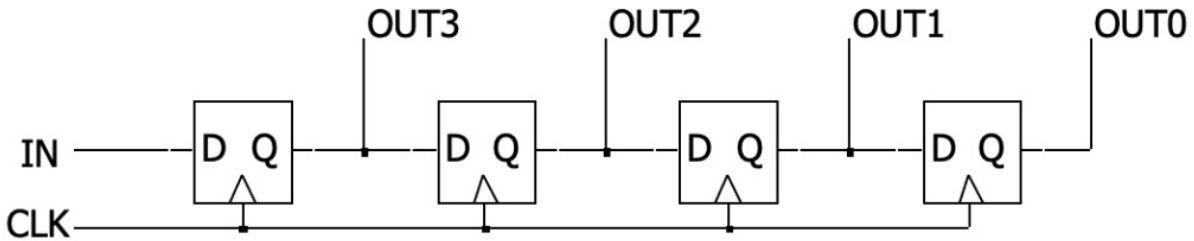
- One type of register
  - Stores binary data
  - Stored data can be shifted right (MSB >> LSM) or left (MSB << LSB)
  - Example: Shift right per clock edge



Register & Shifter

Shifter register

- Implementation:
  - Connect Q output of one flip flop to the D input of the next flip flop
  - 4-bit shift register

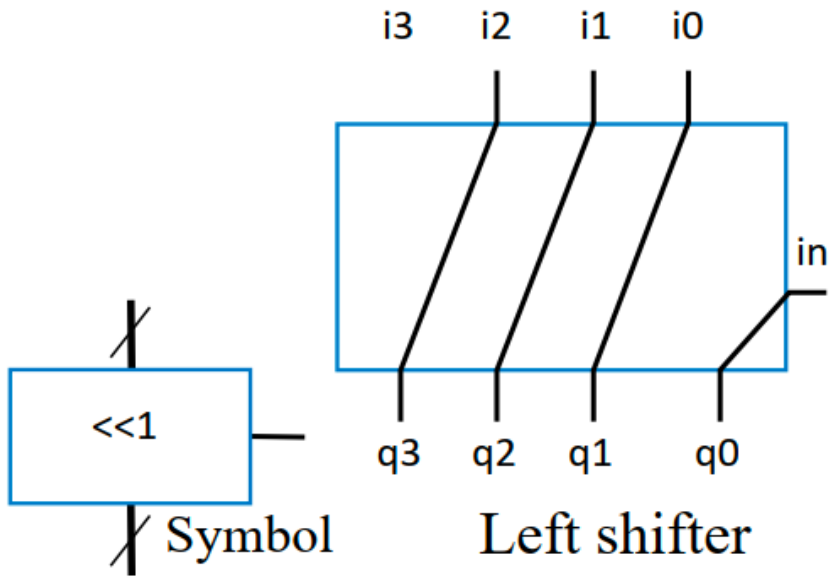


	IN	OUT(3:0)
Initial value:	0	0110
rising edge:	0	0011
rising edge:	0	0001
rising edge:	0	0000
rising edge:	1	1000
rising edge:	0	0100

Register & Shifter

Shifter

- Combinational Datapath component
- Shifting (e.g., left shifting 0011 yields 0110) useful for:
  - Manipulating bits
  - Shift left once is same as multiplying by 2 (0011 (3) becomes 0110 (6))
  - Shift right once same as dividing by 2



Register & Shifter

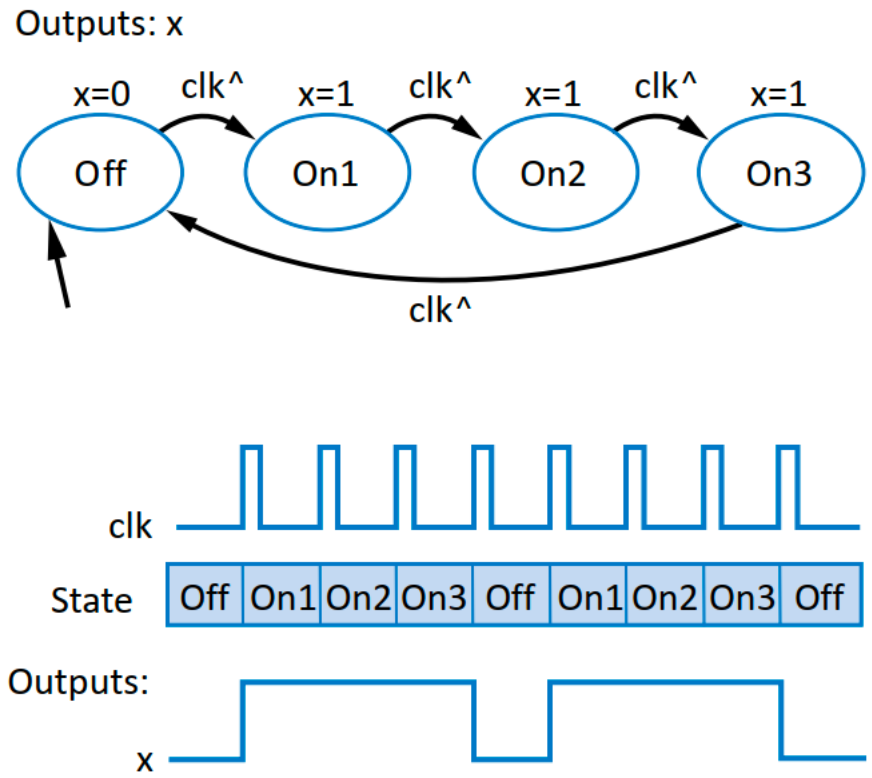
Shifter

[EX1] Create a circuit that approximately computes $P = 0.875 \times Q$ using only shifters and adders. **Roll down** the result to integers. Use wider internal components and wires as necessary to prevent internal overflow.

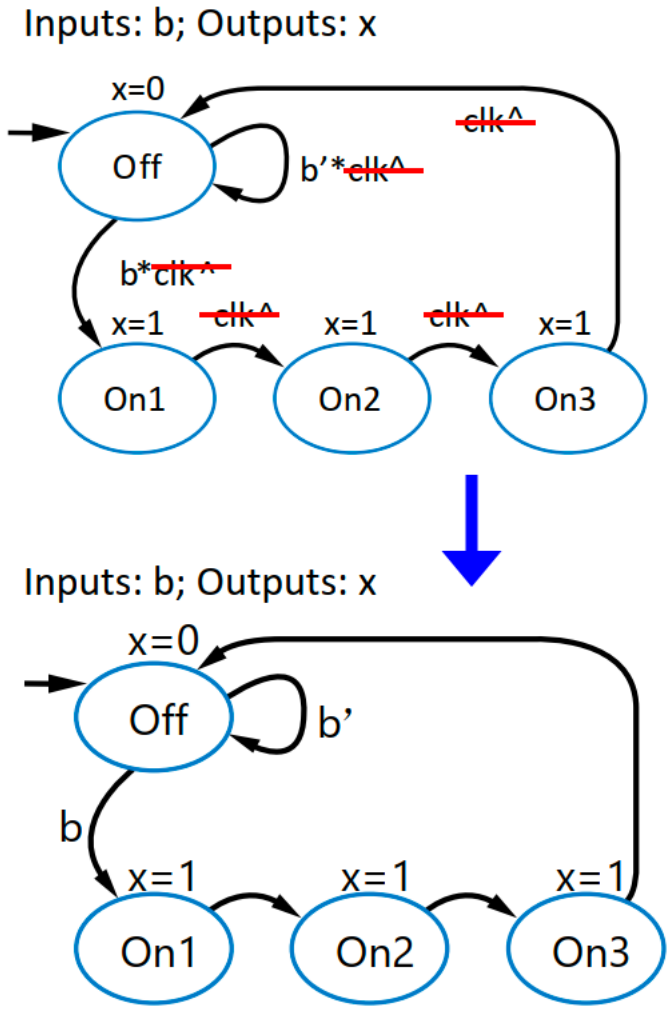
FSM

Definition

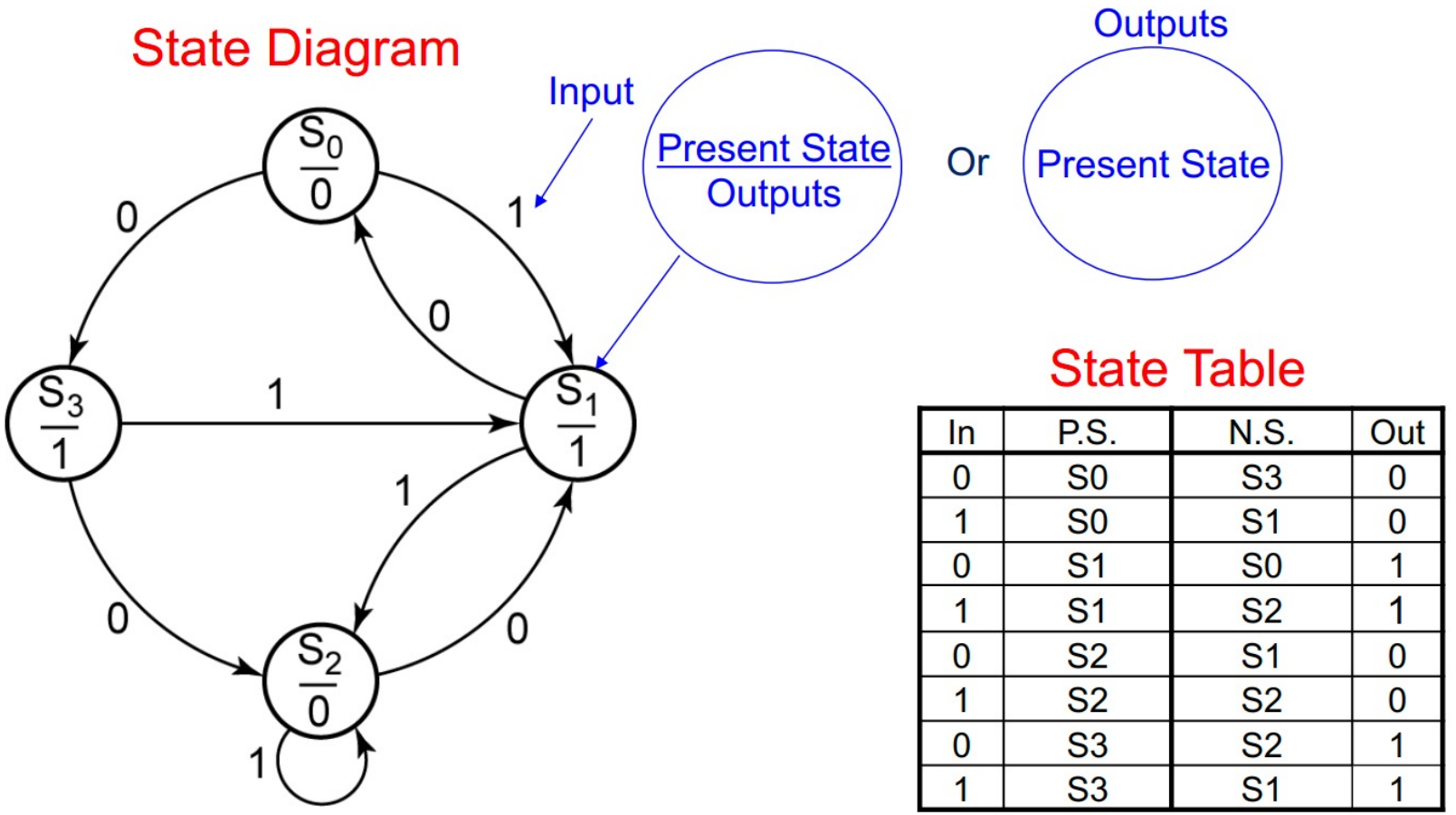
- A way to **design** a sequential circuit by **describing desired behavior** of the sequential circuit
- Consists of a set of states, transitions between states, and maybe inputs and outputs
  - ***present state***: currently happening
  - ***next state***: next to happen



FSM with input & Synchronous FSM



State Diagram and State Table

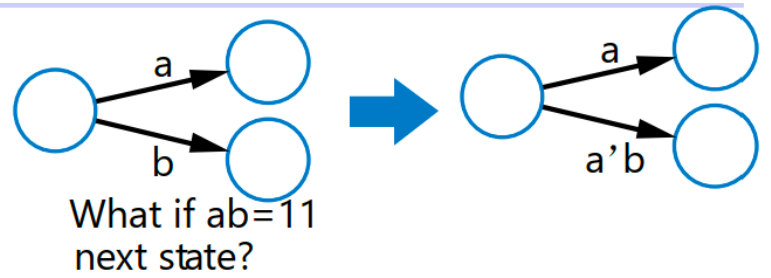


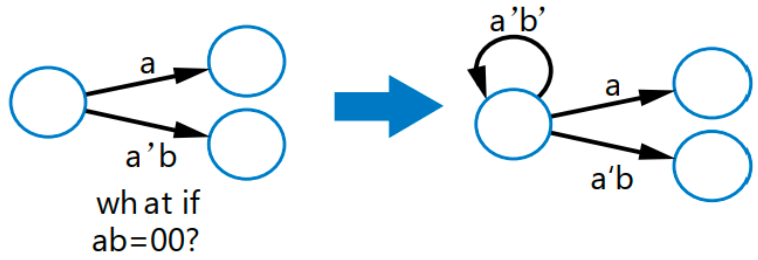
State Table

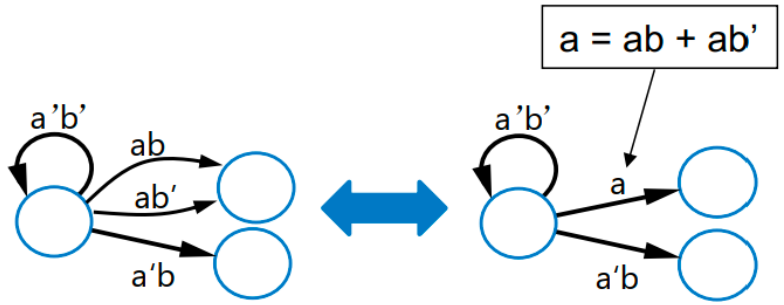
In	P.S.	N.S.	Out
0	S0	S3	0
1	S0	S1	0
0	S1	S0	1
1	S1	S2	1
0	S2	S1	0
1	S2	S2	0
0	S3	S2	1
1	S3	S1	1

Common State Transition Property

- *Only* one condition should be true, among all transitions leaving a state
- *One* condition must be true
  - For any input combination
- All conditions must be considered when leaving a state

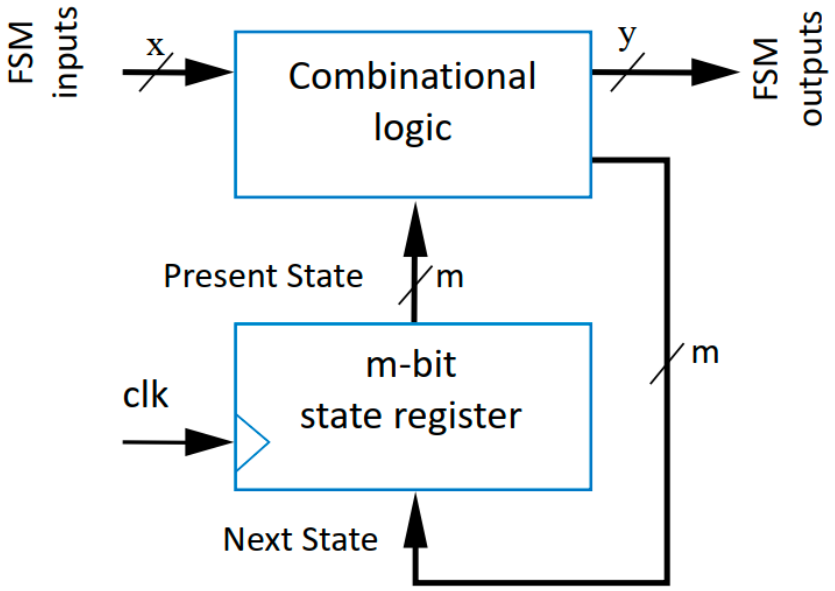






FSM

Standard FSM



- Five step FSM design process

	Step	Description
Step 1	*Capture the FSM*	Create an FSM that describes the desired behavior of the controller.
Step 2	*Create the architecture*	Create the standard architecture by using a state register of appropriate width, and combinational logic with inputs being the state register bits and the FSM inputs and outputs being the next state bits and the FSM outputs.
Step 3	*Encode the states*	Assign a unique binary number to each state. Each binary number representing a state is known as an ***encoding.*** Any encoding will do as long as each state has a unique encoding.
Step 4	*Create the state table*	Create a truth table for the combinational logic such that the logic will generate the correct FSM outputs and next state signals. Ordering the inputs with state bits first makes this truth table describe the state behavior, so the table is a state table.
Step 5	*Implement the combinational logic*	Implement the combinational logic using any method.

FSM Design Example

- Unlock door (u=1) only when buttons pressed in sequence:
  - start, then red, blue, green, red
- Input signals: *s, r, g, b*

if wrong button pressed, returns to “Wait”