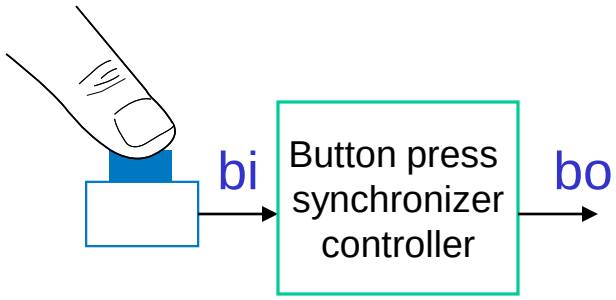
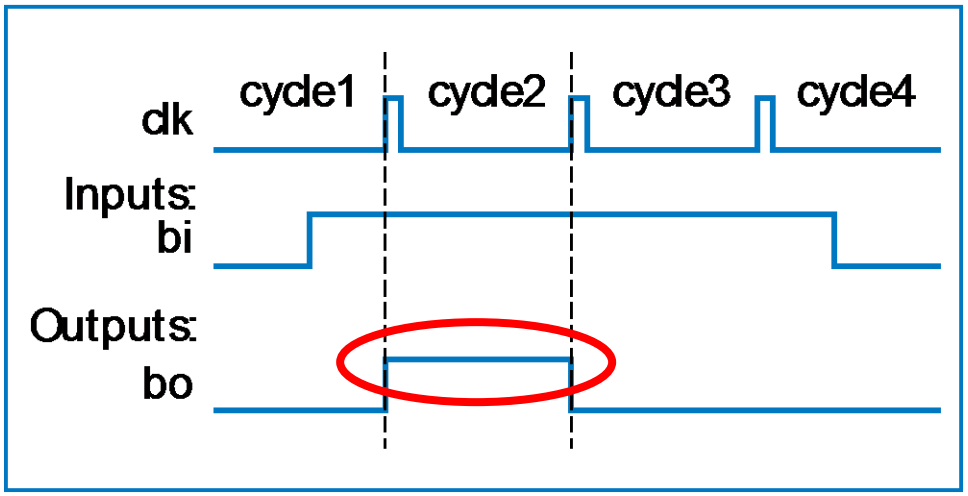
B38DB: Digital Design and Programming
Sequential Logic Design – Controller Examples

Mustafa Suphi Erden

Heriot-Watt University
School of Engineering & Physical Sciences
Electrical, Electronic and Computer Engineering
Room: EM 2.01
Phone: 0131-4514159
E-mail: m.s.erden@hw.ac.uk

Controller Example: Button Press Synchronizer (1/2)

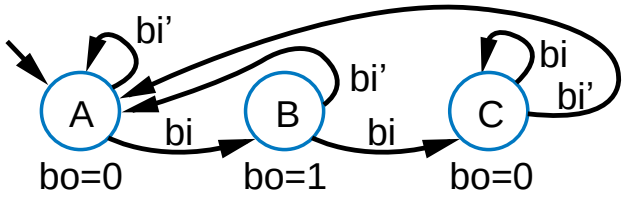




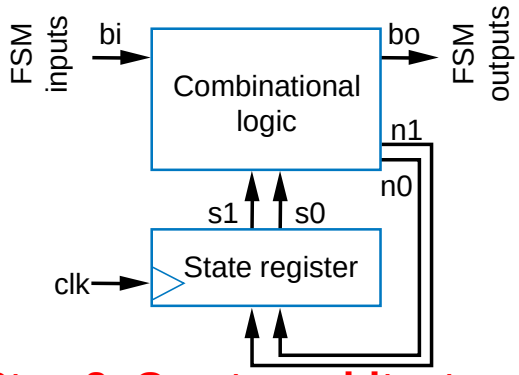
- We want a simple sequential circuit that converts “button-press” to a single cycle duration, regardless of the length of the time that the button is actually pressed
  - We assumed such an ideal button-press signal in earlier examples, like the button in the laser timer controller

Controller Example: Button Press Synchronizer (2/2)

FSM inputs: bi; FSM outputs: bo

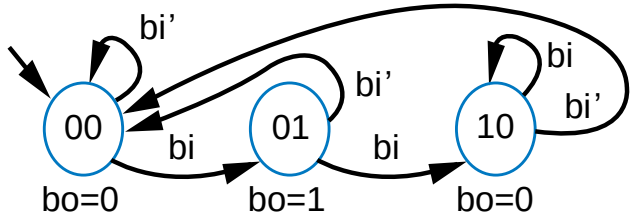


Step 1: FSM



Step 2: Create architecture

FSM inputs: bi; FSM outputs: bo

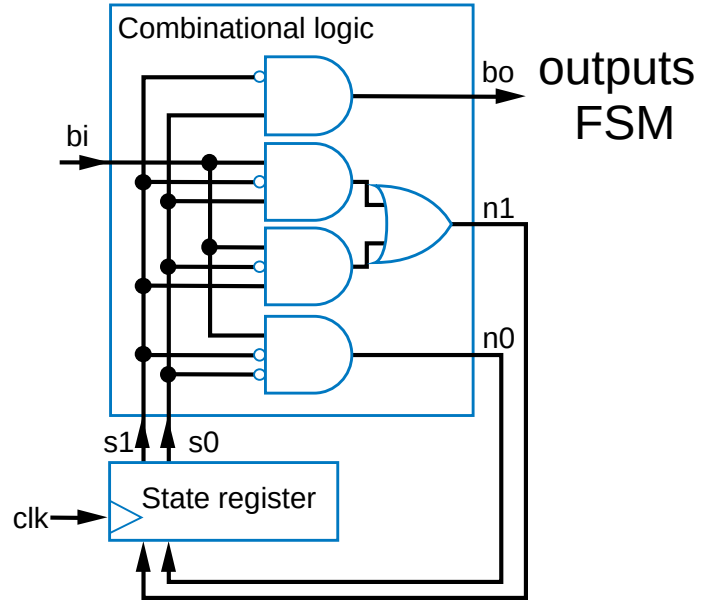


Step 3: Encode states

	Combinational logic Inputs			Outputs		
	s1	s0	bi	n1	n0	bo
A	0	0	0	0	0	0
	0	0	1	0	1	0
B	0	1	0	0	0	1
	0	1	1	1	0	1
C	1	0	0	0	0	0
	1	0	1	1	0	0
unused	1	1	0	0	0	0
	1	1	1	0	0	0

Step 4: State table

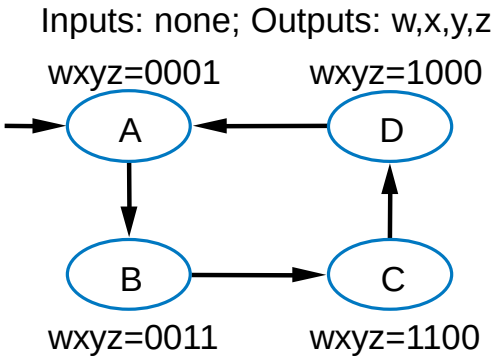
$n1 = s1's0bi + s1s0bi$
$n0 = s1's0'bi$
$bo = s1's0bi' + s1's0bi = s1s0$



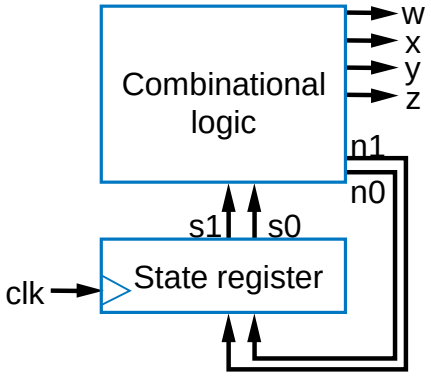
Step 5: Create combinational circuit

Controller Example: Sequence Generator

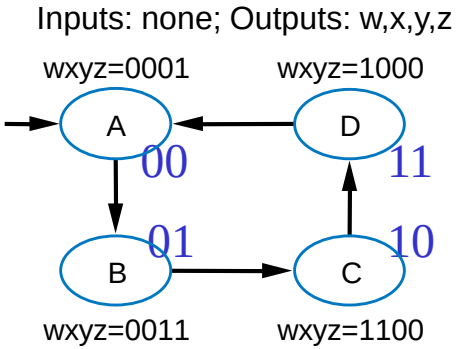
- We want to generate the sequence 0001, 0011, 1100, 1000, (repeats)
  - One binary number for each clock cycle
  - Common examples: to create a pattern with 4 lights, or to control the magnets of a “stepper motor”



Step 1: Create FSM



Step 2: Create architecture

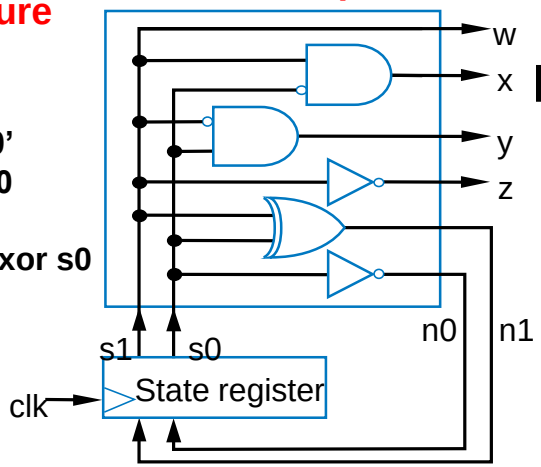


Step 3: Encode states

	Inputs		Outputs					
	s1	s0	w	x	y	z	n1	n0
A	0	0	0	0	0	1	0	1
B	0	1	0	0	1	1	1	0
C	1	0	1	1	0	0	1	1
D	1	1	1	0	0	0	0	0

Step 4: Create state table

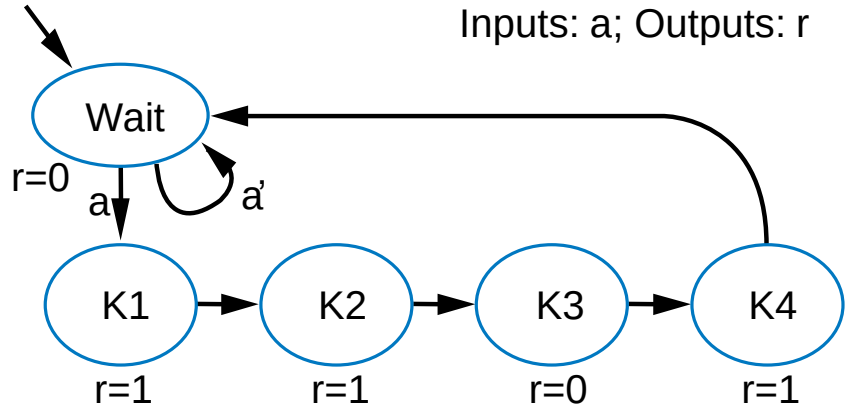
w = s1
x = s1s0’
y = s1’s0
z = s1’
n1 = s1 xor s0
n0 = s0’

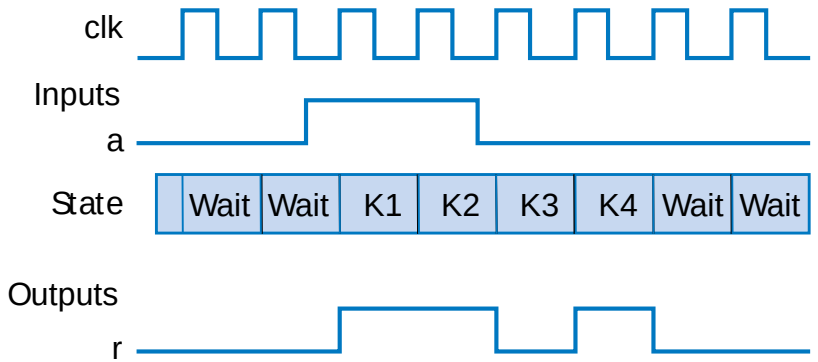


Step 5: Create combinational circuit

Controller Example: Secure Car Key (1/2)

- Many new car keys include a tiny computer chip
  - When the car starts, the car's computer (under engine hood) requests the identifier from key
  - The key transmits the identifier
    - If not, the computer shuts off the car
- FSM
  - Wait until the computer requests ID (a=1)
  - Transmit the ID (in this case, 1101)

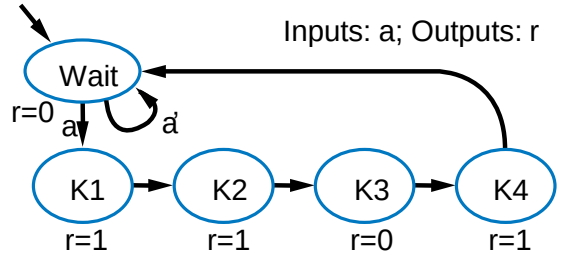




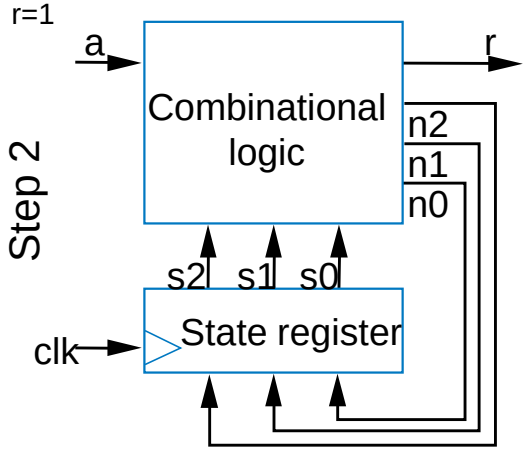
Timing diagrams show states and output values for different input waveforms

Controller Example: Secure Car Key (2/2)

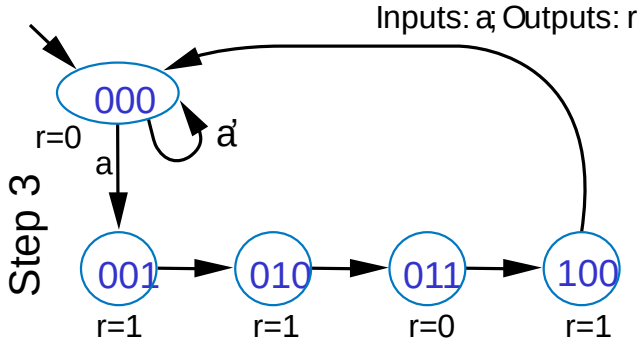
Step 1



Step 2



Step 3



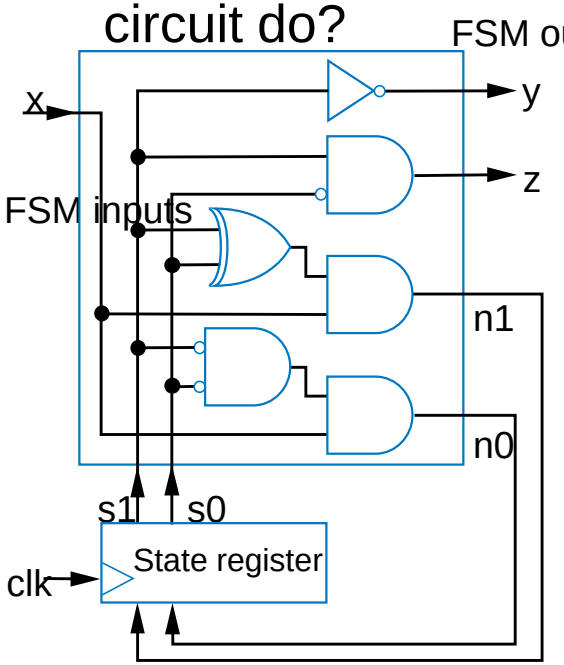
We'll omit Step 5

	Inputs				Outputs			
	s2	s1	s0	a	r	n2	n1	n0
Wait	0	0	0	0	0	0	0	0
	0	0	0	1	0	0	0	1
K1	0	0	1	0	1	0	1	0
	0	0	1	1	1	0	1	0
K2	0	1	0	0	1	0	1	1
	0	1	0	1	1	0	1	1
K3	0	1	1	0	0	1	0	0
	0	1	1	1	0	1	0	0
K4	1	0	0	0	1	0	0	0
	1	0	0	1	1	0	0	0
Unused	1	0	1	0	0	0	0	0
	1	0	1	1	0	0	0	0
	1	1	0	0	0	0	0	0
	1	1	0	1	0	0	0	0
	1	1	1	0	0	0	0	0
	1	1	1	1	0	0	0	0

Step 4

Example: Seq. Circuit to FSM (Reverse Engineering)

What does this circuit do?

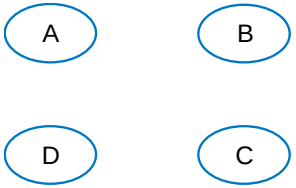


Work backwards

y=s1'
z = s1s0'
n1=(s1 xor s0)x
n0=(s1'*s0')x

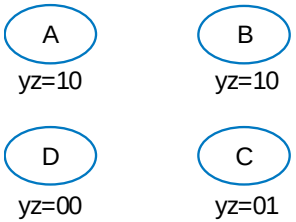
	Inputs			Outputs			
	s1	s0	x	n1	n0	y	z
A	0	0	0	0	0	1	0
	0	0	1	0	1	1	0
B	0	1	0	0	0	1	0
	0	1	1	1	0	1	0
C	1	0	0	0	0	0	1
	1	0	1	1	0	0	1
D	1	1	0	0	0	0	0
	1	1	1	0	0	0	0

Pick any state names you want



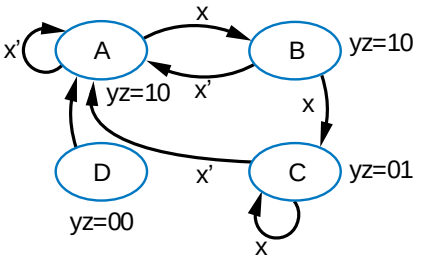
states

Outputs: y, z



states with outputs

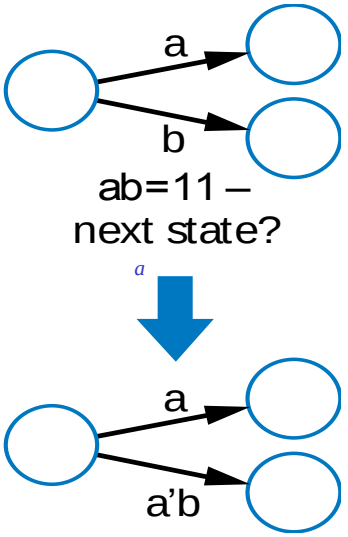
Inputs: x; Outputs: y, z



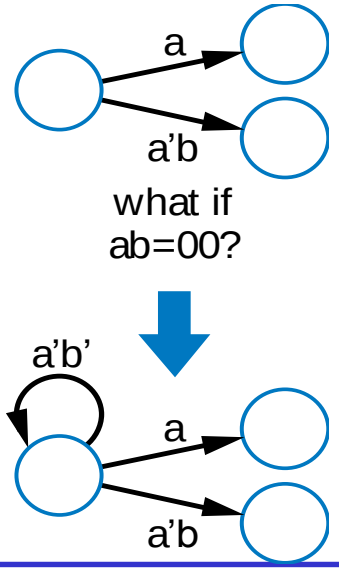
states with outputs and transitions

Common Pitfalls Regarding Transition Properties

- ***Non-exclusive transitions**: Only* one condition should be true for all transitions leaving a state



- ***Incomplete transitions**: One* condition must be true for all transitions leaving a state



Verifying Correct Transition Properties

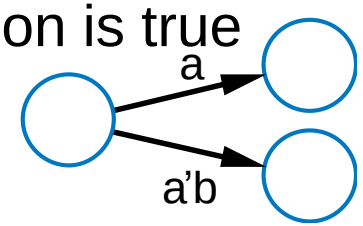
- We can verify by using Boolean algebra
  - Only one condition is true: AND of each condition pair (for transitions leaving a state) should be equal to 0

    → proves that more than one output can never be simultaneously true

  - At least one condition is true: OR of all conditions of the transitions leaving a state should be equal to1

    → proves that at least one condition is true

  - Example



Answer:

a * a'b
= (a * a') * b
= 0 * b
= 0
OK!

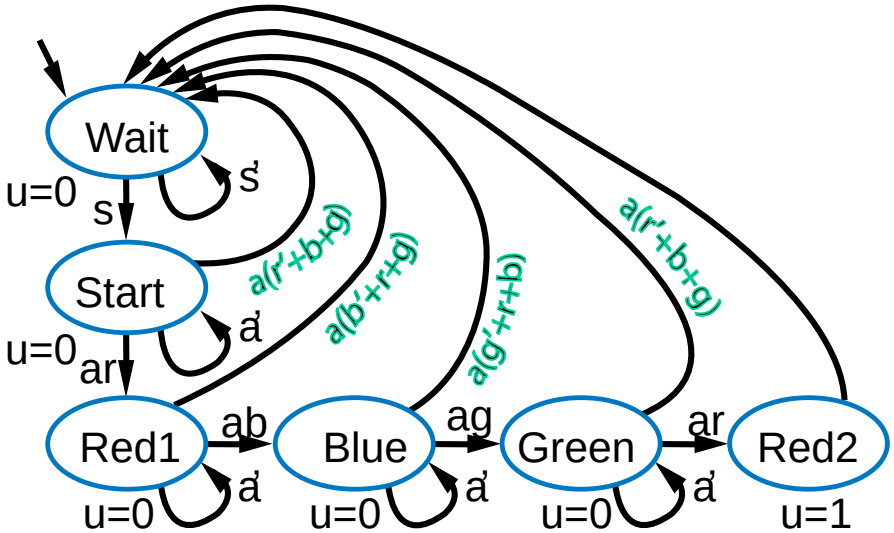
a + a'b
= a*(1+b) + a'b
= a + ab + a'b
= a + (a+a')b
= **a + b**
Fails! Might not be 1 (i.e., a=0, b=0)

Q: Prove whether:

* Only one condition is true (AND of each pair is always 0)
* There is a condition which is true (OR of all transitions is always 1)

Evidence that Pitfall is Common

- Recall the code detector FSM
  - We "fixed" a problem with the transition conditions
  - Do the transitions satisfy the two requirements?
    - Consider transitions of state *Start*, and the "only one true" property



ar * a'
= (a*a')r
= 0

a' * a(r'+b+g)
= 0*r
= 0

ar * a(r'+b+g)
= (a'*a)*(r'+b+g) = 0*(r'+b+g)
= (a*a)*r*(r'+b+g) = a*r*(r'+b+g)
= arr'+arb+arg
= 0 + arb+arg
= arb + arg
= ar(b+g)

Fails! Means that two of Start's transitions could be true at the same time

Intuitively: press the red and blue buttons at the same time: conditions ar, and a(r'+b+g) will be both true. Which one should be taken?

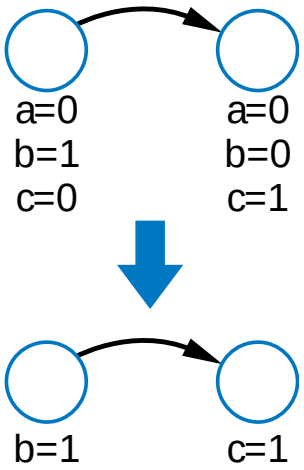
Q: How to solve?

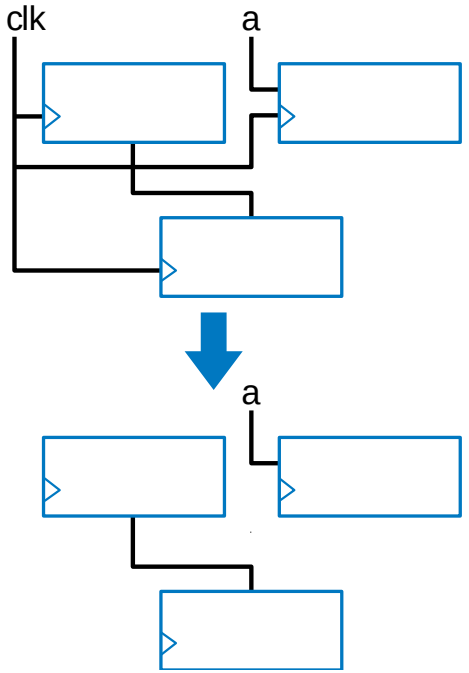
A: ar should be arb'g'
(likewise for ab, ag, ar)

Note: As evidence for that the pitfall is common, we admit that the mistake was not intentional.
A reviewer of the book caught it.

Simplifying Notations

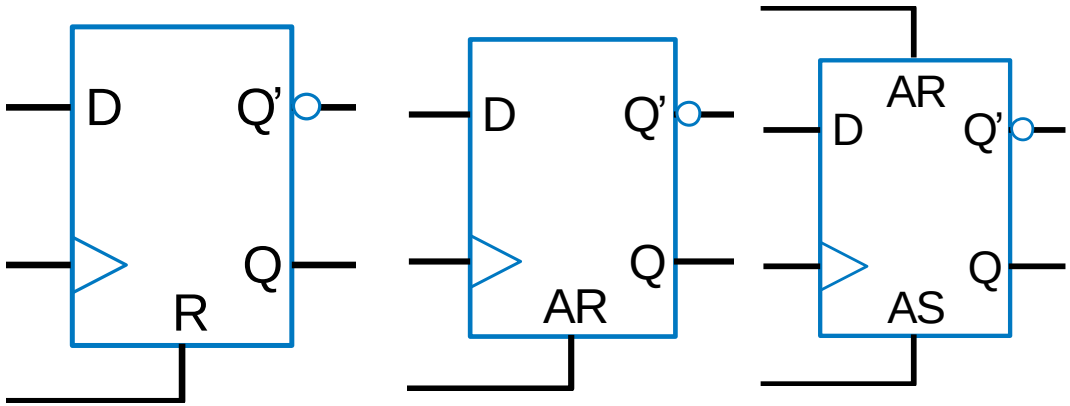
- FSMs
  - Assume any unassigned output implicitly assigned to 0
- Sequential circuits
  - Assume unconnected clock inputs to be connected to the same external clock

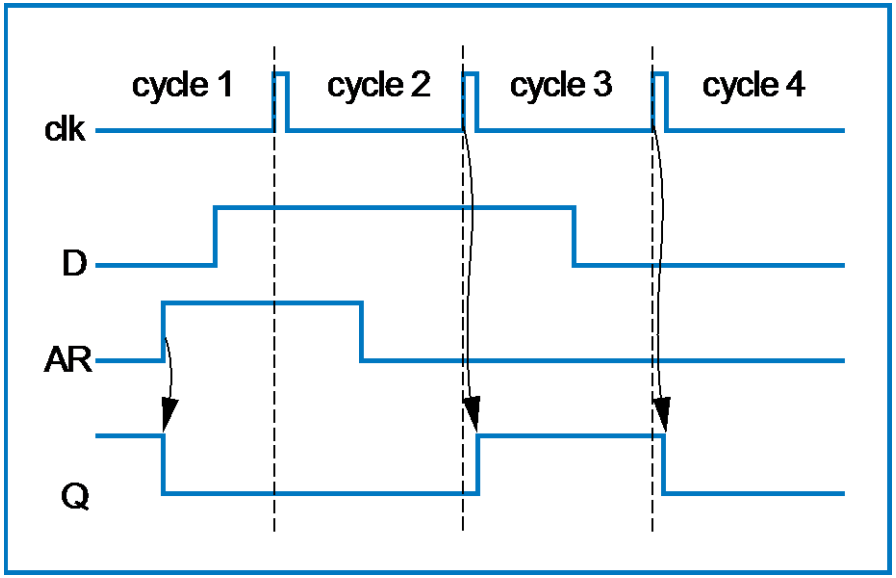




Flip-Flop Set and Reset Inputs

- Some flip-flops have additional inputs
  - Synchronous reset: clears Q to 0 on the next clock edge
  - Synchronous set: sets Q to 1 on the next clock edge
  - Asynchronous reset: clears Q to 0 immediately (not dependent on the clock edge)
    - Example timing diagram shown
  - Asynchronous set: sets Q to 1 immediately





Initial State of a Controller

- All our FSMs had initial state
  - But our sequential circuit designs did not
  - Can we accomplish having an initial state by using flip-flops with reset/set inputs
    - Shown circuit initializes the flip-flops to the **state 01**