ECE 274 Digital Logic

Sequential Logic Design – Sequential Logic Design Process

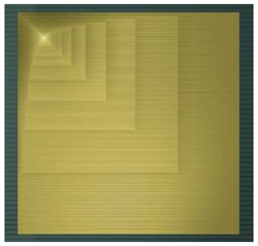
Digital Design 3.4 – 3.5

THE UNIVERSITY OF ARIZONA
TUCSON ARIZONA

Digital Design

Chapter 3: Sequential Logic Design -- Controllers

Slides to accompany the textbook *Digital Design*, First Edition,
by Frank Vahid, John Wiley and Sons Publishers, 2007.
http://www.ddvahid.com



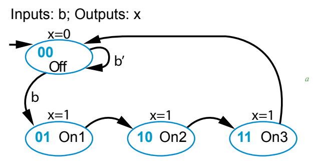
Sequential Logic Design
Controller Design

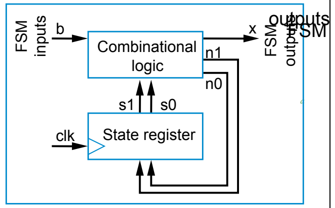
- Five step controller design process

	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.

Sequential Logic Design
Controller Design: Laser Timer Example

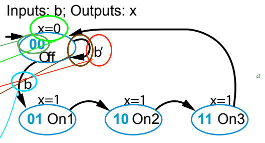
- Step 1: Capture the FSM
  - Already done
- Step 2: Create architecture
  - 2-bit state register (for 4 states)
  - Input b, output x
  - Next state signals n1, n0
- Step 3: Encode the states
  - Any encoding with each state unique will work



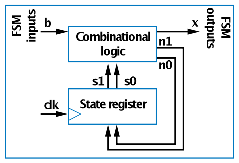


Sequential Logic Design
Controller Design: Laser Timer Example (cont)

- Step 4: Create state table



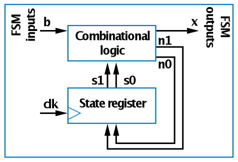
	Inputs			Outputs		
	s1	s0	b	x	n1	n0
Off	0	0	0	0	0	0
	0	0	1	0	0	1
On1	0	1	0	1	1	0
	0	1	1	1	1	0
On2	1	0	0	1	1	1
	1	0	1	1	1	1
On3	1	1	0	1	0	0
	1	1	1	1	0	0



5

Sequential Logic Design
Controller Design: Laser Timer Example (cont)

- Step 5: Implement combinational logic



	Inputs			Outputs		
	s1	s0	b	x	n1	n0
Off	0	0	0	0	0	0
	0	0	1	0	0	1
On1	0	1	0	1	1	0
	0	1	1	1	1	0
On2	1	0	0	1	1	1
	1	0	1	1	1	1
On3	1	1	0	1	0	0
	1	1	1	1	0	0

x = s1 + s0 (note from the table that x=1 if s1 = 1 or s0 = 1)

n1 = s1's0b' + s1's0b + s1s0'b' + s1s0'b

n1 = s1's0 + s1s0'

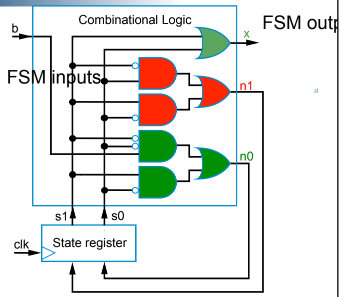
n0 = s1's0'b + s1s0'b' + s1s0'b

n0 = s1's0'b + s1s0'

6

Sequential Logic Design
Controller Design: Laser Timer Example (cont)

- Step 5: Implement combinational logic (cont)



	Inputs			Outputs		
	s1	s0	b	x	n1	n0
Off	0	0	0	0	0	0
	0	0	1	0	0	1
On1	0	1	0	1	1	0
	0	1	1	1	1	0
On2	1	0	0	1	1	1
	1	0	1	1	1	1
On3	1	1	0	1	0	0
	1	1	1	1	0	0

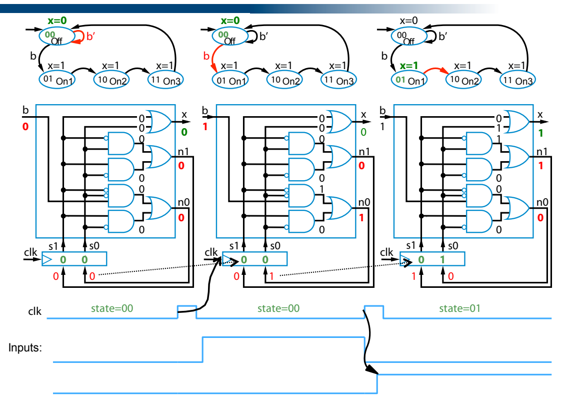
x = s1 + s0

n1 = s1's0 + s1s0'

n0 = s1's0'b + s1s0'

7

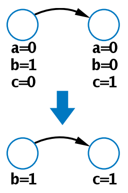
Sequential Logic Design
Understanding the Controller's Behavior



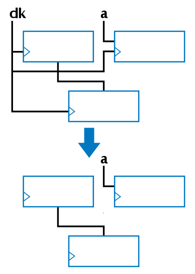
8

Sequential Logic Design
Simplifying Notations

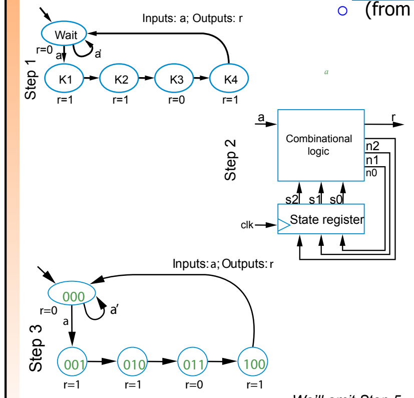
- FSMs
  - Assume unassigned output implicitly assigned 0



- Sequential circuits
  - Assume unconnected clock inputs connected to same external clock



Sequential Logic Design
Controller Example: Secure Car Key



We'll omit Step 5

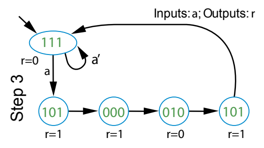
- (from earlier example)

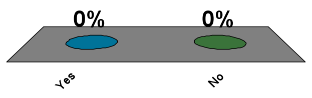
	Inputs				Outputs			
	s2	s1	s0	a	r	n2	n1	n0
Wait	0	0	0	0	0	0	0	0
	0	0	0	1	0	0	0	1
K1	0	0	1	0	1	0	1	0
	0	0	1	1	1	0	1	0
K2	0	1	0	0	1	0	1	1
	0	1	0	1	1	0	1	1
K3	0	1	1	0	0	1	0	0
	0	1	1	1	0	1	0	0
K4	1	0	0	0	1	0	0	0
	1	0	0	1	1	0	0	0
Unused	1	0	1	0	0	0	0	0
	1	0	1	1	0	0	0	0
	1	1	0	0	0	0	0	0
	1	1	0	1	0	0	0	0
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

Step 4

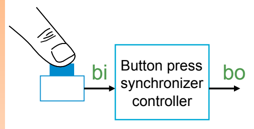
Sequential Logic Design
FSM Example: Code Detector

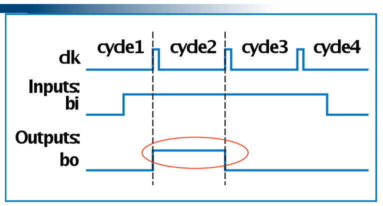
- If we changed the state encoding for the secure car key design to the following, would this affect the final output?
  1. Yes
  2. No





Sequential Logic Design
In Class Exercise: *Button Press Synchronizer*

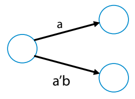


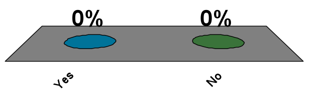


- Want simple sequential circuit that converts button press to single cycle duration, regardless of length of time that button actually pressed
  - We assumed such an ideal button press signal in earlier example, like the button in the laser timer controller

Sequential Logic Design
FSM Transitions

- Is the following FSM valid?
  1. Yes
  2. No

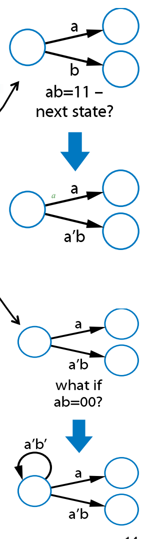




13

Sequential Logic Design
Common Pitfalls Regarding Transition Properties

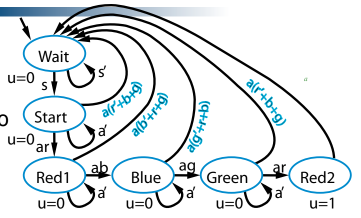
- *Only* one condition should be true
  - For all transitions leaving a state
  - Else, which one?
- *One* condition must be true
  - For all transitions leaving a state
  - Else, where go?
- Can verify using Boolean algebra
  - Only one condition true: AND of each condition pair (for transitions leaving a state) should equal 0 → proves pair can never simultaneously be true
  - One condition true: OR of all conditions of transitions leaving a state) should equal 1 → proves at least one condition must be true



14

Sequential Logic Design
Evidence that Pitfall is Common

- Recall code detector FSM
  - We “fixed” a problem with the transition conditions
  - Do the transitions obey the two required transition properties?
    - Consider transitions of state *Start*, and the “only one true” property



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

Fails! Means that two of Start’s transitions could be true

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

Q: How to solve?

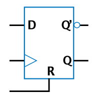
A: ar should be arb'g' (likewise for ab, ag, ar)

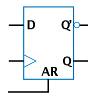
Note: As evidence the pitfall is common, we admit the mistake was not intentional. A reviewer of the book caught it.

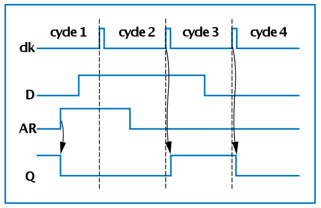
15

Sequential Logic Design
Flip-Flop Set and Reset Inputs

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



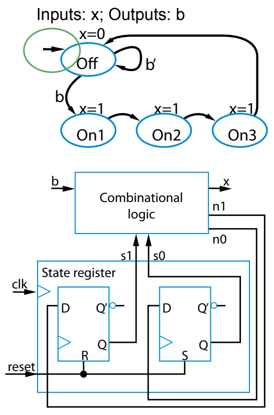




16

Sequential Logic Design
Initial State of a Controller

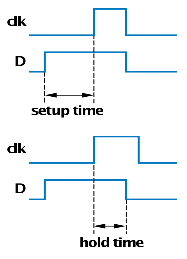
- All our FSMs had initial state
  - But our sequential circuit designs did not
  - Can accomplish using flip-flops with reset/set inputs
    - Shown circuit initializes flip-flops to 01
  - Circuits typically have power on reset circuitry to automatically reset circuit on power up

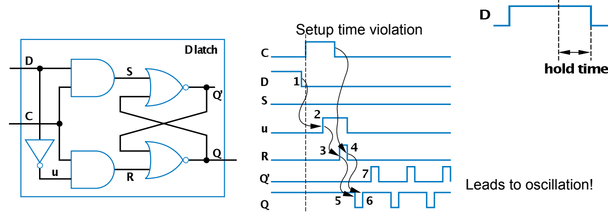


17

Sequential Logic Design
Non-Ideal Flip-Flop Behavior

- Can't change flip-flop input too close to clock edge
  - Setup time: time that D must be stable *before* edge
    - Else, stable value not present at internal latch
  - Hold time: time that D must be held stable *after* edge
    - Else, new value doesn't have time to loop around and stabilize in internal latch

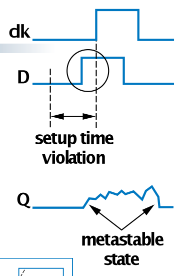


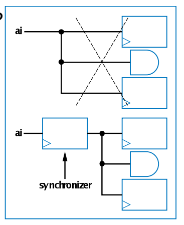


18

Sequential Logic Design
Metastability

- Violating setup/hold time can lead to bad situation known as ***metastable*** state
  - Metastable state: Any flip-flop state other than stable 1 or 0
    - Eventually settles to one or other, but we don't know which
  - For internal circuits, we can make sure observe setup time
  - But what if input comes from external (asynchronous) source, e.g., button press?
- Partial solution
  - Insert synchronizer flip-flop for asynchronous input
    - Special flip-flop with very small setup/hold time
  - Doesn't completely prevent metastability



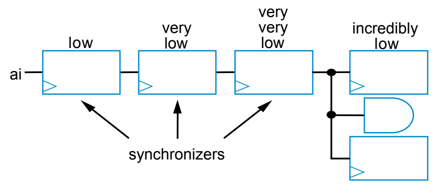


19

Sequential Logic Design
Metastability

- One flip-flop doesn't completely solve problem
- How about adding more synchronizer flip-flops?
  - Helps, but just decreases probability of metastability
- So how solve completely?
  - Can't! May be unsettling to new designers. But we just can't guarantee a design that won't ever be metastable. We can just minimize the mean time between failure (MTBF) -- a number often given along with a circuit

Probability of flip-flop being metastable is…



20