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

EEL7020 – Sistemas Digitais
Aula 7: Circuitos sequenciais – Máquinas de Estado

Prof. Djones Vinicius Lettnin

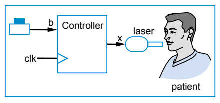
lettnin@eel.ufsc.br

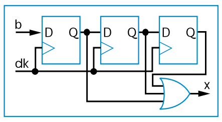
http://lettnin.paginas.ufsc.br/

Disclaimer: slides adapted for EEL7020 by D. Lettnin from the original slides made available by the author F. Vahid.

Finite-State Machines (FSMs) and Controllers

- Want sequential circuit with particular behavior over time
- Example: Laser timer
  - Push button: x=1 for 3 clock cycles
  - How? Let's try three flip-flops
    - b=1 gets stored in first D flip-flop
    - Then 2nd flip-flop on next cycle, then 3rd flip-flop on next
    - OR the three flip-flop outputs, so x should be 1 for three cycles



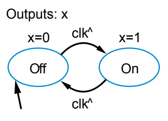


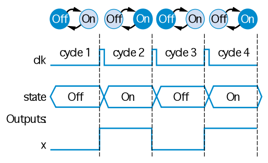
Need a Better Way to Design Sequential Circuits

- Trial and error is not a good design method
  - Will we be able to "guess" a circuit that works for other desired behavior?
    - How about counting up from 1 to 9? Pulsing an output for 1 cycle every 10 cycles? Detecting the sequence 1 3 5 in binary on a 3-bit input?
  - And, a circuit built by guessing may have undesired behavior
    - Laser timer: What if press button again while x=1? x then stays one another 3 cycles. Is that what we want?
- Combinational circuit design process had two important things
  1. A formal way to describe desired circuit behavior
     - Boolean equation, or truth table
  2. A well-defined process to convert that behavior to a circuit
- We need those things for sequence circuit design

Describing Behavior of Sequential Circuit: FSM

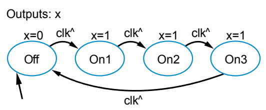
- Finite-State Machine (FSM)
  - A way to describe desired behavior of sequential circuit
    - Akin to Boolean equations for combinational behavior
  - List states, and transitions among states
    - Example: Make x change toggle (0 to 1, or 1 to 0) every clock cycle
    - Two states: "Off" (x=0), and "On" (x=1)
    - Transition from Off to On, or On to Off, on rising clock edge
    - Arrow with no starting state points to initial state (when circuit first starts)

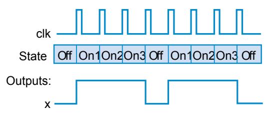




FSM Example: 0,1,1,1,repeat

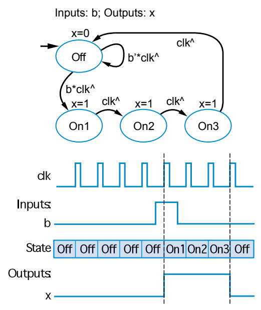
- Want 0, 1, 1, 1, 0, 1, 1, 1, ...
  - Each value for one clock cycle
- Can describe as FSM
  - Four states
  - Transition on rising clock edge to next state





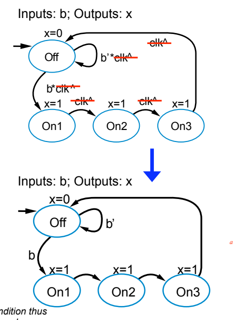
Extend FSM to Three-Cycles High Laser Timer

- Four states
- Wait in “Off” state while b is 0 (b’)
- When b is 1 (and rising clock edge), transition to On1
  - Sets x=1
  - On next two clock edges, transition to On2, then On3, which also set x=1
- So x=1 for three cycles after button pressed



FSM Simplification: Rising Clock Edges Implicit

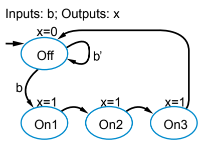
- Showing rising clock on every transition: cluttered
  - Make implicit -- assume every edge has rising clock, even if not shown
  - What if we wanted a transition *without* a rising edge
    - We don’t consider such asynchronous FSMs -- less common, and advanced topic
    - Only consider ***synchronous*** FSMs -- rising edge on *every* transition



Note: Transition with no associated condition thus transitions to next state on next clock cycle

FSM Definition

- FSM consists of
  - Set of states
    - Ex: {Off, On1, On2, On3}
  - Set of inputs, set of outputs
    - Ex: Inputs: {b}, Outputs: {x}
  - Initial state
    - Ex: “Off”
  - Set of transitions
    - Describes next states
    - Ex: Has 5 transitions
  - Set of actions
    - Sets outputs while in states
    - Ex: x=0, x=1, x=1, and x=1

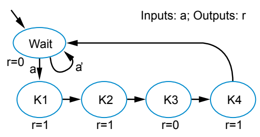


We often draw FSM graphically, known as ***state diagram***

Can also use table (state table), or textual languages

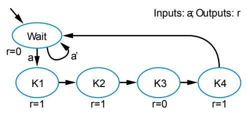
FSM Example: Secure Car Key

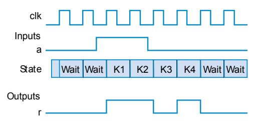
- Many new car keys include tiny computer chip
  - When car starts, car’s computer (under engine hood) requests identifier from key
  - Key transmits identifier
    - If not, computer shuts off car
- FSM
  - Wait until computer requests ID (a=1)
  - Transmit ID (in this case, 1101)



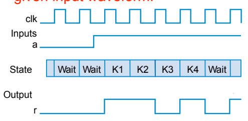
FSM Example: Secure Car Key (cont.)

- Nice feature of FSM
  - Can evaluate output behavior for different input sequence
  - Timing diagrams show states and output values for different input waveforms



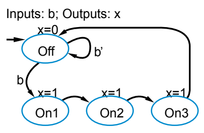


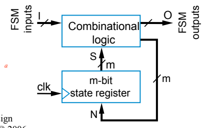
Q: Determine states and r value for given input waveform:



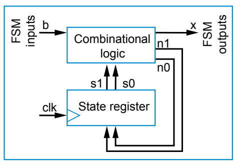
Standard Controller Architecture

- How implement FSM as sequential circuit?
  - Use standard architecture
    - State register -- to store the present state
    - Combinational logic -- to compute outputs, and next state
    - For laser timer FSM
      - 2-bit state register, can represent four states
      - Input b, output x
  - Known as ***controller***



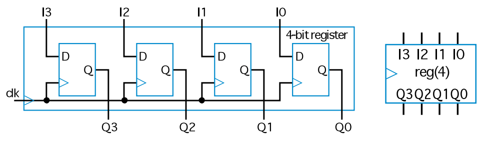


General version



Basic Register

- Typically, we store multi-bit items
  - e.g., storing a 4-bit binary number
- ***Register***: multiple flip-flops sharing clock signal
  - From this point, we'll use registers for bit storage
    - No need to think of latches or flip-flops
    - But now you know what's inside a register



Controller Design

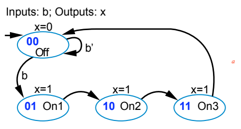
3.4

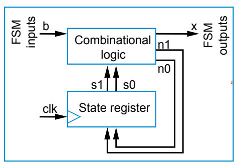
- Five step controller design process

	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.

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

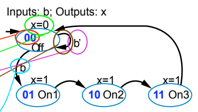


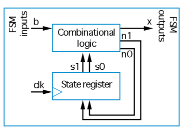


Controller Design: Laser Timer Example (cont)

- 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

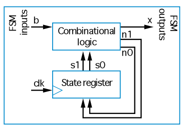




Controller Design: Laser Timer Example (cont)

- 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'

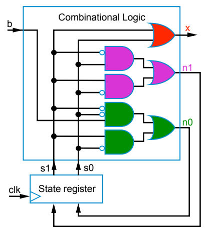
n0 = s1's0'b + s1s0'b' + s1s0'b

n0 = s1's0'b + s1s0'

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



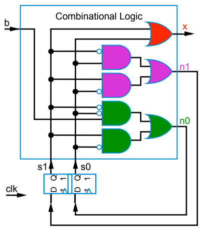
x = s1 + s0

n1 = s1's0 + s1s0'

n0 = s1's0'b + s1s0'

Controller Design: Laser Timer Example (cont)

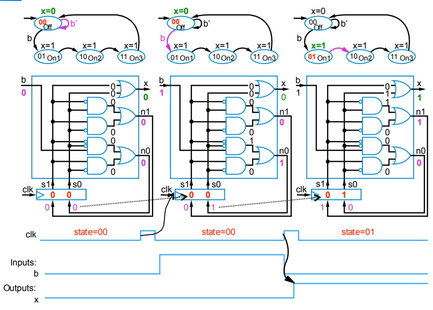
- Step 5: Implement combinational logic (cont)



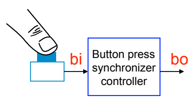
	Inputs			Outputs		
	s1	s0	b	x	n1	n0
Off	0	0	0	0	0	0
	0	0	1	0	0	1
On1	0	1	0	1	1	0
	0	1	1	1	1	0
On2	1	0	0	1	1	1
	1	0	1	1	1	1
On3	1	1	0	1	0	0
	1	1	1	1	0	0

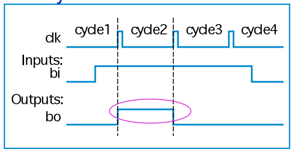
x = s1 + s0
n1 = s1’s0 + s1s0’
n0 = s1’s0’b + s1s0’

Understanding the Controller’s Behavior



Controller Example: Button Press Synchronizer

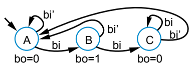




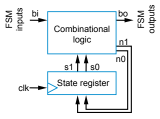
- Want simple sequential circuit that converts button press to single cycle duration, regardless of length of time that button actually pressed
  - We assumed such an ideal button press signal in earlier example, like the button in the laser timer controller

Controller Example: Button Press Synchronizer (cont)

FSM inputs: bi; FSM outputs: bo

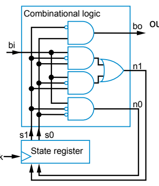


Step 1: FSM

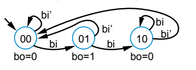


Step 2: Create architecture

n1 = s1’s0bi + s1s0bi
n0 = s1’s0’bi
bo = s1’s0bi’ + s1’s0bi = s1s0



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

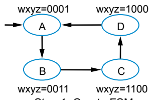
Step 5: Create combinational circuit

Step 5: Create combinational circuit

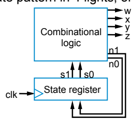
Controller Example: Sequence Generator

- Want generate sequence 0001, 0011, 1100, 1000, (repeat)
  - Each value for one clock cycle
  - Common, e.g., to create pattern in 4 lights, or control magnets of a “stepper motor”

Inputs: none; Outputs: w,x,y,z

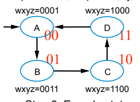


Step 1: Create FSM



Step 2: Create architecture

Inputs: none; Outputs: w,x,y,z

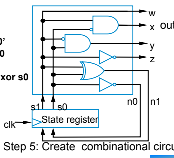


Step 3: Encode states

	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’

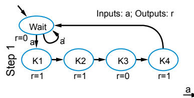


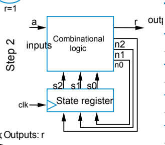
Step 5: Create combinational circuit

Controller Example: Secure Car Key

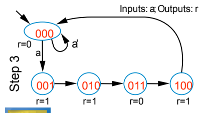
- (from earlier example)

Inputs: a; Outputs: r





Inputs: a; Outputs: r



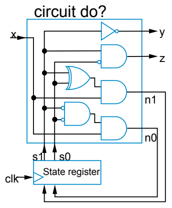
We’ll omit Step 5

	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?

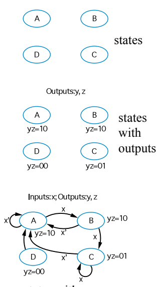


Work backwards

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

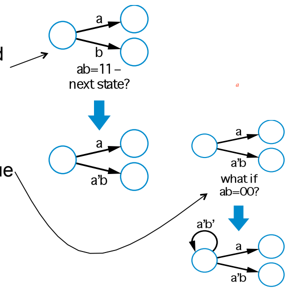
Pick any state names you want



states with outputs and transitions

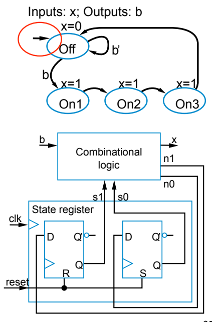
Common Pitfalls Regarding Transition Properties

- *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?



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
  - Designer must ensure reset input is 1 during power up of circuit
    - By electronic circuit design



Chapter Summary

- Sequential circuits
  - Have state
- Created robust bit-storage device: D flip-flop
  - Put several together to build register, which we used to hold state
- Defined FSM formal model to describe sequential behavior
  - Using solid mathematical models -- Boolean equations for combinational circuit, and FSMs for sequential circuits -- is very important.
- Defined 5-step process to convert FSM to sequential circuit
  - Controller
- So now we know how to build the class of sequential circuits known as controllers