Introduction

- Chapter 3: Controllers
  - Control input/output: single bit (or just a few) representing event or state
  - Finite-state machine describes behavior; implemented as state register and combinational logic
- Chapter 4: Datapath components
  - Data input/output: Multiple bits collectively representing single entity
  - Datapath components include registers, adders, ALU, comparators, register files, etc.
- This chapter: custom processors
  - Processor: Controller and datapath components working together to implement an algorithm

RTL Design: Capture Behavior, Convert to Circuit

- Recall
  - Chapter 2: Combinational Logic Design
    - First step: Capture behavior (using equation or truth table)
    - Remaining steps: Convert to circuit
  - Chapter 3: Sequential Logic Design
    - First step: Capture behavior (using FSM)
    - Remaining steps: Convert to circuit
- RTL Design (the method for creating custom processors)
  - First step: Capture behavior (using high-level state machine, to be introduced)
  - Remaining steps: Convert to circuit

RTL Design Method

1. Capture a high-level state machine
   - Describe the system's desired behavior as a high-level state machine.
   - The state machine consists of states and transitions. The state machine is "high-level" because the transition conditions and the state actions are more abstract.
2. Create a datapath
   - Create a datapath to carry out the data operations of the high-level state machine.
3. Convert the datapath to a controller
   - Connect the datapath to a controller block. Connect external Boolean inputs and outputs to the controller block.
4. Derive the controller's FSM
   - Convert the high-level state machine to a finite-state machine (FSM) for the controller, by replacing data operations with setting and reading of control signals to and from the datapath.

RTL Design Method: *Preview* Example

- Soda dispenser
  - c: bit input, 1 when coin deposited
  - a: 8-bit input having value of deposited coin
  - s: 8-bit input having cost of a soda
  - d: bit output, processor sets to 1 when total value of deposited coins equals or exceeds cost of a soda

How can we precisely describe this processor’s behavior?
Preview Example: Step 1 -- Capture High-Level State Machine

- Declare local register tot
- Init state: Set d=0, tot=0
- Wait state: wait for coin
  - If see coin, go to Add state
  - Add state: Update total value: tot = tot + a
    - Remember, a is present coin’s value
    - Go back to Wait state
- In Wait state, if tot >= s, go to Disp(sense) state
- Disp state: Set d=1 (dispense soda)
  - Return to Init state

Inputs: c (bit), a (8 bits), s (8 bits)
Outputs: d (bit)
Local registers: tot (8 bits)

Step 2 -- Create Datapath

- Need tot register
- Need 8-bit comparator to compare s and tot
- Need 8-bit adder to perform tot = tot + a
- Wire the components as needed for above
- Create control inputs/outputs, give them names

Step 3 -- Connect Datapath to a Controller

- Controller’s inputs
  - External input c (coin detected)
  - Input from datapath comparator’s output, which we named tot.lt_s
- Controller’s outputs
  - External output d (dispense soda)
  - Outputs to datapath to load and clear the tot register

Step 4 -- Derive the Controller’s FSM

- Same states and arcs as high-level state machine
- But set/read datapath control signals for all datapath operations and conditions

Implement the FSM as a state register and logic
- As in Ch3
- Table shown on right

Step 1: Create a High-Level State Machine

Let’s consider each step of the RTL design process in more detail

Step 1
- Soda dispenser example
  - Not an FSM because:
    - Multi-bit (data) inputs a and s
    - Local register tot
    - Data operations tot=0, tot=a, tot=0+a
  - Useful high-level state machine:
    - Data types beyond just bits
    - Local registers
    - Arithmetic equations/expressions

Step 2 -- Create Datapath

- Need tot register
- Need 8-bit comparator to compare s and tot
- Need 8-bit adder to perform tot = tot + a
- Wire the components as needed for above
- Create control inputs/outputs, give them names

Step 3 -- Connect Datapath to a Controller

- Controller’s inputs
  - External input c (coin detected)
  - Input from datapath comparator’s output, which we named tot.lt_s
- Controller’s outputs
  - External output d (dispense soda)

Step 4 -- Derive the Controller’s FSM

- Same states and arcs as high-level state machine
- But set/read datapath control signals for all datapath operations and conditions

Implement the FSM as a state register and logic
- As in Ch3
- Table shown on right

Step 1: Create a High-Level State Machine

Let’s consider each step of the RTL design process in more detail

Step 1
- Soda dispenser example
  - Not an FSM because:
    - Multi-bit (data) inputs a and s
    - Local register tot
    - Data operations tot=0, tot=a, tot=0+a
  - Useful high-level state machine:
    - Data types beyond just bits
    - Local registers
    - Arithmetic equations/expressions

Step 2 -- Create Datapath

- Need tot register
- Need 8-bit comparator to compare s and tot
- Need 8-bit adder to perform tot = tot + a
- Wire the components as needed for above
- Create control inputs/outputs, give them names

Step 3 -- Connect Datapath to a Controller

- Controller’s inputs
  - External input c (coin detected)
  - Input from datapath comparator’s output, which we named tot.lt_s
- Controller’s outputs
  - External output d (dispense soda)

Step 4 -- Derive the Controller’s FSM

- Same states and arcs as high-level state machine
- But set/read datapath control signals for all datapath operations and conditions

Implement the FSM as a state register and logic
- As in Ch3
- Table shown on right

Step 1: Create a High-Level State Machine

Let’s consider each step of the RTL design process in more detail

Step 1
- Soda dispenser example
  - Not an FSM because:
    - Multi-bit (data) inputs a and s
    - Local register tot
    - Data operations tot=0, tot=a, tot=0+a
  - Useful high-level state machine:
    - Data types beyond just bits
    - Local registers
    - Arithmetic equations/expressions
Step 1 Example: Laser-Based Distance Measurer

Example of how to create a high-level state machine to describe desired processor behavior

Laser-based distance measurement — pulse laser, measure time T to sense reflection

- Laser light travels at speed of light, \(3 \times 10^8 \text{ m/sec}\)
- Distance is thus \(D = T \text{ sec} \times \frac{3 \times 10^8 \text{ m/sec}}{2}\)

Object of interest: \(D\)

Sensor

Laser

\(T\) (in seconds)

Digital Design
Copyright © 2006
Frank Vahid

Step 1 Example: Laser-Based Distance Measurer

• Inputs/outputs
  - \(B\): bit input, from button to begin measurement
  - \(L\): bit output, activates laser
  - \(S\): bit input, senses laser reflection
  - \(D\): 16-bit output, displays computed distance

Digital Design
Copyright © 2006
Frank Vahid

Step 1 Example: Laser-Based Distance Measurer

• Step 1: Create high-level state machine
  • Begin by declaring inputs and outputs
  • Create initial state, name it \(S_0\)
    - Initialize laser to off (\(L=0\))
    - Initialize displayed distance to 0 (\(D=0\))

Inputs: \(B, S\) (1 bit each)
Outputs: \(L\) (bit), \(D\) (16 bits)

Digital Design
Copyright © 2006
Frank Vahid

Step 1 Example: Laser-Based Distance Measurer

• Add another state, call \(S_1\), that waits for a button press
  - \(B'\) — stay in \(S_1\), keep waiting
  - \(B\) — go to a new state \(S_2\)

Q: What should \(S_2\) do? A: Turn on the laser

Inputs: \(B, S\) (1 bit each)
Outputs: \(L\) (bit), \(D\) (16 bits)

Digital Design
Copyright © 2006
Frank Vahid

Step 1 Example: Laser-Based Distance Measurer

• Add a state \(S_2\) that turns on the laser (\(L=1\))
• Then turn off laser (\(L=0\)) in a state \(S_3\)

Q: What do next? A: Start timer, wait to sense reflection

Inputs: \(B, S\) (1 bit each)
Outputs: \(L\) (bit), \(D\) (16 bits)

Local Registers: \(Dctr\) (16 bits)

Digital Design
Copyright © 2006
Frank Vahid

Step 1 Example: Laser-Based Distance Measurer

• Stay in \(S_3\) until sense reflection (\(S\))
  • To measure time, count cycles for which we are in \(S_3\)
    - To count, declare local register \(Dctr\)
    - Increment \(Dctr\) each cycle in \(S_3\)
    - Initialize \(Dctr\) to 0 in \(S_1, S_2\) would have been O.K. too

Digital Design
Copyright © 2006
Frank Vahid
Step 1 Example: Laser-Based Distance Measurer

- Once reflection detected (S), go to new state S4
  - Calculate distance
  - Assuming clock frequency is $3 \times 10^8$, $D_{ctr}$ holds number of meters, so $D = D_{ctr}/2$
- After S4, go back to S1 to wait for button again

Step 2: Create a Datapath

- Datapath must
  - Implement data storage
  - Implement data computations
- Look at high-level state machine, do three substeps
  - (a) Make data inputs/outputs be datapath inputs/outputs
  - (b) Instantiate declared registers into the datapath (also instantiate a register for each data output)
  - (c) Examine every state and transition, and instantiate datapath components and connections to implement any data computations

Step 2 Example: Laser-Based Distance Measurer

(a) Make data inputs/outputs be datapath inputs/outputs
(b) Instantiate declared registers into the datapath (also instantiate a register for each data output)
(c) Examine every state and transition, and instantiate datapath components and connections to implement any data computations

Step 2 Example Showing Mux Use

- Introduce mux when one component input can come from more than one source

Step 3: Connecting the Datapath to a Controller

- Laser-based distance measurer example
- Easy – just connect all control signals between controller and datapath
Step 4: Deriving the Controller’s FSM

- FSM has same structure as high-level state machine
  - Inputs/outputs all bits now
  - Replace data operations by bit operations using datapath

Inputs: B, S (1 bit each)
Outputs: L (bit), D (16 bits)

Local Registers: Dctr (16 bits)

Datapath
- Dreg_clr
- Dreg_ld
- Dctr_clr
- Dctr_cnt

Inputs: B, S
Outputs: L, Dreg_clr, Dreg_ld, Dctr_clr, Dctr_cnt

S0
S1 S2
S3
S4

L = 0 L = 1 L = 0 L = 0

B'
S'
BS

L = 0

D = Dctr / 2 (calculate D)

Using shorthand of outputs not assigned implicitly assigned 0

Inputs: B, S
Outputs: L, Dreg_clr, Dreg_ld, Dctr_clr, Dctr_cnt

S0
S1 S2
S3
S4

L = 0 L = 1 L = 0 L = 0

B'
S'
BS

L = 0

Dreg_clr = 1
Dreg_ld = 0
Dctr_clr = 0
Dctr_cnt = 0 (laser off)

Dreg_clr = 0
Dreg_ld = 0
Dctr_clr = 1
Dctr_cnt = 0 (clear count)

Dreg_clr = 0
Dreg_ld = 0
Dctr_clr = 0
Dctr_cnt = 0 (laser on)

Dreg_clr = 0
Dreg_ld = 1
Dctr_clr = 0
Dctr_cnt = 0 (load D reg with Dctr/2)

Dreg_clr = 1
Dreg_ld = 0
Dctr_clr = 0
Dctr_cnt = 1 (laser off)

Digital Design
Copyright © 2006 Frank Vahid

Design Challenge

- Design Challenge
  - Create a high-level state machine that describes the following system behavior.
    - The system has an 10-bit input A, a single-bit input C, and a 32-bit output S.
    - On every clock cycle, if C = 1, the system should add A to a running sum and output that sum on S.
    - On every clock cycle, if C = 0, the system should add 2*A to a running sum and output that sum on S.
  
  Hint: declare and use an internal register to keep the sum.
  
  Add a 1-bit input Clr to the system. When Clr = 1, the system should clear the sum, S, to 0.

  - Using the RTL design method, convert the high-level state machine to a datapath and controller

Due Next Lecture (as announced in class)