ECE 274 – Digital Logic

RTL Design: Introduction

Digital Design (Vahid): Ch. 5.1 – 5.2

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 included 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

Step Description
1. Capture a high-level state machine
2. Create a datapath
3. Convert the datapath to a controller block
4. Derive the controller’s FSM

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?
Capture High-Level State Machine

- Declare local register \( t_{ot} \)
- Init state: Set d=0, tot=0
- Wait state: wait for coin
  - If no coin, go to Add state
- Add state: Update total value: \( t_{ot} = t_{ot} + a \)
  - Remember, \( a \) is present coin's value
  - Go back to Wait 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: \( t_{ot} \) (8 bits)

Connect Datapath to a Controller

- Controller's inputs
  - External input c (coin detected)
  - Input from datapath comparator's output, which we named \( t_{ot}_{lt} \)
- Controller's outputs
  - External output d (dispense soda)
  - Outputs to datapath to load and clear the \( t_{ot} \) register

Data operations
- Add \( t_{ot} + a \)
- Compare \( t_{ot} < s \)
- Arithmetic equations/expressions

Step 1: Create a High-Level State Machine

- Let's consider each step of the RTL design process in more detail
- Table shown on right

Step 2 -- Create Datapath

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

Step 3 – Connect Datapath to a Controller

- Create control
- Wire the components as needed for above
- Remember, \( c * (t_{ot} < s) \)
- Step 1
  - Soda dispenser example
  - Not an FSM because:
    - Multi-bit (data) inputs \( a \) and \( s \)
    - Local register \( t_{ot} \)
    - Data operations \( d = h(t) \), etc.
  - 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, $3 \times 10^8$ m/sec
  - Distance is thus $D = T \text{ sec} \times \frac{3 \times 10^8 \text{ m/sec}}{2}$

Object of interest

Laser-based distance measurer

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

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 ($D=0$)

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 $S_2$

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

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
Step 1 Example: Laser-Based Distance Measurer

- Once reflection detected (S), go to new state S4
  - Calculate distance
  - Assuming clock frequency is $3 \times 10^8$, Dctr holds number of meters, so $D = Dctr/2$
- 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

**Datapath**

- Dreg_clr
- Dreg_ld
- Dctr_clr
- Dctr_cnt

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

**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.