CSE140: Components and Design Techniques for Digital Systems

Register Transfer Level (RTL) Design

Tajana Simunic Rosing

Sources: TSR, Katz, Boriello, Vahid, Perkowski
Where we are now…

• What we have covered last time:
  – Combinational and sequential circuits

• What we are covering today:
  – Register Transfer Level (RTL) design

• Deadlines:
  – ZyBook chap 6 due this week
  – Vahid textbook chap 5 available on class schedule page
  – HW#6 out today – due next Tuesday!
  – Exam#3 last day of class
    • 80 minutes long, Comprehensive
    • Bring one 8 ½ x 11” paper with handwritten notes, but nothing else

• Exam #2 has been curved
  – Prior to curve max was 99%, average was 67%, now the average is 75%
  – Regrades have to be requested in writing at the end of this class. Once you walk away with the midterm, the grade stands as is.
  – If you don’t pick it up now, you can get it from TA.
High-Level State Machine

- Some behaviors may be too complex to describe by using classical FSMs
- Soda dispenser
  - c: bit input, 1 when coin deposited
  - a: 8-bit input: value of the deposited coin
  - s: 8-bit input: cost of a soda
  - d: bit output, processor sets it to 1 when total value of deposited coins equals or exceeds cost of a soda
Challenges in High-Level State Machines

Which of the following makes the FSM design of this problem difficult?

A. 8-bit input/output
B. Tracking the current total
C. Multibit comparison
D. All of the above
E. None of the above
Benefits of HLSMs

- High-level state machine (HLSM) extends FSM with:
  - Multi-bit input/output
  - Local storage
  - Arithmetic operations

- Conventions
  - Numbers:
    - Single-bit: '0' (single quotes)
    - Integer: 0 (no quotes)
    - Multi-bit: “0000” (double quotes)
  - == for comparison equal
  - Multi-bit outputs must be registered via local storage
  - // precedes a comment

SodaDispenser

Inputs: c (bit), a (8 bits), s (8 bits)
Outputs: d (bit)  // '1' dispenses soda
Local storage: tot (8 bits)
FSMs vs. HLSMs

How does the HLSM differ from the FSM for this problem?

A. The HLSM stores multibit data, but the FSM doesn’t
B. The FSM stores the state but the HLSM doesn’t
C. Implementing the HLSM requires data registers, data registers are not required to implement the FSM
D. All of the above
E. None of the above
Similarities between FSMs & HLSMs

Which of the following are common between HLSMs and FSMs?

A. Transitions happen at the edge of a clock ✓

B. They both have external complex data ✗

C. All of the above ✗

D. None of the above ✗

Sources: TSR, Katz, Boriello, Vahid, Perkowski

Soda dispenser

Processor

Inputs: c (bit), a (8 bits), s (8 bits)

Outputs: d (bit) // '1' dispenses soda

Local storage: tot (8 bits)

Add

Tot:=tot+a

Disp

d:=1'

SodaDispenser

Init
d:=0'

tot:=0

Wait
c**tot<s

c:=c'*

tot:=tot+a

Disp
d:=1'

tot:=tot+a

Sources: TSR, Katz, Boriello, Vahid, Perkowski
**RTL Design Process**

- **Capture** the behavior
- **Convert** it to a circuit
  - High-level architecture (datapath and control path)
  - Datapath capable of HLSM's data operations
  - Design controller to control the datapath
Step 2: Create Datapath for Soda Dispenser

- Need \textit{tot} register to keep track of the money deposited so far
- Need 8-bit comparator to compare \( s \) (current sum) and \( a \) (target cost)
- Need 8-bit adder to update: \( \text{tot} = \text{tot} + a \)
- Connect everything
- Create control IO
Signals in Soda Dispenser

Inputs: c (bit), a (8 bits), s (8 bits)
Outputs: d (bit) // '1' dispenses soda
Local storage: tot (8 bits)

SodaDispenser

According to the current design, under which of the following conditions does the register output 'tot' change at the rising clock edge?
A. Whenever the value of the coin inserted ('a') changes
B. Whenever the cost of the soda ('s') changes
C. When the signal tot_ld becomes high
D. When the signal tot_clr becomes high
E. Both C. & D.

Sources: TSR, Katz, Boriello, Vahid, Perkowski
Step 3: Connect Datapath to a Controller

- Controller’s inputs
  - External input \( c \) (coin detected)
  - Input from datapath comparator’s output, which we named \( \text{tot}_{-}\text{lt}_s \)
- Controller’s outputs
  - External output \( d \) (dispense soda)
  - Outputs to datapath to load and clear the \( \text{tot} \) register
Step 4 – Derive the Controller’s FSM

- FSM has the same states and arcs as HLSM
- Replace all references to the data elements in the HLSM with appropriate control signals & values
Step 4: Finalize the Implementation

Implement the FSM as a state register and logic

- Inputs: \( c, \text{tot\_lt\_s} \) (bit)
- Outputs: \( d, \text{tot\_ld}, \text{tot\_clr} \) (bit)

Controller diagram:
- States: Init, Wait, Disp
- Transitions:
  - \( c \rightarrow \text{Init} \)
  - \( c \rightarrow \text{Wait} \)
  - \( c \rightarrow \text{Disp} \)
  - \( \text{tot\_lt\_s} \rightarrow \text{Disp} \)
- \( d = 0 \) in Init
- \( \text{tot\_clr} = 1 \) in Disp

Transition table:

<table>
<thead>
<tr>
<th>Cur State</th>
<th>( c )</th>
<th>( \text{tot_lt_s} )</th>
<th>Next State</th>
<th>( d )</th>
<th>( \text{tot_ld} )</th>
<th>( \text{tot_clr} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Init</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Wait</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Disp</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Add</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Disp</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Tot LD</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Sources: TSR, Katz, Boriello, Vahid, Perkowski
Another RTL Design: Laser-Based Distance Measurer

- 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 $D = T \text{ sec} \times 3 \times 10^8 \text{ m/sec} / 2$
Laser-Based Distance Measurer IO

- **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, to display computed distance
Laser-Based Distance Measurer: HLSM

- Declare inputs, outputs, and local storage
  - Dreg required for multi-bit output
- Create initial state, name it S0
  - Initialize laser to off (L:='0')
  - Initialize displayed distance to 0 (Dreg:=0)
Laser-Based Distance Measurer: HLSM

- Add another state, $S1$, that waits for a button press
  - $B'$ – stay in $S1$, keep waiting
  - $B$ – go to a new state $S2$
Laser-Based Distance Measurer: HLSM

- Add a state $S2$ that turns on the laser ($L:='1'$)
- Then turn off laser ($L:='0'$) in a state $S3$
Laser-Based Distance Measurer: HLSM

- Stay in **S3** until sense reflection (S)
- To measure time, count cycles while in **S3**
  - To count, declare local storage **Dctr**
  - Initialize **Dctr** to 0 in **S1**. In **S2** would have been O.K. too.
    - Don't forget to initialize local storage—common mistake
  - Increment **Dctr** each cycle in **S3**
Laser-Based Distance Measurer: HLSM

- 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 $Dreg := Dctr/2$
- After S4, go back to S1 to wait for button again
Laser-Based Distance Measurer: Create a Datapath

- HLSM data I/O → DP I/O
- HLSM local storage → reg
- HLSM state action and transition condition data computation → Datapath components and connections
Laser-Based Distance Measure: Connecting the Datapath to a Controller

Diagram:
- From button B:
  - Controller
  - Dreg_clr
  - Dreg_ld
  - Dctr_clr
  - Dctr_ld
- To display D:
  - 300 MHz Clock
- From sensor S to controller:
  - L to laser
  - 16
- From controller to datapath:
  - Dreg_clr
  - Dreg_ld
  - Dctr_clr
  - Dctr_ld
Laser-Based Distance Measurer: Derive the Controller FSM

- FSM has same states, transitions, and control I/O
- Achieve each HLSM data operation using datapath control signals in FSM

**HLSM**

DistanceMeasurer

*Inputs: B (bit), S (bit)  Outputs: L (bit), D (16 bits)  Local storage: Dreg, Dctr (16 bits)*

- \(S_0\): \(L := '0'\), \(Dctr := 0\)
- \(S_1\): \(L := '0'\), \(Dctr := Dctr + 1\) // calculate \(D\)
- \(S_2\): \(Dreg := Dctr / 2\)
- \(S_3\): \(L := '1'\), \(Dctr := 0\)
- \(S_4\): \(Dreg := 0\), \(Dctr := 0\) (laser off)

**Controller**

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

- \(S_0\): \(L = 0\), \(Dreg clr = 1\) (clear Dreg)
- \(S_1\): \(L = 0\), \(Dreg clr = 0\), \(Dreg ld = 0\)
- \(S_2\): \(L = 1\), \(Dreg clr = 0\), \(Dreg ld = 0\) (clear count)
- \(S_3\): \(Dctr clr = 1\) (laser on)
- \(S_4\): \(Dctr clr = 0\), \(Dctr ld = 1\) (load Dreg with Dctr/2) (stop counting)
Laser-Based Distance Measurer: Simplify the Controller FSM

- Same FSM, using convention of unassigned outputs implicitly assigned 0

**Controller**

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

- **S0**
  - L = 0
  - Dreg_clr = 1
  - (laser off)
  - (clear Dreg)

- **S1**
  - B
  - Dctr_clr = 1
  - (clear count)

- **S2**
  - L = 1
  - (laser on)

- **S3**
  - L = 0
  - Dctr_id = 1
  - (laser off)
  - (load Dreg with Dctr/2)

- **S4**
  - Dreg_id = 1
  - Dctr_id = 0
  - (stop counting)

Some assignments to 0 still shown, due to their importance in understanding desired controller behavior
# RTL Design Process

<table>
<thead>
<tr>
<th>Step</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Step 1:</strong> Capture behavior</td>
<td></td>
</tr>
</tbody>
</table>
  *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 than just Boolean operations on single-bit inputs and outputs. |
| **Step 2:** Convert to circuit  
  2A |  
  *Create a datapath*  
  Create a datapath to carry out the data operations of the high-level state machine. |
| 2B |  
  *Connect the datapath to a controller*  
  Connect the datapath to a controller block. Connect external control inputs and outputs to the controller block. |
| 2C |  
  *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. |