Introduction to RTL Design

Chapter 5: RTL Design


Copyright © 2007 Frank Vahid

Instructors of courses requiring Vahid’s Digital Design textbook published by John Wiley and Sons have permission to modify and use these slides for customary course-related teaching purposes in keeping with this copyright notice in place and unmodified. These slides may be posted as unanimated pdf versions on publicly-accessible course websites. PowerPoint source or pdf with animations may not be posted for public dissemination without the prior written consent of the author. Any other use requires explicit permission. Information may change. Please refer to or obtain special use permissions from Wiley for permission information.

Instructors may make printouts of the slides available to students for a reasonable photocopying charge, without incurring royalties. Any other use requires explicit permission. Information may change. Please refer to or obtain special use permissions from Wiley for permission information.

RTL Design

RTL Design Method

<table>
<thead>
<tr>
<th>Step</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>Capture a high-level state machine.</td>
</tr>
<tr>
<td>2.</td>
<td>Create a datapath to carry out the data operations of the high-level state machine.</td>
</tr>
<tr>
<td>3.</td>
<td>Connect the datapath to a controller block. Connect external Boolean inputs and outputs to the controller block.</td>
</tr>
<tr>
<td>4.</td>
<td>Derive the controller’s FSM.</td>
</tr>
</tbody>
</table>

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?
**RTL Design**

**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` (dispense) 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)

---

**RTL Design**

**Preview Example: 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 input/outputs, give them names

**Controller Datapath**

**Inputs:** `c`, `tot_lt_s` (bit)
**Outputs:** `d`, `tot_ld`, `tot_clr` (bit)

---

**RTL Design**

**Preview Example: 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

**Controller**

**Datapath**

---

**RTL Design**

**Preview Example: 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
RTL Design

**Preview Example: Completing the Design**

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

```
<table>
<thead>
<tr>
<th>a1</th>
<th>a0</th>
<th>c</th>
<th>d</th>
<th>n1</th>
<th>n0</th>
<th>d</th>
<th>p</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
```

**Controller**

**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 \( D = T \text{ sec} \times 3 \times 10^8 \text{ m/sec} / 2 \)

**RTL Design**

**High-Level State Machine**

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

**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: Create high-level state machine
- Begin by declaring inputs and outputs
- Create initial state, name it **S0**
  - Initialize laser to off (L=0)
  - Initialize displayed distance to 0 (D=0)

Q: What should **S2** do?  A: Turn on the laser

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

- Add a state **S2** that turns on the laser (L=1)
- Then turn off laser (L=0) in a state **S3**

**Local Registers:** Dctr (16 bits)
RTL Design

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 $D = \frac{Dctr}{2}$
- After S4, go back to S1 to wait for button again

RTL Design

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
### RTL Design

#### Step 2: Example Showing Mux Use

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

![Diagram showing mux use](image)

#### Step 3: Connecting the Datapath to a Controller

- Laser-based distance measurer example
- Easy – just connect all control signals between controller and datapath

![Diagram showing connection](image)

#### Step 4: Deriving the Controller's FSM

- FSM has same structure as high-level state machine
- Replace data operations by bit operations using datapath

![Diagram showing FSM](image)

---

**Local Registers:**
- EfG, R (16 bits)

**Inputs:**
- B (1 bit)
- S (1 bit)

**Outputs:**
- L (1 bit)
- D (16 bits)

**Local Registers:**
- Dreg (16 bits)
- Dctr (16 bits)

**States:**
- S0, S1, S2, S3, S4

**Transitions:**
- S0 → S1 (L = 0, D = 0)
- S1 → S2 (L = 1)
- S2 → S3 (L = 0, D = Dctr + 1)
- S3 → S4 (L = 0, D = Dctr / 2)
- S4 → S0 (L = 0, D = 0)

**Outputs:**
- Dreg_clr
- Dreg LD
- Dctr_clr
- Dctr_cnt

**Conditions:**
- Laser off
- Laser on
- Load D with Dctr/2
- Stop counting

---

**Inputs:**
- B (1 bit)
- S (1 bit)

**Outputs:**
- L (1 bit)
- D (16 bits)

**Local Registers:**
- Dctr (16 bits)
- Dreg (16 bits)

**States:**
- S0, S1, S2, S3, S4

**Transitions:**
- S0 → S1 (L = 0, D = 0)
- S1 → S2 (L = 1)
- S2 → S3 (L = 0, D = Dctr + 1)
- S3 → S4 (L = 0, D = Dctr / 2)
- S4 → S0 (L = 0, D = 0)

**Outputs:**
- Dreg_clr
- Dreg LD
- Dctr_clr
- Dctr_cnt

**Conditions:**
- Laser off
- Laser on
- Load D with Dctr/2
- Stop counting
RTL Design

Step 4: Deriving the Controller’s FSM

Datapath

Inputs: B, S

Outputs: L, Dreg_clr, Dreg ld, Dctr_clr, Dctr_cnt

Controller

S0 -> S1 -> S2 -> S3 -> S4

L = 0 (laser off)
Dreg_clr = 1 (clear D reg)
D ctr clr = 1 (clear count)
Dreg ld = 0 (load D reg with Dreg)

B’

S1

L = 0 (laser off)
Dreg_clr = 1 (clear D reg)

S2

L = 1 (laser on)
Dctr_cnt = 1 (count up)

S3

L = 0 (laser off)
Dctr clr = 1 (clear count)
Dctr_cnt = 0 (stop counting)

S4

Dreg ld = 1 (load D reg with Dreg2)