RTL Design

RTL Design Method

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

RTL Design

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**ense 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 inputs/outputs, give them names

**Datapath**

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

---

**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**

**Inputs:** \( c \) (bit), \( tot_{lt\_s} \) (bit)
**Outputs:** \( d \) (bit)

---

**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

**Controller**

**Datapath**

**Inputs:** \( c \) (bit), \( tot_{lt\_s} \) (bit)
**Outputs:** \( d \) (bit)
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>s1</th>
<th>s0</th>
<th>c'</th>
<th>c</th>
<th>d'</th>
<th>d</th>
<th>p'</th>
<th>p</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</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>1</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>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Wait</th>
<th>Add</th>
<th>Disp</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Controller

RTL Design

High-Level State Machine

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

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

Outputs:
- d (bit)
- tot (8 bits)

RTL Design

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*10^8 m/sec
  - Distance is thus D = T sec * 3*10^8 m/sec / 2

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
RTL Design
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 S0
  - Initialize laser to off (L=0)
  - Initialize displayed distance to 0 (D=0)

- Add another state, call S1, that waits for a button press
  - B' – stay in S1, keep waiting
  - B – go to a new state S2

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

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

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

- Stay in S3 until sense reflection (S)
- To measure time, count cycles for which we are in S3
  - To count, declare local register Dctr
  - Increment Dctr each cycle in S3
  - Initialize Dctr to 0 in S1. S2 would have been O.K. too
### RTL Design

**Step 1 Example: Laser-Based Distance Measurer**

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

### 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

RTL Design

Step 3: Connecting the Datapath to a Controller

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

RTL Design

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

RTL Design

Step 4: Deriving the Controller’s FSM

- Using shorthand of outputs not assigned implicitly assigned 0
**RTL Design**

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

- **Datapath**
  - Dreg_clr
  - Dreg_ld
  - Dctr_clr
  - Dctr_cnt

- **Inputs:** B, S
- **Outputs:** L, Dreg_clr, Dreg_ld, Dctr clr, Dctr_cnt

- **States:**
  - S0: L = 0, Dreg clr = 1 (laser off, clear D reg)
  - S1: Dreg clr = 1 (clear D reg)
  - S2: L = 1, Dctr clr = 1 (laser on, clear count)
  - S3: Dctr clr = 1 (clear count)
  - S4: Dreg clr = 1 (load D reg with Dctr/2, stop counting)

- **Logic:**
  - Dreg_clr = 1 (clear D reg)
  - Dctr_clr = 1 (clear count)
  - Dctr_cnt = 1 (laser off, count up)
  - Dreg ld = 1 (load D reg with Dctr/2, stop counting)