### RTL Design Method

#### RTL Design Method Examples

**Digital Design 5.3**

Chapter 5: RTL Design


http://www.ddvahid.com

**Copyright © 2007 Frank Vahid**

Instructors of courses requiring Frank 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 the copyright notice above, and to modify these slides for customary course-related teaching purposes in keeping with the copyright notice in the textbook Digital Design, First Edition, by Frank Vahid, John Wiley and Sons Publishers, 2007. These slides may be modified as necessary for teaching purposes and may be used for classroom and homework assignments. However, these slides may not be posted to publicly accessible websites, for internal use by students, or for distribution in any form other than paper or special use permission from Wiley. These slides may be used by non-profit educational and governmental organizations, subject to these requirements.

---

**RTL Design**

**RTL Design Method**

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

---

**Video Compression – Sum of Absolute Differences**

Only difference: UFO moving

- **Frame 1**: Digitized frame 1
  - 1 Mbyte

- **Frame 2**: Digitized frame 2
  - 1 Mbyte

**Difference of 2 from 1**

- 0.01 Mbyte

- Video is a series of frames (e.g., 30 per second)
- Most frames similar to previous frame
  - Compression idea: Just send difference from previous frame
Video Compression – Sum of Absolute Differences

Need to quickly determine whether two frames are similar enough to just send difference for second frame
- Compare corresponding 16x16 “blocks”
  - Treat 16x16 block as 256-byte array
- Compute the absolute value of the difference of each array item
- Sum those differences – if above a threshold, send complete frame for second frame; if below, can use difference method (using another technique, not described)

Want fast sum-of-absolute-differences (SAD) component
- When go=1, sums the differences of element pairs in arrays A and B, outputs that sum

Step 2: Create datapath
- Inputs: A, B (256 byte memory); go (bit)
- Outputs: sad (32 bits)
- Local registers: sum, sad_reg (32 bits); i (9 bits)
- S0: wait for go
- S1: initialize sum and index
- S2: check if done (i>=256)
- S3: add difference to sum, increment index
- S4: done, write to output sad_reg
## RTL Design

### RTL Example: Video Compression – Sum of Absolute Differences

- Step 3: Connect to controller
- Step 4: Replace high-level state machine by FSM

---

### RTL Design

#### Pitfalls and Good Practice

- Considering the high-level state machine shown to the right, what is the final value of D in state F?
  1. 10
  2. 11
  3. 12
  4. 17
  5. 20

Inputs: D (8-bits), Q (8-bits)

Local Registers: R (8-bits)

---

### RTL Design

#### Pitfalls and Good Practice

- Common pitfall: Assuming register is update in the state it’s written
  - All registers updates in each state will happen simultaneously
  - On the next rising clock edge
- Consider the FSM to the right:
  - What is the final value of Q?
  - What is the final state?
  - Answer:
    - Value of Q unknown
    - Final state is C (not D)
- Why?
  - State A: R=99 and Q=R happen simultaneously
  - State B: R not updated with R+1 until next clock cycle, simultaneously with state register being updated

Local registers: R, Q (8-bits)
Considering the high-level state machine shown to the right, what is the final value of D in state F?

1. 10
2. 11
3. 12
4. 17
5. 20

Inputs: D (8-bits), Q (8-bits)
Local Registers: R (8-bits)

RTL Design Pitfalls and Good Practice

- Common pitfall: Reading outputs
  - Outputs should only be written
  - Solution: Introduce additional register, which can be written and read

RTL Design Good Practice

- Good practice: Register all data outputs
  - In fig (a), output P would show spurious values as addition computes
    - Furthermore, longest register-to-register path, which determines clock period, is not known until that output is connected to another component
  - In fig (b), spurious outputs reduced, and longest register-to-register path is clear