**RTL Design Method Examples**

*Digital Design 5.3*

---

**RTL Design**

**RTL Design Method**

<table>
<thead>
<tr>
<th>Step</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Step 1</strong></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><strong>Step 2</strong></td>
<td>Create a datapath Create a datapath to carry out the data operations of the high-level state machine.</td>
</tr>
<tr>
<td><strong>Step 3</strong></td>
<td>Connect the datapath to a controller Connect the datapath to a controller block. Connect external Boolean inputs and outputs to the controller block.</td>
</tr>
<tr>
<td><strong>Step 4</strong></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 Example: Video Compression – Sum of Absolute Differences**

- **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

---

<table>
<thead>
<tr>
<th>Frame 1</th>
<th>Frame 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Digitized frame 1</td>
<td>Digitized frame 2</td>
</tr>
<tr>
<td>1 Mbyte</td>
<td>1 Mbyte</td>
</tr>
</tbody>
</table>

**Difference:**

- Digitized frame 1
  - 1 Mbyte
- Digitized frame 2
  - 0.01 Mbyte

---

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 activities, subject to keeping this copyright notice in place and unmodified. These slides may be posted on course websites or publicly-accessible course servers. PowerPoint source (or pdf with animations) may not be posted for public or widely-accessible websites, but may be posted for classes requiring protected access or distribution directly to students by other electronic means (e.g., course management systems). Any other use requires explicit permission. Instructors may obtain PowerPoint source or obtain special use permissions from Frank Vahid—see http://www.ddvahid.com for information.
**RTL Design**

### RTL Example: 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)

### Step 2: Create datapath

- **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

**Datapath**

- Inputs: A, B (256 byte memory); go (bit)
- Outputs: sad (32 bits)
- Local registers: sum, sad_reg (32 bits); i (9 bits)
**RTL Design**

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

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

![Diagram demonstrating the video compression – sum of absolute differences](image)

**Comparing software and custom circuit SAD**

- Circuit: Two states (S2 & S3) for each \( i \), \( 256 \) \( i \)'s \( \rightarrow \) \( 512 \) clock cycles
- Software: Loop (for \( i = 1 \) to \( 256 \)), but for each \( i \), must move memory to local registers, subtract, compute absolute value, add to sum, increment \( i \) – say about 6 cycles per array item \( \rightarrow \) \( 256 \times 6 = 1536 \) cycles
- Circuit is about 3 times (\( 300\% \)) faster
- Later, we’ll see how to build SAD circuit that is even faster

**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

![Diagram illustrating the high-level state machine](image)

**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
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

---

**Common pitfall:** Reading outputs

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

**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