

# ECE 274 Digital Logic – Fall 2008


## RTL Design Method Examples

*Digital Design 5.3*



# Digital Design

## Chapter 5: RTL Design



Slides to accompany the textbook *Digital Design*, First Edition,  
by Frank Vahid, John Wiley and Sons Publishers, 2007.  
<http://www.ddvahid.com>

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 activities, subject to keeping this copyright notice in place and unmodified. These slides may be posted as [unannotated pdf versions](#) on publicly-accessible course websites, PowerPoint source (or pdf with animations) may [not](#) be posted to publicly-accessible websites, but may be posted for students on internal protected sites or distributed directly to students by other electronic means. Instructors may make [printouts](#) of the slides available to students for a reasonable photocopying charge, without incurring royalties. Any other use requires explicit permission. Instructors may obtain PowerPoint source or obtain special use permissions from Wiley – see <http://www.AbVahid.com> for information.

## RTL Design

*RTL Design Method*

5.2

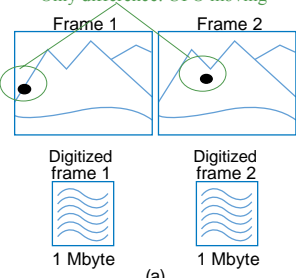
Step	Description
Step 1 <i>Capture a high-level state machine</i>	Describe the system's desired behavior as a high-level state machine. The state machine consists of states and transitions. The state machine is "high-level" because the transition conditions and the state actions are more than just Boolean operations on bit inputs and outputs.
Step 2 <i>Create a datapath</i>	Create a datapath to carry out the data operations of the high-level state machine.
Step 3 <i>Connect the datapath to a controller</i>	Connect the datapath to a controller block. Connect external Boolean inputs and outputs to the controller block.
Step 4 <i>Derive the controller's FSM</i>	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.

3

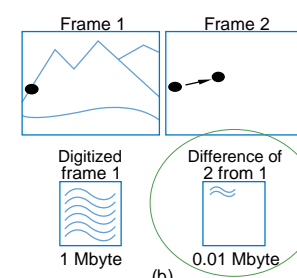
## RTL Design

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

Only difference: UFO moving



(a)



(b)

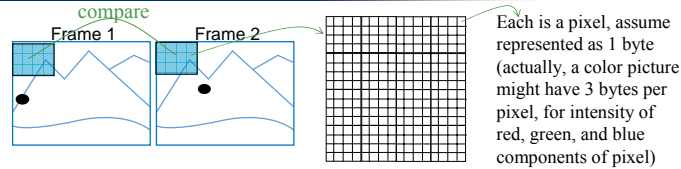
Just send difference

- 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

4

## RTL Design

RTL Example: Video Compression – Sum of Absolute Differences

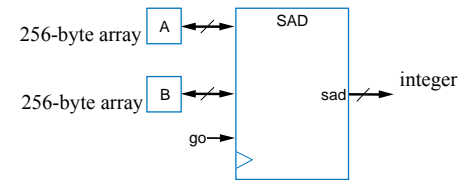


- Need to quickly determine whether two frames are similar enough to just send difference for second frame
  - ▢ Compare corresponding 16x16 “blocks”
    - Treat 16x16 block as 256-byte array
  - ▢ Compute the absolute value of the difference of each array item
  - ▢ Sum those differences – if above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

5

## RTL Design

RTL Example: Video Compression – Sum of Absolute Differences

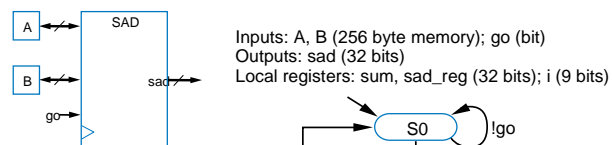


- Want fast sum-of-absolute-differences (SAD) component
  - ▢ When  $go=1$ , sums the differences of element pairs in arrays  $A$  and  $B$ , outputs that sum

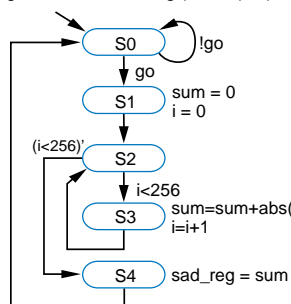
6

## RTL Design

RTL Example: Video Compression – Sum of Absolute Differences



- S0: wait for  $go$
- S1: initialize  $sum$  and  $index$
- S2: check if done ( $i \geq 256$ )
- S3: add difference to  $sum$ , increment index
- S4: done, write to output  $sad\_reg$

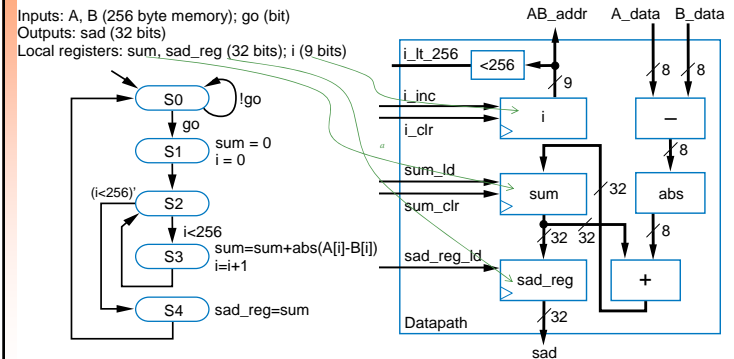


7

## RTL Design

RTL Example: Video Compression – Sum of Absolute Differences

- Step 2: Create datapath

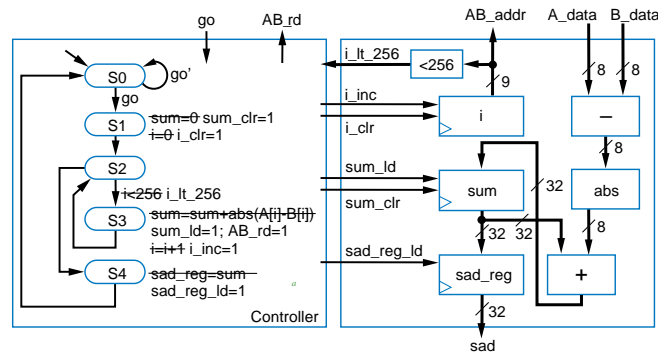


8

## RTL Design

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

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

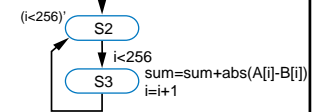


9

## RTL Design

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

- 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  $\rightarrow 256 * 6 = 1536$  cycles
  - Circuit is about 3 times (300%) faster
  - Later, we'll see how to build SAD circuit that is even faster



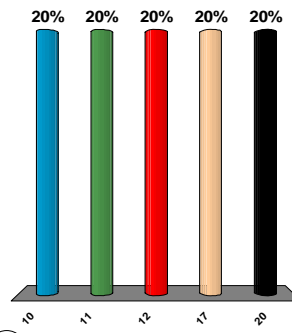
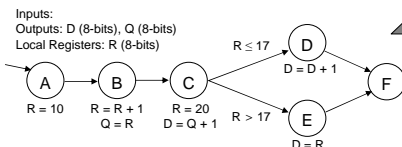
10

## 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?

- 10
- 11
- 12
- 17
- 20

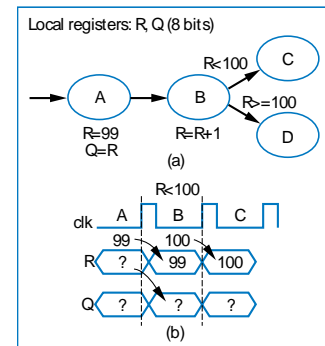


11

## 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

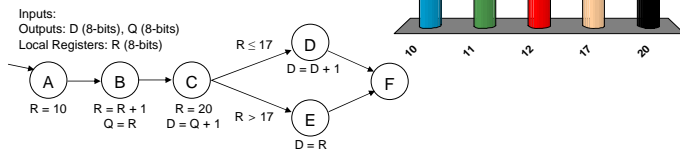


12

## 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?

- 10
- 11
- 12
- 17
- 20

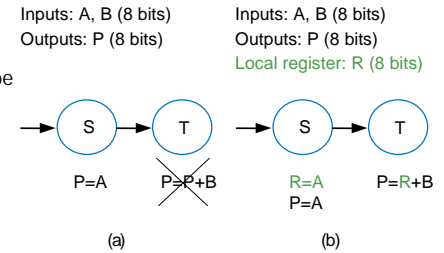


13

## 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

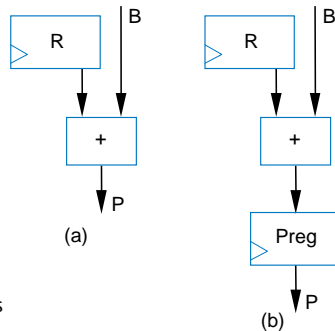


14

## RTL Design Pitfalls and 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



15