


# ECE 274 Digital Logic

## RTL Design – Determining Clock Frequency and Behavioral RTL Design: C-to-Gates

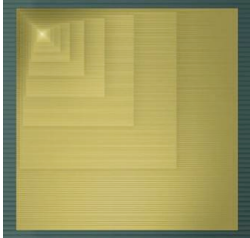
*Digital Design 5.4 – 5.5*



# 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 ~~unmodified~~ 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.ddvahid.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.

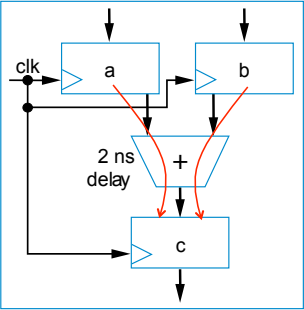
3

## RTL Design

### *Determining Clock Frequency*

5.4

- Designers of digital circuits often want fastest performance
  - Means want high clock frequency
- Frequency limited by *longest register-to-register delay*
  - Known as *critical path*
  - If clock is any faster, incorrect data may be stored into register
  - Longest path on right is 2 ns
    - Ignoring wire delays, and register setup and hold times, for simplicity

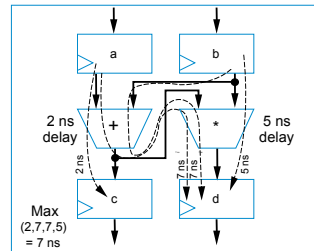


4

## RTL Design

### Critical Path

- Example shows four paths
  - a to c through +: 2 ns
  - a to d through + and \*: 7 ns
  - b to d through + and \*: 7 ns
  - b to d through \*: 5 ns
- Longest path is thus 7 ns
- Fastest frequency
  - $1 / 7 \text{ ns} = 142 \text{ MHz}$

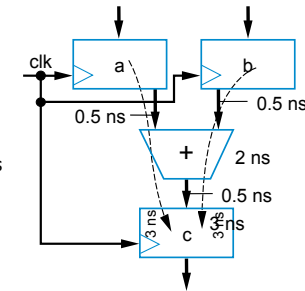


5

## RTL Design

### Critical Path Considering Wire Delays

- Real wires have delay too
  - Must include in critical path
- Example shows two paths
  - Each is  $0.5 + 2 + 0.5 = 3 \text{ ns}$
- Trend
  - 1980s/1990s: Wire delays were tiny compared to logic delays
  - But wire delays not shrinking as fast as logic delays
    - Wire delays may even be greater than logic delays!
- Must also consider register setup and hold times, also add to path
- Then add some time to the computed path, just to be safe
  - e.g., if path is 3 ns, say 4 ns instead

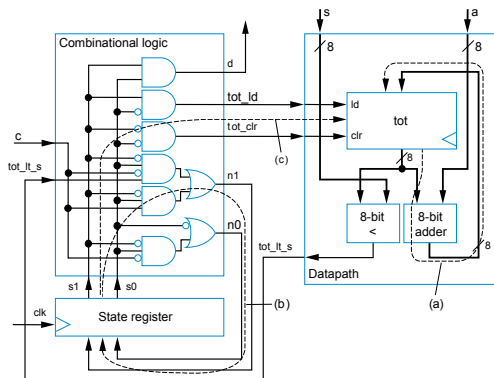


6

## RTL Design

### A Circuit May Have Numerous Paths

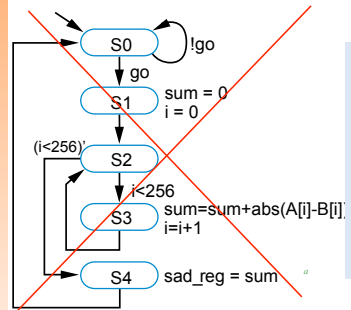
- Paths can exist
  - In the datapath
  - In the controller
  - Between the controller and datapath
  - May be hundreds or thousands of paths
- Timing analysis tools that evaluate all possible paths automatically very helpful



7

## RTL Design

### Behavioral Level Design: C to Gates



```

C code
int SAD (byte A[256], byte B[256]) // not quite C syntax
{
    uint sum; short uint I;
    sum = 0;
    i = 0;
    while (i < 256) {
        sum = sum + abs(A[i] - B[i]);
        i = i + 1;
    }
    return sum;
}
    
```

- Earlier sum-of-absolute-differences example
  - Started with high-level state machine
  - C code is an even better starting point -- easier to understand

8

## RTL Design

Behavioral-Level Design: Start with C (or Similar Language)

- Replace first step of RTL design method by two steps
  - Capture in C, then convert C to high-level state machine
  - How convert from C to high-level state machine?

Step 1A: Capture in C

Step 1B: Convert to high-level state machine

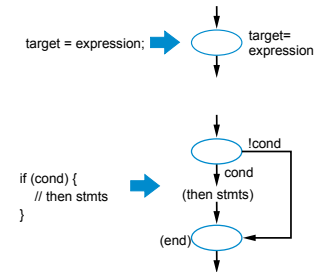
Step	Description
Step 1 <i>Capture a high-level state machine</i>	Describe the system's desired behavior as a high-level state machine. The state machine consists of states and transitions. The state machine is "high-level" because the transition conditions and the state actions are more than just Boolean 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.

9

## RTL Design

Converting from C to High-Level State Machine

- Convert each C construct to equivalent states and transitions
- **Assignment statement**
  - Becomes one state with assignment
- **If-then statement**
  - Becomes state with condition check, transitioning to "then" statements if condition true, otherwise to ending state
    - "then" statements would also be converted to states

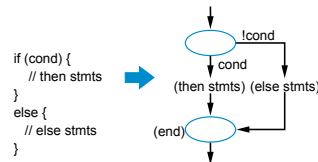


10

## RTL Design

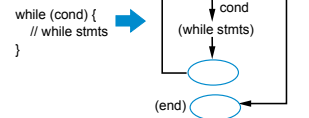
Converting from C to High-Level State Machine

- **If-then-else**
  - Becomes state with condition check, transitioning to "then" statements if condition true, or to "else" statements if condition false



- **While loop statement**

- Becomes state with condition check, transitioning to while loop's statements if true, then transitioning back to condition check



11

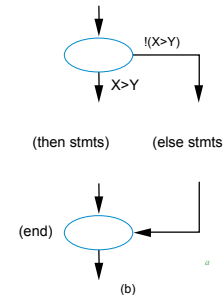
## RTL Design

Simple Example of Converting from C to High-Level State Machine

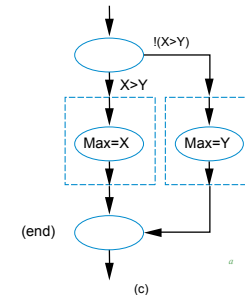
Inputs: uint X, Y  
Outputs: uint Max

```
if (X > Y) {
    Max = X;
}
else {
    Max = Y;
}
```

(a)



(b)



(c)

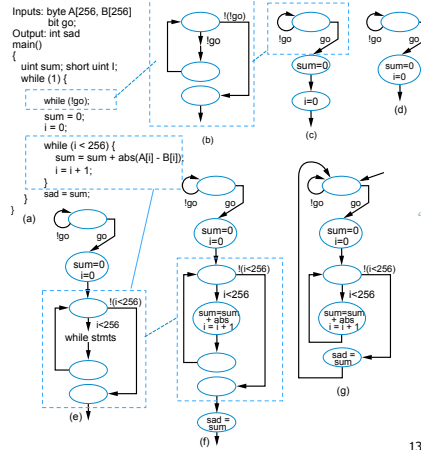
- Simple example: Computing the maximum of two numbers
  - Convert if-then-else statement to states (b)
  - Then convert assignment statements to states (c)

12

## RTL Design

### Example: Sum-of-Absolute-Differences C

- Convert each construct to states
  - Simplify when possible, e.g., merge states
- From high-level state machine, follow RTL design method to create circuit
- Thus, can convert C to gates using straightforward automatable process
  - Not all C constructs can be efficiently converted
  - Use C subset if intended for circuit
  - Can use languages other than C, of course



13