ECE 274 – Digital Logic

RTL Design:
Design Examples

Digital Design (Vahid): Ch. 5.3

Digital Design
Chapter 5:
Register-Transfer Level
(RTL) Design

Slides to accompany the textbook

Digital Design, First Edition,

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 unanimated 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 Examples and Issues

• We’ll use several more examples to illustrate RTL design
• Example: Bus interface
  – Master processor can read register from any peripheral
    • Each register has unique 4-bit address
    • Assume 1 register/periph.
  – Sets rd=1, A=address
  – Appropriate peripheral places register data on 32-bit D lines
    • Peripheral’s address provided on Faddr inputs (maybe from DIP switches, or another register)

5.3

RTL Example: Bus Interface

• Step 1: Create high-level state machine
  – State WaitMyAddress
    • Output "nothing" ("Z") on D, store peripheral’s register value Q into local register Q1
    • Wait until this peripheral’s address is seen (A=Faddr) and rd=1
    – State SendData
    • Output Q1 onto D, wait for rd=0 (meaning main processor is done reading the D lines)

• Step 2: Create datapath
  (a) Datapath inputs/outputs
  (b) Instantiate declared registers
  (c) Instantiate datapath components and connections

Digital Design
Copyright © 2006
Frank Vahid
**RTL Example: Bus Interface**

- Step 3: Connect datapath to controller
- Step 4: Derive controller’s FSM

**Inputs:**
- \( A_{eq} \) (bit)
- \( D_{en} \) (bit)
- \( Q_{ld} \) (bit)
- \( A, Faddr \) (4 bits)
- \( rd \) (bit)

**Outputs:**
- \( D \) (32 bits)

**Local register:**
- \( Q1 \) (32 bits)

**WaitMyAddress**

\[
D = 0 \\
Q1 = Q
\]

\((A = Faddr) \land rd\)

**SendData**

\[
D = Q1 \land rd'
\]

\((A = Faddr) \land rd\')

**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

(a) Digitized frame 2

(b) Digitized frame 1

1 Mbyte

Frame 2

Frame 1

Difference of frame 2 from 1

0.01 Mbyte

Just send difference

**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

**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 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)

**Inputs:**
- \( A, B \) (256 byte memory)
- \( go \) (bit)

**Outputs:**
- \( sad \) (32 bits)

**Local registers:**
- \( sum \), \( sad_reg \) (32 bits)
- \( i \) (9 bits)
RTL Example: Video Compression – Sum of Absolute Differences

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

• Comparing software and custom circuit SAD
  – Circuit: Two states (S2 & S3) for each i, 256 '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 \rightarrow 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

RTL Design Pitfalls and Good Practice

• Common pitfall: Assuming register is update in the state it's written
  – Final value of Q?
  – Final state?
  – Answers may surprise you
    • Value of Q unknown
    • Final state is C, not D
  – Why?
    • State A: R+1 and Q=R happen simultaneously
    • State B: R not updated with R+1 until next clock cycle, simultaneously with state register being updated

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

• Solutions
  – Read register in following state (Q=R)
  – Insert extra state so that conditions use updated value
  – Other solutions are possible, depends on the example

• 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
Control vs. Data Dominated RTL Design

- Designs often categorized as control-dominated or data-dominated
  - Control-dominated design – Controller contains most of the complexity
  - Data-dominated design – Datapath contains most of the complexity
  - General, descriptive terms – no hard rule that separates the two types of designs
  - Laser-based distance measurer – control dominated
  - Bus interface, SAD circuit – mix of control and data
  - Now let’s do a data dominated design

Data Dominated RTL Design Example: FIR Filter

- FIR filter
  - “Finite Impulse Response”
  - Simply a configurable weighted sum of past input values
    \[ y(t) = c_0 x(t) + c_1 x(t-1) + c_2 x(t-2) \]
  - Tens of taps more common
  - RTL design
    - Step 1: Create high-level state machine
      - But there really is none! Data dominated indeed.
    - Go straight to step 2

- Step 2: Create datapath
  - Instantiate registers for c0, c1, c2
  - Instantiate multipliers to compute c*x values
  - Instantiate adders

- Filter concept
  - Suppose X is data from a temperature sensor, and particular input sequence is 180, 180, 181, 240, 180, 181 (one per clock cycle)
  - That 240 is probably wrong!
    - Could be electrical noise
    - Filter should remove such noise in its output Y
  - Simple filter: Output average of last 3 values
    - Small N: less filtering
    - Large N: more filtering, but less sharp output

Data Dominated RTL Design Example: FIR Filter (cont.)

- Step 2: Create datapath (cont.)
  - Instantiate multipliers to compute c*x values

- Supposed sequence is: 180, 181, 240

Data Dominated RTL Design Example: FIR Filter (cont.)

- Instantiate adders

- y(t) = c0*x(t) + c1*x(t-1) + c2*x(t-2)
**Data Dominated RTL Design Example: FIR Filter**

**Step 2: Create datapath (cont.)**
- Add circuitry to allow loading of particular c register

\[ y(t) = c_0 \cdot x(t) + c_1 \cdot x(t-1) + c_2 \cdot x(t-2) \]

**Step 3 & 4: Connect to controller, Create FSM**
- No controller needed
- Extreme data-dominated example
- (Example of an extreme control-dominated design – an FSM, with no datapath)

**Comparing the FIR circuit to a software implementation**
- Circuit
  - Assume adder has 2-gate delay; multiplier has 20-gate delay
  - Longest path goes through one multiplier and two adders
  - 100 x 2 + 20 x 2 = 240 gate delay
  - For 1-megabit filter, would have about a 34-gate delay: 1 multiplier and 7 adders on longest path
- Software
  - 100-tap filter: 100 multiplications, 100 additions. Say 2 instructions per multiplication, 2 per addition. Say 10-gate delay per instruction
  - \((100 \times 2 + 100 \times 2) \times 10 = 4000\) gate delays
- Circuit is more than 100 times faster (10,000% faster). Wow.

**Design Challenge**

- Using the RTL design method, convert the following high-level state machine to a datapath and controller. Design the datapath to structure, but design the controller to the point of an FSM only.

Due Next Lecture (as announced in class)
1 point extra credit (Homework)