CSE140: Components and Design Techniques for Digital Systems

Tajana Simunic Rosing
Announcements and Outline

• Midterm #2 has been graded
• HW #8 due on Tuesday
• Thursday, 5/29, prof. out of town
  – TA will hold the discussion session during the lecture time
• Register Transfer Level design (RTL)
  – Definition
  – Methodology
  – Examples
• Simple CPU design

Sources: TSR, Katz, Boriello, Vahid, Perkowski
CSE140: Components and Design Techniques for Digital Systems

RTL Design Process

Tajana Simunic Rosing
## RTL Design Method

### Step 1: 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.

### Step 2: Create a datapath
Create a datapath to carry out the data operations of the high-level state machine.

### Step 3: Connect the datapath to a controller
Connect the datapath to a controller block. Connect external Boolean inputs and outputs to the controller block.

### Step 4: 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.

Sources: TSR, Katz, Boriello, Vahid, Perkowski
RTL for Datapath & Control

(a) High-level block diagram

(b) Register-transfer-level block diagram
**RTL Design Method Example**

- **Soda dispenser**
  - \( c \): bit input, 1 when coin deposited
  - \( a \): 8-bit input having value of deposited coin
  - \( s \): 8-bit input having cost of a soda
  - \( d \): bit output, processor sets to 1 when total value of deposited coins equals or exceeds cost of a soda

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>0</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>c</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>d</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Sources: TSR, Katz, Boriello, Vahid, Perkowski
Step 1: Capture High-Level State Machine

- Declare local register \(tot\)
- **Init** state: Set \(d=0\), \(tot=0\)
- **Wait** state: wait for coin
  - If see coin, go to **Add** state
- **Add** state: Update total value: \(tot = tot + a\)
  - Remember, \(a\) is present coin’s value
  - Go back to **Wait** state
- In **Wait** state, if \(tot \geq s\), go to **Disp**(ense) state
- **Disp** state: Set \(d=1\) (dispense soda)
  - Return to **Init** state

Not an FSM because:
  - Multi-bit (data) inputs \(a\) and \(s\)
  - Local register \(tot\)
  - Data operations \(tot=0\), \(tot<s\), \(tot=tot+a\).

Useful high-level state machine:
  - Data types beyond just bits
  - Local registers
  - Arithmetic equations/expressions

Sources: TSR, Katz, Boriello, Vahid, Perkowski
Step 2: Create Datapath

- Need \( \text{tot} \) register
- Need 8-bit comparator to compare \( s \) and \( a \)
- Need 8-bit adder to perform \( \text{tot} = \text{tot} + a \)
- Connect everything
- Create control input/outputs

Sources: TSR, Katz, Boriello, Vahid, Perkowski
Step 3: Connect Datapath to a Controller

- Controller’s inputs
  - External input $c$ (coin detected)
  - Input from datapath comparator’s output, which we named $tot_{lt_s}$

- Controller’s outputs
  - External output $d$ (dispense soda)
  - Outputs to datapath to load and clear the $tot$ register

Sources: TSR, Katz, Boriello, Vahid, Perkowski
Step 4 – Derive the Controller’s FSM

- Same states and arcs as high-level state machine
- But set/read datapath control signals for all datapath operations and conditions

Sources: TSR, Katz, Boriello, Vahid, Perkowski
Completing the Design

- Implement the FSM as a state register and logic

**Controller**

- Inputs: c, _tot_lt_s_ (bit)
- Outputs: d, _tot_ld, tot_clr_ (bit)

<table>
<thead>
<tr>
<th>States</th>
<th>s1</th>
<th>s0</th>
<th>c</th>
<th><em>tot_lt_s</em></th>
<th>n1</th>
<th>n0</th>
<th>d</th>
<th><em>tot ld</em></th>
<th><em>tot clr</em></th>
</tr>
</thead>
<tbody>
<tr>
<td>Init</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Wait</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Add</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Disp</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Sources: TSR, Katz, Boriello, Vahid, Perkowski
Laser-based distance measurement – pulse laser, measure time $T$ to sense reflection

- Laser light travels at speed of light, $3 \times 10^8$ m/sec
- Distance is thus $D = T \text{ sec} \times 3 \times 10^8 \text{ m/sec} / 2$
Laser-Based Distance Measurer – I/O Definition

- **Inputs/outputs**
  - *B*: bit input, from button to begin measurement
  - *L*: bit output, activates laser
  - *S*: bit input, senses laser reflection
  - *D*: 16-bit output, displays computed distance

Sources: TSR, Katz, Boriello, Vahid, Perkowski
Step 1: High Level FSM of the Measurer

Inputs: B, S (1 bit each)  Outputs: L (bit), D (16 bits)
Local Registers: Dctr (16 bits)

- Initial state S0: initialize distance to 0 and Laser output to 0
- S1 waits for a button press, once pressed go to a new state S2
- S2 turns on the laser (L=1) - laser pulses!
- S3: set L to 0 and stay in S3 until sense reflection (S) - count cycles for which we are in S3
  - To count, declare local register Dctr, Increment Dctr each cycle in S3
  - Initialize Dctr to 0 in S1. S2 would have been O.K. too
- S4 once reflection detected; calculate distance assuming clk freq 3x10^8, so D=Dctr/2
- After S4, go back to S1 to wait for button again

Sources: TSR, Katz, Boriello, Vahid, Perkowski
Step 2: Create Datapath for the Measurer

Inputs: B, S (1 bit each)
Outputs: L (bit), D (16 bits)
Local Registers: Dctr (16 bits)

3 substeps for a high level FSM
- (a) Make data inputs/outputs be datapath inputs/outputs
- (b) Instantiate declared registers into the datapath (also instantiate a register for each data output)
- (c) Examine every state and transition, and instantiate datapath components and connections to implement any data computations
Step 3: Connecting the Datapath to a Controller

- Easy – just connect all control signals between controller and datapath

Sources: TSR, Katz, Boriello, Vahid, Perkowski
Step 4: Deriving the Controller’s FSM

- FSM has same structure as high-level state machine
  - Inputs/outputs all bits now
  - Replace data operations by bit operations using datapath

Inputs: B, S (1 bit each) Outputs: L (bit), D (16 bits)
Local Registers: Dctr (16 bits)

Outputs: L, Dreg_clr, Dreg_ld, Dctr_clr, Dctr_cnt

Sources: TSR, Katz, Boriello, Vahid, Perkowski
Step 4: Deriving the Controller’s FSM – Short Notation

Inputs: B, S
Outputs: L, DregClr, DregLd, DctrClr, DctrCnt

Sources: TSR, Katz, Boriello, Vahid, Perkowski
RTL Design Example: Bus Interface

- **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
    - Periph’s address provided on $Faddr$ inputs (maybe from DIP switches, or another register)

Sources: TSR, Katz, Boriello, Vahid, Perkowski
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 a datapath

(a) Datapath inputs/outputs
(b) Instantiate declared registers
(c) Instantiate datapath components and connections

Inputs: rd (bit); Q (32 bits); A, Faddr (4 bits)
Outputs: D (32 bits)
Local register: Q1 (32 bits)
Step 3: Connect datapath to controller

Step 4: Derive controller’s FSM

Inputs: rd, A_eq_Faddr (bit)
Outputs: Q1_ld, D_en (bit)

D_en = 0
Q1_ld = 1

D_en = 1
Q1_ld = 0

Sources: TSR, Katz, Boriello, Vahid, Perkowski
**RTL Example: Video Compression**

- 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

Sources: TSR, Katz, Boriello, Vahid, Perkowski
Video Compression – Sum of Absolute Differences

Each is a pixel, assume represented as 1 byte (actually, a color picture might have 3 bytes per pixel, for intensity of red, green, and blue components of pixel).

- If two frames are similar just send a difference instead
  - Compare corresponding 16x16 “blocks”
    - Treat 16x16 block as 256-byte array
  - Compute the absolute value of the difference of each array item
  - Sum the differences
    - if above a threshold, send a complete frame for second frame
    - Else send the difference

Sources: TSR, Katz, Boriello, Vahid, Perkowski
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
Step 1: High-level FSM

- **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

Inputs: A, B (256 byte memory); go (bit)
Outputs: sad (32 bits)
Local registers: sum, sad\_reg (32 bits); i (9 bits)

Sources: TSR, Katz, Boriello, Vahid, Perkowski
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)

Datapath

Sources: TSR, Katz, Boriello, Vahid, Perkowski
Step 3: Connect to controller

Step 4: Replace high-level state machine by FSM

Step 4: Replace high-level state machine by FSM

Sources: TSR, Katz, Boriello, Vahid, Perkowski
Control vs. Data Dominated RTL Design

- Control-dominated design
  - Controller contains most of the complexity
- Data-dominated design
  - Datapath contains most of the complexity
- Laser-based distance measurer – control dominated
- Bus interface, SAD circuit – mix of control and data

Sources: TSR, Katz, Boriello, Vahid, Perkowski
Data Dominated RTL Design Example: FIR Filter

- **FIR filter**
  - “Finite Impulse Response”
  - Simply a configurable weighted sum of past input values
  - \( y(t) = c_0 x(t) + c_1 x(t-1) + c_2 x(t-2) \)
    - Above known as “3 tap”
    - Tens of taps more common
    - Very general filter – User sets the constants \((c_0, c_1, c_2)\) to define specific filter

RTL design
Step 1: Create high-level state machine – there is none
Go straight to step 2

Sources: TSR, Katz, Boriello, Vahid, Perkowski
**Step 2: Create datapath**

- Begin by creating chain of xt registers to hold past values of X
- Instantiate registers for c0, c1, c2
- Instantiate multipliers to compute c*x values
- Instantiate adders
- Add circuitry to allow loading of particular c register

\[ y(t) = c0 \times x(t) + c1 \times x(t-1) + c2 \times x(t-2) \]

**Step 3 & 4:**
Connect to controller,
Create FSM
No controller needed

Sources: TSR, Katz, Boriello, Vahid, Perkowski
Comparing the FIR circuit to a software implementation

- **Circuit**
  - Adder has 2-gate delay, multiplier has 20-gate delay
  - Longest path goes through one multiplier and two adders
    - $20 + 2 + 2 = 24$-gate delay
  - 100-tap filter, would have about a 34-gate delay: 1 multiplier and 7 adders on longest path

- **Software**
  - 100-tap filter: 100 multiplications, 100 additions.
  - If 2 instructions per multiplication, 2 per addition. Say 10-gate delay per instruction.
  - $(100 \times 2 + 100 \times 2) \times 10 = 4000$ gate delays
Determining 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

Sources: TSR, Katz, Boriello, Vahid, Perkowski
Critical Path Considering Wire Delays

- 1980s/1990s: Wire delays tiny compared to logic
- Today wire delay dominates
- Also consider register Tsu & Th

Sources: TSR, Katz, Boriello, Vahid, Perkowski
Example: Critical Path Analysis

- 4 paths:
  - a to c through +: 2 ns
  - a to d through + and *: 7 ns - longest!!!
  - b to d through + and *: 7 ns - longest!!!
  - b to d through *: 5 ns
- Fastest frequency: $1 / 7 \text{ ns} = 142 \text{ MHz}$

Sources: TSR, Katz, Boriello, Vahid, Perkowski
A Circuit May Have Numerous Paths

Sources: TSR, Katz, Boriello, Vahid, Perkowski
RTL Design Issues

Local registers: R, Q (8 bits)

(a)

• Pitfall: Assuming register is updated in the state it’s written
  – Value of Q unknown & final state is C, not D; solution: Read register in following state (Q=R)
  – Reason:
    • State A: R=99 and Q=R happen simultaneously
    • State B: R not updated with R+1 until next clock cycle, simultaneously with state being updated
RTL Design Issues

Inputs: A, B (8 bits)  Outputs: P (8 bits)

Local register: R (8 bits)

Sources: TSR, Katz, Boriello, Vahid, Perkowski

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

• Datapath and Control Design
• RTL Design Steps
  1. Define the high level state machine
  2. Create datapath
  3. Connect datapath with control
  4. Implement the FSM
• Control vs. Data dominated RTL
• Timing analysis – critical path, processor frequency
• RTL Design Issues

Sources: TSR, Katz, Boriello, Vahid, Perkowski