CSE140: Components and Design Techniques for Digital Systems

Register Transfer Level (RTL) Design
based on Vahid chap. 5

Tajana Simunic Rosing
# RTL Design Method

<table>
<thead>
<tr>
<th>Step</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Step 1</td>
<td><strong>Capture a high-level state machine</strong></td>
</tr>
<tr>
<td></td>
<td>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>Step 2</td>
<td><strong>Create a datapath</strong></td>
</tr>
<tr>
<td></td>
<td>Create a datapath to carry out the data operations of the high-level state machine.</td>
</tr>
<tr>
<td>Step 3</td>
<td><strong>Connect the datapath to a controller</strong></td>
</tr>
<tr>
<td></td>
<td>Connect the datapath to a controller block. Connect external Boolean inputs and outputs to the controller block.</td>
</tr>
<tr>
<td>Step 4</td>
<td><strong>Derive the controller’s FSM</strong></td>
</tr>
<tr>
<td></td>
<td>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 Design example: Laser-Based Distance Measurer

- 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 $D = T \text{ sec} \times 3 \times 10^8 \text{ m/sec} / 2$

Object of interest

$2D = T \text{ sec} \times 3 \times 10^8 \text{ m/sec}$
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)

B'  S'

S0  S1  S2  S3  S4
L = 0  D = 0  L = 1  L=0  D = Dctr / 2 (calculate D)

Dreg_clr = 1  Dreg clr  Dreg ld  Dctr clr  Dctr cnt
(laser off)  (clear reg)  (clear count)  (laser on)  (count up)
Dreg_clr = 0  Dreg ld = 0  Dctr clr = 0  Dctr cnt = 0  (load D reg with Dctr/2)
(laser on)  (clear count)  (laser on)  (stop counting)
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>c</th>
<th>a</th>
<th>s</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>100</td>
<td>100</td>
</tr>
</tbody>
</table>
```

Soda dispenser processor: tot: 50

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

- Declare local register \( \text{tot} \)
- **Init** state: Set \( d=0, \text{tot}=0 \)
- **Wait** state: wait for coin
  - If see coin, go to **Add** state
- **Add** state: Update total value: \( \text{tot} = \text{tot} + a \)
  - Remember, \( a \) is present coin’s value
  - Go back to **Wait** state
- In **Wait** state, if \( \text{tot} \geq s \), 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 \( \text{tot} \)
- Data operations \( \text{tot}=0, \text{tot}<s, \text{tot}=\text{tot}+a \).

Useful high-level state machine:
- Data types beyond just bits
- Local registers
- Arithmetic equations/expressions
Step 2: Create Datapath

- Need `tot` register
- Need 8-bit comparator to compare `s` and `a`
- Need 8-bit adder to perform `tot = tot + a`
- Connect everything
- Create control input/outputs

![Datapath Diagram]

**Inputs:** `c` (bit), `a` (8 bits), `s` (8 bits)
**Outputs:** `d` (bit)
**Local registers:** `tot` (8 bits)

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 \( \text{tot}_{-}lt_{-}s \)

- Controller’s outputs
  - External output \( d \) (dispense soda)
  - Outputs to datapath to load and clear the \( \text{tot} \) register
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
Completing the Design

- Implement the FSM as a state register and logic

Inputs: c, tot_lt_s (bit)
Outputs: d, tot_ld, tot_clr (bit)

<table>
<thead>
<tr>
<th></th>
<th>s1</th>
<th>s0</th>
<th>c</th>
<th>tot_lt_s</th>
<th></th>
<th>n1</th>
<th>n0</th>
<th>d</th>
<th>tot_ld</th>
<th>tot_clr</th>
</tr>
</thead>
<tbody>
<tr>
<td>Init</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td></td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td></td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td></td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Wait</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td></td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td></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>0</td>
<td></td>
<td>1</td>
<td>0</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></td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Add</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td></td>
<td>0</td>
<td>0</td>
<td>1</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></td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td>...</td>
</tr>
</tbody>
</table>

Controller

Init > Add > Disp
Disp > tot_clr = 1
Init > d = 0
Wait > tot_clr = 1
Init > tot_clr = 1
Wait > tot_clr = 1
## 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)
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
Step 3: Connect datapath to controller
Step 4: Derive controller’s FSM

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

WaitMyAddress
D_en = 0
Q1_ld = 1
(A_eq_Faddr and rd)

SendMessage
A_eq_Faddr and rd
D_en = 1
Q1_ld = 0

Bus interface

Datapath
A
Faddr
Q
4
4
32
32
32
32
D
Id
Q1
= (4-bit)

A_eq_Faddr
D_en

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

Digitized frame 1
1 Mbyte

Digitized frame 2
1 Mbyte

Difference of 2 from 1
0.01 Mbyte

Only difference: ball moving

Just send difference

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

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

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)
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)
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

AB_addr  A_data  B_data
\[ \begin{array}{c}
\text{i\_lt\_256} \\
i\_inc \\
i\_clr \\
\text{sum\_ld} \\
n\text{sum\_clr} \\
\text{sad\_reg\_ld} \\
s\text{sad\_reg} \\
s\text{sad}
\end{array} \]

\[ \begin{array}{c}
\text{<256} \\
9 \\
8 \quad 8 \\
32 \\
32 \quad 32 \\
32 \\
32
\end{array} \]
Step 3: Connect to controller
Step 4: Replace high-level state machine by FSM

S0
S1
S2
S3
S4

Controller

go
AB_rd

go'

sum=0 sum_clr=1
i=0 i CLR=1

i<256 i_lt_256

sum=sum+abs(A[i]-B[i])

i=i+1 i_inc=1

sad_reg=sum sad_reg ld=1

<256

i

i_clr

i_inc

sum

abs

sum_clr

sum_ld

sad_reg

sad_reg ld

sad

Controller

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

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

Step 3 & 4:
Connect to controller,
Create FSM
No controller needed.
Comparing the FIR circuit to a software implementation

- **Circuit**
  - Adder has 2-gate delay, multiplier has 20-gate delay
  - Longest past 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

\[ y(t) = c_0 \times x(t) + c_1 \times x(t-1) + c_2 \times x(t-2) \]
Critical path analysis in more complex designs

Sources: TSR, Katz, Boriello, Vahid, Perkowski
Simple data encryption/decryption device

- B = 1, set offset O = I [0:31]
- B=0 e=1: encrypt mode: output J = I + O
- B=0 e=0; decrypt mode: get I = J - O
Reaction timer

- On reset ($rst$) reaction timer waits for 10 sec before turning on light ($len=1$)
- Measures the length of time $rtime$ (ms) until user presses button $B$
  - If reaction slower than 2sec, output $slow=1$ and $rtime=2000$
Fast sum of 16 32-bit registers
Hot water detector

- Output warning when average temp over the past 4 samples exceeds a user defined value; clr disables the system
- Inputs (32 bit): CT – current temp; WT – warning temp
- Output: W – high if hot temperature; stays on until clr pressed again
Design from “C” code

Inputs: byte a, byte b, bit go
Outputs: byte gcd, bit done
GCD:
while(1) {
    while(!go);
    done = 0;
    while ( a != b ) {
        if( a > b ) {
            a = a - b;
        } else {
            b = b - a;
        }
    }
    gcd = a;
    done = 1;
}
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

• Timing analysis – critical path in more complex circuits
  – Watch out for all possible long paths (e.g. datapath to FSM, FSM control logic, datapath logic etc)