Lecture 21:
RTL Design

CSE 140: Components and Design Techniques for Digital Systems

Diba Mirza
Dept. of Computer Science and Engineering
University of California, San Diego
## RTL Design

- **Combinational design (Logic Level)**
  - *Capture* Comb. behavior: Equations, truth tables
  - *Convert* to circuit: AND + OR + NOT → Comb. Logic

- **Controller Design (Logic Level)**
  - *Capture* sequential behavior: FSMs
  - *Convert* to circuit: Register + Comb. logic → Controller

- **Processor Design (Register Transfer Level)**

- **Standard Modules** (used in Processor design)
  - Adders, shifters, counters, decoder, muxes,…

---

Levels of digital design abstraction
RTL Design

- Processor Design (Register Transfer Level)
  - Capture behavior: High-level state machine or pseudocode
  - Convert to circuit: Controller + Datapath → Processor
  - Known as “RTL” (register-transfer level) design

Levels of digital design abstraction

Processors:
- Programmable (microprocessor)
- Custom
High-Level State Machines (HLSMs)

- Some behaviors too complex for equations, truth tables, or FSMs
- Ex: 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
High-Level State Machines (HLSMs)

• Which of the following makes the FSM design of this problem difficult?
  A. 8-bit input/output
  B. Tracking the current total
  C. All of the above
HLSMs

- High-level state machine (HLSM) extends FSM with:
  - Multi-bit input/output
  - Local storage
  - Arithmetic operations

- Conventions
  - Numbers:
    - Single-bit: '0' (single quotes)
    - Integer: 0 (no quotes)
    - Multi-bit: “0000” (double quotes)
  - == for equal, := for assignment
  - Multi-bit outputs must be registered via local storage
  - // precedes a comment
Q: How does the HLSM differ from the FSM for this problem?

A. The HLSM requires storing data, but the FSM doesn’t
B. The FSM required storing the state but the HLSM doesn’t
C. Implementing the HLSM requires registers, registers are not required to implement the FSM

**Inputs:** c (bit), a (8 bits), s (8 bits)

**Outputs:** d (bit) // '1' dispenses soda

**Local storage:** tot (8 bits)

```
SodaDispenser

Init

Wait

tot:=0

c:=0

d:=0

Add

tot:=tot+a

c:=c

d:=d

Disp

tot==(tot<s)
c:=c

d:=1
```
Q: Which of the following is common between HLSMs and FSMs?
A. Transitions happen at the rising edge of the clock
B. They both have external data and control inputs and outputs
RTL Design Process

- **Capture** behavior
- **Convert** to circuit
  - High-level architecture (datapath and controlpath)
  - Datapath capable of HLSM's data operations
  - Controller to control datapath
### RTL Design Process

- **Capture behavior**
- **Convert to circuit**
  - High-level architecture (datapath and controlpath)
  - Datapath capable of HLSM's data operations
  - Controller to control datapath

### Table: RTL Design Process

<table>
<thead>
<tr>
<th>Step</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Step 1:</strong> Capture behavior</td>
<td><strong>Capture a high-level state machine</strong>&lt;br&gt;Describe the system's desired behavior as a high-level state machine. The state machine consists of states and transitions. The state machine is &quot;high-level&quot; because the transition conditions and the state actions are more than just Boolean operations on single-bit inputs and outputs.</td>
</tr>
<tr>
<td>2A</td>
<td><strong>Create a datapath</strong>&lt;br&gt;Create a datapath to carry out the data operations of the high-level state machine.</td>
</tr>
<tr>
<td><strong>Step 2:</strong> Convert to circuit</td>
<td><strong>Connect the datapath to a controller</strong>&lt;br&gt;Connect the datapath to a controller block. Connect external control inputs and outputs to the controller block.</td>
</tr>
<tr>
<td>2B</td>
<td><strong>Derive the controller's FSM</strong>&lt;br&gt;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>
Design datapath (2A)

Step 1

Inputs: c (bit), a (8 bits), s (8 bits)
Outputs: d (bit) // '1' dispenses soda
Local storage: tot (8 bits)

Init
- tot := 0
- d := '0'

Wait
- c' *(tot<s)
- tot := tot + a
- c
- d := '1'

Disp

Soda Dispenser

Step 2A

Datapath

8-bit <

tot ld

tot clr

tot lt s

Controller

Tot ld

tot cl r

tot lt s

c

8-bit adder

8

Step 2B

d := '0'
tot := 0

c'*(tot<s)
c

d := '1'
tot := tot + a

What do we need for DP?
Design the datapath (2A)

Inputs: c (bit), a (8 bits), s (8 bits)
Outputs: d (bit) // '1' dispenses soda
Local storage: tot (8 bits)

Q: According to the current design, under which of the following conditions does the register output ‘tot’ change?

A. Whenever the value of the coin inserted (‘a’) changes
B. Whenever the cost of the soda (‘s’) changes
C. When the signal tot_ld becomes high
D. When the signal tot_clr becomes high
Connect data path to controller (2B)

Step 2B

Controller

Datapath

c

d

tot_ld

tot_clr

tot_lt_s

Step 2A

Datapath

8-bit

<

8-bit adder

ld

clr

tot

s

8

a

8

8
**Design control path FSM**

**Inputs:** c (bit), a (8 bits), s (8 bits)

**Outputs:** d (bit) // '1' dispenses soda

**Local storage:** tot (8 bits)

---

**SodaDispenser**

**Step 1**

- **Init**
  - d = '0'
  - tot = 0

- **Wait**
  - c
  - c' \cdot (\text{tot < s})
  - tot := tot + a

- **Disp**
  - d = '1'

---

**Step 2B**

- **Controller**
  - c
  - d

- **Datapath**
  - tot ld
  - tot clr
  - tot lt s

---

**Step 2C**

- **Controller**
  - c
  - d

- **Add**
  - tot ld = 1
  - tot clr = 1

- **Disp**
  - d = 1

---

**Controller**

- **Add**
  - tot ld = 1
  - tot clr = 1

- **Disp**
  - d = 1

---

**Inputs:** c, tot lt s (bit)

**Outputs:** d, tot ld, tot clr (bit)

---

Digital Design 2e
Copyright © 2010
Frank Vahid
Implement controller FSM

Use controller (FSM) design process from previous lectures to complete the design
RTL Design Process—Step 2A: Create a datapath

- Need component library from which to choose

```
clk^ and clr=1: Q=0
clk^ and ld=1: Q=I
else Q stays same
S = A+B

(unsigned)
A<B: lt=1
A=B: eq=1
A>B: gt=1
shiftL1: <<1
shiftL2: <<2
shiftR1: >>1
...`
Step 2A: Create a Datapath—Simple Examples

(a) \( P_{\text{reg}} = X + Y + Z \)

(b) \( P_{\text{reg}} = P_{\text{reg}} + X \)

(c) \( P_{\text{reg}} = X + Y; \quad \text{reg}Q = Y + Z \)

(d) \( k=0: P_{\text{reg}} = Y + Z \quad k=1: P_{\text{reg}} = X + Y \)
More RTL Design

- Additional datapath components

\[
S = A - B \quad \text{(signed)}
\]
\[
P = A \times B \quad \text{(unsigned)}
\]
\[
Q = |A| \quad \text{(unsigned)}
\]

- For subtraction, \(S = A - B\) when \(A\) and \(B\) are signed.
- For multiplication, \(P = A \times B\) is unsigned.
- For absolute value, \(Q = |A|\) is unsigned.

- \(clk^\wedge\) and \(clr=1\): \(Q=0\)
- \(clk^\wedge\) and \(inc=1\): \(Q=Q+1\)
- Else \(Q\) stays the same.
Summary

- Modern digital design involves creating processor-level components
- High-level state machines
- RTL design process
  - 1. Capture behavior: Use HLSM
  - 2. Convert to circuit
    - A. Create datapath  B. Connect DP to controller  C. Derive controller FSM