Datapath Components – Multifunction

Registers

- Can store data, very common in datapaths
- Basic register of Ch 3: Loaded every cycle
  - Useful for implementing FSM -- stores encoded state
  - For other uses, may want to load only on certain cycles

Basic register loads on every clock cycle

How extend to only load on certain cycles?

Add 2x1 mux to front of each flip-flop
- Register’s load input selects mux input to pass
  - Either existing flip-flop value, or new value to load
Datapath Components

Register Example: Above-Mirror Display

- Instead of connecting car’s computer to display using 32 wires, can we use fewer wires?
- To reduce wires: Car’s computer can write 1 value at a time, loads into one of four registers with display

Datapath Components

Shift Registers

- Shift right
  - Move each bit one position right
  - Shift in 0 to leftmost bit
- Shift Register
  - Connect register’s flip-flop’s outputs to next flip-flop’s input
  - This design would always shift on every clock cycle
  - How can we control it?

Datapath Components

Shift Registers

- What is the result after shifting 10011 four times to the right?
  1. 10011
  2. 00010
  3. 10000
  4. 00001

Datapath Components

Shift Register

- To allow register to either shift or retain, use 2x1 muxes
  - shr: 0 means retain, 1 shift
  - shr_in: value to shift in
    - May be 0, or 1
  - Note: Can easily design shift register that shifts left instead
Datapath Components

**Rotate Register**

- **Rotate right**
  - Like shift right, but leftmost bit comes from rightmost bit

![Diagram of Rotate Register](image)

**Shift Register Example: Above-Mirror Display**

- Earlier example: 8 wires from car's computer to above-mirror display's four registers
  - Better than 32 wires, but 11 still a lot — want fewer for smaller wire bundles
- Use shift registers
  - Wires: 1 + 2 + 1 = 4
  - Computer sends one value at a time, one bit per clock cycle

![Diagram of Shift Register](image)

Datapath Components

**Multifunction Registers**

- Many registers have multiple functions
  - Load, shift, clear (load all 0s)
  - And retain present value, of course
- Easily designed using muxes
  - Just connect each mux input to achieve desired function

![Diagram of Multifunction Registers](image)

**Functions:**

<table>
<thead>
<tr>
<th>$s_1$</th>
<th>$s_0$</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>Maintain present value</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>Parallel load</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>Shift right</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>Shift left</td>
</tr>
</tbody>
</table>

![Table of Multifunction Registers](image)
Datapath Components
Multifunction Registers with Separate Control Inputs

<table>
<thead>
<tr>
<th>Id</th>
<th>shr</th>
<th>shl</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>Maintain present value</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>Shift left</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>Shift right</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>Parallel load</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>Parallel load – Id has priority</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>Parallel load – Id has priority</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>Parallel load – Id has priority</td>
</tr>
</tbody>
</table>

Truth table for combinational circuit

\[
s_1 = \text{ld} \ast \text{shl} + \text{ld} \ast \text{shr} + \text{ld} \ast \text{shl} + \text{ld} \\

s_0 = \text{ld} \ast \text{shl} + \text{ld}
\]

Datapath Components
Register Operation Table

- Register operations typically shown using compact version of table
  - X means same operation whether value is 0 or 1
    - One X expands to two rows
    - Two Xs expand to four rows
  - Put highest priority control input on left to make reduced table simple

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Outputs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Id</td>
<td>shr</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Datapath Components
Register Design Process

- Can design register with desired operations using simple four-step process

**TABLE 4.1** Four-step process for designing a multifunction register.

<table>
<thead>
<tr>
<th>Step</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>Determine max size: Count the number of operations (don’t forget the maintain present value operation!) and add to front of each flip-flop a mux with at least that number of inputs.</td>
</tr>
<tr>
<td>2.</td>
<td>Create mux operation table: Create an operation table defining the desired operation for each possible value of the input select lines.</td>
</tr>
<tr>
<td>3.</td>
<td>Connect mux inputs: For each operation, connect the corresponding mux data input to the appropriate external input or flip-flop input (possibly passing through some logic) to achieve the desired operation.</td>
</tr>
<tr>
<td>4.</td>
<td>Map control lines: Create a truth table that maps external control lines to the internal mux select lines, with appropriate priorities, and then design the logic to achieve this mapping.</td>
</tr>
</tbody>
</table>

Datapath Components
Register Design Example

- Desired register operations
  - Load, shift left, synchronous clear, synchronous set

**Step 1:** Determine mux size

5 operations: above, plus maintain present value (don’t forget this one!)

\[ \text{Use 8x1 mux} \]

**Step 2:** Create mux operation table

**Step 3:** Connect mux inputs

**Step 4:** Map control lines

\[
s_2 = \text{clr} \ast \text{set} \\

s_1 = \text{clr} \ast \text{set} \ast \text{ld} \ast \text{shl} + \text{clr} \\

s_0 = \text{clr} \ast \text{set} \ast \text{ld} + \text{clr}
\]
Datapath Components
Register Design Example

Step 4: Map control lines

- \( s2 = \text{clr} \cdot \text{set} \)
- \( s1 = \text{clr} \cdot \text{set} \cdot \text{ld} \cdot \text{shl} + \text{clr} \)
- \( s0 = \text{clr} \cdot \text{set} \cdot \text{ld} \cdot \text{clr} \)

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Outputs</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>clr, set, ld, shl</td>
<td>s2, s1, s0</td>
<td></td>
</tr>
</tbody>
</table>