EECS150 - Digital Design
Lecture 21 - High-level Design and Optimization
April 16, 2002
John Wawrzynek

Introduction
• High-level Design Specifies:
  – How data is moved around and operated on.
  – The architecture (sometimes called micro-architecture):
    • The organization of state elements and combinational logic blocks
    • Functional specification of combinational logic blocks
• Optimization
  – Deals with the task of modifying a architecture and data movement procedure to meet some particular design requirement:
    • performance, cost, power, or some combination.
• Most designers spend most of their time on high-level organization and optimization
  – modern CAD tools help fill in the low-level details and optimization
    • gate-level minimization, state-assignment, etc.
  – A great deal of the leverage on effecting performance, cost, and power comes at the high-level.

A Standard High-level Organization
• Controller
  – accepts control input, generates control output and sequences the movement of data in the datapath.
• Datapath
  – is responsible for data manipulation. Usually includes a limited amount of storage.
• Memory
  – optional block used for long term storage of data structures.

A Standard Model for CPUs, micro-controllers, many other digital sub-systems.
Usually not nested.
Often cascaded:

A Standard High-level Organization

Register Transfer Level Descriptions
• A standard high-level representation for describing systems.
• It follows from the fact that all synchronous digital system can be described as a set of state elements connected by CL blocks:
• RTL comprises a set of register transfers with optional operators as part of the transfer.
• Example:
  \[
  \text{regA} \leftarrow \text{IN}; \\
  \text{regB} \leftarrow \text{IN}; \\
  \text{regC} \leftarrow \text{regA} + \text{regB}; \\
  \text{regB} \leftarrow \text{regC};
  \]
  \( \text{if (start==1)} \text{regA} \leftarrow \text{regC} \)
• Personal style:
  – use "]->" to separate transfers that occur on separate cycles.
  – Use "}" to separate transfers that occur on the same cycle.
• Example (2 cycles):
  
  In this case: RTL description is used to sequence the operations on the datapath.
  It becomes the high-level specification for the controller.
  Design of the FSM controller follows directly from the RTL sequence. FSM controls movement of data by controlling the multiplexor control signals.

Example of Using RTL

Example of Using RTL

Example of Using RTL

What does the datapath look like:

The controller:
**List Processor Example**

- RTL gives us a framework for making high-level optimizations.

- General design procedure outline:
  1. Problem, Constraints, and Component Library Spec.
  2. "Algorithm" Selection
  3. Micro-architecture Specification
  4. Analysis of Cost, Performance, Power
  5. Optimizations, Variations
  6. Detailed Design

---

**1. Problem Specification**

- Design a circuit that forms the sum of all the 2's complements integers stored in a linked-list structure starting at memory address 0:

- All integers and pointers are 8-bit. The link-list is stored in a memory block with an 8-bit address port and 8 bit data port, as shown below. The pointer from the last element in the list is 0.

---

**1. Other Specifications**

- Design Constraints:
  - Usually the design specification puts a restriction on cost, performance, power or all. We will leave this unspecified for now and return to it later.

- Component Library:

<table>
<thead>
<tr>
<th>Component</th>
<th>delay</th>
</tr>
</thead>
<tbody>
<tr>
<td>n-bit register</td>
<td>0.5ns</td>
</tr>
<tr>
<td>n-bit 2-1 multiplexer</td>
<td>3ns</td>
</tr>
<tr>
<td>n-bit adder</td>
<td>10ns read (asynchronous read)</td>
</tr>
<tr>
<td>memory</td>
<td>10ns read (asynchronous read)</td>
</tr>
<tr>
<td>zero compare</td>
<td>0.5 log(n)</td>
</tr>
</tbody>
</table>

  Are these reasonable?

---

**2. Algorithm Specification**

- In this case the memory only allows one access per cycle, so the algorithm is limited to sequential execution. If in another case more input data is available at once, then a more parallel solution may be possible.

- Assume datapath state registers NEXT and SUM.

  ```
  if (START==1) NEXT<->0, SUM<->0; 
  until (NEXT==0) {
      SUM<->SUM + Memory[NEXT+1];
      NEXT<->Memory[NEXT];
  }
  R<->SUM, DONE<->1;
  ```

---

**3. Architecture #1**

*Direct implementation of RTL description:*

---

**New Component**

- Register with Load Enable:

  - Allows register to be loaded on selected clock posedge or to retain its previous value.
4. Analysis of Cost, Performance, and Power

- Skip Power for now.
- Cost:
  - How do we measure it? # of transistors? # of gates? # of CLBs?
  - Depends on implementation technology. Usually we are interested in comparing the relative cost of two competing implementations.
    (Save this for later)
- Performance:
  - 2 clock cycles per number added.
  - What is the minimum clock period?
  - Detailed timing next page:

4. Analysis of Performance

- Detailed timing:
  - Clock period (T) = max (clock period for each state)
  - T > 32ns, F < 31 MHz
  - Assumes that the controller delay does not limit the performance.
- Conclusion:
  - COMPUTE_SUM state does most of the work. Most of the components are inactive in GET_NEXT state.
  - GET_NEXT does: Memory access + …
  - COMPUTE_SUM does: 8-bit add, memory access, 15-bit add + …

  Move one of the adds to GET_NEXT.

5. Optimization

- Architecture #2:
- Incremental cost:
  - addition of another clearable, load_enabled register.

5. Optimization, Architecture #2

- New timing:
  - Clock Period (T) = max (clock period for each state)
  - T > 24ns, F < 41.67MHz
  - Is this worth the extra cost?
  - Can we lower the cost?
  - Notice that the circuit now only performs one add on every cycle. Why not share the adder for both cycles?
5. Optimization, Architecture #3

• Datapath:

• Incremental cost:
  – Addition of another mux and control. Removal of an 8-bit adder.

• Performance:
  – mux adds 1ns to cycle time. 25ns, 40MHz.

• Is the cost savings worth the performance degradation?