EECS150 - Digital Design

Lecture 21 - High-level Design and Optimization

Nov 5, 2003

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 an 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 external and control input, generates control and external 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.

- **Standard model for CPUs, micro-controllers, many other digital sub-systems.**
- **Usually not nested.**
- **Often cascaded:**
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 combination logic (CL) blocks:

  - RTL comprises a set of register transfers with optional operators as part of the transfer.
  - Example:
    $$\text{regA} \leftarrow \text{regB}$$
    $$\text{regC} \leftarrow \text{regA} + \text{regB}$$
    $$\text{if} (\text{start}==1) \text{regA} \leftarrow \text{regC}$$

- My personal style:
  - use “;” to separate transfers that occur on separate cycles.
  - Use “,” to separate transfers that occur on the same cycle.
  - Example (2 cycles):
    $$\text{regA} \leftarrow \text{regB}, \text{regB} \leftarrow 0;$$
    $$\text{regC} \leftarrow \text{regA};$$
Example of Using RTL

- In this case: RTL description is used to sequence the operations on the datapath (dp).
- 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.

\[
\begin{align*}
\text{ACC} &\leftarrow \text{ACC} + \text{R0}, \text{R1} \leftarrow \text{R0}; \\
\text{ACC} &\leftarrow \text{ACC} + \text{R1}, \text{R0} \leftarrow \text{R1}; \\
\text{R0} &\leftarrow \text{ACC};
\end{align*}
\]
Example of Using RTL

- Sometimes RTL is used as a starting point for designing both the dp and the control:
  - example:
    ```
    regA ← IN;
    regB ← IN;
    regC ← regA + regB;
    regB ← regC;
    ```
  - From this we can deduce:
    - IN must fanout to both regA and regB
    - regA and regB must output to an adder
    - the adder must output to regC
    - regB must take its input from a mux that selects between IN and regC

- 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. At least one node in list.

I/Os:
- START resets to head of list and starts addition process.
- DONE signals completion
- R, Bus that holds the final result
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>clk-to-Q=0.5ns, setup=0.5ns</td>
</tr>
<tr>
<td>n-bit 2-1 multiplexor</td>
<td>1ns</td>
</tr>
<tr>
<td>n-bit adder</td>
<td>(2 log(n) + 2)ns</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>

(single ported memory)

Are these reasonable?
New Component

- Register with Load Enable:

  - Allows register to be either be loaded on selected clock posedge or to retain its previous value.
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.
  - NEXT holds a pointer to the node in memory.
  - SUM holds the result of adding the node values to this point.

    If (START==1) NEXT←0, SUM←0;
    repeat { 
        SUM←SUM + Memory[NEXT+1];
        NEXT←Memory[NEXT];
    } until (NEXT==0);
    R←SUM, DONE←1;
3. Architecture #1

Direct implementation of RTL description:

If (START==1) NEXT←0, SUM←0;
repeat {
    SUM←SUM + Memory[NEXT+1];
    NEXT←Memory[NEXT];
} until (NEXT==0);
R←SUM, DONE←1;
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
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

- Add new register named NUMA, for address of number to add.
- Update RTL to reflect our change:

If (START==1) NEXT←0, SUM←0, NUMA←1;
repeat {
    SUM←SUM + Memory[NUMA];
    NUMA←Memory[NEXT] + 1,
    NEXT←Memory[NEXT] ;
} until (NEXT==0);
R←SUM, DONE←1;
5. Optimization

• Architecture #2:

If (START==1) NEXT<-0, SUM<-0, NUMA<-1;
repeat {
    SUM<-SUM + Memory[NUMA];
    NUMA<-Memory[NEXT] + 1, NEXT<-Memory[NEXT] ;
} until (NEXT==0);
R<-SUM, DONE<-1;

• Incremental cost: addition of another register.
5. Optimization, Architecture #2

- New timing:
  Clock Period \( T \) = max (clock period for each state)
  \[ T > 24\text{ns}, F < 41.67\text{MHz} \]

- 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?