Embedded Systems Design: A Unified Hardware/Software Introduction

Chapter 2: Custom single-purpose processors

Introduction

- Processor
  - Digital circuit that performs a computation task
  - Controller and datapath
  - General-purpose: variety of computation tasks
  - Single-purpose: one particular computation task
  - Custom single-purpose: non-standard task
- A custom single-purpose processor may be
  - Fast, small, low power
  - But, high NRE, longer time-to-market, less flexible

CMOS transistor on silicon

- Transistor
  - The basic electrical component in digital systems
  - Acts as an on/off switch
  - Voltage at “gate” controls whether current flows from source to drain
  - Don’t confuse this “gate” with a logic gate

Outline

- Introduction
- Combinational logic
- Sequential logic
- Custom single-purpose processor design
- RT-level custom single-purpose processor design
Embedded Systems Design: A Unified Hardware/Software Introduction, (c) 2000 Vahid/Givargis

Combinational logic design

- Complementary Metal Oxide Semiconductor
- We refer to logic levels
  - Typically 0 is 0V, 1 is 5V
- Two basic CMOS types
  - nMOS conducts if gate=1
  - pMOS conducts if gate=0
  - Hence “complementary”
- Basic gates
  - Inverter, NAND, NOR

Basic logic gates

CMOS transistor implementations

- nMOS transistor
- pMOS transistor

RT-Level Combinational components

- nAB
- nsum
- carry
- Comparator
- ALU

Combinational circuit - a digital circuit whose output is purely a function of its present inputs.

A) Problem description
y is 1 if a is 1 or b and c are 1. x is 1 if b or c is to 1, but not both, or if all are 1.

B) Truth table

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Outputs</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>b</td>
</tr>
<tr>
<td>0</td>
<td>0</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>

C) Output equations
\[ y = a'bc + ab'c + abc' + abc \]
\[ z = ab'c + a'bc + ab + abc \]

D) Minimized output equations
\[ y' = a'bc + ab'c + abc' + abc \]
\[ z' = ab'c + a'bc + ab + abc \]

E) Logic Gates

- Inverter
- NAND gate
- NOR gate
Sequential components

Sequential circuit - a digital circuit whose outputs are a function of the present inputs as well as the previous inputs.

Sequential logic design

A) Problem Description
You want to construct a clock divider. Slow down your pre-existing clock so that you output a 1 for every four clock cycles.

E) Minimized Output Equations

\[ Q = Q_1'Q_0a + Q_1a' + Q_0'Q_0a \]

Sequential logic design (cont.)

Custom single-purpose processor basic model

D) State Table (Moore-type)

- Given this implementation model
  - Sequential logic design quickly reduces to combinational logic design
Example: greatest common divisor

- First create algorithm
- Convert algorithm to “complex” state machine
  - Known as FSMD: finite-state machine with datapath
  - Can use templates to perform such conversion

State diagram templates

Creating the datapath

- Create a register for any declared variable
- Create a functional unit for each arithmetic operation
- Connect the ports, registers and functional units
  - Based on reads and writes
  - Use multiplexors for multiple sources
- Create unique identifier
  - for each datapath component

Creating the controller’s FSM

- Same structure as FSMD
- Replace complex actions/conditions with datapath configurations
Splitting into a controller and datapath

Completing the GCD custom single-purpose processor design

- We finished the datapath
- We have a state table for the next state and control logic
  - All that’s left is combinational logic design
- This is not an optimized design, but we see the basic steps

Controller state table for the GCD example

- We often start with a state machine
  - Rather than algorithm
  - Cycle timing often too central to functionality
- Example
  - Bus bridge that converts 4-bit bus to 8-bit bus
  - Start with FSMD
  - Known as register-transfer (RT) level
  - Exercise: complete the design
RT-level custom single-purpose processor design (cont’)

Optimizing single-purpose processors

- Optimization is the task of making design metric values the best possible
- Optimization opportunities
  - original program
  - FSMD
  - datapath
  - FSM

Optimizing the original program

- Analyze program attributes and look for areas of possible improvement
  - number of computations
  - size of variable
  - time and space complexity
  - operations used
    - multiplication and division very expensive

Optimizing the original program (cont’)

<table>
<thead>
<tr>
<th>original program</th>
<th>optimized program</th>
</tr>
</thead>
<tbody>
<tr>
<td>0: int x, y;</td>
<td>0: int x, y, r;</td>
</tr>
<tr>
<td>1: while (1) {</td>
<td>1: while (1) {</td>
</tr>
<tr>
<td>2: while (x &gt; y);</td>
<td>2: while (x &gt; y);</td>
</tr>
<tr>
<td>3: x = x - y;</td>
<td>// x must be the larger number</td>
</tr>
<tr>
<td>4: y = y - x;</td>
<td>3: if (x, i := y, i)</td>
</tr>
<tr>
<td>5: while (x &gt; y)</td>
<td></td>
</tr>
<tr>
<td>6: if (x &lt; y)</td>
<td></td>
</tr>
<tr>
<td>7: y = y - x;</td>
<td></td>
</tr>
<tr>
<td>else</td>
<td></td>
</tr>
<tr>
<td>8: x = x - y;</td>
<td></td>
</tr>
<tr>
<td>9: d_o = x;</td>
<td></td>
</tr>
<tr>
<td>}</td>
<td>4: x = x_i;</td>
</tr>
<tr>
<td></td>
<td>5: y = y_i;</td>
</tr>
<tr>
<td></td>
<td>6: else {</td>
</tr>
<tr>
<td></td>
<td>7: x = x_i;</td>
</tr>
<tr>
<td></td>
<td>8: y = y_i;</td>
</tr>
<tr>
<td></td>
<td>9: while (y &gt; 0)</td>
</tr>
<tr>
<td></td>
<td>10: r = x * y;</td>
</tr>
<tr>
<td></td>
<td>11: x = y;</td>
</tr>
<tr>
<td></td>
<td>12: y = r;</td>
</tr>
<tr>
<td></td>
<td>13: d_o = x;</td>
</tr>
<tr>
<td>}</td>
<td>}</td>
</tr>
</tbody>
</table>

GCD(42, 8) - 9 iterations to complete the loop
x and y values evaluated as follows: (42, 8), (43, 8), (26, 8), (13, 8), (10, 8), (2, 8), (2, 6), (2, 4), (2, 2)

GCD(42, 8) - 3 iterations to complete the loop
x and y values evaluated as follows: (42, 8), (8, 2), (2, 0)
Optimizing the FSMD

- Areas of possible improvements
  - merge states
    - states with constants on transitions can be eliminated, transition taken is already known
    - states with independent operations can be merged
  - separate states
    - states which require complex operations (a*b*c*d) can be broken into smaller states to reduce hardware size
  - scheduling

Optimizing the datapath

- Sharing of functional units
  - one-to-one mapping, as done previously, is not necessary
  - if same operation occurs in different states, they can share a single functional unit
- Multi-functional units
  - ALUs support a variety of operations, it can be shared among operations occurring in different states

Optimizing the FSM

- State encoding
  - task of assigning a unique bit pattern to each state in an FSM
  - size of state register and combinational logic vary
  - can be treated as an ordering problem
- State minimization
  - task of merging equivalent states into a single state
    - state equivalent if for all possible input combinations the two states generate the same outputs and transitions to the next same state
Summary

- Custom single-purpose processors
  - Straightforward design techniques
  - Can be built to execute algorithms
  - Typically start with FSMD
  - CAD tools can be of great assistance