EECS 150 - Components and Design Techniques for Digital Systems

Lec 07 – PLAs and FSMs
9/21-04

David Culler
Electrical Engineering and Computer Sciences
University of California, Berkeley

http://www.eecs.berkeley.edu/~culler
http://www-inst.eecs.berkeley.edu/~cs150

Outline

• Programmable Logic to Implement Sum of Products
• Designing with PLAs
• Announcements
• FSM Concept
• Example: history sensitive computation
• Example: Combo lock
• Encodings and implementations

Review: minimum sum-of-products expression from a Karnaugh map

• Step 1: choose an element of the ON-set
• Step 2: find “maximal” groupings of 1s and Xs adjacent to that element
  – consider top/bottom row, left/right column, and corner adjacencies
  – this forms prime implicants (number of elements always a power of 2)
• Repeat Steps 1 and 2 to find all prime implicants
• Step 3: revisit the 1s in the K-map
  – if covered by single prime implicant, it is essential, and participates in final cover
  – 1s covered by essential prime implicant do not need to be revisited
• Step 4: if there remain 1s not covered by essential prime implicants
  – select the smallest number of prime implicants that cover the remaining 1s

Big Idea: boolean functions <-> gates

• $2^n$ boolean functions of $n$ inputs
  – Each represented uniquely by a Truth Table
  – Describes the mapping of inputs to outputs
• Each boolean function represented by many boolean expressions
  – Axioms establish equivalence
  – Transform expressions to optimize
• Each boolean expression has many implementations in logic gates
  – Canonical: Sum of Products, Product of Sums
  – Minimal
  – K-maps as a systematic means of reducing Sum of Products
• Any acyclic network of gates implements a boolean function

How to quickly implement SofPs?

One Answer: Xilinx 4000 CLB

Figure 1: Simplified Block Diagram of Xilinx 4000 Series CLB (BYP and Carry Logic Functions not shown)
**Programmable Logic**

- **Regular logic**
  - Programmable Logic Arrays
  - Multiplexers/Decoders
  - ROMs
- **Field Programmable Gate Arrays (FPGAs)**
  - Xilinx

**Programmable Logic Arrays (PLAs)**

- Pre-fabricated building block of many AND/OR gates
  - Actually NOR or NAND
  - "Personalized" by making or breaking connections among gates
  - Programmable array block diagram for sum of products form

**Shared Product Terms**

Example:

<table>
<thead>
<tr>
<th>Product Term</th>
<th>Inputs</th>
<th>Outputs</th>
<th>Personality Matrix</th>
</tr>
</thead>
<tbody>
<tr>
<td>AB</td>
<td>1 1</td>
<td>0 1 1 0</td>
<td>1 = uncomplemented in term</td>
</tr>
<tr>
<td>BC</td>
<td>0 0 0 0</td>
<td>0 0 0 0</td>
<td>0 = complemented in term</td>
</tr>
<tr>
<td>AC</td>
<td>1 0 0 0</td>
<td>1 0 1 0</td>
<td>1 = does not participate</td>
</tr>
<tr>
<td>BF</td>
<td>0 0 0 1</td>
<td>0 0 1 0</td>
<td>0 = no connection to output</td>
</tr>
<tr>
<td>A</td>
<td>1 - - -</td>
<td>1 0 0 1</td>
<td>1 = term connected to output</td>
</tr>
</tbody>
</table>

**Before Programming**

- All possible connections available before "programming"
  - In reality, all AND and OR gates are NANDs
After Programming

- Unwanted connections are “blown”
  - Fuse (normally connected, break unwanted ones)
  - Anti-fuse (normally disconnected, make wanted connections)

Alternate Representation for High Fan-in Structures

- Short-hand notation--don’t have to draw all the wires
  - Signifies a connection is present and perpendicular signal is an input to gate

Programmable Logic Array Example

- Multiple functions of A, B, C
  - F1 = A B C
  - F2 = A + B + C
  - F3 = A’ B’ C’
  - F4 = A’ + B’ + C’
  - F5 = A xor B xor C
  - F6 = A xor B xor C

PLA Design Example

- BCD to Gray code converter

PLA Design Example (cont’d)

- Code converter: programmed PLA

PLA Design Example (revisited)

- BCD to Gray code converter

• Signifies a connection is present and perpendicular signal is an input to gate

Programmable Logic Array Example

- Multiple functions of A, B, C
  - F1 = A B C
  - F2 = A + B + C
  - F3 = A’ B’ C’
  - F4 = A’ + B’ + C’
  - F5 = A xor B xor C
  - F6 = A xor B xor C

PLA Design Example

- BCD to Gray code converter

PLA Design Example (revisited)

- BCD to Gray code converter

Programmable Logic Array Example

- Multiple functions of A, B, C
  - F1 = A B C
  - F2 = A + B + C
  - F3 = A’ B’ C’
  - F4 = A’ + B’ + C’
  - F5 = A xor B xor C
  - F6 = A xor B xor C

PLA Design Example

- BCD to Gray code converter

PLA Design Example (revisited)

- BCD to Gray code converter
PLAs Design Example

- BCD to Gray code converter

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>W</th>
<th>X</th>
<th>Y</th>
<th>Z</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Minimized functions:

W = X = Y = Z

PLA Second Design Example

- Magnitude comparator

Other Options

- Muxes
- DeMuxes
- ROMs
- LUTs

We will return to these later

Announcements

- Reading: Katz 1.4.2 (again), 7.1-3, pp 174-186
- Mid term th 10/7

Recall: What makes Digital Systems tick?

Recall 61C: Single-Cycle MIPS

1. Instruction Fetch
2. Register Read
3. Execute
4. Memory
5. Reg. Write
Recall 61C: 5-cycle Datapath - pipeline

Typical Controller: state

Typical Controller: state + output

Typical Controller: state + output + input

Two Kinds of FSMs

Parity Checker Example

A string of bits has "even parity" if the number of 1's in the string is even.

- Design a circuit that accepts a bit-serial stream of bits and outputs a 0 if the parity thus far is even and outputs a 1 if odd:

  bit stream → Parity Checker → OUT

  0 even parity 1 if odd parity

  example: 0 even 0 even 1 odd 1 odd 0 odd 1 even

  CLK

  IN

  OUT

  State / out

  State

  Input / Out

- Can you guess a circuit that performs this function?
The circuit is in.

Another example

Effect of past inputs is

Output depends on which state

state       OUT  IN   state

0                    0      0                0
EVEN       0     1     ODD
0                    0      0                0

5. Derive Logic Equations

Derive logic equations

Inputs: sequence of input values, reset

- XOR gate for NS calculation
- DFF to hold present state
- no logic needed for output

CL (combinational logic) calculates next state and output

Remember: The FSM follows exactly one edge per cycle.
Implementation in software

```c
int combination_lock() {
    int v1, v2, v3;
    int error = 0;
    static int c[3] = 3, 4, 2;
    while (!new_value()) {
        v1 = read_value();
        if (v1 != c[0]) then error = 1;
        v2 = read_value();
        if (v2 != c[1]) then error = 1;
        v3 = read_value();
        if (v2 != c[2]) then error = 1;
        if (error == 1) then return(0); else return(1);
    }
}
```

Sequential example: abstract control

- Finite-state diagram
  - States: 5 states
    - Represent point in execution of machine
    - Each state has outputs
  - Transitions: 6 from state to state, 5 self transitions, 1 global
    - Changes of state occur when clock says it's ok
    - Based on value of inputs
  - Inputs: reset, new, results of comparisons
  - Output: open/closed

data-path vs. control

- Internal structure
  - Data-path
    - Storage for combination
    - Comparators
  - Control
    - Finite-state machine controller
    - Control for data-path
    - State changes controlled by clock

Sequential example (cont’d): finite-state machine

- Finite-state machine
  - Refine state diagram to include internal structure

Sequential example (cont’d): finite-state machine

- Finite-state machine
  - Generate state table (much like a truth-table)
Encode state table
- state can be: 51, 52, 53, OPEN, or ERR
- needs at least 3 bits to encode: 000, 001, 010, 011, 100
- and as many as 5: 00001, 00010, 00100, 01000, 10000
- choose 4 bits: 0001, 0010, 0100, 1000, 0000

Encode outputs
- output mux can be: C1, C2, or C3
- needs 2 to 3 bits to encode
- choose 3 bits: 001, 010, 100
- output open/closed can be: open or closed
- needs 1 or 2 bits to encode
- choose 1 bits: 1, 0

One-hot encoded FSM

In General:
- FFs must be initialized for correct operation (only one 1)
- In General:
  - state FF input
  - to other state FF logic and/or output

One-hot encoded FSM

Sequential example: encoding
- Encode state table
  - state can be: S1, S2, S3, OPEN, or ERR
  - needs at least 3 bits to encode: 000, 001, 010, 101, 110
  - and as many as 5: 00001, 00010, 00100, 01000, 10000
  - choose 4 bits: 0001, 0010, 0100, 1000, 0000

Sequential example (cont’d): encoding
- Encode state table
  - state can be: S1, S2, S3, OPEN, or ERR
  - choose 4 bits: 0001, 0010, 0100, 1000, 0000
  - choose 3 bits: 001, 010, 100
  - output open/closed can be: open or closed
  - choose 1 bits: 1, 0

Sequential example: encoding
- Encode state table
  - state can be: S1, S2, S3, OPEN, or ERR
  - needs at least 3 bits to encode: 000, 001, 010, 100
  - and as many as 5: 00001, 00010, 00100, 01000, 10000
  - choose 4 bits: 0001, 0010, 0100, 1000, 0000

Sequential example (cont’d): encoding
- Encode state table
  - state can be: S1, S2, S3, OPEN, or ERR
  - choose 4 bits: 0001, 0010, 0100, 1000, 0000
  - choose 3 bits: 001, 010, 100
  - output open/closed can be: open or closed
  - choose 1 bits: 1, 0

One-hot encoded FSM

- In General:
  - state FF input
  - to other state FF logic and/or output