Philosophy

★ All Theoretical Subjects Should Be Backed Up With Practical Exercises
★ Students Should Verify All “Paper” Designs to Verify Their Correctness
★ Students Should Verify Exhaustive Proofs by Using an Exhaustive Simulation
★ Students Should be Exposed to the Principles of Testing
**Fundamental Laws**

<table>
<thead>
<tr>
<th>Equation</th>
<th>Law</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a \cdot 1 = a$</td>
<td>And Identity Law</td>
</tr>
<tr>
<td>$a + 0 = a$</td>
<td>Or Identity Law</td>
</tr>
<tr>
<td>$ab = ba$</td>
<td>Commutativity of and</td>
</tr>
<tr>
<td>$a + b = b + a$</td>
<td>Commutativity of or</td>
</tr>
<tr>
<td>$(ab)c = a(bc)$</td>
<td>Associativity of and</td>
</tr>
<tr>
<td>$(a+b)+c = a+(b+c)$</td>
<td>Associativity of or</td>
</tr>
<tr>
<td>$a(b+c) = ab+bc$</td>
<td>And distributes over or</td>
</tr>
<tr>
<td>$a+bc = (a+b)(a+c)$</td>
<td>Or distributes over and</td>
</tr>
<tr>
<td>$aa' = 0$</td>
<td>And inverse law</td>
</tr>
<tr>
<td>$a+a' = 1$</td>
<td>Or inverse law</td>
</tr>
</tbody>
</table>
Identity Laws

Ident: circuit
inputs a
outputs xa, xo

one i1
zero i0
g1: and (a, i1), xa
g2: or (a, i0), xo
endcircuit
Associativity of AND

**FHDL**

- **aa:** circuit
  - inputs a, b, c
  - outputs left, right

- **g1:** and (a, b), l1
- **g2:** and (l1, c), left
- **h1:** and (b, c), r1
- **h2:** and (a, r1), right

**VECTORS**

- 0, 0, 0
- 0, 0, 1
- 0, 1, 0
- 0, 1, 1
- 1, 0, 0
- 1, 0, 1
- 1, 1, 0
- 1, 1, 1
Karnaugh Maps

- Start with A Truth Table
- Construct A Boolean Function
- Translate the Boolean Function into FHDL
- Simulate The FHDL to Reconstruct the Truth Table
Design With Gates

- Determine Number of Inputs & Outputs
- Derive Truth-Tables for Outputs
- Derive Boolean Functions for each Output
- Draw Logic Diagram
- Encode Logic Diagram in FHDL
- Simulate FHDL to Verify Correct Operation
Design with MSI Functions

- Implement Example MSI Functions
- Simulate to Verify Correctness
- Design using MSI Components Directly
- Implement Using FHDL Functional Blocks and Hierarchical Design
- Simulate to Verify Correctness
Sequential Circuits

- Design and Simulate RS, D, and Edge-Triggered D flip-flops
- Illustrate Oscillations using the RS flip-flop
- Illustrate The Need for Edge-Triggering, or Master/Slave Synchronization
- Illustrate Logic Hazards, and the Danger They Pose to Asynchronous Circuits
Registers And Memory

- Registers, with Variations are Directly Simulated by FHDL
- Registers Act as Collections of Unsynchronized D Flip-Flops
- RAM Can be Simulated Directly
- Simple ROMs and PLAs are Easy to Construct
Sequential Control

- State Machines Can be Constructed from Basic FHDL Components
- The ROM Preprocessor Allows RTL-Like Specifications
- The PLA Preprocessor Can Be Used To Create More Exotic State Machines
- ASMs can be specified and simulated directly
Testing: An Example

Distinguish Between The Two
How To Do It

Test 1
0,0,1,0,0,0  Presence or Absence of Carry
0,0,0,0,0,0  Must Be Detectable

Test 2
0,0,1,1,0,0  Carry Must Be Tested In Both States
0,0,0,1,0,0

Unacceptable Substitute For Test 2!
0,0,1,1,1,0  Error Is Masked, Because C2 and
0,0,0,1,1,0  C3 are Identical in Both Tests
Future Development

Fault Simulation in FHDL
Logic Design

With Simulation Exercises
Philosophy

- All Theoretical Subjects Should Be Backed Up With Practical Exercises
- Students Should Verify All “Paper” Designs to Verify Their Correctness
- Students Should Verify Exhaustive Proofs by Using an Exhaustive Simulation
- Students Should be Exposed to the Principles of Testing
# Fundamental Laws

<table>
<thead>
<tr>
<th>Equation</th>
<th>Law</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a \cdot 1 = a$</td>
<td>And Identity Law</td>
</tr>
<tr>
<td>$a + 0 = a$</td>
<td>Or Identity Law</td>
</tr>
<tr>
<td>$ab = ba$</td>
<td>Commutativity of and</td>
</tr>
<tr>
<td>$a + b = b + a$</td>
<td>Commutativity of or</td>
</tr>
<tr>
<td>$(ab)c = a(bc)$</td>
<td>Associativity of and</td>
</tr>
<tr>
<td>$(a + b) + c = a + (b + c)$</td>
<td>Associativity of or</td>
</tr>
<tr>
<td>$a(b + c) = ab + bc$</td>
<td>And distributes over or</td>
</tr>
<tr>
<td>$a + bc = (a + b)(a + c)$</td>
<td>Or distributes over and</td>
</tr>
<tr>
<td>$aa' = 0$</td>
<td>And inverse law</td>
</tr>
<tr>
<td>$a + a' = 1$</td>
<td>Or inverse law</td>
</tr>
</tbody>
</table>
Identity Laws

**FHDL**

<table>
<thead>
<tr>
<th>Ident:</th>
<th>circuit</th>
</tr>
</thead>
<tbody>
<tr>
<td>inputs</td>
<td>a</td>
</tr>
<tr>
<td>outputs</td>
<td>xa, xo</td>
</tr>
</tbody>
</table>

| g1: | and (a, i1), xa |
| g2: | or (a, i0), xo |

**VECTORS**

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>
## Associativity of AND

<table>
<thead>
<tr>
<th>FHDL</th>
<th>VECTORS</th>
</tr>
</thead>
<tbody>
<tr>
<td>aa:</td>
<td>0,0,0</td>
</tr>
<tr>
<td></td>
<td>inputs a,b,c</td>
</tr>
<tr>
<td></td>
<td>outputs left,right</td>
</tr>
<tr>
<td></td>
<td>endcircuit</td>
</tr>
<tr>
<td>g1:</td>
<td>1,0,0</td>
</tr>
<tr>
<td></td>
<td>and (a,b),l1</td>
</tr>
<tr>
<td>g2:</td>
<td>1,1,0</td>
</tr>
<tr>
<td></td>
<td>and (l1,c),left</td>
</tr>
<tr>
<td>h1:</td>
<td>1,1,1</td>
</tr>
<tr>
<td></td>
<td>and (b,c),r1</td>
</tr>
<tr>
<td>h2:</td>
<td></td>
</tr>
<tr>
<td></td>
<td>and (a,r1),right</td>
</tr>
</tbody>
</table>


Karnaugh Maps

- Start with A Truth Table
- Construct A Boolean Function
- Translate the Boolean Function into FHDL
- Simulate The FHDL to Reconstruct the Truth Table
Design With Gates

- Determine Number of Inputs & Outputs
- Derive Truth-Tables for Outputs
- Derive Boolean Functions for each Output
- Draw Logic Diagram
- Encode Logic Diagram in FHDL
- Simulate FHDL to Verify Correct Operation
Design with MSI Functions

- Implement Example MSI Functions
- Simulate to Verify Correctness
- Design using MSI Components Directly
- Implement Using FHDL Functional Blocks and Hierarchical Design
- Simulate to Verify Correctness
Sequential Circuits

- Design and Simulate RS, D, and Edge-Triggered D flip-flops
- Illustrate Oscillations using the RS flip-flop
- Illustrate The Need for Edge-Triggering, or Master/Slave Synchronization
- Illustrate Logic Hazards, and the Danger They Pose to Asynchronous Circuits
Registers And Memory

- Registers, with Variations are Directly Simulated by FHDL
- Registers Act as Collections of Unsynchronized D Flip-Flops
- RAM Can be Simulated Directly
- Simple ROMs and PLAs are Easy to Construct
Sequential Control

- State Machines Can be Constructed from Basic FHDL Components
- The ROM Preprocessor Allows RTL-Like Specifications
- The PLA Preprocessor Can Be Used To Create More Exotic State Machines
- ASMs can be specified and simulated directly
Testing: An Example

Distinguish Between The Two
# How To Do It

<table>
<thead>
<tr>
<th>Test 1</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0,0,1,0,0,0</td>
<td>Presence or Absence of Carry</td>
</tr>
<tr>
<td>0,0,0,0,0,0</td>
<td>Must Be Detectable</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Test 2</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0,0,1,1,0,0</td>
<td>Carry Must Be Tested In Both States</td>
</tr>
<tr>
<td>0,0,0,1,0,0</td>
<td></td>
</tr>
</tbody>
</table>

Unacceptable Substitute For Test 2!

| 0,0,1,1,1,0     | Error Is Masked, Because C2 and C3 are Identical in Both Tests |
Future Development

Fault Simulation in FHDL