Outline

- Last time:
  - Review State Tables & State Transition Diagrams
  - Implementation Using D Flip-Flops
  - Machine Equivalence
  - Incompletely Specified Machines
  - State Assignment & State Coding Schemes
  - Design Example: Assign Codes to States
  - Design Example: Implement Using D flip-flops
  - Design Example: Implement Using T flip-flops
- This lecture:
  - Introduction to VHDL

Overall Sequential Design Flow

- Schematic Entry
- Logic Optimization & Technology Mapping
- FPGA Netlist
- CLB Assignment & Routing

Common Representations Used in the Design Process

- Specification
  - Embodiment of the Requirements a System is to Satisfy
- Behavioral Description of the Function of the Design
- Implementation-Independent Description
- Register, Gate, Switch-Level Netlists
- Electrical Schematic
- Transistor-Level Netlist, Voltages, Currents and Detailed Models
- Mask-Level Layout
- Logic Values and Strengths
- Schematic Levels

Structured Custom Chip Layout

- Standard Cell Layout
- Layout Abstractions for Cell-Based Design
- Mask-Level Layout
- Cell Abstraction for Automatic Placement
### Standard Cells vs. Gate Array

- **Standard Cells**
  - Two tracks required and all connections routed.

- **Gate Array**
  - Shorter wire length but three tracks required.

In a Standard Cell design, an additional track could be added while in a Gate-Array, the designer is faced with extra wire length or no connection.

### Sequential Control-Flow Model

- Based on Von Neumann "fetch;execute;store" with single (serial control flow) or synchronized multiple (parallel control flow) thread(s) of control.
- Proposes an ordering of computation based on strict, temporal sequencing of operations, in line with above cycle.
- Not well suited to the description of hardware since arbitrary hardware rarely fits such a model of computation.
- Ability to combine several operations and treat them as a unit, or block (break-end, push-pull, ...) or allow an operation to be executed in a loop.

```plaintext
for(i=0 to 9)
  x[i] = y[i] + c; /* "forall" */

j = 0;
while(j < 10)
  x[j] = y[j] + c;
  j = j + 1;
```

### Converting Procedural Descriptions to a Dataflow-Oriented Representation

- Determine scope of variables and apply single-assignment in the scope
- Convert Complex data-structures into simple types
- Unroll loops with constant loop-counts (if appropriate)
- Perform simple syntactic optimizations:
  - Move operations out of loops where possible
  - Simplify complex expressions
  - Extract common sub-expressions

### Complications to Dataflow Analysis

- **Unrestricted goto statements (jumps)**
  - Introduce overly-pessimistic dependencies during static analysis
  - Values of variables references in the statements following a label depend on the assignments to variables in the code sections that can jump to the label.
- **Global variables**
  - i.e. variables declared outside of any function and therefore able to be shared by all functions
  - Last-use cannot be determined without examining the entire program.

Languages versus Models: A Software Analogy

8.2.13

Behavior and Structure: Two Faces of the Same Coin

8.2.14

Behavior and Structure

8.2.15

State and Statements

8.2.16

Data and Control

8.2.17

VHDL: The "nroff/latex" of Design

8.2.18

Languages versus Models: A Software Analogy

<table>
<thead>
<tr>
<th>Languages</th>
<th>Models</th>
</tr>
</thead>
<tbody>
<tr>
<td>f77</td>
<td>f</td>
</tr>
<tr>
<td>&quot;C&quot;</td>
<td>&quot;C&quot;</td>
</tr>
<tr>
<td>Lisp</td>
<td>&quot;D&quot;</td>
</tr>
<tr>
<td>Pascal</td>
<td>&quot;D&quot;</td>
</tr>
<tr>
<td>Intermediate Form</td>
<td>Hardware Implementation</td>
</tr>
<tr>
<td>Von Neumann Computer</td>
<td>Hardware Intermediate Form</td>
</tr>
</tbody>
</table>

Behavior and Structure: Two Faces of the Same Coin

8.2.14

Behavioral Information Carried In:

- "Behavioral" Semantics
- "Structural" Syntax

State and Statements

Consider symbolic state vector:

\[
\text{serial} \\
\text{(statement A)} \\
\text{(statement B)} \\
\text{parallel} \\
\text{(statement C)} \\
\text{serial} \\
\text{(statement D)} \\
\text{(statement E)} \\
\text{))))
\]

Statements Executed:

<table>
<thead>
<tr>
<th>State Number</th>
<th>Statements Executed</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>A</td>
</tr>
<tr>
<td>2</td>
<td>B</td>
</tr>
<tr>
<td>3</td>
<td>C</td>
</tr>
<tr>
<td>3.1</td>
<td>D</td>
</tr>
<tr>
<td>3.2</td>
<td>E</td>
</tr>
</tbody>
</table>

Every time (serial (parallel (serial occurs, pLevel++; stateVector[pLevel] becomes current.

Data and Control

- Control is a necessary by-product of the synthesis process of mapping a behavior into the "real world" (i.e. to hardware and to "slow time")
  - While it is a useful abstraction for certain classes of design, it should not be treated in a special way in the fundamental notation used for describing behavior/structure.
  - Consider the "Microscope Test": Control and data signals appear the same. The notion of control is a design-style oriented concept (and is therefore important) but control is not fundamentally different from other artifacts of the description (e.g. "datapath").
- Disclaimer: This is the view of this presenter but this view is not held today by the majority of synthesis researchers (e.g. VT, DDL, VHDL).

VHDL: The "nroff/latex" of Design

VHDL-Based Synthesis System
3-Bit Parity Function: "Control-Oriented"

parity <= '1' when A = '1' and B = '1' and C = '1';
else parity <= '0';
end if;

else
  if C = '1' then parity <= '0';
  else parity <= '1';
  end if;
end if;

3-Bit Parity Function: "Dataflow-Oriented"

parity <= ((A xor B) xor C);

3-bit Parity Function: Possible VHDL Implementation

ENTITY parityFunction IS
  PORT( A, B, C : IN t_wlogic; parity : OUT t_wlogic )
END parityFunction;

ARCHITECTURE full OF parityFunction IS
BEGIN
  PROCESS (A, B, C)
  VARIABLE count : integer;
  BEGIN
    count := 0;
    IF A = '1' THEN count := count +1; END IF;
    IF B = '1' THEN count := count +1; END IF;
    IF C = '1' THEN count := count +1; END IF;
    IF (count MOD 2) = 0 THEN
      OUT <= '0';
    ELSE
      OUT <= '1';
    END IF;
  END PROCESS;
END full;

"I synthesize from C" or "I synthesize from VHDL"

- No you don't!
  (with one known exception [AT&T Cones])
  OR
- If you do, you're probably being very silly!
Remember:
- A subset of a language is a different language
- A sequential language has sequential semantics

* a = 1;
  a = b+c;
  d = 0+1;

Representing Time for Behavioral Description

- Must distinguish sequence (causality) vs. passage of time
- Continuous time variable:
  A after B
  A, B independent
- "Quantized" time variable:
  A after B
  A, B independent
  A, B at the same time
where the meaning of "at the same time" is across an interval (must be valid at the end of the interval).
Why Should Time be Discrete?

Such an interval is called a "timeslot"; analogous to a "control state" in control-flow-based synthesis.

Timeslot may have many models associated with it: unspecified, fixed, interval (EDIF minNoMax), distribution.

System temporal behavior is relative; external inputs map behavior to "slow time."

Encoding Information in Time & Space

In Most HDLs, "wires" are declared but the passage of time is embedded in the control structures.

We are caught up (once again!) with imperative, sequential thinking and a Von Neumann model.

We need a way of capturing both temporal and spatial encoding in a single, unified mathematical model.

Use a type mechanism: "τ-types"

Design Representation

• Some variation of entity-relationship model most common today.
• Data model usually based on existing interchange format or design language (e.g. EDIF, VHDL).
• Open Issue: Forced Consistency (conventional database approach) versus Periodic Check ("Lint" approach).
• Integration Environment provides lingua franca for design representation. Useful side-effect of standards meetings is a common understanding of terms and data objects.
What's in a Name?

- Just about Everything!
- Efficient name resolution - resolving references to design objects - is one of the most important, "undecided" research problems.
- Strongly related to multiprocessor distributed cache consistency problem, distributed file system problem, general distributed data management problem.
- Ultimate issue is efficient pruning of "global search." (replication of read-only data, use of "hints," management of domains and dynamic data migration are all important.)