Design Automation

- Requirements
  - RFP
- Specifications
- Design CAD
- Simulation
- Prototype
- Production Support
- Revisions
Design and Simulation

Design Specification

HDL Entry

Synthesis

Net List

Simulation

Place and Route

Timing Simulation

Production Drawings

Part Library

logical

physical

Component specifications

function, delays

layout, pins

electrical characteristics
Parts are **generic objects** with:
- symbol
- pins
- part name (Part A)
- instance name (U1)

Wires connect pins
- may be 1 or more bits (bus)
- have a signal name
- may be implicit (DX)

Netlist
- connected signal names
  - U1.a, AG, U3.g
  - U2.w, WF, U3.f
  - etc.
- each part instance
  - U1 is a Part A
  - U2 is a Part B
  - U3 is a Part C
Schematic Capture

- Part library provides
  - Part symbol
  - Location of pins
  - Part name
  - Pin names

- Hierarchical tools
  - Allow user to define parts
  - Allow design to be subdivided
  - Support reuse of designs
  - User must define above info
  - Later user provides part netlist

- Hierarchical netlists
  - Look like subroutines
  - Instance names differentiate
  - multiple instances of a subcircuit
Schematic Capture - Hierarchical

Logic Drawing

Body Drawing

Part A

Part B

Part C

MYPART.LOGIC.1

MYPART.BODY.1
Netlist Output

- Each tool’s unique format
  - complex set of translators
- EDIF
  - early attempt at a standard format
  - various versions
- HDLs (structural)
  - Verilog
  - VHDL (gov’t approved)
Simulation

- Part library defines part as netlist based on a set of *primitive* parts
- Simulator provides computer model of primitives, and simulation engine to compute their interactions
- Simulator computes node voltages (simulator outputs) given input voltages
Time-Driven Simulation

- Primitives are resistors, capacitors, transistors, etc.
- Node voltages can be computed at any moment in the simulation
- Simulator computes node voltages every $\Delta t$
- Adjusts $\Delta t$ as needed to control error
- Approach used by $SPICE$
- Can be VERY expensive (time-wise)
Event-Driven Simulation

- Only recompute node voltages at interesting times - when something changes
- Estimate logic levels (0 and 1)
- Estimate rise and fall delay
- Ignore rise and fall time
- Use basic logic operations (AND, OR, NOT) as primitives
Simulation Styles

![Diagram showing simulation styles: event-based electrical, switch-based logical, time-based SPICE, and rapid simulation using VERILOG/VHDL.]}
Rise/Fall Time and Delay

IN

Fall Delay

OUT

Fall Time
Event-Driven Simulation

- Simulator keeps sorted list of future events
- Simulator execution
  - remove next event from list
  - set sim time to time of the event
  - simulate the event (primitives)
  - insert new events (caused by the current event) into the event list
  - run until no more events
Event-Driven Simulation

- Example - an inverter
  - Primitive model
    
    PRIMITIVE inverter BEGIN
    when input goes high to low
    schedule output high in RDELAY cycles
    when input goes low to high
    schedule output low in FDELAY cycles
    END

- Use netlist to locate all other gates that are connected to output
Event-Driven Simulation

- Simulation output
  - list of changes to signals (high-low, low-high)
  - usually used to create a timing diagram
  - also may be used as input to another simulation

- Simulation cost
  - reduced to time needed to schedule and cause events - computation is minimal
  - no longer sensitive to circuit density, speed, or technology
Basic Logic Values

- Each node in one state
  - 0
  - 1
  - U (unknown or undefined)
  - Z (high impedance)
TSB and TG Problems

- TSBs and TGs can charge a node and then stop driving it
  - Add Z+
  - Add Z-
  - Add charge decay
- TGs are bi-directional, and can exhibit charge-sharing
  - VERY hard to simulate
Different Driver Strength

- Might WANT to have one gate override another
Dealing with OC Outputs

Wire-AND configuration

F = A’B’C’
Dealing with OE Outputs

Wire-OR configuration

\[ F = A' + B' + C' \]
Model of OC and OE Outputs
Simulating OC and OE Parts

- Model OC devices to output 0 or Z
- Model OE devices to output 1 or Z
- Model pull-up resistor to output 1-
- Model pull-down resistor to output 0+
- Z and 1- is 1-; Z and 0+ is 0+
- 0 and 1- is 0+; 1 and 0+ is 1-
- Must model node as multi-input device
## Multi-Valued Logic

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>0+</th>
<th>1-</th>
<th>1</th>
<th>Z</th>
<th>U</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>U</td>
<td>0</td>
<td>U</td>
</tr>
<tr>
<td>0+</td>
<td>0</td>
<td>0</td>
<td>U</td>
<td>1</td>
<td>0+</td>
<td>U</td>
</tr>
<tr>
<td>1-</td>
<td>0</td>
<td>U</td>
<td>1</td>
<td>1</td>
<td>1-</td>
<td>U</td>
</tr>
<tr>
<td>1</td>
<td>U</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>U</td>
</tr>
<tr>
<td>Z</td>
<td>0</td>
<td>0+</td>
<td>1-</td>
<td>1</td>
<td>Z</td>
<td>U</td>
</tr>
<tr>
<td>U</td>
<td>U</td>
<td>U</td>
<td>U</td>
<td>U</td>
<td>U</td>
<td>U</td>
</tr>
</tbody>
</table>
Advanced Logic Simulators

- Can operate at switch or gate level
- Node Model
  - logic level
  - level strength
  - path strength
- Switch simulations model for both 0 and 1
- Table driven engine evaluates node states
Register Transfer Level (RTL) Simulators

- Restricted to synchronous circuits in a standard Mealy model
- Languages specify logic as state diagrams or truth tables
  - PIC, SYSPLD, ABEL
- Translate nicely to PALs
Hybrid (Mixed-Mode) Simulation

- Electrical (spice) and logic simulation together
- Partition problem into those areas needing critical timing analysis, and perform full electrical simulation of only those parts
Simulation Models

- Inputs to simulator
  - netlist
  - stimulus
  - device models
- Models composed of
  - primitive logic functions
  - timing
Simulation Models

- Primitive logic functions
  - AND, OR, NOT, NAND, NOR, XOR
  - TSB, TG, pull-up, etc.
  - no-delay

- Timing
  - rise time: best, worst, typical
  - fall time: best, worst, typical
  - setup time, hold time, pulse width
Typical FF Model

• Behavior
  • primitive, no-delay gates, or HDL model

• Timing
  • rise/fall delay from clock to Q
  • setup time, hold time
  • clock minimum high/low pulse width
  • preset/reset delay to Q
  • min preset/reset pulse width
  • min preset/reset to clock spacing
Typical FF Timing

- a - minimum clock pulse width high
- b - minimum clock pulse width low
- c - minimum setup time
- d - minimum hold time
- e - clock to Q fall delay (avg, worst)
- f - clock to Q rise delay (avg, worst)
- g - preset to Q rise delay (avg, worst)
- h - minimum preset removal to clock
- i - minimum preset pulse width
Routing

- Router inputs
  - netlist from schematic capture
  - physical part data from part library
    - physical dimension
    - pin data
      - placement, size
      - mount type
      - electrical characteristics
  - technology and user settings
Routing - Inputs

- Netlist
  - table of part numbers and type
  - connection list by part and pin number
- Pin data
  - signal name
  - special flags
  - position, size
  - resistance and capacitance
Routing - Parameters

• Router settings
  • dimensions of layout space
  • placement of connectors, some parts
  • routing layers, spacing, interconnects
  • electrical characteristics (resistance, capacitance)
  • routing goals (trace length, power density, etc.)
Routing - Outputs

- Layout
  - part placement
  - interconnect
  - power distribution
- Information
  - errors
  - routing density
  - routing data by net in netlist - feed back to simulator
Back Annotation

• Output data from router added to schematic
• Simulations to estimate timing
  • identify critical paths
  • full logical simulation
  • full electrical simulation
• Leads to re-routing, or re-design
Hardware Definition Languages

- Provide
  - textual definition of netlists
  - user-defined primitive models
  - high level functionality models
  - mixed-level simulation

- Major examples
  - VHDL
  - Verilog HDL
HDLs

- Support hierarchical design methodology
- Behavioral models
  - describe behavior, not implementation
  - supports top-down design
  - supports mixed-mode simulation
  - means for user-defined primitives
- Structural models
  - textual netlists
  - support synthesis
Top-Down Design: Requirements Analysis

Behavioral Model of Complete System
# Top-Down Design: Problem Decomposition

<table>
<thead>
<tr>
<th>Structural Model</th>
</tr>
</thead>
<tbody>
<tr>
<td>Behavioral Model of Component</td>
</tr>
</tbody>
</table>

*copyright(c) 1996 W. B. Ligon III*
# Top-Down Design: Problem Decomposition

## Structural Model

<table>
<thead>
<tr>
<th>Structural Model</th>
<th>Structural Model</th>
<th>Structural Model</th>
<th>Structural Model</th>
</tr>
</thead>
<tbody>
<tr>
<td>B</td>
<td>B</td>
<td>B</td>
<td>B</td>
</tr>
<tr>
<td>e</td>
<td>e</td>
<td>e</td>
<td>e</td>
</tr>
<tr>
<td>h</td>
<td>h</td>
<td>h</td>
<td>h</td>
</tr>
<tr>
<td>a</td>
<td>a</td>
<td>a</td>
<td>a</td>
</tr>
<tr>
<td>v</td>
<td>v</td>
<td>v</td>
<td>v</td>
</tr>
<tr>
<td>e</td>
<td>e</td>
<td>e</td>
<td>e</td>
</tr>
</tbody>
</table>

Copyright (c) 1996 W. B. Ligon III
Top-Down Design: Problem Decomposition
Top-Down Design: Hybrid Simulation

<table>
<thead>
<tr>
<th>Structural Model</th>
<th>Structural Model</th>
<th>Structural Model</th>
<th>Behavioral Model of Component</th>
<th>Behavioral Model of Component</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Behavioral Model of Component</td>
<td>Behavioral Model of Component</td>
<td>Structural Model</td>
<td>Structural Model</td>
</tr>
<tr>
<td></td>
<td>Behavioral Model of Component</td>
<td>Behavioral Model of Component</td>
<td>Structural Model</td>
<td>Structural Model</td>
</tr>
</tbody>
</table>

Unit Under Simulation