Introduction
Hardware Description Languages (HDLs)

- Highly portable and readable
- Provide different descriptive styles
  - Structural
  - Register Transfer Level (RTL)
  - Behavioral
- Serve as input to synthesis
- Support structured design methodology
- Support hardware concurrency
- Support rapid prototyping
- Serve as IP repository
Major HDLs

- **Verilog**
  - Started by Gateway in 1984
  - Became open to public by Cadence in 1990
  - IEEE standard 1364 in 1995
  - Slightly better at gate/transistor level
  - Language style close to C/C++
  - Pre-defined data type, easy to use

- **VHDL**
  - Started by VHSIC project in 1980s
  - IEEE standard 1076 in 1987, 1993
  - Slightly better at system level
  - Language style close to Ada/Pascal
  - User-defined data type, more flexible

- **Equally effective, personal preference**
Steps in HDL-based Design Flow

- Create behavior
- Consider testability
- Verify/simulate functionality
- Synthesize gate-level netlist
- Compare gate-level and behavior
- Conduct fault analysis
- Verify timing
- Place & route
- Verify design rules
- Extract parasitics
Design Styles

Issues of VLSI circuits

Performance
Area
Cost
Time-to-market

Different design styles

Full custom
Standard cell
Gate array
FPGA
CPLD
SPLD
SSI

Performance, Area efficiency, Cost, Flexibility
SSI/SPLD Design Style

(a) 4-bit comparator.

(b) SSI implementation.

(c) SPLD (PLA) implementation.
FPGA Design Style

• Illustrated by a symmetric array-based FPGA:
Gate Array Design Style

Diagram with labeled components including:
- I/O pads
- Pins
- Prefabricated transistor array
- Customized wiring
Standard Cell Design Style
Full Custom Design Style
Design Style Trade-offs

Turnaround Time (Days)

Logic Capacity (Gates)
Design Styles

- Restricting design styles to reduce complexity.
- Choosing design style according to design time, performance, size and cost, etc.
Hardware Modeling with Verilog
Design Entry

- **Schematic**
  - Focus on structural detail of a design

- **Hardware Description Language**
  - Describe a design by structure, behavior or a mixture of the two
Block Diagram & Schematic
Verilog Structural Description

```verilog
module Add_half (sum, c_out, a, b);
    input a, b;
    output sum, c_out;
    wire c_out_bar;
    xor (sum, a, b);
    nand (c_out_bar, a, b);
    not (c_out, c_out_bar);
endmodule
```

Note: All bold-faced items are Verilog keywords.
Verilog Behavioral Description

module Add_half (sum, c_out, a, b);
  input a, b;
  output sum, c_out;
  reg sum, c_out;
  always @ (a or b)
    begin
      sum = a ^ b; // Exclusive or
      c_out = a & b; // And
    end
endmodule
Another Behavioral Description

module add_half (sum, c_out, a, b);
  input     a, b;
  output    sum, c_out;
  
  assign {c_out, sum} = a + b; //continuous assignment

endmodule  Concatenation
Behavioral Description of D FF

module Flip_flop (q, data_in, clk, rst);
  input data_in, clk, rst;
  output q;
  reg q;

  always @ (posedge clk) begin
    if (rst == 1) q = 0;
    else q = data_in;
  end
endmodule
Module Instantiation

• Accomplished by entering
  – Module name as a module item within a parent module
  – Signal identifiers at appropriate ports

• Module instantiation needs a module identifier

• A module is never declared within another module

• Connection by position: the order of ports in instantiation usually matches the order in module declaration
module Add_full (sum, c_out, a, b, c_in); // parent module
input a, b, c_in;
output c_out, sum;
wire w1, w2, w3;

Add_half M1 (w1, w2, a, b); // child module
Add_half M2 (sum, w3, w1, c_in); // child module
or (c_out, w2, w3); // primitive instantiation
endmodule
Nested module instantiation is the mechanism for hierarchical decomposition of a design.
Verilog Predefined Primitives

- Combinational logic
  - and, nand, or, nor, xor, xnor, buf, not
- Tri-State
  - bufif0, bufif1, notif0, notif1
- MOS
  - nmos, pmos, rnmos, rpmos
- CMOS
  - cmos, rcmos
- Bi-directional
  - tran, tranif0, tranif1, rtran, rtranif0, rtranif1
- Pull
  - pullup, pulldown
Verilog Primitives

- Never used stand-alone in a design, must be instantiated within a module
- Predefined or user-defined
- Identifier (instance name) is optional
- Default delay = 0
- Port elements of an instantiated primitive must be outputs followed by inputs
module Add_half_1 (sum, c_out, a, b);
    input a, b;
    output sum, c_out;
    wire c_out_bar;

    xor #2 (sum, a, b);
    nand #2 (c_out_bar, a, b);
    not #1 (c_out, c_out_bar);
endmodule
Another Example

module nanf201 (O, A, B);
  input A, B;
  output O;
  nand (O, A, B);
specify
  specparam
    T01 = 1.13:3.09:7.75,  \text{Min delay}
    T10 = 0.93:2.50:7.34,  \text{Typical delay}
  (A \Rightarrow O) = (T01, T10),  \text{Max delay}
  (B \Rightarrow O) = (T01, T10),  \text{Falling time}
endspecify
endmodule
Smart Primitives

```verilog
module nand3 (O, A1, A2, A3);
    input  A1, A2, A3;
    output O;

    nand (O, A1, A2, A3);
endmodule
```

*Same primitive can be used to describe for any number of inputs*
Explicit Structural Description

```verilog
module Add_half (sum, c_out, a, b);
  input a, b;
  output sum, c_out;
  wire c_out_bar;

  xor  g1 (sum, a, b);
  nand g2 (c_out_bar, a, b);
  not  g3 (c_out, c_out_bar);
endmodule

g1, g2, g3 are primitive identifiers.
An undeclared identifier is assumed to be a wire.
```
Implicit Structural Description

module nand2_1 (y, x1, x2);
    input x1, x2;
    output y;
    assign y = x1 & x2;
endmodule

module nand2_2 (y, x1, x2);
    input x1, x2;
    output y;
    wire y = x1 & x2;
endmodule

Explicit continuous assignment

Continuous assignment –
Static binding between LHS and RHS
No mechanism to eliminate or alter the binding
Multiple Instantiations & Assignments

module Multiple_Gates (y1, y2, y3, a1, a2, a3, a4);
    input  a1, a2, a3, a4;
    output y1, y2, y3;

    `and #1 G1(y1, a1,a2,a3), (y2, a2,a3,a4), (y3, a1, a4);
endmodule

module Multiple_Gates(y1, y2, y3, a1, a2, a3, a4);
    input  a1, a2, a3, a4;
    output y1, y2, y3;

    assign #1 y1 = a1&a2&a3, y2 = a2&a3&a4, y3 = a1&a4;
endmodule
Module Port Connections

- By position
- By name
- Empty port

**module** child (a, b, c);

...  
**endmodule**

**module** parent;

  **wire** u, v, w;
  child m1 (u, v, w);
  child m2 (.c(w), .a(u), .b(v));
  child m3(u, , w);

**endmodule**

*An unconnected input is driven to "z", and an unconnected output is not used.*
Behavioral Descriptions: RTL/Data Flow

module and4_rtl (y_out, x_in);
  input [3:0] x_in;
  output y_out;

  assign y_out = & x_in; // Alternative:
  // x_in[1] & x_in[0];
endmodule

reduction AND operator
bitwise AND operator
module Flip_flop (q, data_in, clk, set, rst);
    input data_in, clk, set, rst;
    output q;
    reg q;

    always @ (posedge clk)
        begin
            if (rst == 0) q = 0;
            else
                if (set == 0) q = 1;
                else q = data_in;
        end
endmodule
Behavioral Descriptions: Algorithm-based

module and4_alg (y_out, x_in);
input [3:0] x_in;
output y_out;
reg y_out;
integer k;

always @ (x_in)
begin: and_loop
  y_out=1;
  for (k=0; k<= 3; k=k+1)
    if (x_in[k]==0)
      begin
        y_out=0;
        disable and_loop;
      end
end
endmodule
Structured Design Methodology

- Design: top-down
- Verification: bottom-up
16-bit Ripple-carry Adder
Hierarchical Decomposition

module Add_rca_16 (sum, c_out, a, b, c_in);
  output [15:0] sum;
  output c_out;
  input [15:0] a, b;
  input c_in;
  wire c_in, c_in4, c_in8, c_in12, c_out;
  Add_rca_4 M1 (sum[3:0], c_in4, a[3:0], b[3:0], c_in);
  Add_rca_4 M2 (sum[7:4], c_in8, a[7:4], b[7:4], c_in4);
  Add_rca_4 M3 (sum[11:8], c_in12, a[11:8], b[11:8], c_in8);
  Add_rca_4 M4 (sum[15:12], c_out, a[15:12], b[15:12], c_in12);
endmodule

module Add_rca_4 (sum, c_out, a, b, c_in);
  output [3:0] sum;
  output c_out;
  input [3:0] a, b;
  input c_in;
  wire c_in4, c_in3, c_in2;
  Add_full G1 (sum[0], c_in2, a[0], b[0], c_in);
  Add_full G2 (sum[1], c_in3, a[1], b[1], c_in2);
  Add_full G3 (sum[2], c_in4, a[2], b[2], c_in3);
  Add_full G4 (sum[3], c_out, a[3], b[3], c_in4);
endmodule

CS 4120
Mixed Descriptive Styles

module Add_rca_16 (sum, c_out, a, b, c_in);
  output [15:0] sum;
  output c_out;
  input [15:0] a, b;
  input c_in;
  wire c_in, c_in4, c_in8, c_in12, c_out;
  Add_rca_4 M1 (sum[3:0], c_in4, a[3:0], b[3:0], c_in);
  Add_rca_4 M2 (sum[7:4], c_in8, a[7:4], b[7:4], c_in4);
  Adder_4RTL M3 (sum[11:8], c_in12, a[11:8], b[11:8], c_in8);
  Adder_4RTL M4 (sum[15:12], c_out, a[15:12], b[15:12], c_in12);
endmodule

module Adder_4RTL (sum, c_out, a, b, c_in);
  output [3:0] sum;
  output c_out;
  input [3:0] a, b;
  input c_in;
  assign {c_out, sum} = a + b + c_in;
endmodule
module array_of_nor (y, a, b);
    input [0:7] a, b;
    output [0:7] y;
    nor [0:7] (y, a, b);
endmodule
4-bit Adder

module array_of_adders (sum, c_out, a, b, c_in);
  input [3:0] a, b;
  input c_in;
  output [3:0] sum;
  output c_out;
  wire [3:1] carry;

  Add_full M[3:0] (sum, {c_out, carry[3:1]}, a, b, {carry[3:1], c_in});
endmodule

Add_full M[0](sum[0], carry[1], a[0], b[0], c_in)
Add_full M[1](sum[1], carry[2], a[1], b[1], carry[1])
Add_full M[2](sum[2], carry[3], a[2], b[2], carry[2])
Add_full M[3](sum[3], c_out, a[3], b[3], carry[3])
Four-stage Pipeline of Registers

module array_of_flops (q, data_in, clk, set, rst);
input [7:0] data_in;
input clk, set, rst;
output [7:0] q;

Flip_flop M [7:0] (q, data_in, clk, set, rst);
endmodule

module array_pipeline (q, data_in, clk, set, rst);
input [7:0] data_in;
input clk, set, rst;
output [7:0] q;
wire [23:0] pipe;

array_of_flops M [3:0] ({q, pipe}, {pipe, data_in}, clk, set, rst);
endmodule
2-bit Comparator

\[ A_{\lt B} = \overline{A_1} B_1 + \overline{A_1} \overline{A_0} B_0 + \overline{A_0} B_1 B_0 \]
\[ A_{\gt B} = A_1 B_1 + A_0 B_1 B_0 + A_1 A_0 B_0 \]
\[ A_{eq\_B} = \overline{A_1} \overline{A_0} \overline{B_1} B_0 + \overline{A_1} \overline{A_0} B_1 \overline{B_0} + A_1 A_0 B_1 B_0 + A_1 \overline{A_0} B_1 B_0 \]

<table>
<thead>
<tr>
<th>A&gt;B</th>
<th>A1</th>
<th>B1</th>
<th>A0</th>
<th>B0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</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>1</td>
</tr>
</tbody>
</table>

CS 4120
module compare_2_str (A_lt_B, A_gt_B, A_eq_B, A0, A1, B0, B1);
input A0, A1, B0, B1;
output A_lt_B, A_gt_B, A_eq_B;
wire w1, w2, w3, w4, w5, w6, w7;
or (A_lt_B, w1, w2, w3);
nor (A_gt_B, A_lt_B, A_eq_B);
and (A_eq_B, w4, w5);
and (w1, w6, B1);
and (w2, w6, w7, B0);
and (w3, w7, B0, B1);
not (w6, A1);
not (w7, A0);
xnor (w4, A1, B1);
xnor (w5, A0, B0);
endmodule
RTL Description

• No explicit binding to primitive gates

```verbatim
module compare_2a (A_lt_B, A_gt_B, A_eq_B, A1, A0, B1, B0);
  input   A1, A0, B1, B0;
  output  A_lt_B, A_gt_B, A_eq_B;

  assign A_lt_B = (~A1) & B1 | (~A1) & (~A0) & B0 | (~A0) & B1 & B0;
  assign A_gt_B = A1 & (~B1) | A0 & (~B1) & (~B0) | A1 & A0 & (~B0);
  assign A_eq_B = (~A1) & (~A0) & (~B1) & (~B0) | (~A1) & A0 & (~B1) & B0 | A1 & A0 & B1 & B0 | A1 & (~A0) & B1 & (~B0);
endmodule
```
Alternatives

module compare_2b (A_lt_B, A_gt_B, A_eq_B, A1, A0, B1, B0);
input A1, A0, B1, B0;
output A_lt_B, A_gt_B, A_eq_B;
assign A_lt_B = ({A1, A0} < {B1, B0});
assign A_gt_B = ({A1, A0} > {B1, B0});
assign A_eq_B = ({A1, A0} == {B1, B0});
endmodule

module compare_2ca (A_lt_B, A_gt_B, A_eq_B, A, B);
input [1:0] A, B;
output A_lt_B, A_gt_B, A_eq_B;
assign A_lt_B = (A < B);
assign A_gt_B = (A > B);
assign A_eq_B = (A == B);
endmodule
Algorithmic Description

- Procedure-style behavior code

```verilog
module compare_2_algo (A_lt_B, A_gt_B, A_eq_B, A, B);

input [1:0] A, B;
output A_lt_B, A_gt_B, A_eq_B;
reg A_lt_B, A_gt_B, A_eq_B;

always @ (A or B) // Behavior and event expression
begin
  A_lt_B = 0;
  A_gt_B = 0;
  A_eq_B = 0;
  if (A==B) A_eq_B = 1;
  else if (A > B) A_gt_B = 1;
  else A_lt_B = 1;
end
endmodule
```
Synthesized Result
Language Conventions

- Case sensitive
- Instance of a module must be named
- Keywords are lower-case
- Identifiers:
  - May use upper and lower case alphabetical characters, decimal digits (0,1,...,9) and underscore (_)
  - May contain up to 1024 characters, and the first character must not be a digit
- Single-line comments start with “//”, and blocked (multi-line) comments being with “/\*” and end with “\*/”
- Compiler directives begin with grave accent: “``” (e.g., “``define state0 3 b001’’)
- All names beginning with “$” denote built-in system tasks or functions (e.g., $monitor)
- Others
Number Representation

• 8’b1010
  – Eight-bit binary 1010 -> a decimal value of 10

• <[size]><base_format><number>
  – size: a decimal value (optional) indicating the number of bits for storing the value
  – base_format:
    • Decimal (d or D)
    • Hex (h or H)
    • Octal (o or O)
    • Binary (b or B)
  – Number: value expressed in indicated base
Examples

<table>
<thead>
<tr>
<th>Num.</th>
<th>#bits</th>
<th>Base</th>
<th>Dec</th>
<th>Stored</th>
</tr>
</thead>
<tbody>
<tr>
<td>2’b10</td>
<td>2</td>
<td>binary</td>
<td>2</td>
<td>10</td>
</tr>
<tr>
<td>3’d5</td>
<td>3</td>
<td>decimal</td>
<td>5</td>
<td>101</td>
</tr>
<tr>
<td>3’o5</td>
<td>3</td>
<td>octal</td>
<td>5</td>
<td>101</td>
</tr>
<tr>
<td>8’h5</td>
<td>8</td>
<td>hex</td>
<td>5</td>
<td>00000101</td>
</tr>
<tr>
<td>3’b5</td>
<td></td>
<td>invalid!</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>