## Hardware description languages

- Describe hardware at varying levels of abstraction
- Structural description
  - textual replacement for schematic
  - hierarchical composition of modules from primitives
- Behavioral/functional description
  - describe what module does, not how
  - synthesis generates circuit for module
- Simulation semantics

## HDLs

- Abel (circa 1983) - developed by Data-I/O
  - targeted to programmable logic devices
  - not good for much more than state machines
- ISP (circa 1977) - research project at CMU
  - simulation, but no synthesis
- Verilog (circa 1985) - developed by Gateway (now part of Cadence)
  - similar to Pascal and C
  - delays is only interaction with simulator
  - fairly efficient and easy to write
  - IEEE standard
- VHDL (circa 1987) - DoD sponsored standard
  - similar to Ada (emphasis on re-use and maintainability)
  - simulation semantics visible
  - very general but verbose
  - IEEE standard

## Verilog

- ⌘ Supports structural and behavioral descriptions
- ⌘ Structural
  - explicit structure of the circuit
  - e.g., each logic gate instantiated and connected to others
- ⌘ Behavioral
  - program describes input/output behavior of circuit
  - many structural implementations could have same behavior
  - e.g., different implementation of one Boolean function
- ⌘ We'll only be using behavioral Verilog
  - rely on schematic for structural constructs

## Structural model

```
module xor_gate (out, a, b);
  input     a, b;
  output    out;
  wire      abar, bbar, t1, t2;

  inverter invA (abar, a);
  inverter invB (bbar, b);
  and_gate and1 (t1, a, bbar);
  and_gate and2 (t2, b, abar);
  or_gate  or1 (out, t1, t2);

endmodule
```

## Simple behavioral model

⌘ Continuous assignment

```
module and_gate (out, in1, in2);
  input          in1, in2;
  output         out;
  reg            out;

  assign #2 out = in1 & in2;

endmodule
```

**delay from input change to output change**

## Simple behavioral model

⌘ always block

```
module and_gate (out, in1, in2);
  input          in1, in2;
  output         out;
  reg            out;

  always @(in1 or in2) begin
     #2 out = in1 & in2;
  end

endmodule
```

simulation register - keeps track of value of signal

specifies when block is executed ie. triggered by which signals

## Driving a simulation

```
module stimulus (a, b);
  output        a, b;
  reg [1:0]     cnt;

  initial begin
    cnt = 0;
    repeat (4) begin
      #10 cnt = cnt + 1;
      $display ("@ time=%d, a=%b, b=%b, cnt=%b",
        $time, a, b, cnt); end
    #10 $finish;
  end

  assign a = cnt[1];
  assign b = cnt[0];
endmodule
```

2-bit vector

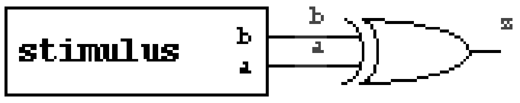
initial block executed only once at start of simulation

print to a console

directive to stop simulation

## Complete Simulation

⌘ Instantiate stimulus component and device to test in a schematic



## Comparator Example

```
module Compare1 (A, B, Equal, Alarger, Blarger);
  input     A, B;
  output    Equal, Alarger, Blarger;

  assign #5 Equal = (A & B) | (~A & ~B);
  assign #3 Alarger = (A & ~B);
  assign #3 Blarger = (~A & B);
endmodule
```

## Hardware Description Languages vs. Programming Languages

- Program structure
  - instantiation of multiple components of the same type
  - specify interconnections between modules via schematic
  - hierarchy of modules (only leaves can be HDL in DesignWorks)
- Assignment
  - continuous assignment (logic always computes)
  - propagation delay (computation takes time)
  - timing of signals is important (when does computation have its effect)
- Data structures
  - size explicitly spelled out - no dynamic structures
  - no pointers
- Parallelism
  - hardware is naturally parallel (must support multiple threads)
  - assignments can occur in parallel (not just sequentially)