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 (absorbed by 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 in DesignWorks
  - rely on schematic when we want structural descriptions

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 xor_gate (out, a, b);
  input         a, b;
  output        out;
  reg           out;

  assign #6 out = a ^ b;

endmodule
```

simulation register - keeps track of value of signal

delay from input change to output change

Simple behavioral model

- always block

```
module xor_gate (out, a, b);
  input         a, b;
  output        out;
  reg           out;

  always @(a or b) begin
     #6 out = a ^ b;
  end

endmodule
```

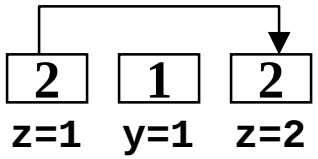
specifies when block is executed
ie. triggered by which signals

Event Queue in Verilog

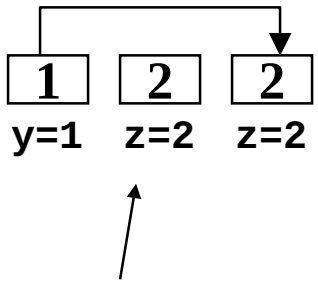
```
module xor_gate (w,x,y,z);
  input  [7:0]  x, w;
  output [7:0]  y, z;
  reg    [7:0]  y, z;

  always @(x)                  1
     y = x + 1;
  always @(y or w)             2
     z = y + w;
endmodule
```

Initially y = 0

Case 1: at time zero: w=1, x = 0



Case 2: at time zero: x=0, w=1



Good simulators will not execute this event

Point: Simulation semantics are simulator/order independent