# EECS150 - Digital Design
## Lecture 8 - Hardware Description Languages

February 14, 2002

John Wawrzynek



## Outline

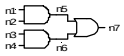
- Netlists
- Design flow
- What is a HDL?
- Verilog
  - history
  - examples



## Netlist

- A key data structure (or representation) in the design process is the "netlist":
  - Network List
- Netlist lists components and connects them with nodes:

ex:




```
g1 "and" n1 n2 n5
g2 "and" n3 n4 n6
g3 "or" n5 n6 n7
```

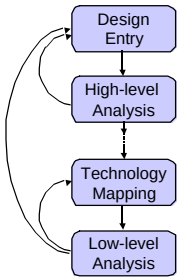
Alternative format:

```
n1 g1.in1
n2 g1.in2
n3 g2.in1
n4 g2.in2
n5 g1.out g3.in1
n6 g2.out g3.in2
n7 g3.out
g1 "and"
g2 "and"
g3 "or"
```

- Netlist is what is needed for simulation and implementation.
- Could be at the transistor level, gate level, ...
- Could be hierarchical or flat.
- How do we generate a netlist?

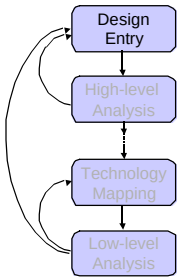


## Design Flow





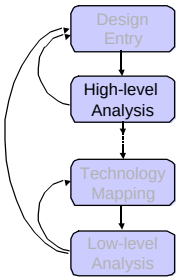

## Design Flow




- Circuit is described and represented:
  - Graphically (Schematics)
  - Textually (HDL)
- Result of circuit specification is a netlist of:
  - generic primitives - logic gates, flip-flops, or
  - technology specific primitives - LUTs/CLBs, transistors, discrete gates, or
  - higher level library elements - adders, ALUs, register files, decoders, etc.

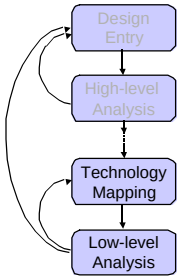


## Design Flow




- High-level Analysis is used to verify:
  - correct function
  - rough:
    - timing
    - power
    - cost
- Common tools used are:
  - simulator - check functional correctness, and
  - static timing analyzer
    - estimates circuit delays based on timing model and delay parameters for library elements (or primitives).

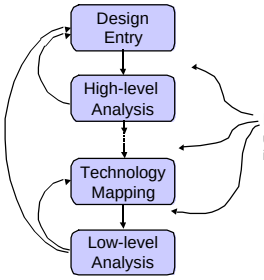
## Design Flow




- Technology Mapping:
  - Converts netlist to implementation technology dependent details
    - Expands library elements,
    - performs:
      - partitioning,
      - placement,
      - routing
- Low-level Analysis
  - Simulation and Static Tools perform low-level checks with:
    - accurate timing models,
    - wire delay
  - For FPGAs this step could use the actual device.



## Design Flow




Netlist:
used between and internally for all steps.



## Design Entry

- Schematic entry/editing was the standard method in industry
- Used in EECS150 last semester
- ☺ Schematics are intuitive. They match our use of gate-level or block diagrams.
- ☺ Somewhat physical. They imply a physical implementation.
- ☹ Require a special tool (editor).
- ☹ Unless hierarchy is carefully designed, schematics can be confusing and difficult to follow.

- Hardware Description Languages are the new standard
  - except for PC boards



## HDLs

- Basic Idea:
  - Language constructs describe circuit structure
  - *Structural descriptions* similar to hierarchical netlist.
  - *Behavioral descriptions* use higher-level constructs (similar to conventional programming).
- Originally designed to help in abstraction and simulation.
  - Now "logic synthesis" tools exist to automatically convert from behavioral descriptions to gate netlist.
  - This may lead you to falsely believe that hardware design can be reduced to writing programs!
- "Structural" example:

```
Decoder(output x0,x1,x2,x3;
            inputs a,b)
{
  wire abar, bbar;
  inv(bbar, b);
  inv(abar, a);
  nand(x0, abar, bbar);
  nand(x1, abar, b   );
  nand(x2, a,    bbar);
  nand(x3, a,    b   );
}
```

- "Behavioral" example:

```
Decoder(output x0,x1,x2,x3;
            inputs a,b)
{
  case [a b]
    00: [x0 x1 x2 x3] = 0x0;
    01: [x0 x1 x2 x3] = 0x2;
    10: [x0 x1 x2 x3] = 0x4;
    11: [x0 x1 x2 x3] = 0x8;
  endcase;
}
```



## Verilog

- A brief history:
  - Originated at Automated Integrated Design Systems (renamed Gateway) in 1985. Acquired by Cadence in 1989.
  - Invented as simulation language. Synthesis was an afterthought. Many of the basic techniques for synthesis were developed at Berkeley in the 80's and applied commercially in the 90's.
  - Around the same time as the origin of Verilog, the US Department of Defense developed VHDL. Because it was in the public domain it began to grow in popularity.
  - Afraid of losing market share, Cadence opened Verilog to the public in 1990.
  - An IEEE working group was established in 1993, and ratified IEEE Standard 1394 in 1995.
  - Verilog is the language of choice of Silicon Valley companies, initially because of high-quality tool support and its similarity to C-language syntax.
  - VHDL is still popular within the government, in Europe and Japan, and some Universities.
  - Most major CAD frameworks now support both.
  - Latest HDL. C++ based. OSCI (Open System C Initiative).



## Basic Example

```
//2-input multiplexor in gates
module mux2 (in0, in1, select, out);
   input in0,in1,select;
   output out;
   wire s0,w0,w1;

   not
        (s0, select);
   and
        (w0, s0, in0),
        (w1, select, in1);
   or
        (out, w0, w1);

endmodule // mux2
```

- Notes:
  - comments
  - "module"
  - port list
  - declarations
  - wire type
  - primitive gates

## Hierarchy & Bit Vectors

```
//2-input mux in gates
module mux2 (in0, in1, select, out);
       .
       .
       .
endmodule // mux2

//4-input mux built from 3 2-input muxes
module mux4 (in0, in1, in2, in3, select, out);
   input in0,in1,in2,in3;
   input [1:0] select;
   output      out;
   wire        w0,w1;

   mux2
     m0 (.select(select[0]), .in0(in0), .in1(in1), .out(w0)),
     m1 (.select(select[0]), .in0(in2), .in1(in3), .out(w1)),
     m3 (.select(select[1]), .in0(w0), .in1(w1), .out(out));
endmodule // mux4
```

- Notes:
  - instantiation similar to primitives
  - select is 2-bits wide
  - named port assignment



## Behavioral with Bit Vectors

```
//Behavioral model of 32-bit
// wide 2-to-1 multiplexor.
module mux32 (in0,in1,select,out);
   input [31:0] in0,in1;
   input      select;
   output [31:0] out;
   //
   reg [31:0] out;
   always @ (in0 or in1 or select)
     if (select) out=in0;
     else out=in0;
endmodule // Mux
```

- Notes:
  - inputs, outputs 32-bits wide
  - behavioral descriptions use the keyword `always` followed by *procedural* assignments
  - Target output of procedural assignments must of of type `reg`
  - Unlike `wire` types where the target output of an assignment may be continuously updated, a `reg` type retains it value until a new value is assigned (the assigning statement is executed).



## "Dataflow" Descriptions of Logic

```
//Dataflow description of mux
module mux2 in0, in1, select, out);
   input in0,in1,select;
   output out;
   assign out = (~select & in0)
                  | (select & in1);
endmodule // mux2
```

Alternative:

```
   assign out = select ? in1 : in0;
```

- Notes:
  - dataflow modeling provides a way to describe combinational logic by its function rather than gate structure.
  - The `assign` keyword is used to indicate a *continuous assignment*. Whenever anything on the RHS changes the LHS is updated.



## Testbench

```
module testmux;
  reg a, b, s;
  wire f;
  reg expected;

  mux2 myMux (.select(s), .in0(a), .in1(b), .out(f));

  initial
      begin
            s=0; a=0; b=1; expected=0;
         #10 a=1; b=0; expected=1;
         #10 s=1; a=0; b=1; expected=1;
      end
  initial
      $monitor(
          "select=%b in0=%b in1=%b out=%b, expected out=%b time=%d",
           s, a, b, f, expected, $time);
   endmodule // testmux
```

- Notes:
  - initial block similar to always except only executes once (at beginning of simulation)
  - #n's needed to advance time
  - $monitor - prints output