Hardware Description Languages

Introduction and Combinational Logic

Joe Crop

Oregon State University

ECE 271

Outline

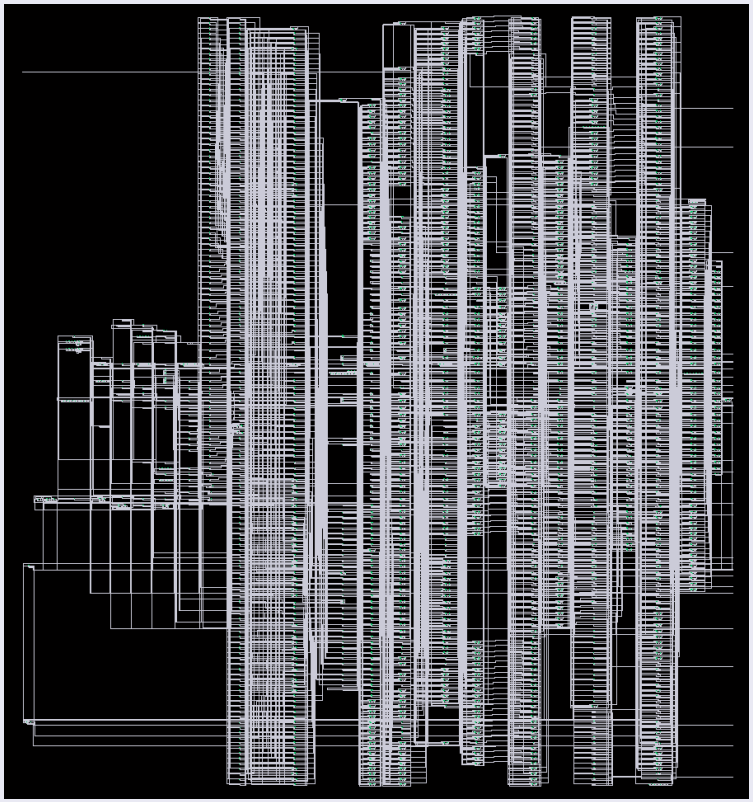
Outline

What is an HDL?

A Programming language that describes hardware.

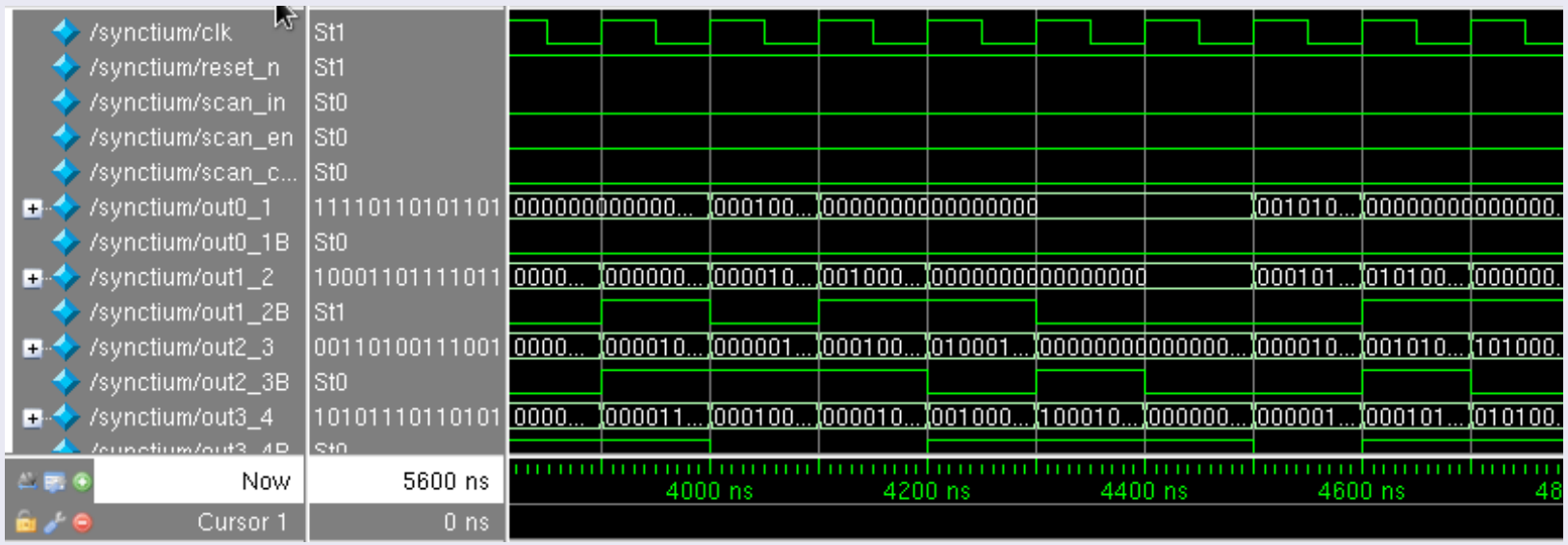
Why HDL?

1.) Schematics can get large and complicated.



Why HDL?

2.) The ability to simulate is imperative.



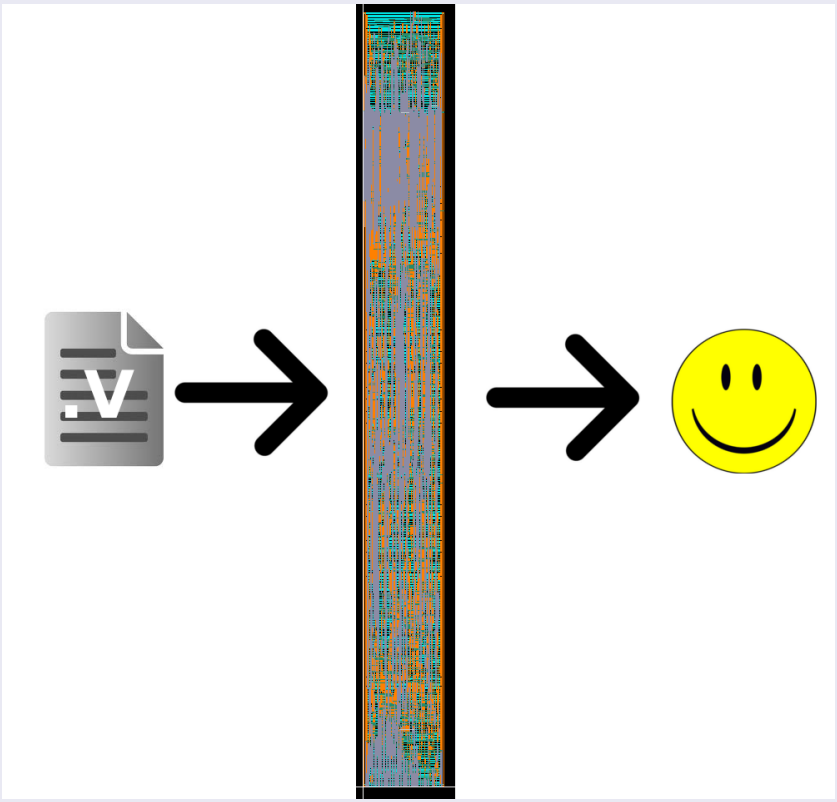
Why HDL?

3.) Logic minimization and optimization are a pain.

```
Beginning Mapping Optimizations  (Ultra High effort)
-----------------------------
 ELAPSED                 WORST NEG TOTAL NEG  DESIGN
  TIME       AREA          SLACK     SLACK   RULE COST
-------- --------- --------- --------- ----------
  0:00:18  372324.0      0.00       0.0        0.6
  0:00:19  372324.0      0.00       0.0        0.6
  0:00:21  356715.6      0.00       0.0       34.4
  0:00:23  356836.8      0.00       0.0        0.0
  0:00:24  355791.0      0.00       0.0        0.0
  0:00:28  355689.0      0.00       0.0       31.1
  0:00:33  355379.1      0.00       0.0        0.0
Optimization Complete
```

Why HDL?

4.) Faster design time is needed.



Types of HDLs

- Verilog
- VHDL
- System-C
- Bluespec
- C/C++
- Matlab

Modules

Definition: A block of hardware with inputs and outputs.

Examples:

- AND gate
- multiplexer
- Adder
- CPU

VHDL/Verilog Comparison

VHDL

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.all;
3
4  entity gates is
5   port(in1, in2: in STD_LOGIC;
6        out1: out STD_LOGIC);
7  end;
8
9  architecture synth of gates is
10  signal in1_b: STD_LOGIC;
11 begin
12
13 in1_b <= not in1;
14 out1 <= in1_b and in2;
15
16 end
```

Verilog

```
1  module gates(in1, in2, out1);
2
3  input  in1;
4  input  in2;
5  output out1;
6
7  wire in1_b;
8
9  assign in1_b = ~in1;
10 assign out1 = in1_b & in2;
11
12 endmodule
```

Typical Design Cycle

1. Concept
2. Design
3. Simulate
4. Synthesize
5. Implement
6. Sign-off

Outline

Bitwise Operators

operator	function
~	NOT
*, /, %	Multiply, Divide, Modulo
<<, >>	Logical Shift Left, Right
<<<, >>>	Arithmetic Shift Left, Right
<, <=, >, >=	Relative Comparison
==, !=	Equality Comparison
&	AND
~&	NAND
^	XOR
~^	XNOR
|	OR
~|	NOR
?:	Conditional

Bitwise Operators: Example

gates.v

```
1  module gates( a, b, y1, y2, y3, y4, y5);
2      input  a;
3      input  b;
4      output y1;
5      output y2;
6      output y3;
7      output y4;
8      output y5;
9
10     assign y1 = a & b;         // AND
11     assign y2 = a | b;         // OR
12     assign y3 = a ^ b;         // XOR
13     assign y4 = ~(a & b);      // NAND
14     assign y5 = ~(a | b);      // NOR
15 endmodule
```

Buses

Buses of wires

```
1  module gates(in1, in2, out1);
2
3  input  [0:2] in1;
4  input  [2:0] in2;
5  output [0:2] out1;
6
7  wire [0:2] in1_b;
8
9  assign in1_b = ~in1;
10 assign out1 = in1_b & in2;
11
12 endmodule
```

Comments

Comments in Verilog

```
1  module gates( a, y);
2      input  a;
3      output y;
4
5  /* A block coment in Verilog
6   * consists of either C/C++
7   * style comment statements
8   * */
9
10     assign y1 = a & b;        // an and gate
11
12 endmodule
```

Reduction Operators

Buses of wires

```
1  module gates( a, b, y1, y2, y3, y4, y5);
2      input  [3:0] a;
3      input  b;
4      output y1;
5      output y2;
6      output y3;
7      output [1:0] y4;
8      output y5;
9
10     assign y1 = a[0] & b;         // AND
11     assign y2 = a[1] | b;         // OR
12     assign y3 = a[2] ^ b;         // XOR
13     assign y4 = ~(a[1:2] & b);      // NAND
14     assign y5 = ~(a[3] | b);      // NOR
15 endmodule
```

Multiplexers

Multiplexers using the Conditional operator

```
1  module mux( in_0, in_1, sel, y);
2      input  in_0;
3      input  in_1;
4      input  sel;
5      output y;
6
7      assign y = sel ? in_1 : in_0;        // A simple MUX
8
9  endmodule
```

Internal Variables

Internal Variables

```
1  module gates( a, b, y);
2      input  a;
3      input  b;
4      output y;
5
6      /* an internal "variable" */
7      wire var_or;
8
9      assign var_or = a | b;        // OR
10     assign y = a & var_or;        // AND
11 endmodule
```

Formatting Numbers

- Format: (# of bits) ' (base) (value)
  - 3'b101 = binary 101
  - 8'b101 = binary 00000101
  - 3'd6 = decimal (binary 110)
  - 8'hAB = hexadecimal (binary 10101011)
  - 6'o42 = octal (binary 100010)
- Things to (almost) never do:
  - 'b11 = 000...0011
  - 42 = 000...0101010
  - 4'bz = high-impedance (zzzz)
  - 4'bx = not defined (xxxx)

Tristate Buffers

Example

```
1  module tristate( in, enable, y);
2      input  in;
3      input  enable;
4      output y;
5
6      assign y = sel ? in : 1'bz;        // A tristate buffer
7
8  endmodule
```

Bit Swizzling

More commonly known as concatenation

```
1  module gates(in1, in2, out1);
2
3  input  [0:2] in1;
4  input  [2:0] in2;
5  output [0:2] out1;
6
7  wire [0:2] in1_b;
8
9  assign in1_b = {~in1[0:1], in2[2]};
10 assign out1 = in1_b & in2;
11
12 endmodule
```

Simulating Delays

Example

```
1  module gates( a, b, y1, y2, y3, y4, y5);
2      input  a;
3      input  b;
4      output y1;
5      output y2;
6      output y3;
7      output y4;
8      output y5;
9
10     assign #2 y1 = a & b;         // AND
11     assign #3 y2 = a | b | y1;    // OR
12     assign #1 y3 = a ^ b;         // XOR
13     assign #1 y4 = ~(a & b);      // NAND
14     assign #1 y5 = ~(a | b);      // NOR
15 endmodule
```

Feedback

- Please give me your feedback
  - Email: cropj@eecs.oregonstate.edu
  - Text: (541) 250-0253
  - Twitter: @JoeCrop

Outline

Structural Modeling

Behavioral Modeling: Describing a module in terms of inputs and outputs.

Structural Modeling: Describing a module in terms of how it is composed of sub-modules.

A Simple Structure

Example

```
1  module mux4( in_00, in_01, in_10, in_11 sel, y);
2      input  in_00, in_01, in_10, in_11;
3      input  [1:0] sel;
4      output y;
5
6      wire [3:0] low, high;
7
8      mux LOWMUX(in_00, in_01, sel[0], low);
9      mux HIGHMUX(in_10, in_11, sel[0], high);
10     mux FINALMUX(low, high, sel[1], y);
11 endmodule //mux4
12
13 module mux( in_0, in_1, sel, y);
14     input  in_0, in_1, sel;
15     output y;
16
17     assign y = sel ? in_1 : in_0;        // A simple MUX
18 endmodule //mux
```

The Correct way to do it!

Example

```
1  module mux4( in_00, in_01, in_10, in_11 sel, y);
2      input  in_00, in_01, in_10, in_11;
3      input  [1:0] sel;
4      output y;
5
6      wire [3:0] low, high;
7
8      mux LOWMUX(
9                  .in_0   (in_00),
10                 .in_1   (in_01),
11                 .sel    (sel[0]),
12                 .y      (low)
13             );
14     mux HIGHMUX(
15                 .in_0   (in_10),
16                 .in_1   (in_11),
17                 .sel    (sel[0]),
18                 .y      (high)
19             );
```

Outline

Registers (DFF)

Positive Edge

```
1  module flop(clk, d, q);
2      input clk;
3      input d;
4      input q;
5
6      reg q;
7
8      always @(posedge clk)
9          q <= d;
10
11 endmodule //flop
```

Negative Edge

```
1  module neg_flop(clk, d, q);
2      input clk;
3      input d;
4      input q;
5
6      reg q;
7
8      always @(negedge clk)
9      begin
10         q <= d;
11     end
12
13 endmodule //neg_flop
```

Resettable Registers (DFFR)

Example

```
1  module flopr(clk, d, q, reset);
2      input clk, reset;
3      input [0:1] d;
4      input [0:1] q;
5
6      reg [1:0] q;
7
8      always @(posedge clk) //synchronous reset
9          if(resest)  q[0] <= 0;
10         else        q[0] <= d[0];
11
12     always @(posedge clk, posedge reset) //asynchronous reset
13         if(resest)  q[0] <= 0;
14         else        q[0] <= d[0];
15
16 endmodule //flopr
```

Enable Registers (DFFE)

Example

```
1  module flope(clk, d, q, reset, en);
2      input clk, reset, en, d;
3      output q;
4
5      reg q;
6
7      always @(posedge clk, posedge reset) //asynchronous reset
8      begin
9          if(resest)
10         begin
11             q <= 0;
12         end
13         else if(en)
14         begin
15             q <= d;
16         end
17     end
18 endmodule //flope
```

Multiple Registers

Example

```
1  module flop(clk, d, q);
2      input clk, d;
3      output q1;
4
5      reg q0, q1;
6
7      always @(posedge clk)
8      begin
9          q0 <= d;
10         q1 <= q0;
11     end
12 endmodule //flop
```

Latches

Transparent when clock is high.

Example

```
1  module latch(clk, d, q);
2      input clk, d;
3      output q;
4
5      reg q;
6
7      always @(clk, d) //senitivity list
8          if(clk) q <= d;
9
10 endmodule //latch
```

More Cool stuff

- Case Statements (4.5.1)
- Combinational if/else (4.5.2)
- Blocking / Non-Blocking Assignments (4.5.4)

Feedback

- Please give me your feedback
  - Email: cropj@eecs.oregonstate.edu
  - Text: (541) 250-0253
  - Twitter: @JoeCrop

Outline

Feedback

- Please give me your feedback
  - Email: cropj@eecs.oregonstate.edu
  - Text: (541) 250-0253
  - Twitter: @JoeCrop

Feedback

- Please give me your feedback
  - Email: cropj@eecs.oregonstate.edu
  - Text: (541) 250-0253
  - Twitter: @JoeCrop