Lecture 5
Verilog Language 1

ASIC Design Flow

- Application-specific Integrated Circuit (ASIC) Design Flow for Large-scale Digital Circuits

Design Language and Tools:

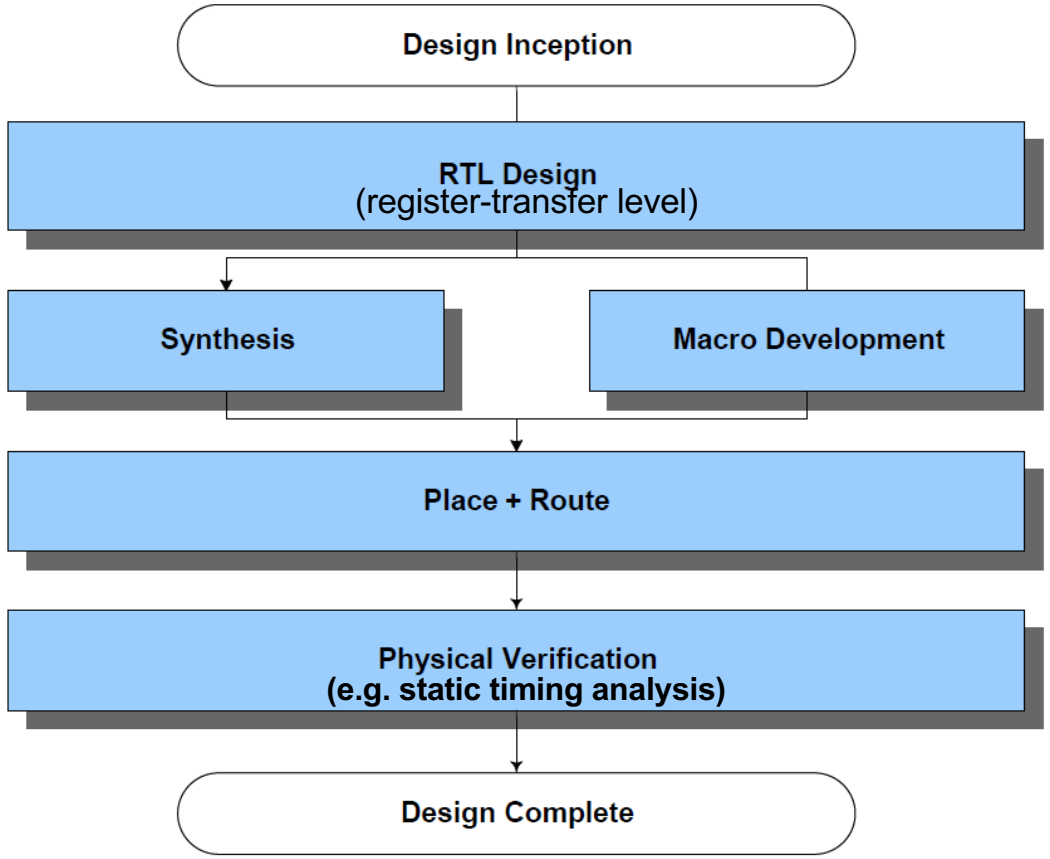
Matlab / C

Verilog/VHDL (sim by Modelsim, NC-sim)

Design Compiler / RTL Compiler

Encounter

Encounter/Primetime / ETS



Verilog and VHDL

- Hardware description language (HDL)
  - Text description
- Used to define digital logic and circuits
- Purpose:
  - Model and simulate digital system
  - Synthesize digital circuits
    - Convert from high level logic description to low level gate netlists

Verilog and VHDL

- Two popular languages
  - Verilog
    - IEEE Standard 1364-1995/2001/2005
    - Based on the C language
  - VHDL: **V**ery High Speed Integrated Circuits **H**ardware **D**escription **L**anguage
    - Developed by Department of Defense (DOD) from 1983
    - IEEE Standard 1076-1987/1993/200x
    - Based on the ADA language

Verilog and VHDL

- Verilog/VHDL has become industry standard language for hardware description
- Widely used for design:
  - General purpose CPU/GPU
  - Application Specific Integrated Circuits (ASIC)
  - FPGA
  - System level design model and verification
    - High level model
    - Mixed-signal Circuits: analog and digital co-design

Verilog and VHDL

- Verilog vs VHDL
  - Both of them are supported by commercial design tools
  - Both of them are sufficient to complete the design and verification task
  - Verilog: simpler, easier to use, C-like
    - Slightly more popular in industry
  - VHDL: more sophisticated, precise, more data types, less error prone

Types of Verilog Codes

- Three types of Verilog codes
  - Behavioral:
    - Describe behavior of system
    - No indication of hardware realization
    - Not always synthesizable
    - Used to build model and testbench
  - Register-Transfer-Level (RTL):
    - Describe circuit and data operation between registers
    - Synthesizable (Can be directly converted into hardware)
    - Used to build real digital circuits
  - Gate Level:
    - Describe physical gate level implementation
    - Usually generated from RTL code

Basic Syntax-General

- Case sensitive
- // used as comment line
- Keywords are case sensitive: e.g. module, input, output, etc.
- Start with “module”, end with “endmodule”
- Separate sentence by “;”
- Separate keywords by “,”

Modules

- Logic is contained within modules
- Ports: declared set of inputs, outputs, and "inout"s
- Internal contents:
  - Wires (wire)
  - Registers (reg)
  - Submodules
- ports, wire, reg without explicit range defaults to single bit

Verilog HDL Models

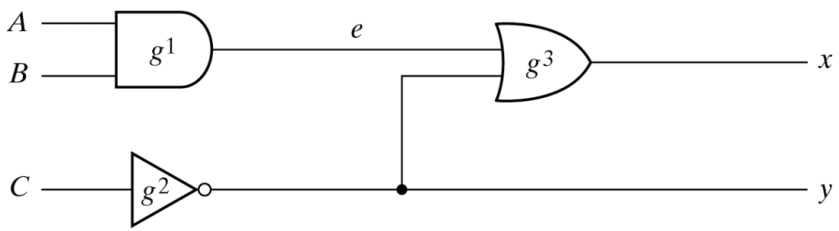
- HDL model specifies the relationship between input signals and output signals
- HDL uses special constructs to describe hardware concurrency, parallel activity flow, time delays and waveforms

Verilog code for a AND gate

```
module and_gate(y, x1, x2);
input x1, x2;
output y;
and(y, x1, x2);
endmodule
```

Modules

```
module my_ckt(A,B,C,x,y);
    input A,B,C;          // ports
    output x,y;           // ports
    wire e;               // internal signals
    and g1(e,A,B);        // submodules
    not g2(y,C);          // submodules
    or g3(x,e,y);         // submodules
endmodule
```



In class example

```
1  module class_example (a, b, c, x, y);
2    //list ports and directions and range
3    input a, b, c;
4    output x, y;
5
6    wire e; //internal connections are of type wire
7
8    //structural or gate level
9   // and g1(e, a, b);
10  // not g2(y, c);
11 //  or g3(x, e, y);
12
13   //rtl style
14
15   assign e = a & b;
16   assign y = !c;
17   assign x = e | y;
18
19 endmodule
```

In class example: testbench

```
1  `timescale 1ns / 1ps
2  module example_tb ();
3
4    // connections for probes that link the test vectors
5    // and collecte results to/from the design under test
6
7    // test vectors supplied by the testbench are always reg
8    reg in_x1;
9    reg in_x2;
10   reg in_x3;
11
12   // signals continuously driven by the DUT are wire
13   wire out_x;
14   wire out_y;
15
16   // DUT instantiation
17   class_example dut (
18     .a     (in_x1),
19     .b     (in_x2),
20     .c    (in_x3),
21     .x     (out_x),
22     .y     (out_y)
23   );
24
25   // Test stimulus
26   initial begin
27     // Use the monitor task to display the FPGA IO
28     $monitor("time=%3d, a=%b, b=%b, c=%b, x=%b, y=%b \n",
29              $time, in_x1, in_x2, in_x3, out_x, out_y);
30     // Generate each input with a 20 ns delay between them
31     in_x1 = 1'b0;
32     in_x2 = 1'b0;
33     in_x3 = 1'b0;
34     #20
35     in_x3 = 1'b1;
36     #20
37     in_x1 = 1'b1;
38     in_x2 = 1'b1;
39     #20
40     in_x1 = 1'b0;
41   end
42 endmodule : example_tb
43
```

Ports

- Declare ports as “input”, “output”
- Also declare ports with number of bits (bus)
- Examples:

```
input clk;
input [15:0] data_in; //explicit range
output [7:0] count;   //explicit range
```

Numbers in Verilog

- Integer number can be specified in
  - Decimal;
  - Hexadecimal;
  - Octal;
  - Binary;
- If size is unspecified, stored as 32-bit binary number

Numbers in Verilog

Integer	Stored as
1	00000000000000000000000000000001
8'hAA	10101010
6'b100011	100011
'hF	00000000000000000000000000001111
16’ bZ	ZZZZZZZZZZZZZZZZ
8’ bx	xxxxxxxx

Numbers in Verilog

- Logic values representing electrical phenomena, other than high voltage and ground voltage:
  - X, Z are not synthesizable, only used for simulation.

Logic Value	Description
0	zero, low, false
1	one, high, true
z or Z	high impedance, floating
x or X	unknown, uninitialized, contention

- Real numbers: e.g. 1.2, 0.6
  - Not synthesizable (only used for behavior simulation)

Numbers in Verilog

- Unsigned and signed integers
  - By default, all numbers are "unsigned"
  - Use "-" sign for negative number
    - e.g. -16'h1234 (EDCC in HEX or 1110110111001100 in binary)
  - Negative number is stored in 2's complement format

Numbers in Verilog

```
1  module test_number(a,b,c);
2    input signed [3:0] a;
3    input signed [3:0] b;
4
5    output signed [3:0] c;
6
7      assign c=a*b;
8
9  endmodule
10
```

Signed operations must always be assigned to signed outputs

Numbers in Verilog

```
1  `timescale 1ns / 1ps
2  module number_rep_tb ();
3
4    // connections for probes that link the test vectors
5    // and collecte results to/from the design under test
6
7    // test vectors supplied by the testbench are always reg
8    reg signed [3:0] in_a;
9    reg signed [3:0] in_b;
10
11
12   // signals continuously driven by the DUT are wire
13   wire signed [3:0] out_c;
14
15
16   // DUT instantiation
17   test_number dut (
18     .a      (in_a),
19     .b      (in_b),
20     .c      (out_c)
21   );
22
23   // Test stimulus
24   initial begin
25     // Use the monitor task to display the FPGA IO
26   //  $monitor("time=%3d, a=%b, b=%b, c=%b \n",
27           //    $time, in_a, in_b, out_c);
28     $monitor("time=%3d, a=%d, b=%d, c=%d \n",
29               $time, in_a, in_b, out_c);
30
31         in_a= -2;
32         in_b= 1;
33   end
34 endmodule : example_tb
```

```
xcelium> run
time=  0, a=1110, b=0001, c=1110
```

Numbers in Verilog

```
1  module test_number(a,b,c);
2    input signed [3:0] a;
3    input signed [3:0] b;
4
5    output [3:0] c;
6
7      assign c=a*b;
8
9  endmodule
10
```

If c is defined as unsigned, the simulation result would be incorrect, c = 14.

```
xcelium> run
time=  0, a=1110, b=0001, c=1110
```

Signals

- Two types of signals:
  - wire: connection between components
    - Need to have ONLY one driver
    - Does not retain value, i.e. no memory
    - Always accesible
  - reg: registers
    - Treated as flip-flops
    - Retain values until it is updated
- Which one to use?
  - Think about circuit realization, e.g. reg is used in sequential circuits to store values
  - reg is also used in behavior simulation as internal stimulus

Examples: reg a;

```
wire signed [31:0] b;  //this is a bus
```

Verilog Operators

- Arithmetic Operators:

  +, -, *, /, % (modulus)

- Relational Operators:

Operator	Description
a	a less than b
a>b	a greater than b
a<=b	a less than or equal to b
a>=b	a greater than or equal to b

Verilog Operators

- Equality Operators

Operator	Description
a === b	a equal to b, including x and z (Case equality)
a !== b	a not equal to b, including x and z (Case inequality)
a == b	a equal to b, resulting may be unknown (logical equality)
a != b	a not equal to b, result may be unknown (logical equality)

(if either operand is x or z, the result is x for == or !=)

Verilog Operators

- Logical Operators

Operator	Description
!	logic negation
&&	logical and
||	logical or

- Bit-wise Operators

Operator	Description
~	negation
&	and
|	inclusive or
^	exclusive or
^~ or ~^	exclusive nor (equivalence)

(For operands with unequal length, the shorter operand is 0-filled at MSBs)

Verilog Operators

- Shift Operators

Operator	Description
<<	left shift
>>	right shift

  - Example:

```
4'b1001 >> 1;  // result is 4'b0100
4'b1001 << 1;  // result is 4'b0010
```

Verilog Operators

- Concatenation Operators
  - Group multiple bits into a bus;
  - { }
  - Examples:

{a, b[3:0], c, 4’b1001} // result has 24 bits if a, c are 8-bits.

Verilog Operators

- Conditional Operators

  cond_expr ? true_expr : false_expr

Examples:

```
assign out = (sel==0) ? in0: in1;
```

(This code is synthesized into a multiplexer)

Verilog Operators

- Operator Precedence

Operator	Symbols
Unary, Multiply, Divide, Modulus	!, ~, *, /, %
Add, Subtract, Shift	+, – , <>
Relation, Equality	,<=,>=,==,!=,===,!===
Reduction	&, !&,^,^~,|,~|
Logic	&&, ||
Conditional	?