Lect 19

Verilog HDL

CS221: Digital Design

Dr. A. Sahu
Dept of Comp. Sc. & Engg.
Indian Institute of Technology Guwahati

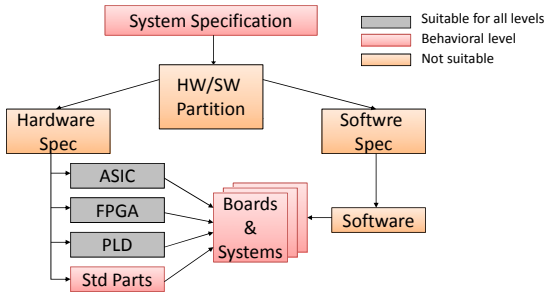
Outline

- HDL Programming : Verilog HDL
- HDL Rules
- HDL Module and Examples
- HDL levels : Data flow, Structural and Behavioral, UDP
- Testing and Simulation

What is Verilog

- Hardware Description Language (HDL)
- Developed in 1984
- Standard: IEEE 1364, Dec 1995

Application Areas of Verilog

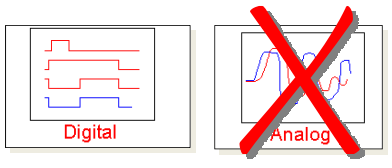


HDL, Area of Application

- Design Entry
- Logic Simulation
- Functional Verification
- Digital Circuit Synthesis
- Timing Verification
- Fault Simulation
- Documntation

Basic Limitation of Verilog

Description of digital systems only



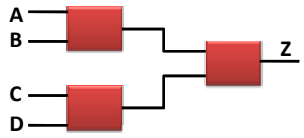
Abstraction Levels in Verilog

- Behavioral
- RTL
- Gate
- Layout (VLSI)

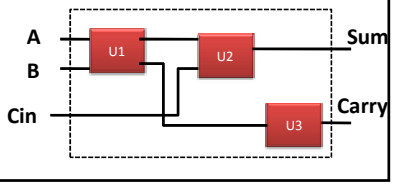
Our focus

Main Language Concepts (i)

- Concurrency

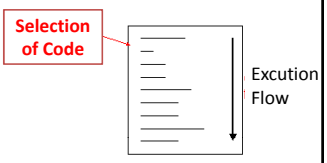


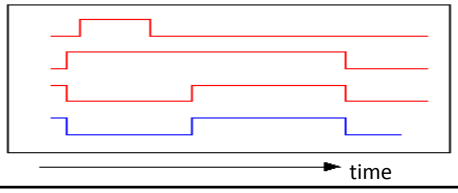
- Structure



Main Language Concepts (ii)

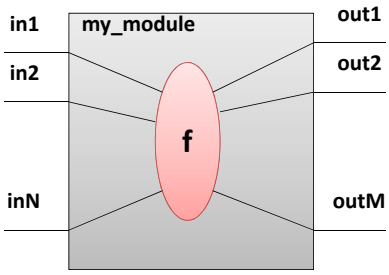
- Procedural Statements
- Time





Lets Start with an Example of Verilog HDL module

Module



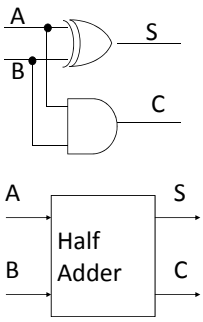
Everything you write in Verilog must be inside a module
exception: compiler directives

Module

```
module my_module(out1, .., inN);
output out1, .., outM;
input in1, .., inN;
.. // declarations
.. // description of f (maybe
.. // sequential)
endmodule
```

Everything you write in Verilog must be inside a module
exception: compiler directives

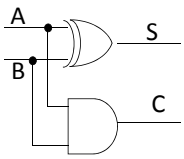
Example: Half Adder




```
module half_adder(S,C,A,B);
output S, C;
input A, B;

wire S, C, A, B;

assign S = A ^ B;
assign C = A & B;

endmodule
```

9/9/2018

Example: Half Adder
(Data Flow Model: using Boolean Equations)




```
module half_adder(S,C,A,B);
output S, C;
input A, B;

wire S, C, A, B;

assign S = A ^ B;
assign C = A & B;

endmodule
```

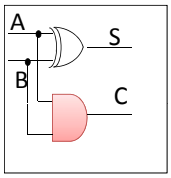
```
assign S = A ^ B;
assign C = A & B;
```

Same

```
assign C = A & B;
assign S = A ^ B;
```

9/9/2018

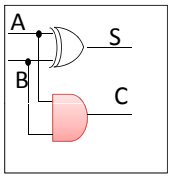
Example: Half Adder, 2nd Implementation




```
module half_adder(S,C,A,B);
output S, C;
input A, B;

wire S, C, A, B;

xor  (S, A, B);
and  (C, A, B);

endmodule
```

9/9/2018

Example: Half Adder, 2nd Implementation Using Structural/gate primitive inter connection




```
module half_adder(S,C,A,B);
output S, C;
input A, B;

wire S, C, A, B;

xor  (S, A, B);
and  (C, A, B);

endmodule
```

```
xor  (S, A, B);
and  (C, A, B);
```

Same

```
and  (C, A, B);
xor  (S, A, B);
```

Verilog HDL Languages

9/9/2018

User Identifiers

- Formed from {[A-Z], [a-z], [0-9], _, $}
- Can't begin with $ or [0-9]
  - `myidentifier`
  - `m_y_identifier`
  - `3my_identifier`
  - `$my_identifier`
  - `_myidentifier$`
- Case sensitivity : `myid` ≠ `Myid`

9/9/2018

Comments : same as C++ Style

- `// The rest of the line is a comment`
- `/* Multiple line comment */`
- `/* Nesting /* comments */ do NOT work */`

Verilog Value Set

- ***0*** represents low logic level or false condition
- ***1*** represents high logic level or true condition
- ***x*** represents unknown logic level
- ***z*** represents high impedance logic level == > open circuit

Truth Tables (Updated..)

AND	0	1	X	Z
0	0	0	0	0
1	0	1	X	X
X	0	x	x	X
Z	0	x	x	x

OR	0	1	X	Z
0	0	1	X	X
1	1	1	1	1
X	X	1	X	X
Z	X	1	X	X

XOR	0	1	X	Z
0	0	1	X	X
1	1	0	X	X
X	X	x	x	X
Z	x	x	x	x

NOT	0	1	X	Z
OUT	1	0	X	X

Sorry: There were two mistakes in this Slide, now corrected

Numbers in Verilog (i)

<size>'<radix> <value>

- No of bits
- Binary → b or B; Octal → o or O; Decimal → d or D; Hexa → h or H
- Consecutive chars 0-f, x, z

Examples:

– 8'h ax = 1010 xxxx

– 12'o 3zx7 = 011 zzz xxx 111

Numbers in Verilog (ii)

- You can insert "_" for readability
  - 12'b 000_111_010_100
  - 12'b 000111010100
  - 12'o 07_24

  } Represent the same number
- Bit extension
  - MS bit = 0, x or z ⇒ extend this
    - 4'b x1 = 4'b xx_x1
  - MS bit = 1 ⇒ zero extension
    - 4'b 1x = 4'b 00_1x

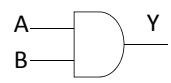
Numbers in Verilog (iii)

- If *size* is ommitted it
  - is inferred from the *value* or
  - takes the simulation specific number of bits or
  - takes the machine specific number of bits
- If *radix* is ommitted too .. **decimal is assumed**
  - 15 = <size>'d 15

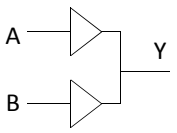
Nets (i)

- Can be thought as hardware wires driven by logic
- Equal *z* when unconnected
- Various types of nets
  - `wire`
  - `wand` (wired-AND)
  - `wor` (wired-OR)
  - `tri` (tri-state)
- In following examples: Y is evaluated, ***automatically***, every time A or B changes

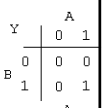
Nets (ii)



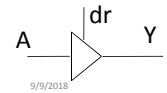

```
wire Y;  // declaration
assign Y = A & B;
```




```
wand Y; // declaration
assign Y = A;
assign Y = B;
```




```
wor Y;  // declaration
assign Y = A;
assign Y = B;
```




```
tri Y;  // declaration
assign Y = (dr) ? A : z;
```


Registers

- Variables that store values
- Do not represent real hardware
- *But real hardware can be implemented with registers*
- Only one type: `reg`

```
reg A, C; // declaration
// assignments are always done inside a
//procedure
  A = 1;
  C = A; // C gets the logical value 1
  A = 0; // C is still 1
  C = 0; // C is now 0
```

- Register values are updated explicitly!!

Vectors

- Represent buses

```
wire [3:0] busA;
reg  [1:4] busB;
reg  [1:0] busC;
```

- Left number is MS bit
- Slice management

```
busC = busA[2:1];
```

⟺

```
busC[1] = busA[2];
busC[0] = busA[1];
```

- Vector assignment (***by position!!***)

```
busB = busA;
```

⟺

```
busB[1] = busA[3];
busB[2] = busA[2];
busB[3] = busA[1];
busB[4] = busA[0];
```


Integer & Real Data Types

- Declaration

```
integer i, k;
real r;
```

- Use as registers (inside procedures)

```
 i = 1;// Asgnmts occur in procedure
 r = 2.9;
 k = r; // k is rounded to 3
```

- Integers are not initialized!!
- Reals are initialized to *0.0*

Time Data Type

- Special data type for simulation time measuring
- Declaration

```
time my_time;
```

- Use inside procedure

```
my_time = $time;
// get current sim time
```

- Simulation runs at simulation time, not real time

Strings

- Implemented with regs:

```
reg [8*13:1] string_val; // can hold up to 13 chars
string_val = "Hello Verilog";
string_val = "hello"; // MS Bytes are filled with 0
string_val = "I am overflowed"; // "I " is truncated
```

- Escaped chars:
  - \n newline \t tab
  - %% % \\ \
  - \" "

Logical Operators

- && → logical AND
- || → logical OR
- ! → logical NOT
- Operands evaluated to ONE bit value: *0*, *1* or *x*
- Result is ONE bit value: *0*, *1* or *x*

```
A = 6;
B = 0;
C = x;
```

→

```
A && B → 1 && 0 → 0
A || !B → 1 || 1 → 1
C || B → x || 0 → x
```

but C && B =0

Bitwise Operators (i)

- & → bitwise AND
- | → bitwise OR
- ~ → bitwise NOT
- ^ → bitwise XOR
- ~^ or ^~ → bitwise XNOR
- Operation on bit by bit basis
- Two operand needed

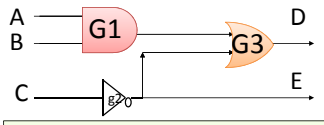
Reduction Operators

- & → AND
- | → OR
- ^ → XOR
- ~& → NAND
- ~| → NOR
- ~^ or ^~ → XNOR

- **One multi-bit operand → One single-bit result**

```
a = 4'b1001;
c = |a; // c = 1|0|0|1 = 1
```


Example: using primitive gates



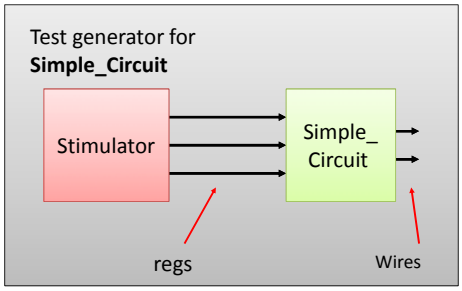

```
module    Simple_Circuit (A,B,C,D,E);
          output    D,E;
          input     A,B,C;
          wire      w1;
          and       G1 (w1,A,B);
          not       G2 (E,C);
          or        G3 (D,w1,E);
Endmodule
```


Delays

- Propagation delay
  - Specified in terms of time units
  - Specified by the symbol #

    and #(10) G1 (w1, A,B)
- Association of time unit and time scale is made with the compiler directive 'timescale
  - Specified before the declaration of a module
  - 'timescale 1 ns/100ps indicates unit of measurement for time delay

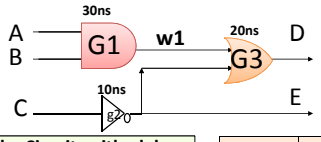
Test Bench



Test Bench for Simple_Circuit

```
module t_Simple_Circuit_delay;
wire    D,E;
reg     A,B,C;
Simple_Circuit_delay M1 (A,B,C,D,E);  //instantiation of M1
initial
  begin
    A= 1'b0; B=1'b0; C=1'b0;
   #100 A=1'b1; B=1'b1; C=1"b1;
  end
Initial #200 $finish;
endmodule
```

Example With Delay




```
module     Simple_Circuit_with_delay (A,B,C,D,E);
        output      D,E;
        input       A,B,C;
        wire         w1;
        and         #(30) G1 (w1,A,B);
        not         #(10)  G2 (E,C);
        or          #(20)  G3 (D,w1,E);
endmodule
```

Time (ns)	ABC	Ew1D
Initial	0 0 0	1 0 1
initial	1 1 1	1 0 1
10	1 1 1	0 01
20	1 1 1	0 0 1
30	1 1 1	0 1 0
40	1 1 1	0 1 0
50	1 1 1	0 1 1

Example