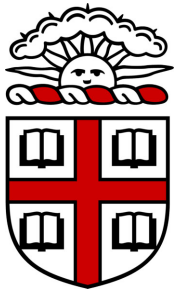
EN164: Design of Computing Systems

Lecture 05: Lab Foundations / Verilog 1

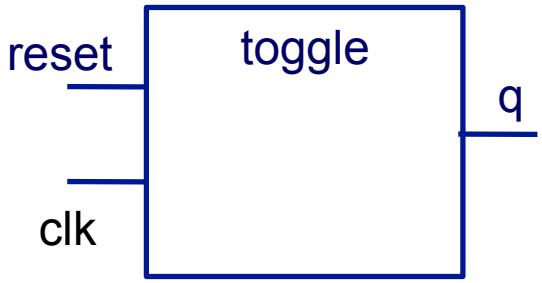
Professor Sherief Reda
http://scale.engin.brown.edu
Electrical Sciences and Computer Engineering
School of Engineering
Brown University
Spring 2011



Introduction to Verilog

- Differences between hard definition language (HDL) and software languages? Concurrency, propagation of time, signal dependency or sensitivity
- Verilog is case sensitive and syntax is similar to C
- Comments are designated by // to the end of a line or by /* to */ across several lines.
- Many online textbooks available from Brown library
  - Introduction to Logic Synthesis using Verilog HDL
  - Verilog Digital System Design
  - Verilog Quickstart
  - The Verilog Hardware Description Language

Verilog modules



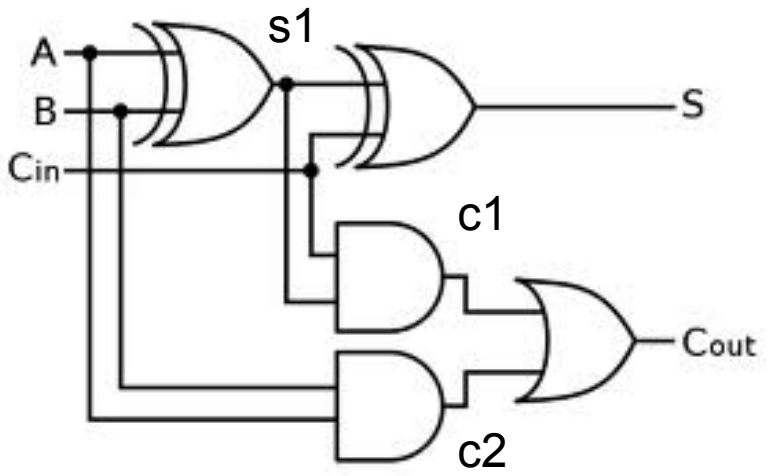
The functionality of each module can be defined with three modeling levels:

1. *Structural or gate level*
2. *Dataflow level*
3. *Behavioral or algorithmic level*

Verilog allows different levels of abstraction to be mixed in the same module.

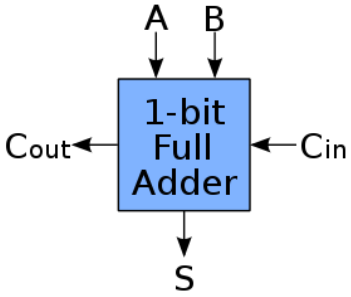
1. Structural modeling -- Data types: bits

- Nets represent connections between hardware elements. They are continuously driven by output of connected devices. They are declared using the keyword `wire`.
  - `wire s1;`
  - `wire c1, c2;`
  - `wire d=0;`



Modules and ports

```
module FA(A, B, Cin, S, Cout);
input A, B, Cin;
output S, Cout;
...
endmodule
```



- All port declarations (input, output, inout) are implicitly declared as `wire`.
- If an output should hold its value, it must be declared are `reg`

```
module FA(input A, input B, input Cin, output S, output Cout);
...
endmodule
```

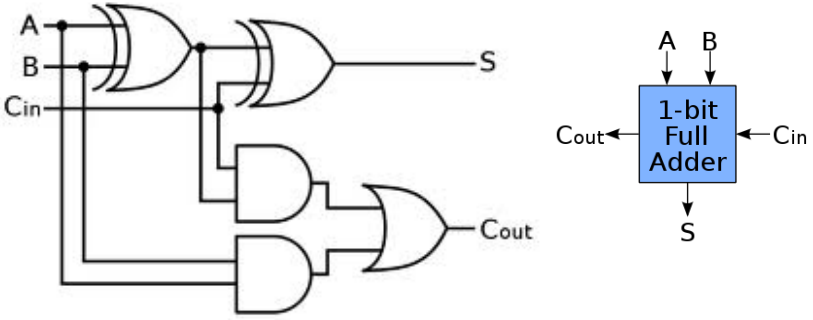
Gate level modeling

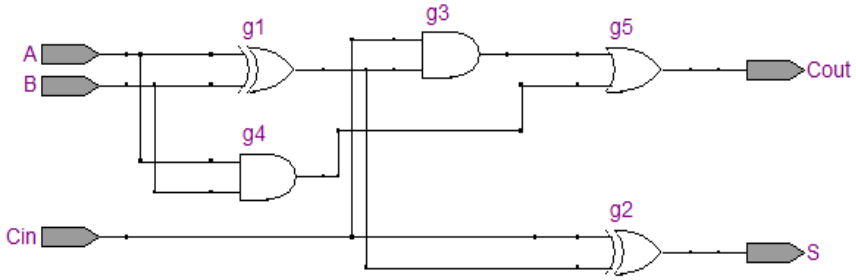
```
wire Z, Z1, OUT, OUT1, OUT2, IN1, IN2;

and a1(OUT1, IN1, IN2);
nand na1(OUT2, IN1, IN2);
xor x1(OUT, OUT1, OUT2);
not (Z, OUT);
buf final (Z1, Z);
```

- Essentially describes the topology of a circuit
- All instances are executed concurrently just as in hardware
- Instance name is not necessary
- The first terminal in the list of terminals is an output and the other terminals are inputs
- Not the most interesting modeling technique for our class

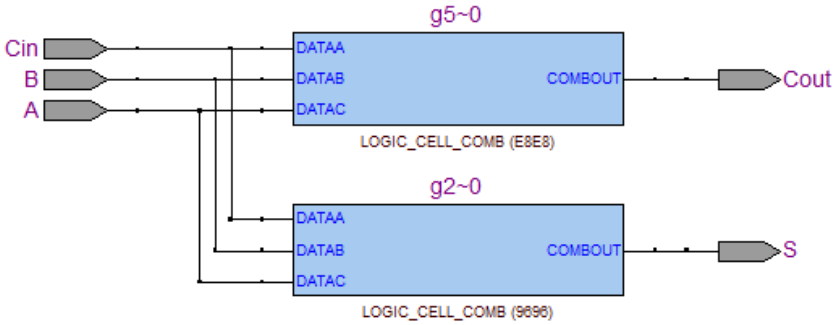
1 bit full adder example

```
module FA(A, B, Cin, S, Cout);
input A, B, Cin;
output S, Cout;
wire s1, c1, c2;

xor g1(s1, A, B);
xor g2(S, s1, Cin);
and g3(c1, s1, Cin);
and g4(c2, A, B);
or g5(Cout, c1, c2);
endmodule
```





[from Tools ➔ Netlist Viewers ➔ RTL Viewer]



[from Tools ➔ Netlist Viewers ➔ after technology mapping]

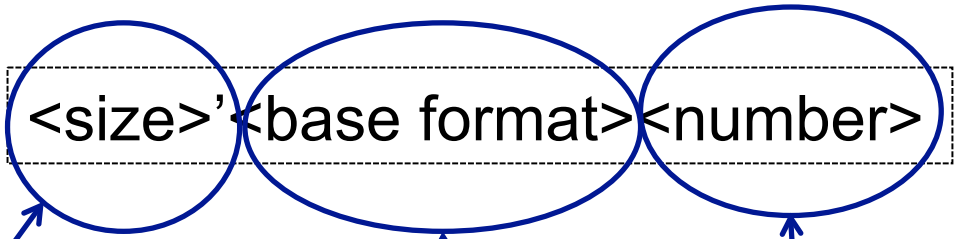
Data types: vectors

- A net or register can be declared as *vectors*. Example of declarations:
  - `wire a;`
  - `wire [7:0] bus;`
  - `wire [31:0] busA, busB, busC;`

- It is possible to access bits or parts of vectors
  - `busA[7]`
  - `bus[2:0]`
  - `virt_addr[0:2]`

Specifying values for wires and variables

- Number specification.



specifies the number of bits in the number

d or D for decimal
h or H for hexadecimal
b or B for binary
o or O for octal

Number depends on the base

Examples:

- `4'b1111`
- `12'habc`
- `16'd235`
- `12'h13x`
- `-6'd3`
- `12'b1111_0000_1010`

X or x: don't care
Z or z: high impedence
_ : used for readability

Modules with input / output vectors

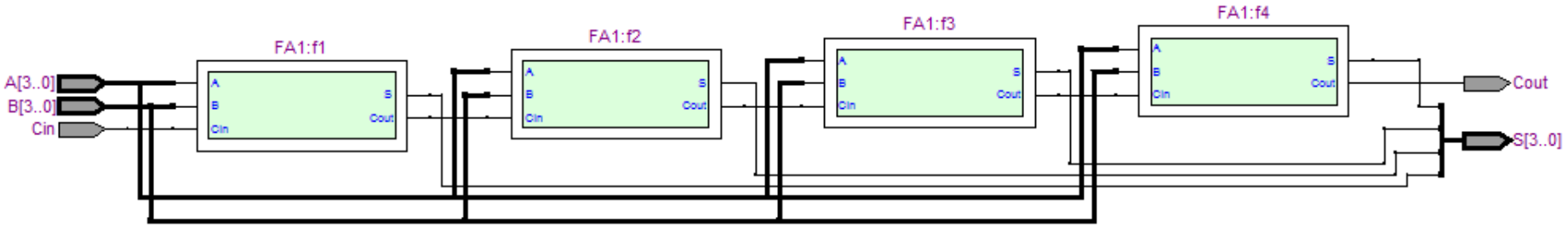
```
module fulladd4(output reg[3:0] sum,
                output reg c_out,
                input [3:0] a, b,
                input c_in);
…
…
endmodule
```

OR

```
module fulladd4(sum, c_out, a, b, input c_in);
output reg [3:0] sum
output reg c_out;
input [3:0] a, b;
input c_in;
…
…
endmodule
```

Module instantiation

```
module fulladd4(A,  B,  Cin,  S, Cout);
input [3:0] A, B;
input Cin;
output [3:0] S;
output Cout;
wire C1, C2, C3;

FA f1(A[0], B[0], Cin, S[0], C1);
FA f2(A[1], B[1], C1, S[1], C2);
FA f3(A[2], B[2], C2, S[2], C3);
FA f4(A[3], B[3], C3, S[3], Cout);
endmodule
```



[from Tools ➔ Netlist Viewers ➔ RTL Viewer]

Alternative form of module instantiation

```
module FA(A, B, Cin, S, Cout);
input A, B, Cin;
output S, Cout;
…
endmodule
```

```
wire a1, a2, a3, a4, a5
```

```
FA f1(a1, a2, a3, a4, a5);
```

OR

```
FA1 f1(.Cout(a5), .S(a4), .B(a2), .A(a1), .C_IN(a3));
```

Instantiation an array of gates

```
wire [7:0] OUT, IN1, IN2;

// array of gates instantiations
nand n_gate [7:0] (OUT, IN1, IN2);

// which is equivalent to the following
nand n_gate0 (OUT[0], IN1[0], IN2[0]);
nand n_gate1 (OUT[1], IN1[1], IN2[1]);
nand n_gate2 (OUT[2], IN1[2], IN2[2]);
nand n_gate3 (OUT[3], IN1[3], IN2[3]);
nand n_gate4 (OUT[4], IN1[4], IN2[4]);
nand n_gate5 (OUT[5], IN1[5], IN2[5]);
nand n_gate6 (OUT[6], IN1[6], IN2[6]);
nand n_gate7 (OUT[7], IN1[7], IN2[7]);
```

2. Dataflow modeling

- Module is designed by specifying the data flow, where the designer is aware of how data flows between hardware registers and how the data is processed in the design
- The continuous assignment is one of the main constructs used in dataflow modeling
  - `assign out = i1 & i2;`
  - `assign addr[15:0] = addr1[15:0] ^ addr2[15:0];`
  - `assign {c_out, sum[3:0]}=a[3:0]+b[3:0]+c_in;`
- A continuous assignment is always active and the assignment expression is evaluated as soon as one of the right-hand-side variables change
- Assign statements describe hardware that operates concurrently − ordering does not matter
- Left-hand side must be a scalar or vector net. Right-hand side operands can be wires, (registers, integers, and real)

Operator types in dataflow expressions

- Operators are similar to C except that there are no ++ or –

- Arithmetic: *, /, +, -, % and **
- Logical: !, && and ||
- Relational: >, <, >= and <=
- Equality: ==, !=, === and !==
- Bitwise: ~, &, |, ^ and ^~
- Reduction: &, ~&, |, ~|, ^ and ^~
- Shift: << and >>
- Concatenation: { }
- Replication: {{}}
- Conditional: ?: