IS C351
Computer Architecture & Organization
Lab

Week #1

Getting Started With Verilog

Getting Started With Verilog

- Motivation
- Introduction to Verilog
- First Verilog Program
- Testing Verilog Program
- Styles of Coding
- Structural Coding

Getting Started With Verilog

Motivation

Introduction to Verilog

First Verilog Program

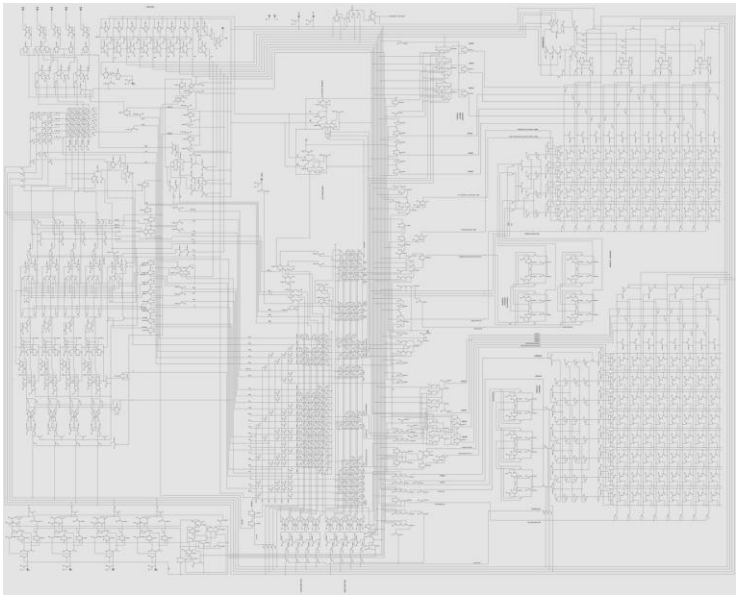
Testing Verilog Program

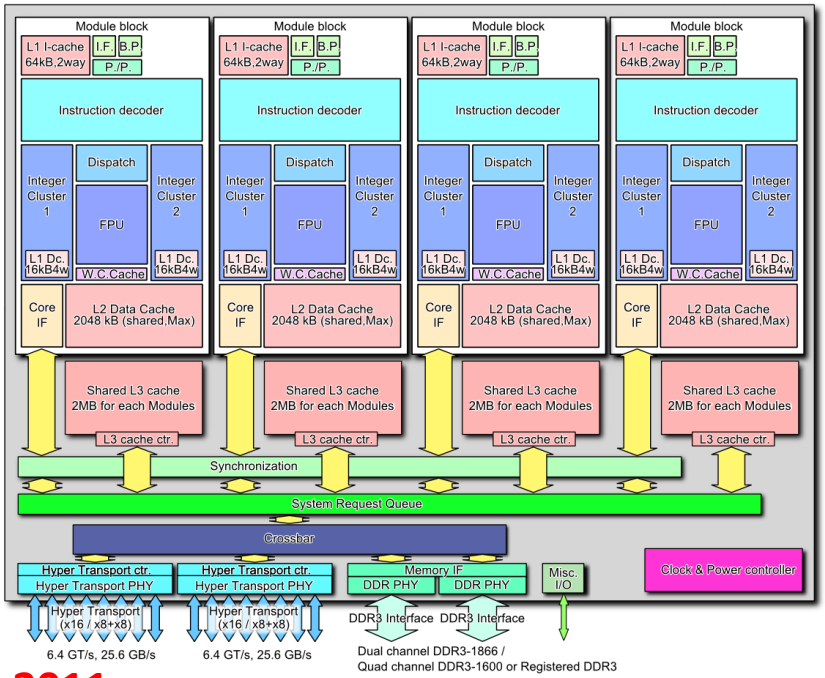
Styles of Coding

Structural Coding

Motivation

Scalability - From **gates** to **microprocessors**





1971

Intel 4004 microprocessor

~2,300 transistors @ 1 MHz

Hand-drawn schematic (circuit diagram)

One-man team

2011

AMD Bull Dozer microprocessor

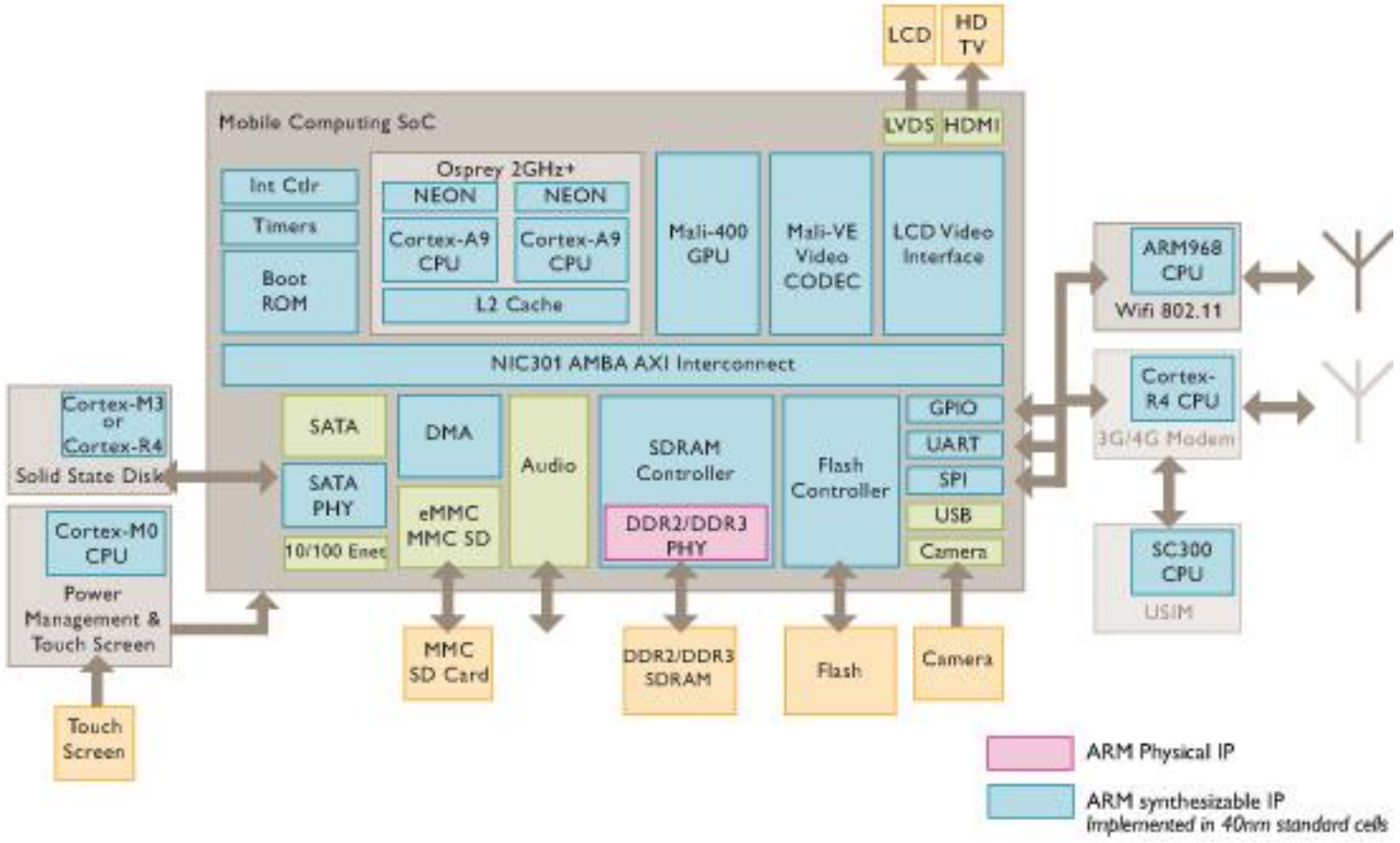
>1.2 Billon transistors @ 1GHz

Hierarchical Block Diagram

> 50 people

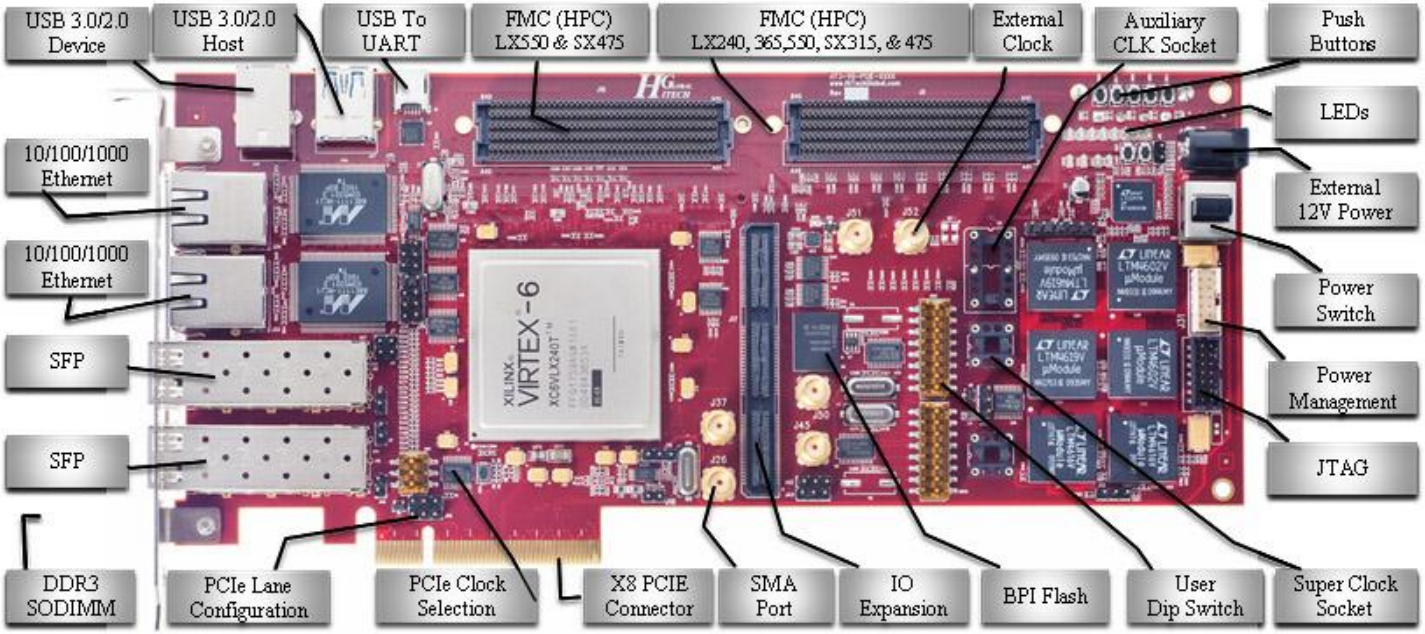
Motivation

Reuse – ARM Processors and related Intellectual Property



Motivation

Rapid Prototyping



How does it benefit to you?

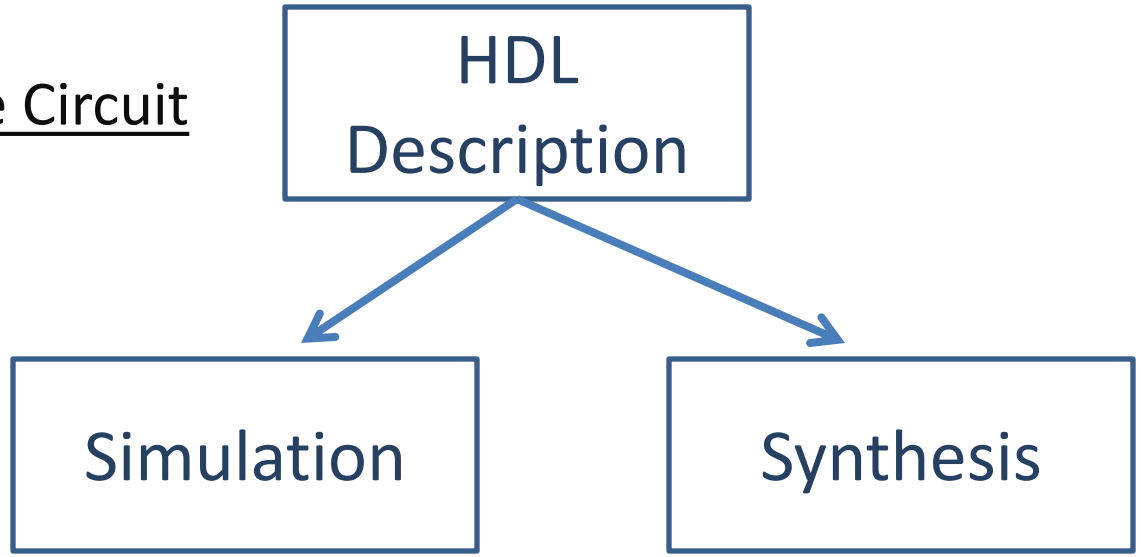
Makes hardware development approachable for the software engineers!!

Getting Started With Verilog

- Motivation
- **Introduction to Verilog**
- First Verilog Program
- Testing Verilog Program
- Styles of Coding
- Structural Coding

Context

Describe the Circuit



Verify the Circuit – functional, timing and power constraints

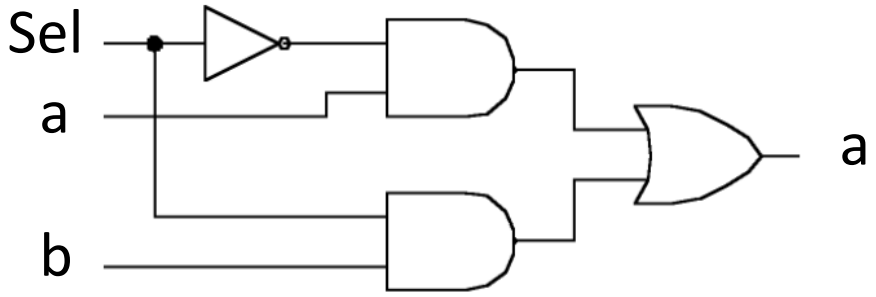
Create the Circuit – mapping to implementation platform

Context – Simple Example

HDL Description

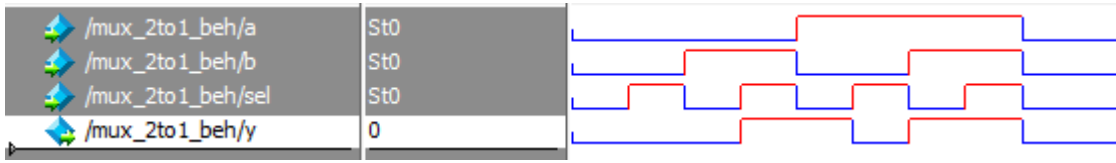
```
module mux(y,a,b,sel);
    output reg y;
    input wire a,b,sel;

    always @(*)
        begin
            if (sel)
                y = a;
            else
                y = b;
        end
endmodule
```



Synthesis

Simulation



Introduction to Verilog

- Originated at Automated Integrated Design Systems (renamed Gateway) in 1985
- C-like (not C)
- Originally for simulation; synthesis added later.
- Case sensitive, weakly typed language
- New additions to the language – System Verilog and Verilog-AMS

Getting Started With Verilog

- Motivation
- Introduction to Verilog
- **First Verilog Program**
- Testing Verilog Program
- Styles of Coding
- Structural Coding

Simple AND Gate in Verilog

```
module and_gate(out,in1,in2);
        output wire out;
        input wire in1,in2;
        //use and gate primitive
        and   AND0(out,in1,in2);
endmodule
```

Keywords in Verilog

```
module and_gate(out,in1,in2);
        output wire out;
        input wire in1,in2;
        //use and gate primitive
        and   AND0(out,in1,in2);
endmodule
```

■ **Verilog key words**

Module Details in Verilog

module name

```
module and_gate(out,in1,in2);
        output wire out;
        input wire in1,in2;
        //use and gate primitive
        and   AND0(out,in1,in2);
endmodule
```

ports* with output ports followed by input ports

Component instantiation

port – input / output connection(wire) of a digital circuit

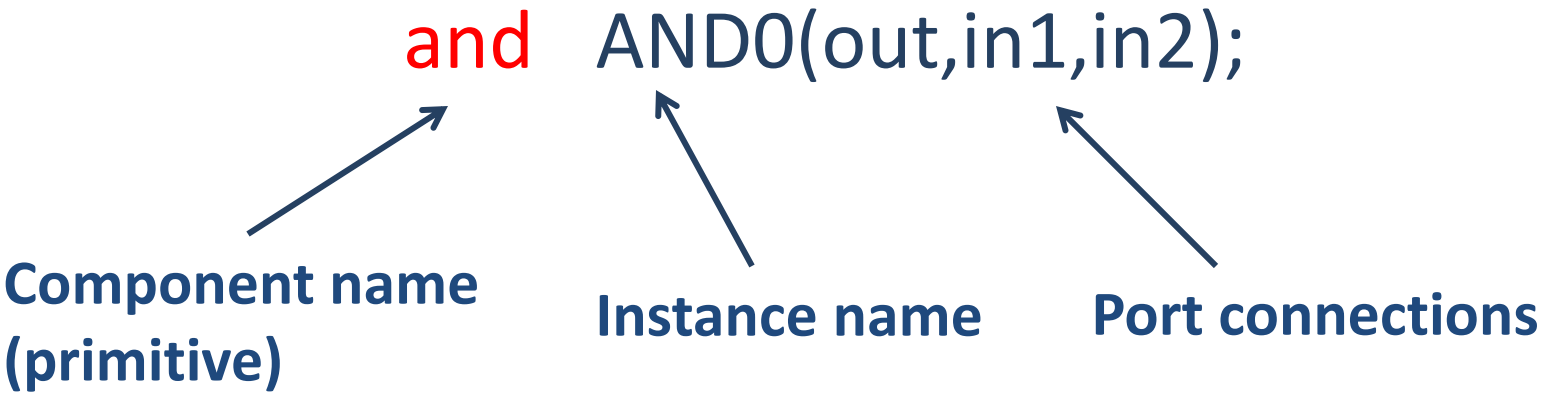
Port Declarations in Verilog


```
module and_gate(out,in1,in2);
        output wire out;
        input wire in1,in2;
        //use and gate primitive
        and   AND0(out,in1,in2);
endmodule
```

output port declaration

input port declaration

Component Instantiation in Verilog



- **Primitive – basic component provided by Verilog**
- **User defined primitives (UDP) are also possible (discussed later)**

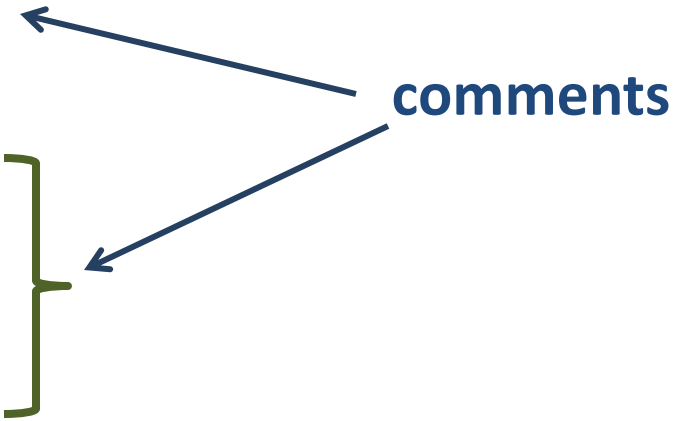
Semi-colons in Verilog


```
module and_gate(out,in1,in2);
        output wire out;
        input wire in1,in2;
        //use and gate primitive
        and   AND0(out,in1,in2);
endmodule
```

most statements end with semi-colon

Comments in Verilog

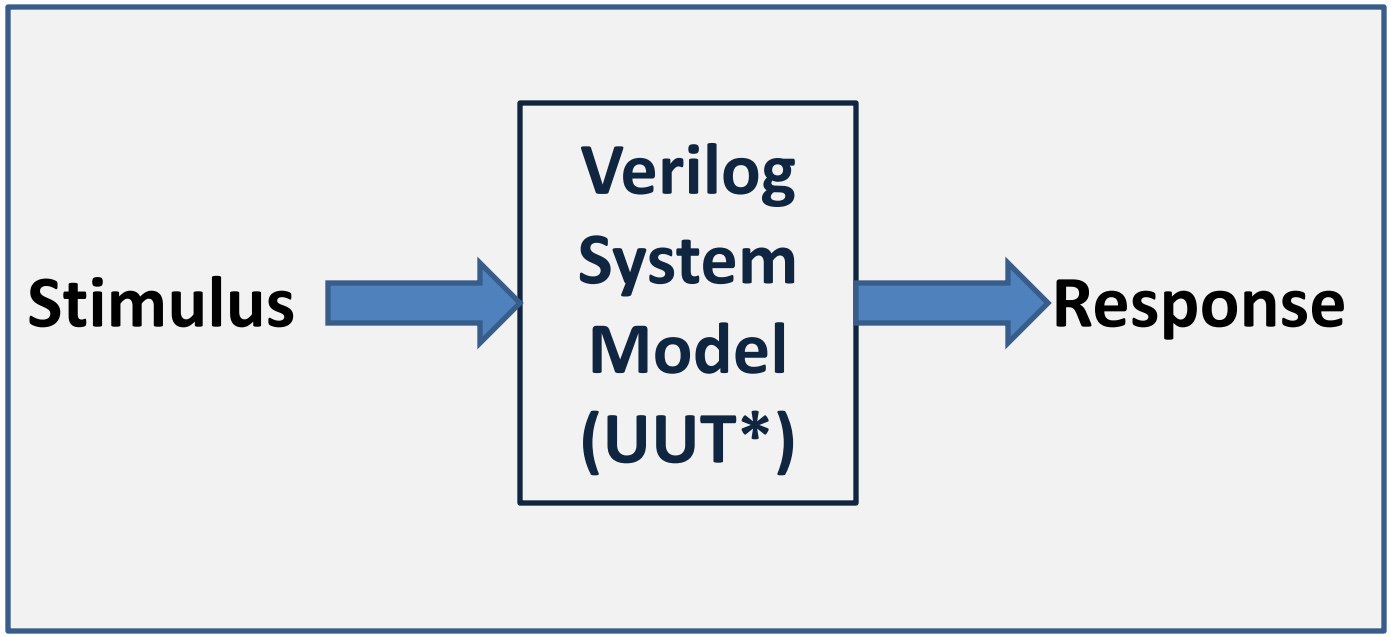
```
module and_gate(out,in1,in2);
        output wire out;
        input wire in1,in2;
        //use and gate primitive
        and    AND0(out,in1,in2);
       /*
        multi-line comment
       */
endmodule
```



Getting Started With Verilog

- Motivation
- Introduction to Verilog
- First Verilog Program
- **Testing Verilog Program**
- Styles of Coding
- Structural Coding

Testbench



UUT – Unit Under Test

Testbench Structure

```
`timescale 1ns/1ps
module tb_<foo_module_name> ();
        //declare all outputs of UUT as wires
        //declare all inputs of UUT as registers

        //instantiate the component (UUT – unit under test)

        //optional block
        initial
           begin
                //initialization code
           end
```

…continued

Testbench Structure

```
        //optional block
        always
            begin
                    //iterative code
            end
endmodule
```

Testbench for AND gate

```
`timescale 1ns/1ps

module tb_and_gate();
        wire   out;          //all outputs become wires
        reg    in1,in2;      //all inputs become registers

        and_gate   AND0(out,in1,in2);
```

…continued

Testbench for AND gate

```
//executed at the beginning of simulation
initial
   begin
                in1 = 1'b0; in2 = 1'b0;
      #10       in1 = 1'b0; in2 = 1'b1;
      #10       in1 = 1'b1; in2 = 1'b0;
      #10       in1 = 1'b1; in2 = 1'b1;
      #10       $stop;
   end
endmodule
```

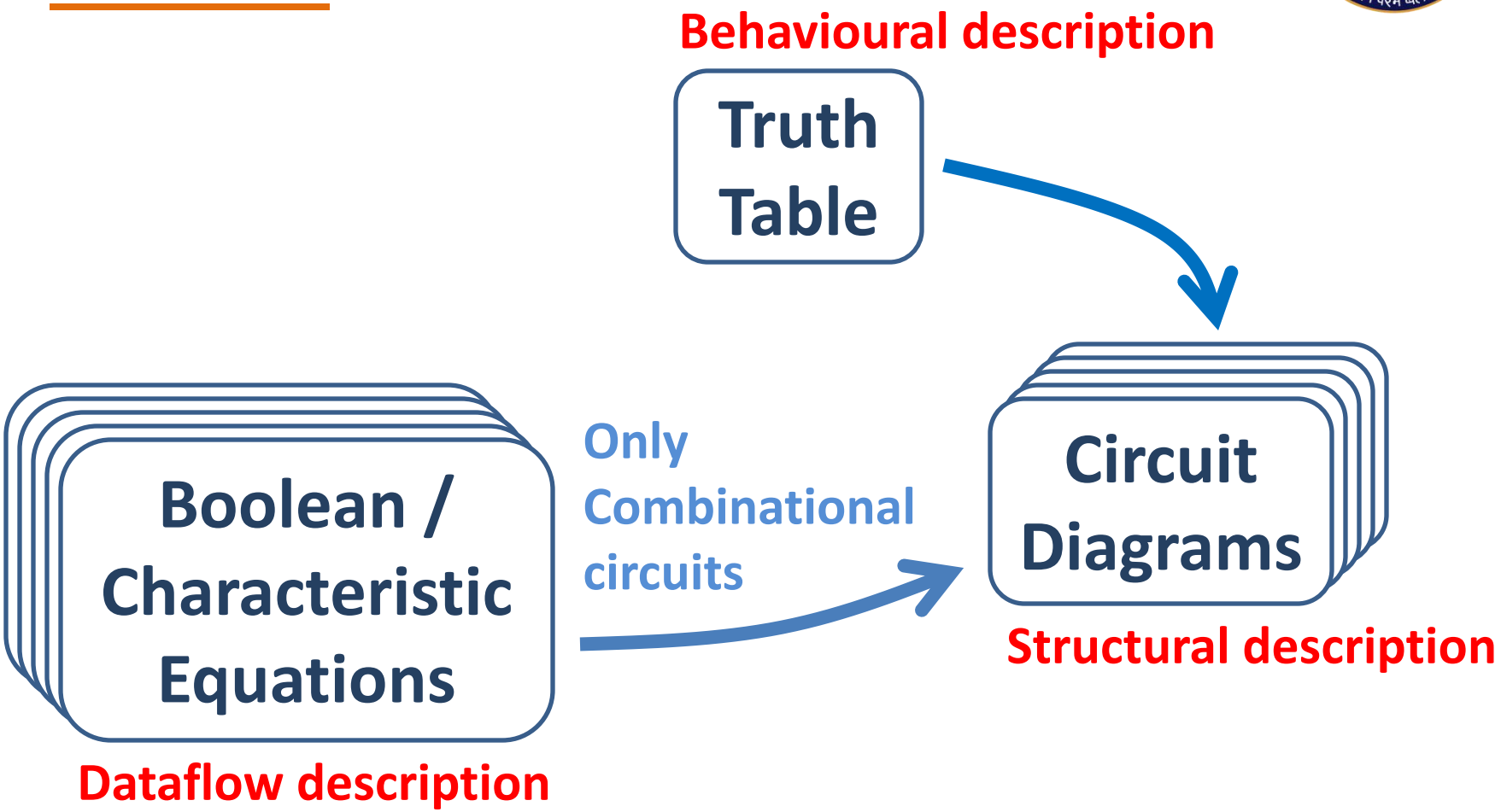
assignment delay

system task

Getting Started With Verilog

- Motivation
- Introduction to Verilog
- First Verilog Program
- Testing Verilog Program
- **Styles of Coding**
- Structural Coding

How Do We Describe a Digital Circuit?



Behavioural Description

- Describe the behaviour (truth-table) of the circuit
- Scalable description
- Synthesizable – Synthesizer generates the circuit automatically
- **Most preferred form for practical designs**

Behavioural Coding Example – Half Adder Circuit

Half Adder Truth Table

a	b	sum	carry
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Behavioural Coding Example – Half Adder Circuit


```
module half_adder (sum, carry, a, b);
  output  reg sum,carry;
  input wire a,b;

  always @(*)
        begin
```

...continued

Behavioural Coding Example – Half Adder Circuit


```
        casex ({a,b})
            2'b00:      sum = 1'b0; carry = 1'b0;
            2'b01:      sum = 1'b1; carry = 1'b0;
            2'b10:      sum = 1'b1; carry = 1'b0;
            2'b11:      sum = 1'b0; carry = 1'b1;
            default:    sum = 1'bx; carry = 1'bx;
        endcase
    end
endmodule
```

Dataflow Description

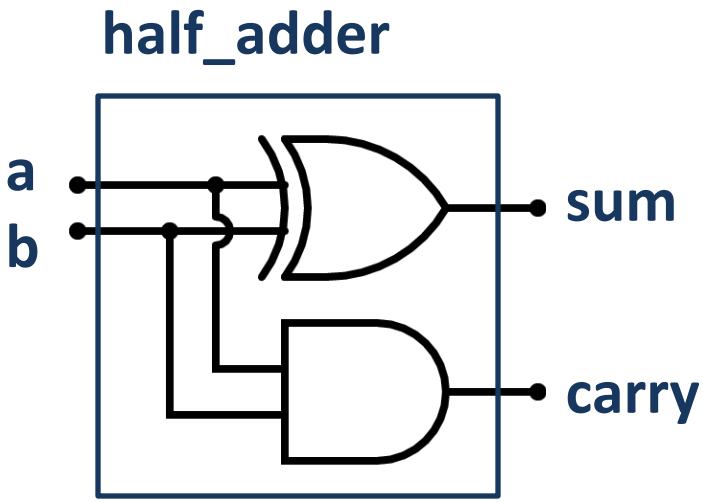
- Describe the functionality as boolean equations
- Mostly useful for functional verification
- Not always synthesizable
- Use with caution

Dataflow Coding Example – Half Adder Circuit

```
module half_adder (sum, carry, a, b);
  output  wire sum,carry;
  input wire a,b;

 assign sum = a ^ b;
 assign carry = a & b;

endmodule
```



Structural Description

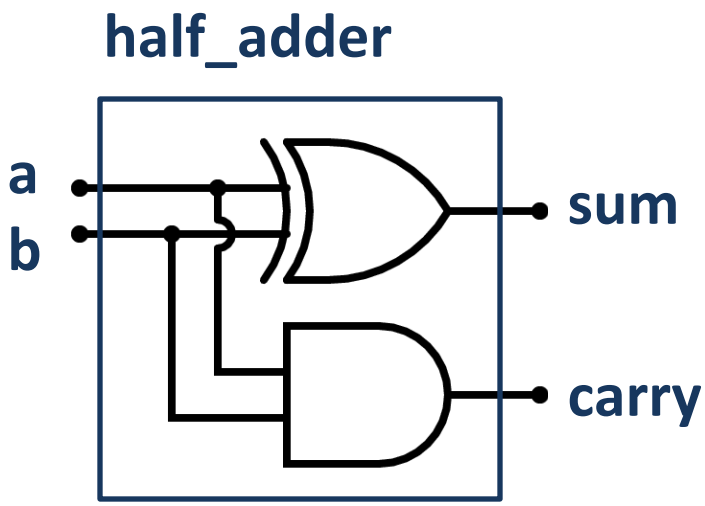
- Describe the circuit one module (component) at a time
- One-to-One correspondence with the synthesized circuit
- Useful in hierarchical design
- Very painful
- **What you say is what you get**

Structural Coding Example – Half Adder Circuit

```
module half_adder (sum, carry, a, b);
  output  wire sum,carry;
  input wire a,b;

  xor    XOR0(sum,a,b);
  and    AND0(carry,a,b);

endmodule
```



One Last Word on Styles

What's in a name? that which we call a rose

By any other name would smell as sweet;

- Juliet to Romeo

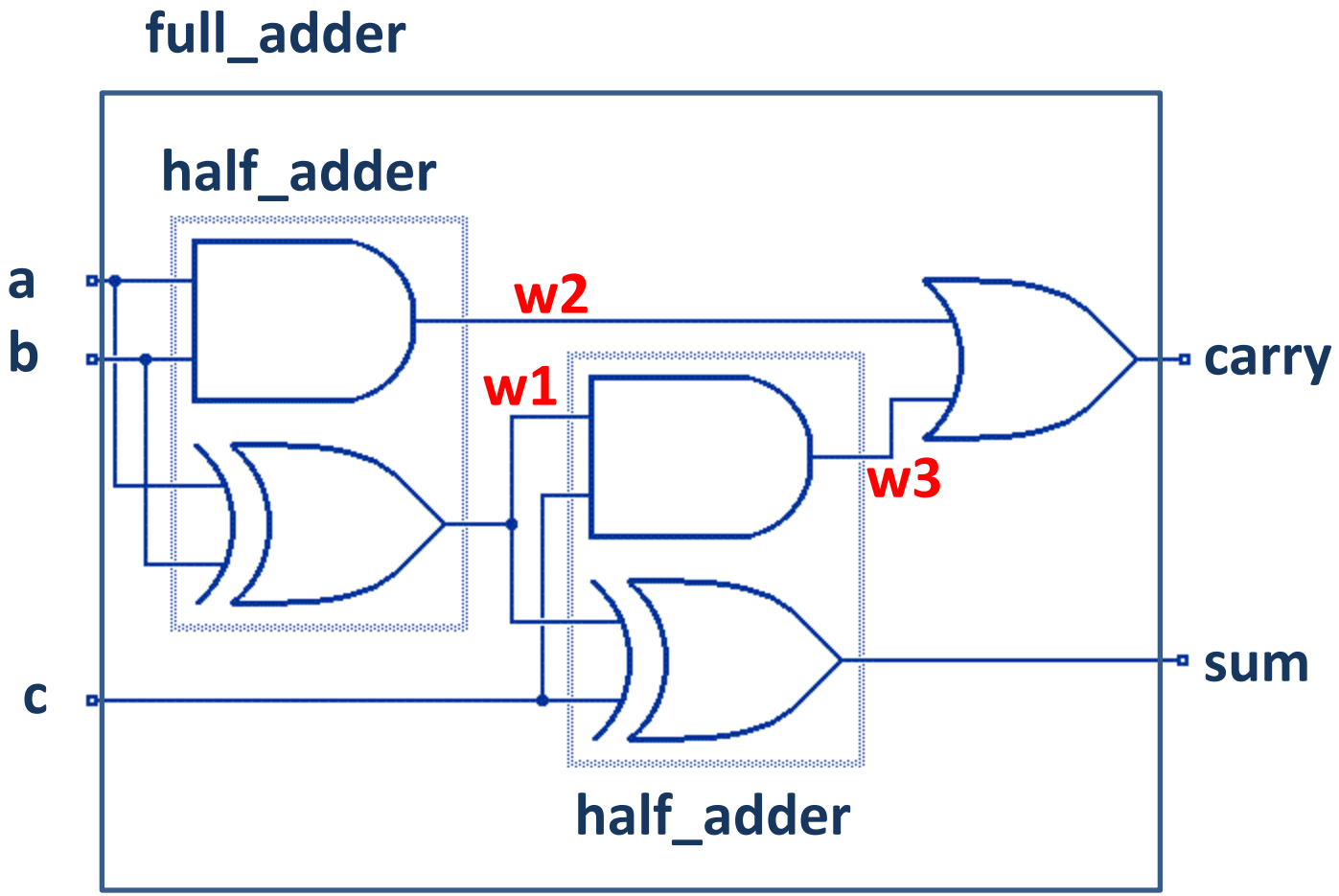
Things are different in Verilog;

Coding Style matters to others!!!

Getting Started With Verilog

- Motivation
- Introduction to Verilog
- First Verilog Program
- Testing Verilog Program
- Styles of Coding
- **Structural Coding**

Full-Adder Circuit



Full-Adder Circuit

```
module full_adder (sum,carry,a,b,c);
    output wire carry,sum;
    input wire a,b,c;
    wire  w1,w2,w3;

   //connect by position
   //port order in ha_struct is (sum,carry,a,b)

   half_adder HA0(w1,w2,a,b);
```

All ports and intermediate connections become wires

one to one mapping

…continued

Full-Adder Circuit

```
//port order in ha_struct is (sum,carry,a,b)
//connect by name
half_adder HA1(.carry(w3),
               .sum(sum),
               .a(w1),
               .b(c));
or  OR0(carry,w2,w3);
endmodule
```

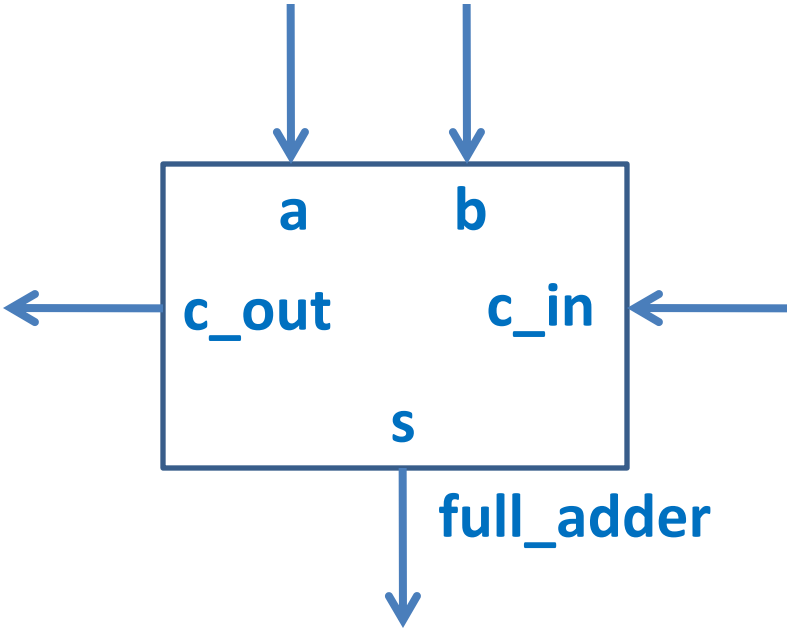
port name

wire name

Order is unimportant

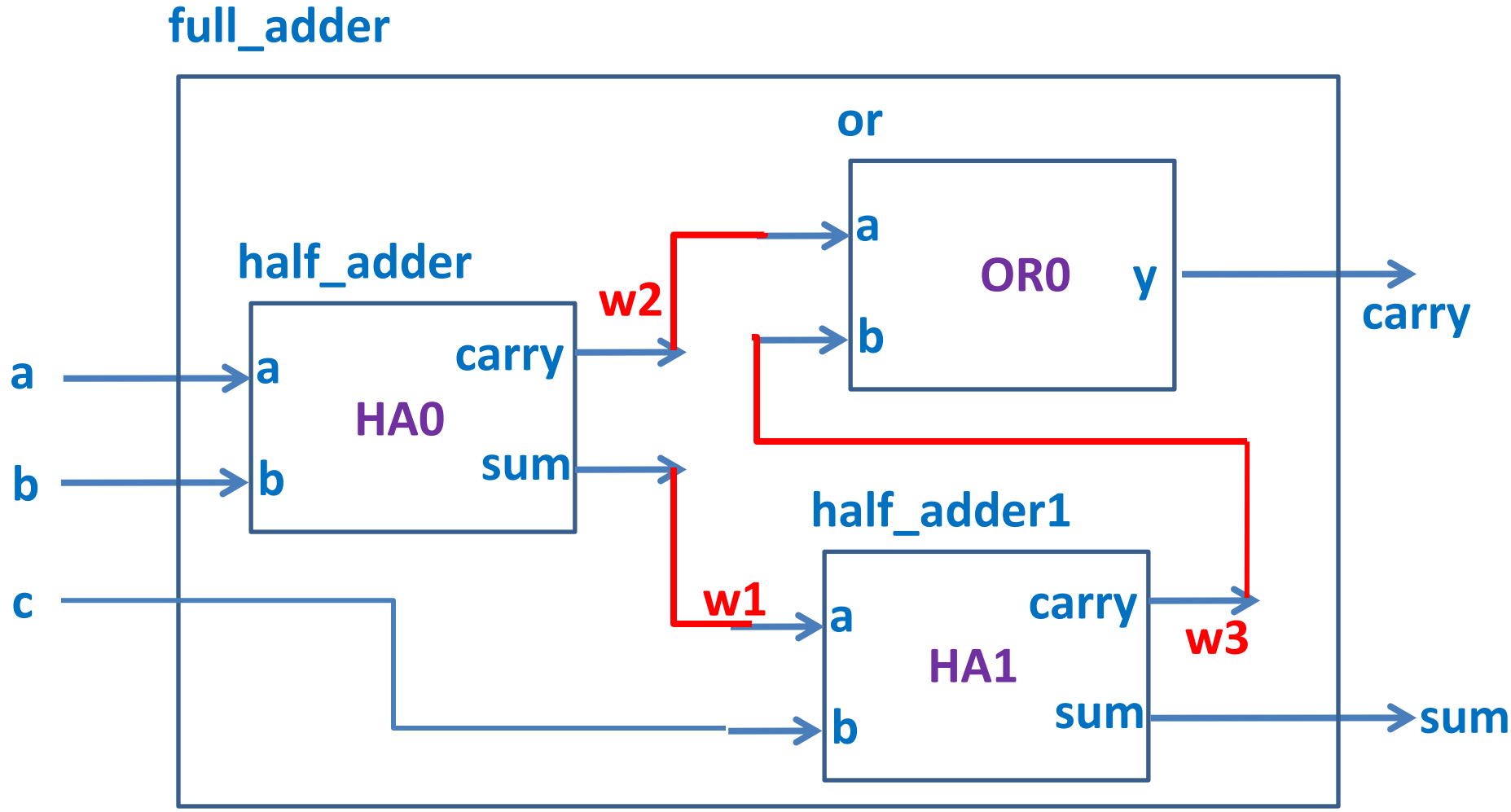
Recommended!!

Full-Adder Block Diagram

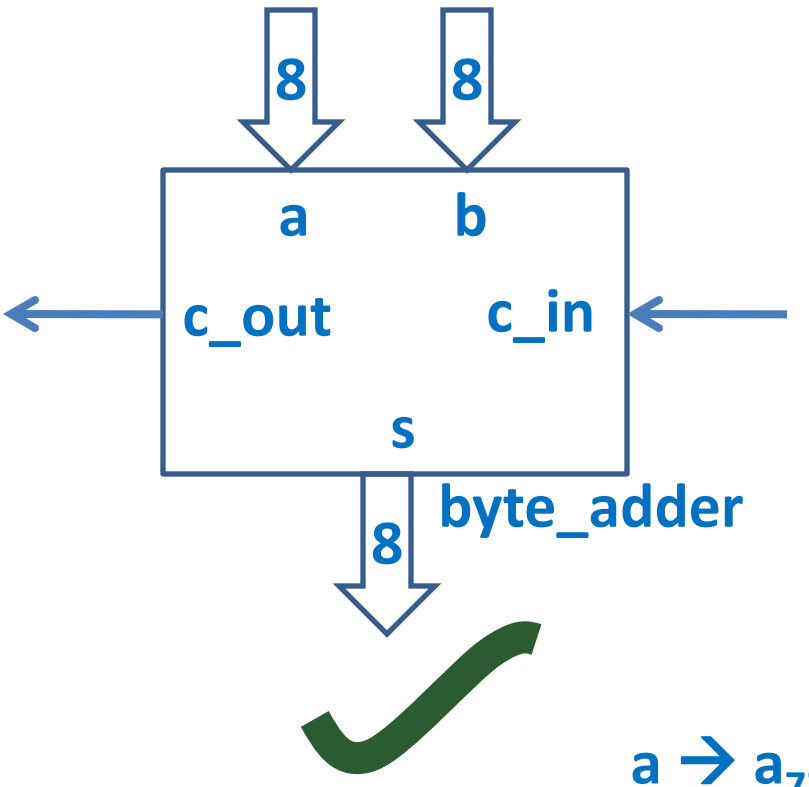


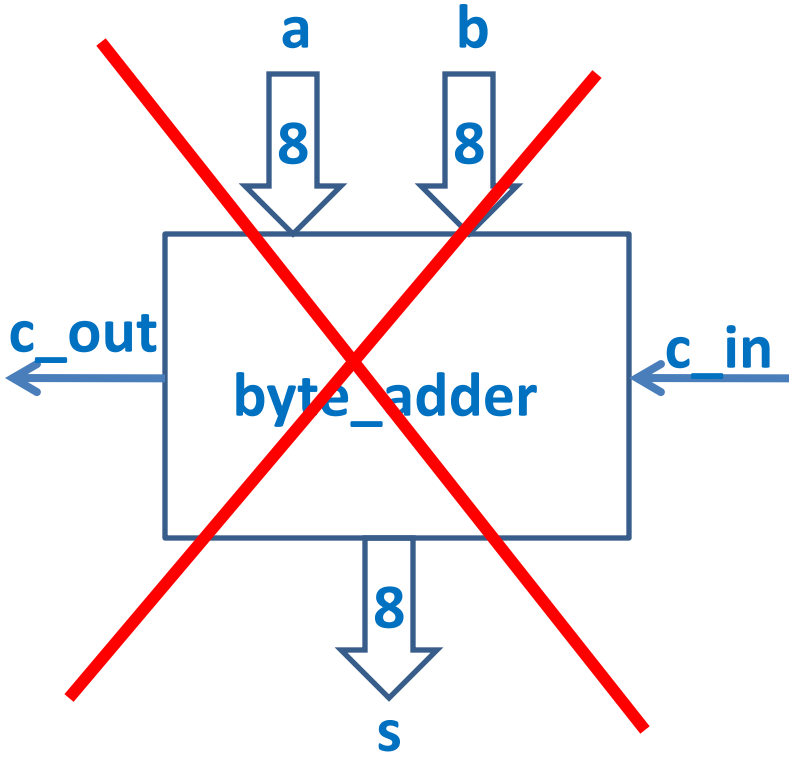
- Port directions are indicated visually
- Ports names are inside the box
- Entity name is outside the box

Full-Adder Hierarchical Diagram



Byte Adder Block Diagram



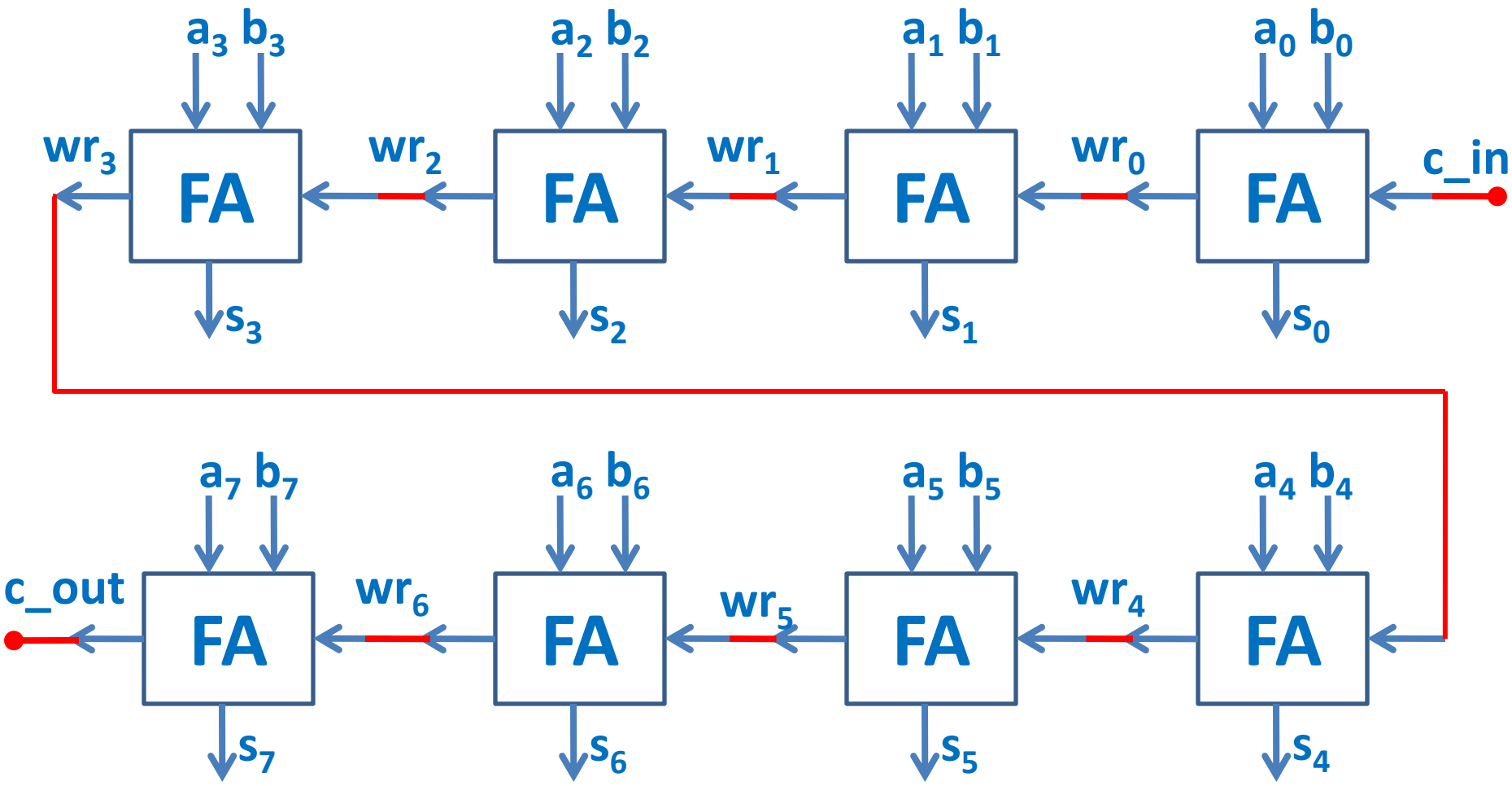


$a \rightarrow a_7a_6a_5a_4a_3a_2a_1a_0$

$b \rightarrow b_7b_6b_5b_4b_3b_2b_1b_0$

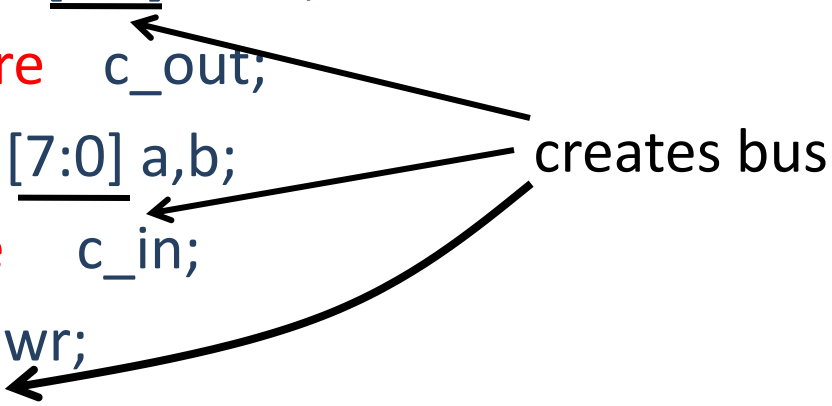
$s \rightarrow s_7s_6s_5s_4s_3s_2s_1s_0$

Byte Adder From Full-Adder



Byte-Adder in Verilog

```
module byte_adder_struct(sum, c_out, a, b, c_in);
    output wire [7:0]  sum;
    output  wire    c_out;
    input wire [7:0] a,b;
    input  wire     c_in;
    wire [7:0]  wr;
```




```
  //full adder instance for 0th bit
  full_adder_beh  FA_BEH_0(…);
  //full adder instance for 1st bit
  full_adder_beh  FA_BEH_1(…);
```

Byte-Adder in Verilog

```
    //full adder instance for 2nd bit
    full_adder_beh  FA_BEH_2(…);
    //full adder instance for 3rd bit
    full_adder_beh  FA_BEH_3(…);
    //full adder instance for 4th bit
    full_adder_beh  FA_BEH_4(…);
    //full adder instance for 5th bit
    full_adder_beh  FA_BEH_5(…);
    //full adder instance for 6th bit
    full_adder_beh  FA_BEH_6(…);
    //full adder instance for 7th bit
    full_adder_beh  FA_BEH_7(…);
endmodule
```

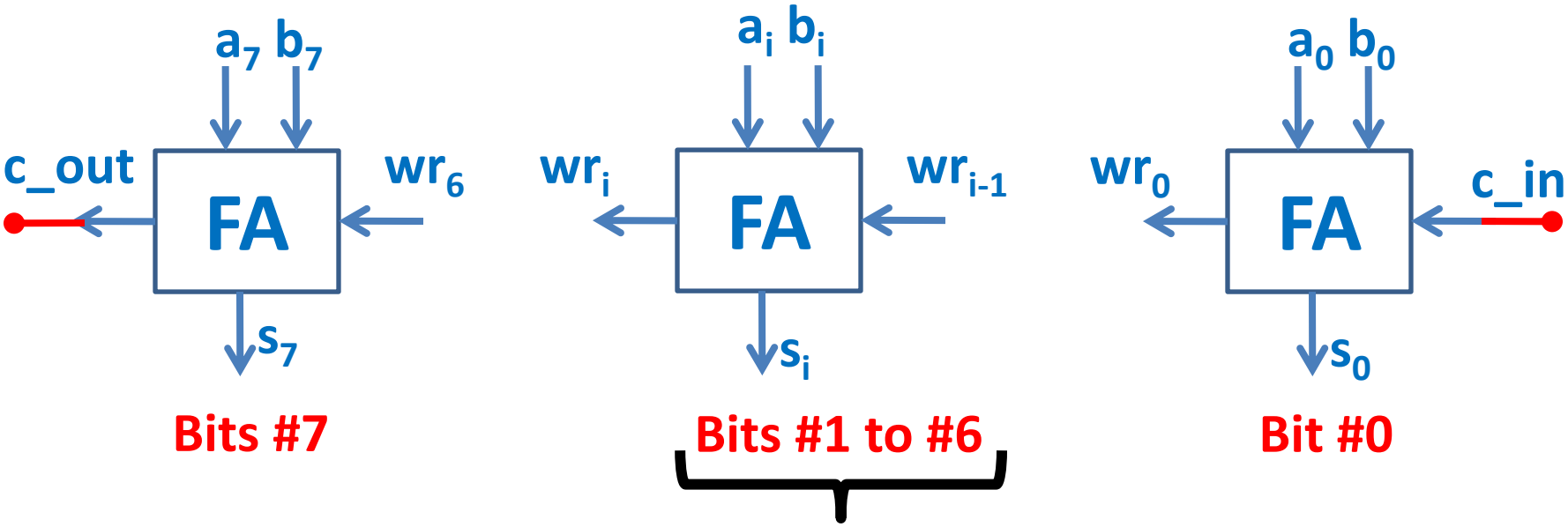
Wait A Minute!!!

Are you saying that I will repeat code this way even for 64-bit adder?

There has to be a better way!!!

Use generate for instantiating repetitive blocks

A Symmetrical View of Byte-Adder



generate is useful here

Generate Block Syntax

```
genvar i;
generate
   for(i = 0; i < size; i = i + 1)
        begin
              //generate block code
        end
endgenerate
```

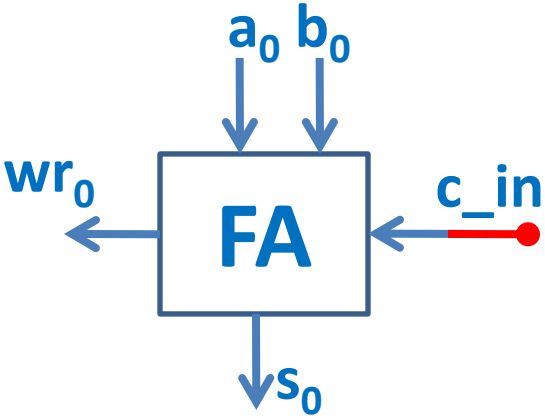
Byte Adder Using Generate


```
module byte_adder_struct(sum, c_out, a, b, c_in);
    output wire [7:0]  sum;
    output  wire    c_out;
    input wire [7:0] a,b;
    input  wire     c_in;
    wire [7:0]  wr;
```

…continued

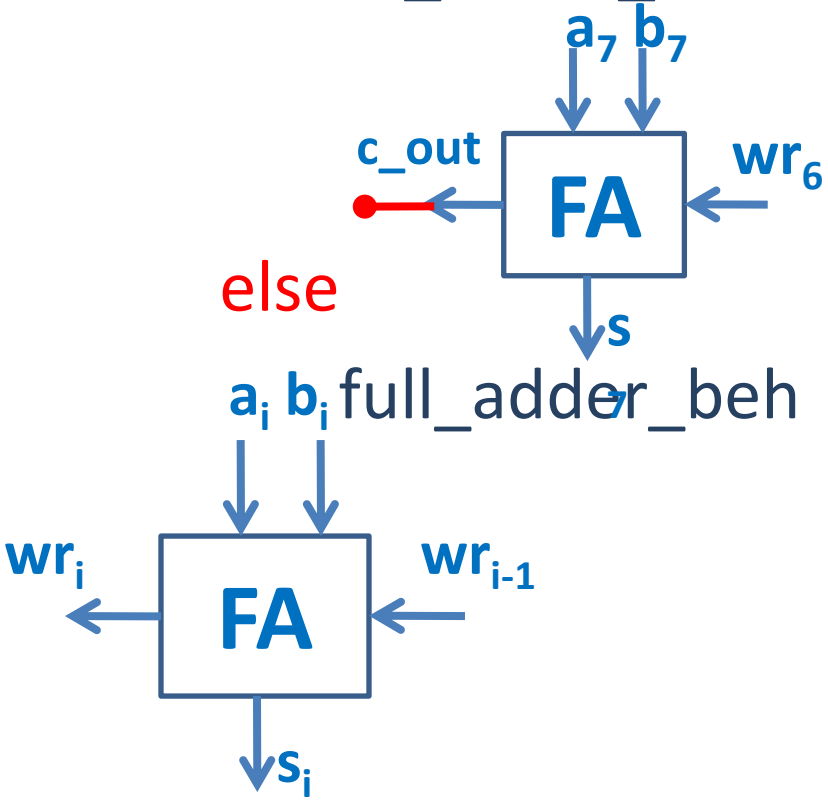
Byte Adder Using Generate

```
genvar i;
 generate
    for( i = 0; i < 8; i = i + 1)
     begin
        if(i == 0)
           full_adder_beh  FA_BEH (.sum(sum[i]),
                                   .carry(wr[i]),
                                .a(a[i]),.b(b[i]),.c_in(c_in));
                                   …continued
```



Byte Adder Using Generate

```
else if (i == 7)
    full_adder_beh  FA_BEH (.sum(sum[i]),
                            .carry(c_out),
                            .a(a[i]),.b(b[i]),.c_in(wr[i-1]));
else
    full_adder_beh  FA_BEH (.sum(sum[i]),
                            .carry(wr[i]),
                            .a(a[i]),.b(b[i]),.c_in(wr[i-1]));
                            ...continued
```



Byte Adder Using Generate

```
    end
  endgenerate
endmodule
```

References

- Ciletti, Michael D., *Advanced Digital Design with Verilog HDL*, Prentice Hall India, 2003.
- Palnitkar, Samir, *Verilog HDL: A Guide to Design and Synthesis*, Sunsoft Press, 1996.
- Brad Hutchings, Verilog – A Hardware Description Language, ECEn 224 Fundamentals of Digital Systems, Fall 2011, Dept. of ECE, Brigham Young University, accessed on 27.06.2012.
- Prof. Stephen A. Edwards, Verilog 1995, 2001 and SystemVerilog 3.1 , Languages for Embedded Systems, Spring 2009, University of Trento, Italy , accessed on 28.06.2012.
- Charles R. Kime and Michael J. Schulte, Verilog HDL Introduction, ECE 554 Digital Engineering Laboratory, Fall 2009, Dept. of ECE, University of Wisconsin, Madison accessed on 27.06.2012.
- J. Wawrzynek, CS61c: Representation of Combinational Logic Circuits, Lecture Notes, CS61c Great Ideas in Computer Science, Dept. of CS, University of California, Berkeley accessed on 27.06.2012.
- Full-Adder schematic diagram http://www.physics.udel.edu/~watson/scen103/colloq2000/fulladder.html accessed on 27.06.2012.

References

- Intel 4004 Microprocessor Schematic http://xenia.media.mit.edu/~mcnerney/2009-4004/i4004-schematic.gif accessed on 26.06.2012.
- ARM Mobile Computing Platform http://www.arm.com/images/processor/Mobile_Computing_Diagram_550.jpg accessed on 26.06.2012.
- AMD Bull Dozer family block diagram http://upload.wikimedia.org/wikipedia/commons/e/ec/AMD_Bulldozer_block_diagram_%288_core_CPU%29.PNG accessed on 26.06.2012
- AMD transistor count http://news.softpedia.com/news/Ex-AMD-Engineer-Blames-Bulldozer-s-Low-Performance-on-Lack-of-Fine-Tuning-227816.shtml accessed on 26.06.2012.
- Xilinx Vertex 6 Development Board http://hitechglobal.com/Boards/Virtex6_PCIExpress_Board.htm accessed on 26.06.2012.