Introduction to Verilog:

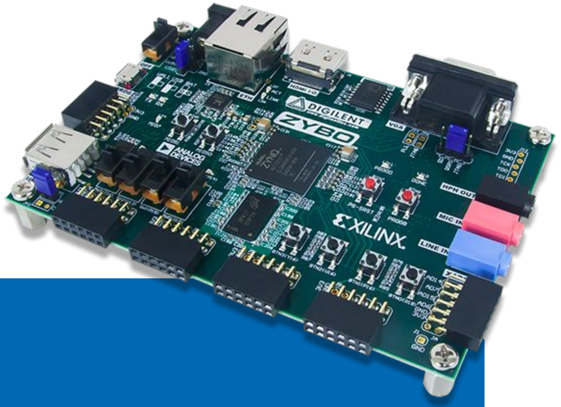
From code to FPGA

Beatriz Garrido

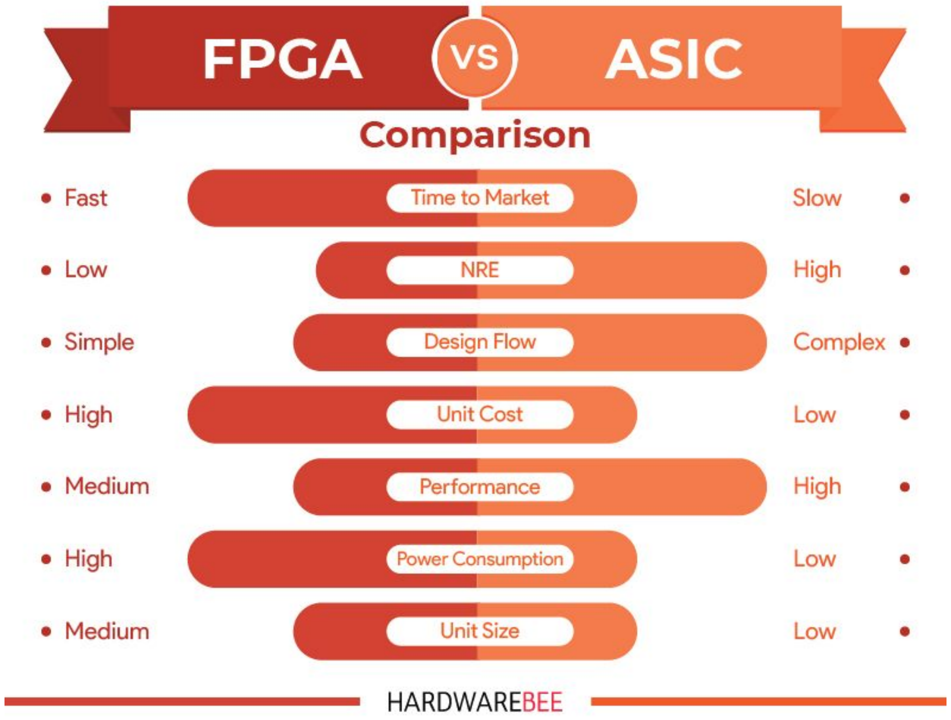
IEEE Robotics & Automation Society
University of Porto

IEEE
Advancing Technology for Humanity

Utility


DIGILENT
ZYBO
XILINX
ANALOG DEVICES
ZYNQ
VGA
HPH OUT
MIC IN
LINE IN

What’s the difference?



FPGA VS ASIC

Comparison

FPGA		ASIC
Fast	Time to Market	Slow
Low	NRE	High
Simple	Design Flow	Complex
High	Unit Cost	Low
Medium	Performance	High
High	Power Consumption	Low
Medium	Unit Size	Low

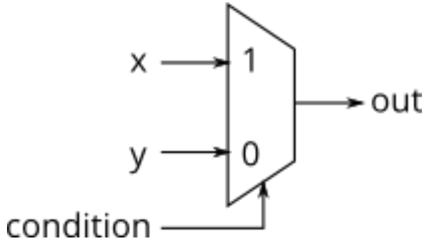
HARDWAREBEE

Hello World

```
module hello_world() ;
        initial begin
        $display ("Hello World!");
        $finish; // stop the simulator
       end
endmodule
```

Verilog Basis

```
if (condition) begin
        out = x;
end
else begin
        out = y;
end
```



or more simply:

```
assign out = (condition) ? x : y:
```

Loops

```
while (this_cond) begin
       $display ("This is going great!");
 end
```

```
for (i = 0; i < 16; i = i +1) begin
       $display ("Current value of i is %d", i);
 end
```

Operators

Operator Type	Operator Symbol	Operation Performed
Arithmetic	*	Multiply
	/	Division
	+	Add
	-	Subtract
	%	Modulus
Logical	!	Logical negation
	&&	Logical and
	||	Logical or

Operator Type	Operator Symbol	Operation Performed
Equality	==	Equality
	!=	inequality
Shift	>>	Right shift
	<<	Left shift
Relational	>	Greater than
	<	Less than
	>=	Greater than or equal
	<=	Less than or equal

Operator Type	Operator Symbol	Operation Performed
Reduction	~	Bitwise negation
	~&	nand
	|	or
	~|	nor
	^	xor
	^~ OR ~^	xnor
Concatenation	{}	Concatenation

Switch Cases

```
case (two_bit_variable)
        2’b00: $display ("print1"); //if the variable has value 00 do this thing
        2’b01: $display ("print2”); //if the variable has value 01 do this other thing
        default: $display ("print3"); //don’t forget this one
endcase
```

Bit Values

4’b10_00

underscores are ignored (only for easier reading)

base format (b, d, o, h)

decimal number representing size in bits

Binary

→ 8’b0000_0000

Hexadecimal

→ 32’h0a34_40f1

→ 16’ha630

Decimal

→ 32’d42

Simple combinational example

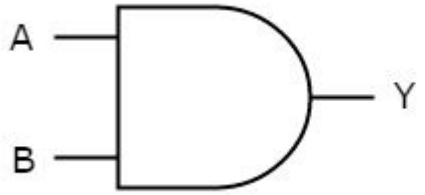
```
module and_gate(input a, input b, output y ) ;
```

$\rightarrow$ ( ) all of the module's inputs and outputs

```
        input a, b;

        output y;
```

$\}$ declare every input and output

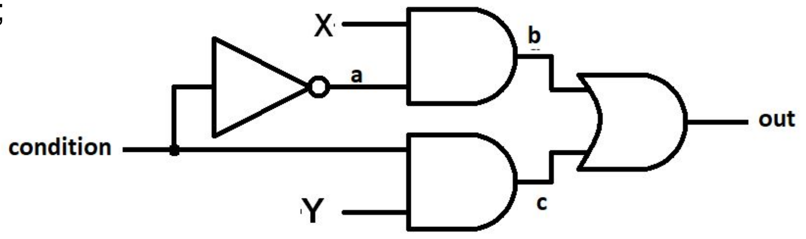
```
        wire a, b, y;

        assign y = a & b;


endmodule
```



Modules inside Modules

```
module and_gate(input in1, input in2, output out);
```

```
module mux( input x, y, condition, output out);

          wire a, b, c;

          not_gate dut(.in(condition),         .out(a))

          and_gate dut( .in1(x),               .in2(a),         .out(b) ) ;

          and_gate dut( .in1(condition),       .in2(y),         .out(c) ) ;

          or_gate dut( .in1(b),                .in2(c),         .out(out) );

endmodule
```

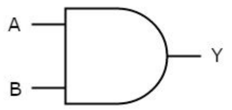


Wires and Registers

```
input a, b;

output c;

wire a, b, c;

assign a = b ^ c;
// less common, but reg can also appear on assigns
```

```
module circuit (
        input clk, a, reset,
        output y);

        reg  y;

        wire a;


        always  @ (posedge clk ) begin

                if (reset == 0) begin

                        y <= 0;

                end else begin

                        y <= a;

                end

        end

endmodule
```

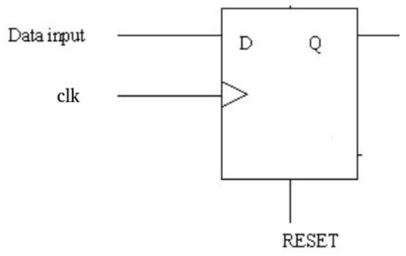
Always @ block



```
module and_comb_example( a, b, y );

        input a, b;

        output y;

        wire y, a, b, sel;

        always @ ( a or b ) begin

                y = a & b;
        end

        always @ ( * ) begin

                sel = 0;
        end

endmodule
```

This process is always "executed"

```
module reg_seq_example( clk, reset, d, q);
        input clk, reset, d;
        output q;
        reg   q;
        wire clk, reset, d;
        always @ (posedge clk or posedge reset) begin
                if (reset) begin
                                q <= 1'b0;
                end
                else begin
                                q <= d;
                end
        end
endmodule
```



Sequential vs. Combinational

```
always  @ (posedge clk )
          if (reset == 0) begin
                    y <= 0;
          end else if (sel == 0) begin
                    y <= a;
          end else begin
                    b <= a;
                    c <= b;
                    y <= c;
          end
end
```

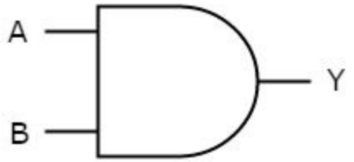
all in parallel

```
always  @ (a or b or sel) begin
          if (sel == 0) begin
                    d = a;
          end else begin
                    b = a;
                    c = b;
                    d = c;
          end
end
```

order matters

Test Benches

```
module simple_example(
   input a,
   input b,
   output y
 );

 assign y = a & b;

endmodule
```



```
module and_tb();

        reg a;
        reg b;
        wire y;

        simple_example  DUT(a,b,y);
        initial begin
                $monitor("a=%b, b=%b, y=%b", a, b, y);
                a=0; b=0; #10;
                a=0; b=1; #10;
                a=1; b=0; #10;
                a=1; b=1; #10;

                $finish;
        end
endmodule
```

module instantiation

delay 10 units of time

Self-checking Testbench

```
module testbench();
   reg a, b;
   wire y;

   sillyfunction DUT(.a(a), .b(b), .y(y));

   initial begin
             a=0; b=0;  #10; //apply input, wait
             if (y!==0) $display("00 failed"); //check

             a=1; #10;
             if (y!==0) $display("001 failed");

             b=1; a=0; #10;
             if (y!==0) $display("010 failed");

             //etc.. etc..

   end
endmodule
```

Let’s practice!

Please install Icarus Verilog

Linux:

> *sudo apt install iverilog*

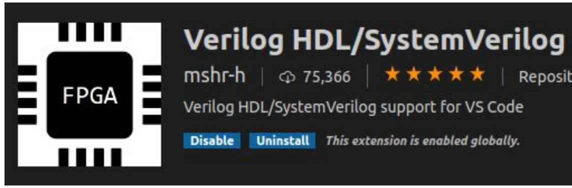
Windows:

http://bleyer.org/icarus/

MasOS:

http://macappstore.org/icarus-verilog/

You can install Verilog HDL extension on Visual Studio Code:


Verilog HDL/SystemVerilog
mshr-h | 75,366 | ★★★★★ | Reposit
Verilog HDL/SystemVerilog support for VS Code
Disable Uninstall This extension is enabled globally.

<TÍTULO>

<SUBTÍTULO>

se windows nao der:

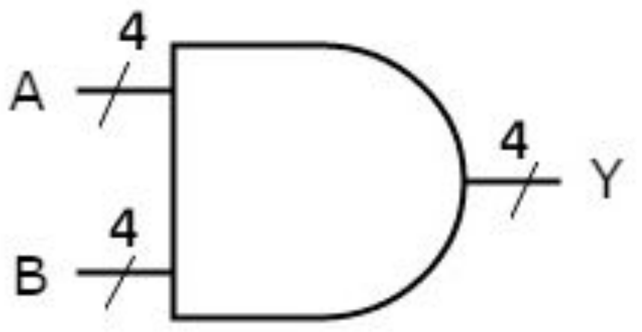
https://www.howtogeek.com/249966/how-to-install-and-use-the-linux-bash-shell-on-windows-10/

```
>sudo apt update
```

```
>sudo apt upgrade
```

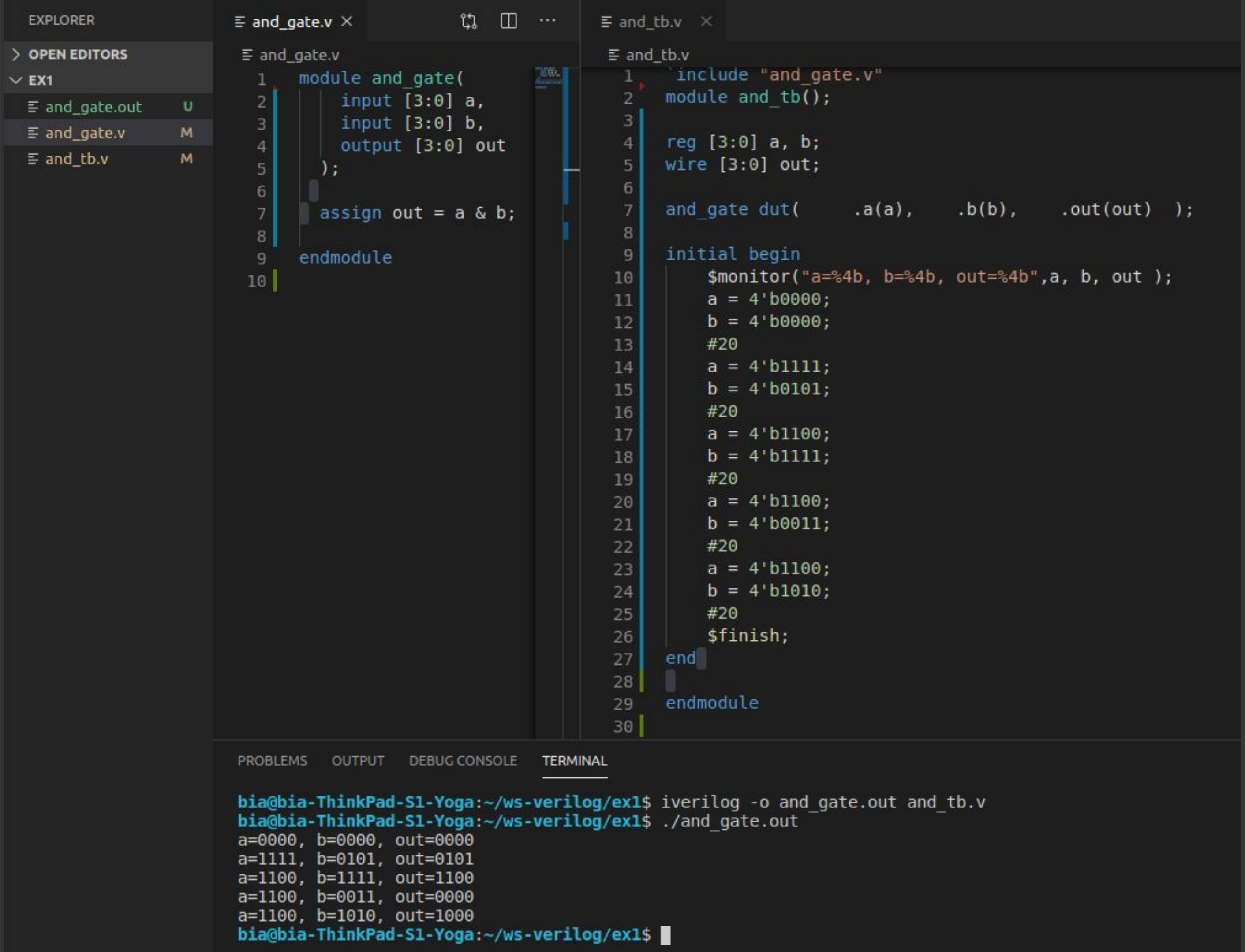
Let’s Practice

Try and write the Verilog code and TestBench for an and gate with 4 bit inputs:



To run your code, write these commands on the terminal:

```
> iverilog tb_name.v

> ./a.out
```



```verilog
module and_gate(
    input [3:0] a,
    input [3:0] b,
    output [3:0] out
  );

  assign out = a & b;

endmodule
```

```verilog
`include "and_gate.v"
module and_tb();

reg [3:0] a, b;
wire [3:0] out;

and_gate dut(     .a(a),    .b(b),    .out(out)  );

initial begin
    $monitor("a=%4b, b=%4b, out=%4b",a, b, out );
    a = 4'b0000;
    b = 4'b0000;
    #20
    a = 4'b1111;
    b = 4'b0101;
    #20
    a = 4'b1100;
    b = 4'b1111;
    #20
    a = 4'b1100;
    b = 4'b0011;
    #20
    a = 4'b1100;
    b = 4'b1010;
    #20
    $finish;
end

endmodule
```

```
bia@bia-ThinkPad-S1-Yoga:~/ws-verilog/ex1$ iverilog -o and_gate.out and_tb.v
bia@bia-ThinkPad-S1-Yoga:~/ws-verilog/ex1$ ./and_gate.out
a=0000, b=0000, out=0000
a=1111, b=0101, out=0101
a=1100, b=1111, out=1100
a=1100, b=0011, out=0000
a=1100, b=1010, out=1000
bia@bia-ThinkPad-S1-Yoga:~/ws-verilog/ex1$
```

Try and write the Testbench for this code

```
module fulladder(
        input x,
        input y,
        input cin,

        output A,
        output cout
);
        assign {cout,A} =  cin + y + x;

endmodule
```

Carry in	Input y	Input x	Carry out	Output A
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

desired output

Solution

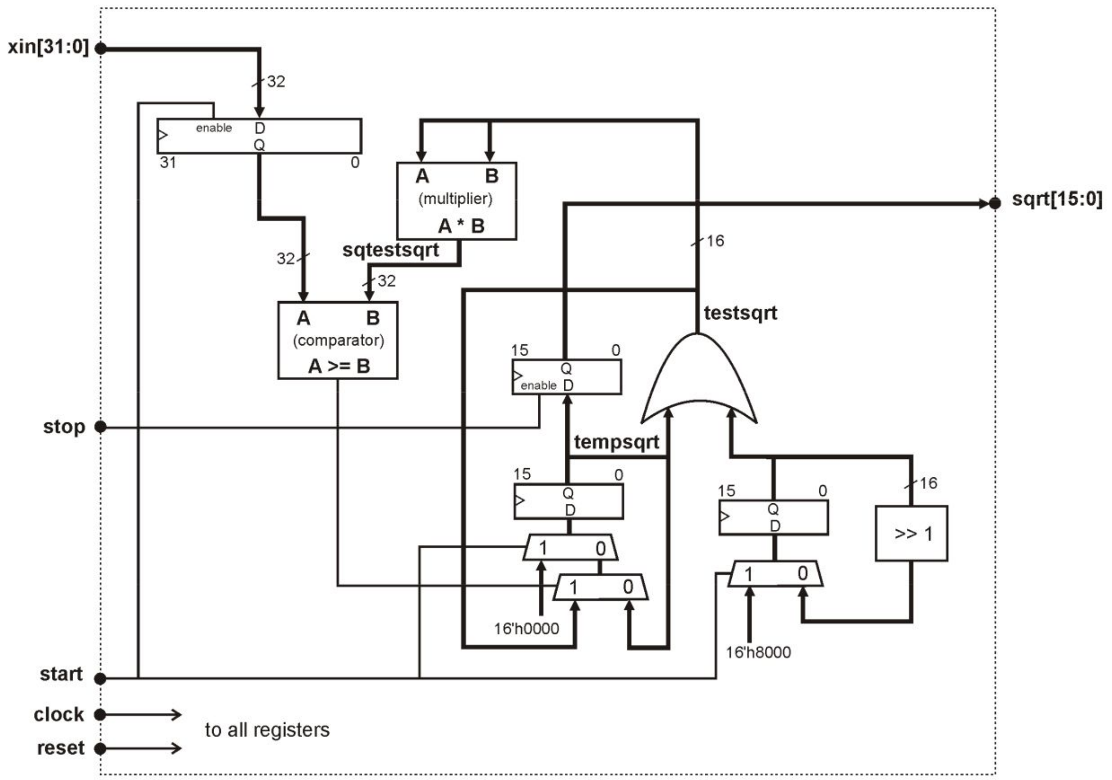
https://github.com/ieeeupsb/workshop-verilog/blob/master/ex3/fulladder_tb.v

Sequential circuit exercises

Write a sequential circuit that passes this TestBench:

https://github.com/ieeeupsb/workshop-verilog/blob/master/ex4/counter_tb.v

For the Pros



xin[31:0]
32
enable
D
Q
31
0
A
B
(multiplier)
A * B
sqrt[15:0]
16
sqtestsqrt
32
32
A
B
(comparator)
A >= B
testsqrt
15
0
Q
enable D
stop
tempsqrt
15
0
Q
D
1
0
1
0
16'h0000
15
0
Q
D
16
>> 1
1
0
16'h8000
start
clock
reset
to all registers

For the Pros

https://github.com/ieeeupsb/workshop-verilog/tree/master/ex5

Want to learn more?

http://www.asic-world.com/verilog

IEEE Robotics & Automation Society

IEEE

Thank you!

nuieee@fe.up.pt
nuieee.fe.up.pt
facebook.com/ieeeupsb