Verilog For Combinational Logic

Instructor: Dr. Vinicius Prado da Fonseca (vpradodafons@online.mun.ca)

MEMORIAL UNIVERSITY
www.mun.ca

Verilog

- Hardware description language used for simulation and synthesis
- Allows a digital logic system to be analyzed at the behavioural, functional, and structural levels
- Simulation
  - Verify that the digital system operates as expected in outputs and timing
- Synthesis translate the HDL model into
  - ASICs (Application Specific Integrated Circuits)
  - FPGAs (Field Programmable Gate Arrays).
- Some features of verilog are only used for simulation and play no part in synthesis

MEMORIAL UNIVERSITY
www.mun.ca

Using the VSCode devcontainer

- The software tools mentioned here are not mandatory.
- The lab is still required to submit your answers.
- You may use the tools below to test your code outside the lab app and to study the examples presented in class.

Repo: https://github.com/vncprado/COMP2003-devcontainer

Instructions: D2L > Contents > Labs > Lab support files > Software Tools

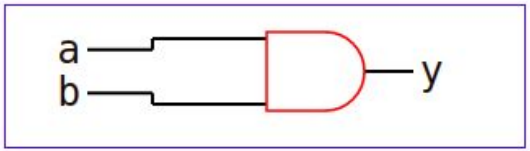
MEMORIAL UNIVERSITY
www.mun.ca

Verilog modules

- A design is modelled as a set of modules
- Each module encapsulates a set of definitions
- Can be used elsewhere
  - Blackbox idea
  - Set of inputs, outputs and functional/timing definitions

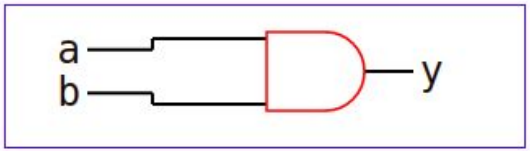
MEMORIAL UNIVERSITY
www.mun.ca

AND gate module (and_gate.v)



MEMORIAL UNIVERSITY
www.mun.ca

AND gate module (and_gate.v)




```
/* declare a module called and_gate */
module and_gate(output logic y, input logic a, b); // three connection
    // a and b are inputs, only connected to inputs of gates
    // x is an output, must be connected to an output in this module

    // and is the function
    // #2 is the delay from input to output
    // g1 is the name of the gate
    // both delay and gate's name are optional
    and #2 g1(y, a, b);
    // y is the output, a, b are inputs
endmodule
```


MEMORIAL UNIVERSITY
www.mun.ca

Verilog Test Bench

- Most modules deal with the functionality of the system
- Some of the modules deal with testing the design

$ iverilog -g2012 and_gate_tb.v and_gate.v -o and_gate

$ vvp and_gate

- The initial block is started at the beginning of the simulation
- The print statement in Verilog is $display
- and_gate_tb.v

MEMORIAL UNIVERSITY
www.mun.ca

Verilog Test Bench

- if and if-else
- case, casex -- case is similar to switch in Java and C
- loops
  - while
  - do-while
  - for (init; test; incr)
  - repeat(count)
- Examples in the notes

MEMORIAL UNIVERSITY
www.mun.ca

Testing all inputs (loop example)

- 2 input requires 2^2 input combinations.
- For n inputs, 2^n combinations are required.
- Using an integer to test all possible inputs:

```
integer all_inputs;

assign {a,b} = all_inputs[1:0];
```

```
all_inputs=00000

t=010 a=0 b=0 y=0

all_inputs=00001

t=020 a=0 b=1 y=1

all_inputs=00010

t=030 a=1 b=0 y=1

all_inputs=00011

t=040 a=1 b=1 y=1
```


MEMORIAL UNIVERSITY
www.mun.ca

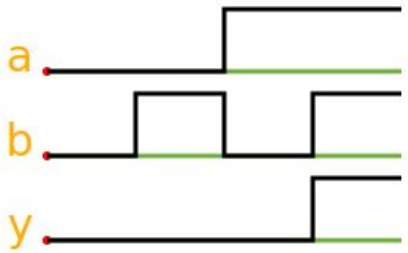
Testing all inputs (and_gate.vcd)

- Using an integer to test all possible inputs:

```
integer all_inputs;
assign {a,b} = all_inputs[1:0];
```

- View also the the waveform:
  - $dumpfile("and_gate.vcd");
  - $dumpvars(0, testing);
  - or $dumpvars();

```
all_inputs=00000
t=010 a=0 b=0 y=0
all_inputs=00001
t=020 a=0 b=1 y=1
all_inputs=00010
t=030 a=1 b=0 y=1
all_inputs=00011
t=040 a=1 b=1 y=1
```



MEMORIAL UNIVERSITY
www.mun.ca

Examining Delays

- Include a #2 delay in the and_gate.v module
- Analise delays with dump.vcd in gtkwave or WaveTrace plugin
- Shown in the waveform

```
t=0000 a=0 b=0 y=x
t=0001 a=0 b=0 y=x
t=0002 a=0 b=0 y=0
```

- Only after the delay (2ns) has a valid value
- Also in tb_and_mod_v2.v (notes)

MEMORIAL UNIVERSITY
www.mun.ca

Delay Paths

- Resulting output change depends on the path it propagate along
- A real issue in real-life circuits
- Waveform analysis

```
wire n1, n2;

and #2  g1(n1, a, b);

or #3   g2(n2, n1, c);

and #2  g3(y, n2, d);
```

- gate_timing.v -> gate_timing.vcd -> (gtkwave or WaveTrace, include n1 and n2 wires)

MEMORIAL UNIVERSITY
www.mun.ca

Timing

- If you have more than one initial begin block
  - All initial begin blocks execute in parallel
- “gate_timing.v” in the notes
  - Two initial blocks: (A,B,C) and (X,Y,Z) :

```
At 0005, A
At 0010, X
At 0015, B
At 0020, Y
    ...
```


MEMORIAL UNIVERSITY
www.mun.ca

Functional AND module (and_gate.v)

- First AND example was structural, gate-level
- Boolean expressions can also be used in verilog
- The and_gate module can be defined using a Boolean expression with
- Use the “assign” instruction

MEMORIAL UNIVERSITY
www.mun.ca

Functional AND module (and_gate.v)

- First AND example was structural, gate-level
- Boolean expressions can also be used in verilog
- The and_gate module can be defined using a Boolean expression with

```
module and_gate(output logic y, input logic a, b); // three connection
    // continuous assignment, boolean expression
    assign #2 y = a & b;
    // the output is still delayed by 2ns
endmodule
```


MEMORIAL UNIVERSITY
www.mun.ca

Behavioural AND module (and_gate.v)

- Design using a truth table
- assign #2 y = t; // output connected to a temp
- Use the always_comb block
- “Infinite loop” idea
- Runs whenever a, or b changes
- 2'b00 the 2 specifies the number of bits
- 'b specifies the radix is binary

MEMORIAL UNIVERSITY
www.mun.ca

Behavioural AND module (and_gate.v)

- Design using a truth table
- assign #2 y = t; // output connected to a temp
- Use the always_comb block
- “Infinite loop” idea
- Runs whenever a, or b changes
- 2'b00 the 2 specifies the number of bits
- 'b specifies the radix is binary

```
// model the delay
assign #2 y = t;

// a behavioural description
// always is like an infinite loop, it always runs
// when ever a, or b changes.
always_comb begin
    case( {a,b} ) // {a,b} concatenates a and b
        2'b00: t = 0; // if a==0 and b==0
        2'b01: t = 0; // if a==0 and b==1
        2'b10: t = 0; // if a==1 and b==0
        2'b11: t = 1; // if a==1 and b==1
    endcase
    // 2'b00 the 2 specifies the number of bits
    // 'b specifies the radix is binary
end
```


MEMORIAL UNIVERSITY
www.mun.ca

Functional, Structural, Behavioural

- Functional
  - Boolean expressions

```
assign y = a&b;
```

- Structural
  - gates/modules

```
and (y, a, b);
```

- Behavioural
  - boolean expression or always_comb block

```
always_comb begin
    case ({a, b})
        2'b00 : y = 0;
        2'b01 : y = 0;
        2'b10 : y = 0;
        2'b11 : y = 1;
    endcase
end
```


MEMORIAL UNIVERSITY
www.mun.ca

Case vs casez

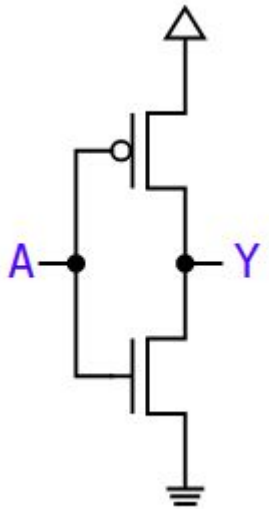
Several examples in the notes:
11a - Design Using Structural and Functional Verilog

MEMORIAL UNIVERSITY
www.mun.ca

CMOS Inverter and Test Bench

[Skipped for new version of the lab]

Verilog can also model and simulate NMOS and PMOS transistors (not_gate_CMOS.v)



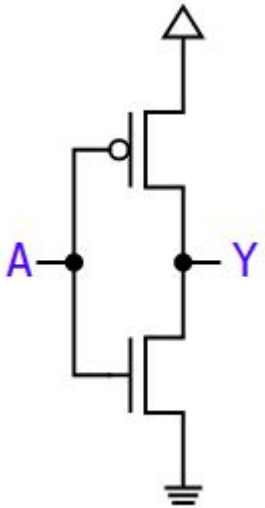
MEMORIAL UNIVERSITY
www.mun.ca

CMOS Inverter and Test Bench

```
module main; // main has no outside connections
    logic a, y; // used to connect components

    // instantiate and connect module to be tested
    // this statement is declarative
    not_gate testing( .y(y), .a(a) );
    // the order of named connections is not important
    // not_gate testing( .a(a), .y(y) );


    initial begin
        a = 0;
        #10; // delay 10 units (10ns in this case)
        $display("t=%04d a=%b y=%b", $time, a,y);
        a = 1;
        #10; // delay 10ns
        $display("t=%04d a=%b y=%b", $time, a,y);
        $finish; // terminate the simulation
    end
endmodule
```



MEMORIAL UNIVERSITY
www.mun.ca

Functional POS and SOP XOR Example

a	b	c	XOR	Minterms Maxterm
0	0	0	0	a|b|c
0	0	1	1	~a&~b&c
0	1	0	1	~a&b&~c
0	1	1	0	a|~b|~c
1	0	0	1	a&~b&~c
1	0	1	0	~a|b|~c
1	1	0	0	~a|~b|c
1	1	1	1	a&b&c

```
assign y = ~a & ~b & c | ~a & b & ~c | a
& ~b & ~c | a & b & c;

assign z = (~a | ~b | c) & (~a | b | ~c)
& (a | ~b | ~c) & (a | b | c);
```

- sop_pos.v
- sop_pos.vcd

MEMORIAL UNIVERSITY
www.mun.ca

Bitwise Inverter (bitwise_not.v, bitwise_not.vcd)

- Nets in Verilog can be grouped into buses (vectors)

```
logic [3:0] y

logic [3:0] a
```

- Four nets labelled y[3], y[2], y[1], and y[0]
- Similar to bitwise operators in C, Java, Python
- The Boolean operators &, |, ^, and ~ operator on the group of nets

```
assign y = ~a;
```

- bitwise_and.v, bitwise_and.vcd

MEMORIAL UNIVERSITY
www.mun.ca

Online Notes

- D2L and repo has all examples presented here
- Lots more examples
- Fibonacci implementations

MEMORIAL UNIVERSITY
www.mun.ca

Multiplexer Implementation (Try yourself)

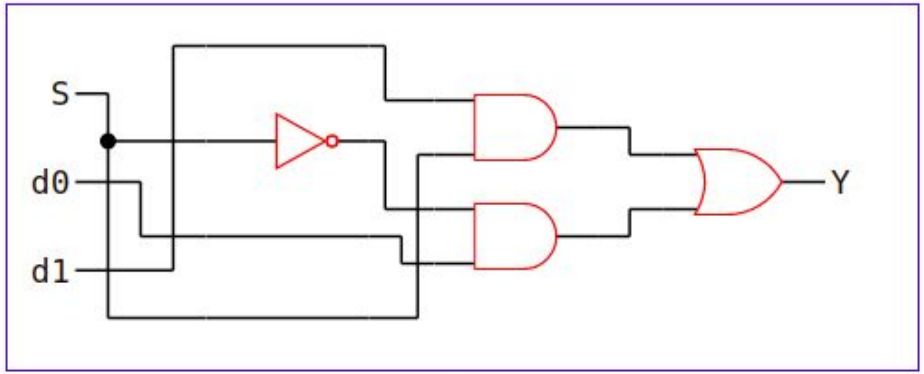
- 2-to-1 MUX verilog implementation:

```
module mux2_1_assign(output logic Y, input logic a, b, s);

     assign Y = ~s & a | s & b;

endmodule
```

- muxes2_1.v, tb_muxes2_1.v



2-to-4 Decoder (dec2_4.v) (Try yourself)

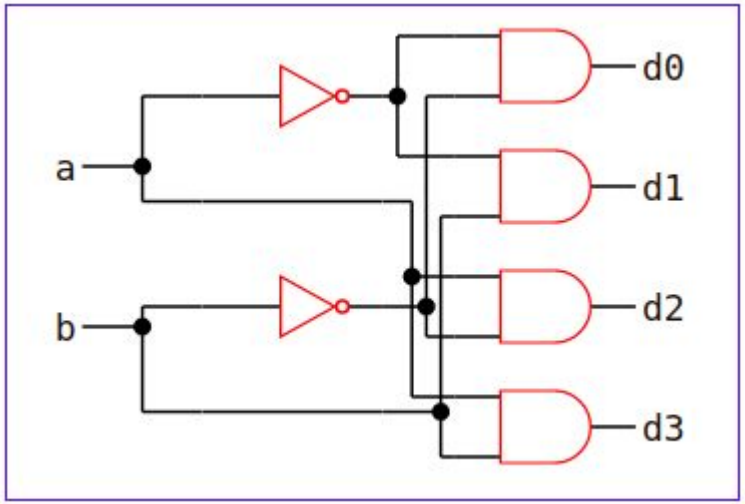
A 2-to-4 decoder provides all of the minterms gate-level implementation:

```
and d0_gate(d0, a_not, b_not);

and d1_gate(d1, a_not, b);

and d2_gate(d2, a, b_not);

and d3_gate(d3, a, b);
```



MEMORIAL UNIVERSITY
www.mun.ca

Questions?

Next: Digital Design with Truth tables and K-Maps

MEMORIAL UNIVERSITY
www.mun.ca

Installing Icarus Verilog

- The version provided by most Ubuntu's is not the most recent. Labnet at MUN provides the most recent version.
- Verilog is set up in the labnet environment with

  $ PATH="/usr/local/bin/:$PATH"

- Permanent by adding the above line to either ~/.bashrc or ~/.profile
- Local Ubuntu machine
  - Check if the version is v11
  - Install from source:
    - https://steveicarus.github.io/iverilog/usage/installation.html
- Gtkwave (or vscode plugin next slide) should also be installed. For Ubuntu:

  $ sudo apt install gtkwave (visualizer)

MEMORIAL UNIVERSITY
www.mun.ca

Installing Icarus Verilog

- My lectures setup
- Windows 11 machine + Vscode
  - WSL2 Ubuntu 20.04
  - iverilog installed from source
  - WaveTrace plugin

MEMORIAL UNIVERSITY
www.mun.ca