EEE3096S: Embedded Systems II

LECTURE 24:

VERILOG SYNTAX

Presented by:

STANLEY MBEWE

Electrical Engineering
University of Cape Town

OUTLINE OF LECTURE

- Learning Verilog HDL
- Building blocks
- Non-synthesizable data types

Starting on Verilog

Hopefully you already have an idea from last lecture what you're letting in to your headspace ☺

LEARNING VERILOG

Best way to learn HDL…

is through practice coding with it!

See the Verilog Cheat Sheet included in resources

Online learning and reference sources for Verilog programming

Textbook Appendix A.4 is recommended reading

Since we are not going in-depth into Verilog an appropriate reference is the one provided by **Wikibooks** (these links are concise resources which would be adequate for introduction level what we are doing) :

https://en.wikibooks.org/wiki/Programmable_Logic

https://en.wikibooks.org/wiki/Programmable_Logic/Verilog

ASIC World is an excellent web resource, the goto place for useful support and examples (at a introductory to intermediate level) :

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

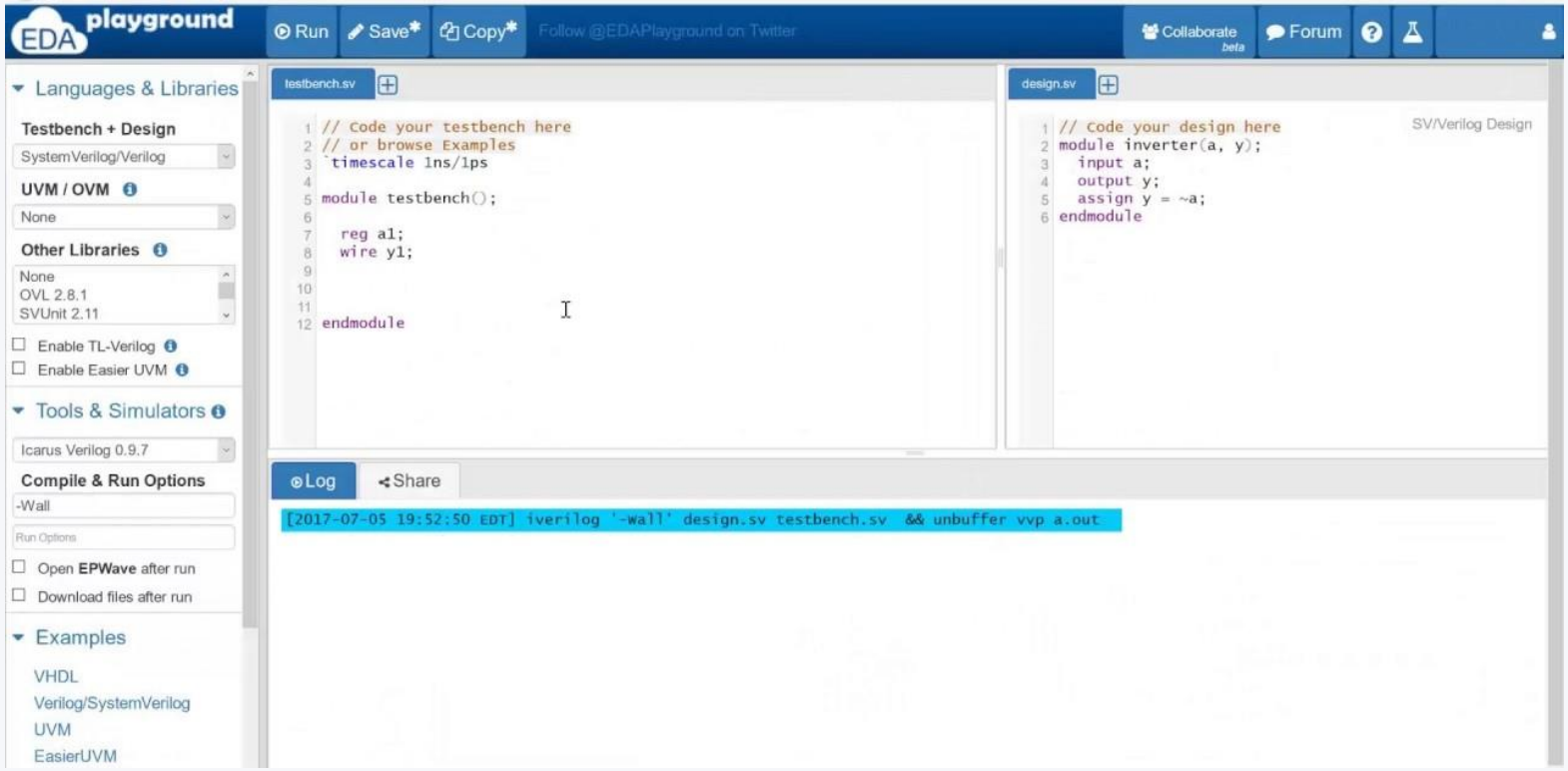
http://www.asic-world.com/examples/verilog/

The textbook, Patterson & Hennessy (2017), provides useful recap of digital logic in Appendix A, and further introduction and explanations of HDL (and Verilog) in Appendix A.4 (recommended reading).

EDA Playground

one of Simon's favourite!

An easy to access tool that you can get started with! Why not try it out.



https://www.edaplayground.com/

NOTE: How to set it up…

- Log in via Gmail or create a **register a new login** so that you can save designs
- Choose the simulator: from Tools & Simulators select **Icarus Verilog 0.10.0**

iVerilog : Icarus Verilog

If you would prefer using Verilog offline, or run simulations on your own computer, or develop more substantial designs, then you may want to use Icarus Verilog. It's free!

iVerilog is a compiler:
iVerilog will parse the Verilog code and generate an executable that the PC can run (called a.out if you don't use the flags to change the output executable file name)

```
swinberg@forge:~/TestVeri$ iverilog mynand.v
swinberg@forge:~/TestVeri$ ./a.out
A = 0, B = 0, Nand output w = 1

A = 0, B = 1, Nand output w = 1

A = 1, B = 0, Nand output w = 1

A = 1, B = 1, Nand output w = 0

swinberg@forge:~/TestVeri$
```

http://iverilog.icarus.com/

Available on Vula in Resources/Software/iverilog-10.1.1-x64_setup

For Ubuntu or Debian you can install it (if linked to the leg server) using:

```
apt-get install iverilog
```

one of Simon's favourites!

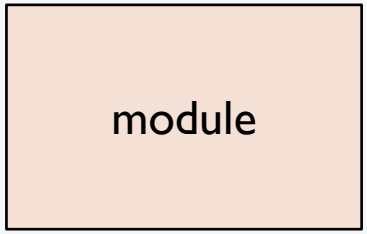
Online IDEs: edaplayground or jdoodle.com/execute-verilog-online/

let’s get going

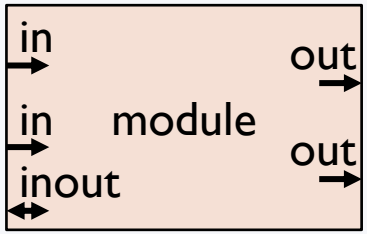
The Verilog coding paradigm…

the four essential ingredients

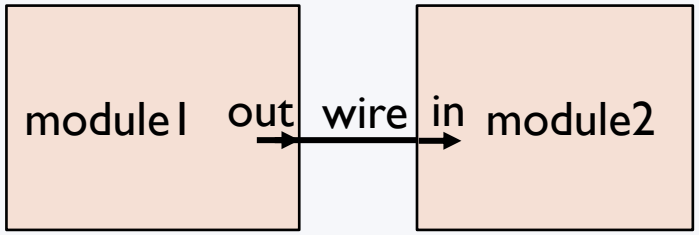
MODULES



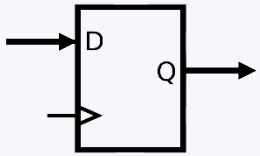
PORTS



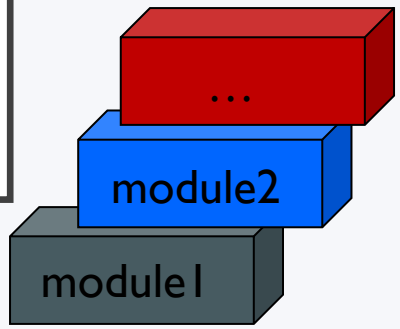
WIRES



REGISTERS



MODULE: BUILDING BLOCK OF VERILOG PROGRAMS



Module: the basic block that does something and can be connected to (i.e. equivalent to entity in VHDL)

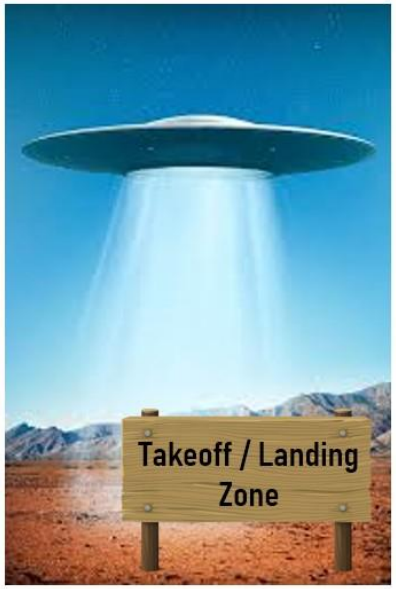
Modules are hierarchical. They can be individual elements (e.g. comprise standard gates) or can be a composition of other modules.

SYNTAX:

module <*module name*> (<ports*>);
…
<*module implementation*>
…
endmodule

* *Technically referred to as a 'module terminal list' in the documentation.*

MODULE PORTS: BUILDING BLOCK OF VERILOG PROGRAMS



Port: Interface that is used to move signals into or out of the module.

Ports can also be inout ports, that can either listen to signals or send signals out (these are not as easy as just a &x variable parameter in C++)

SYNTAX: **module** <*module name*> (<port names or declaration>);
[<port declarations>]

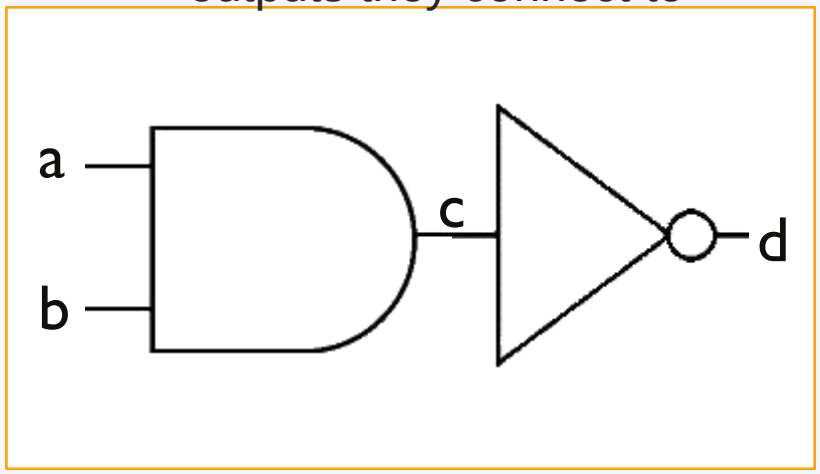
SYNTAX for port declarations:

Verilog '95: specify only the port names in brackets, then under the module declaration define each named port .

Verilog 2001 & later: can still use '95 version or can define ports together with the port name inside the brackets (see later).

WIRES

- Wires (or nets) are used to connect elements (e.g. ports of modules)
- Wires have values continuously driven onto them by outputs they connect to




```
// Defining the wires
// for this circuit:

wire a;
wire a, b, c;
```

REGISTERS

- Registers store data
- Registers retain their data until another value is put into them (i.e. works like a FF or latch)
- A register needs no continuous driver

```
reg myregister;  // declare a new register  (defaults to 1 bit)

myregister = 1'b1;  // set the value to 1
```

This notation explained in a moment…

MODULE ABSTRACTION LEVELS

- Switch Level Abstraction (lowest level)
  - Implementing using only switches and interconnects.
- Gate Level (slightly higher level)
  - Implementing terms of gates like (i.e., AND, NOT, OR etc) and using interconnects between gates.
- Dataflow Level
  - Implementing in terms of dataflow between registers
- Behavioral Level (highest level)
  - Implementing module in terms of algorithms, not worrying about hardware issues (much). Close to C programming.

Arguably the best thing about Verilog!!

SYNTACTIC ISSUES: CONSTANT VALUES IN VERILOG

- Number format:

  <size>'<base><number>

- Some examples:
  - 3'b111 – a three bit number (i.e. 7_{10})
  - 8'ha1 – a hexadecimal (i.e. $A1_{16} = 161_{10}$)
  - 24'd165 – a decimal number (i.e. 165_{10})

Defaults:

100 – 32-bit decimal by default if you don't have a '

'hab – 32-bit hexadecimal unsigned value ($AB_{16} = 171_{10}$)

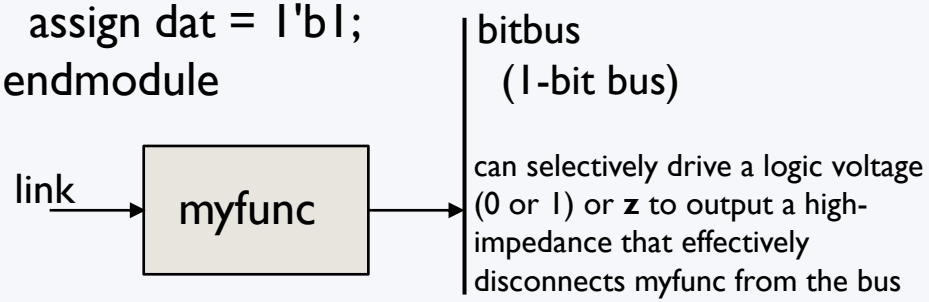
'o77 – 32-bit Octal unsigned value ($77_8 = 63_{10}$)

SYNTACTIC ISSUES: CONSTANT VALUES IN VERILOG

Constant	Hardware Condition
0	Low / Logic zero / False
1	High / Logic one / True
x	Unknown
z	Floating / High impedance

NB!

```
// Example for linking a data bit to a bus if module
// is  asked to put value on the bus via a link input
module myfunc (output busbit, input link  );
  reg dat; // stores result worked on
  assign busbit = link ? dat : 1'bz;
  // do stuff to compute dat
  assign dat = 1'b1;
endmodule
```



Next lecture introduces Verilog program structure and test benches

Try examples online: https://www.edaplayground.com/x/3tUK

VECTORS OF WIRES AND REGISTERS

// Define some wires:

wire a; // a bit wire

wire [7:0] abus; // an 8-bit bus

wire [15:0] bus1, bus2; // two 16-bit busses

// Define some registers

reg active; // a single bit register

reg [0:17] count; // a vector of 18 bits

NON-SYNTHESISABLE DATA TYPES

These datatypes are used both during the compilation and simulation stages to do various things like checking loops, calculations.

- Integer 32-bit value

  integer i; // e.g. used as a counter

- Real 32-bit floating point value

  real r; // e.g. floating point value for calculation

- Time 64-bit value

  time t; // e.g. used in simulation for delays

MEMORY JOGGER…

Q: Explain the Verilog value 10'h10 …

A:

(a) 10-bit value 10_{10}

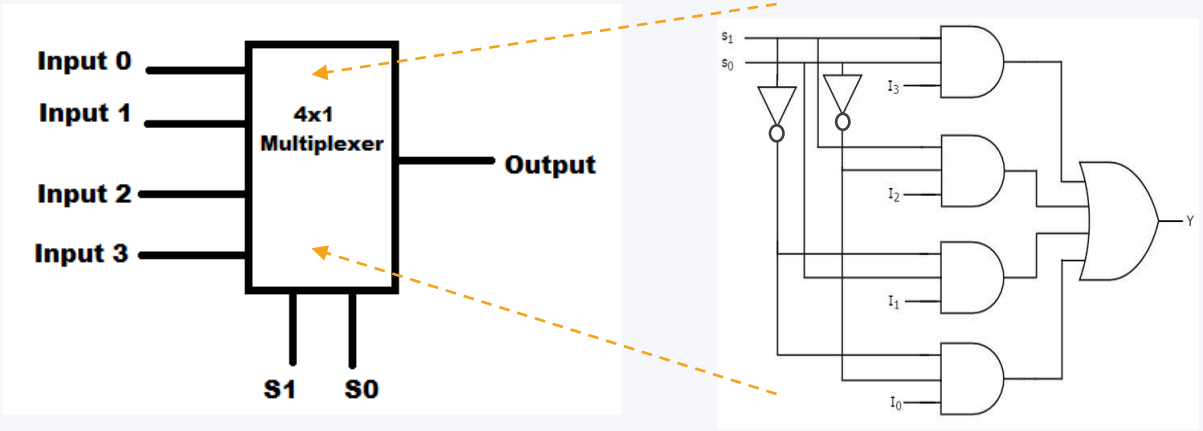
(b) Arbitrary size value 16_{10}

(c) 10-bit value 10_{16}

(d) array [16,16,16,16,16,16,16,16,16,16]

CLASS/TAKEHOME EXERCISE

Pseudo code thinking exercise… (see EEE3096S-L24-Activity.pdf)
Remember how a multiplexer is made using AND, NOT and OR gates?



I know you don't know the full Verilog code yet, but you know there are registers and wires that connect things (i.e. other modules) up.
Experiment with your own pseudocode on how you might express this circuit as code. You could just use binary expressions and that would likely be close enough.
We will revisit this next lecture, using actual Verilog.

MUX 4X1 PSEUDOCODE SAMPLE SOL.

```
// Multiplexer implemented using gates only
start module mux2to1
{
  inputs:  a,b,sel.
  output: y.
  wires: sel,asel,bsel,invsel;
  invsel = ~sel;    // not
  asel = a & invsel;   // and these together, save to asel
  bsel = b & sel;  // and these also, save to bsel
  y = asel | bsel;   // or these lines together for output
}   // wait for next lecture to see proper syntax!
```

Comments/questions:

Q: Would you get marks for this in a test if asked for Verilog?

A: Maybe some marks for showing some logic and understanding but you'd miss out a fair bit if not proper Verilog and appropriate syntax.

Why this approach: I want you to think more about the process and action of converting circuit to text, before diving down into syntax issues.