COMPUTER ORGANIZATION (ECS 409)

HARDWARE DESCRIPTION LANGUAGE AND VERILOG

Dr. Sukarn Agarwal

EECS
Indian Institute of Science Education and Research Bhopal

Agenda for Today and Next Lecture

- Hardware Description Languages
- Implementing Combinational Logic (in Verilog)
- Implementing Sequential Logic (in Verilog)

Required Readings

- Hardware Description Languages and Verilog
  - H&H Chapter 4 in full

Grading Policy

Marks Distribution:

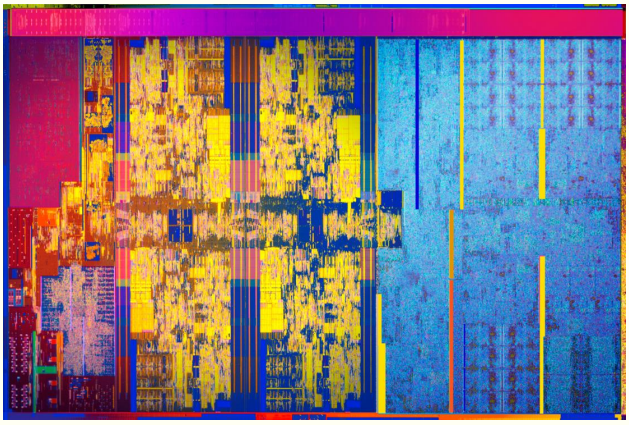
- Assignment: For Practise
- Mid Sem: 40%
- End Sem: 60%

Assignment Type:

- Implementing Combinational Circuit in Verilog
  - Using Structural Verilog Module Coding
  - Using Behavioral Verilog Module Coding
- Implementing Sequential Circuit in Verilog
  - Latches, Counters and Registers
  - State Digram

Hardware Description Languages & Verilog

2017: Intel Kaby Lake



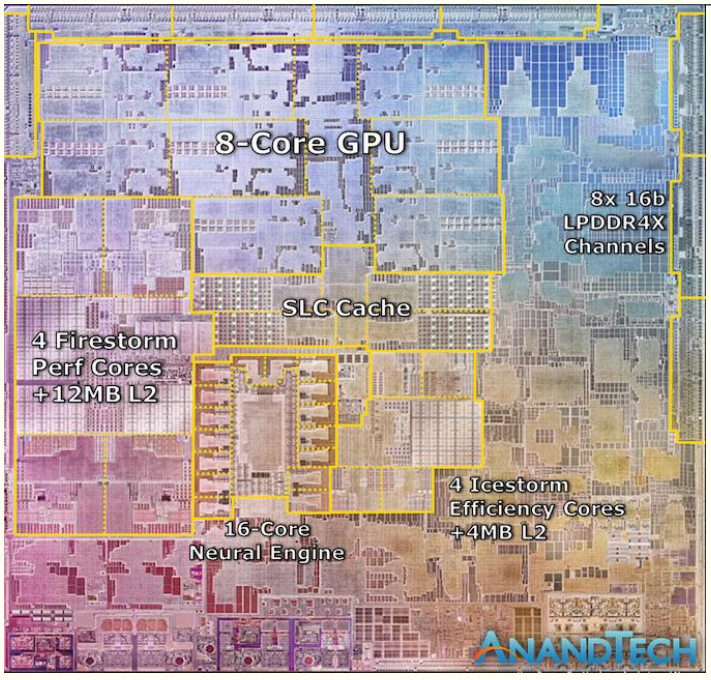
https://en.wikichip.org/wiki/intel/microarchitectures/kaby_lake

- 64-bit processor
- 4 cores, 8 threads
- 14-19 stage pipeline
- 3.9 GHz clock freq.

- 1.75B transistors

- In ~47 years, about 1,000,000-fold growth in transistor count and performance!

2021: Apple M1

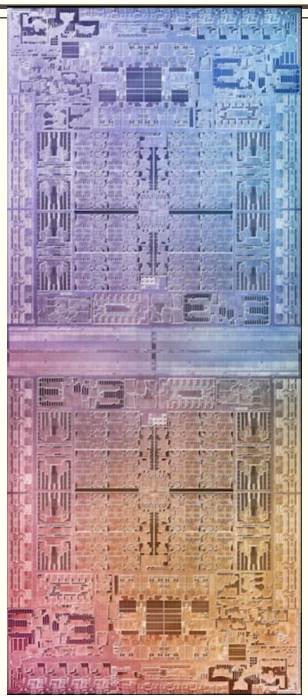


- 4 High-Perf GP Cores
- 4 Efficient GP Cores
- 8-Core GPU
- 16-Core Neural Engine
- Lots of Cache
- Many Caches
- 8x Memory Channels

- 16B transistors

Source: https://www.anandtech.com/show/16252/mac-mini-apple-m1-tested

2022: Apple M1 Ultra

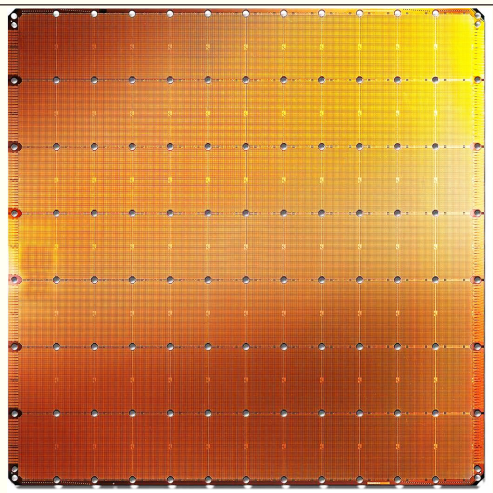


- 16 High-Perf GP Cores
- 4 Efficient GP Cores
- 64-Core GPU
- 32-Core Neural Engine
- Lots of Cache
- Many Caches
- 32x Memory Channels
- 128 GB DRAM

- 114B transistors

https://www.theverge.com/2022/3/9/22968611/apple-m1-ultra-gpu-nvidia-rtx-3090-comparison

2019: Cerebras Wafer Scale Engine



- The largest ML accelerator chip (2019)
- 400,000 cores

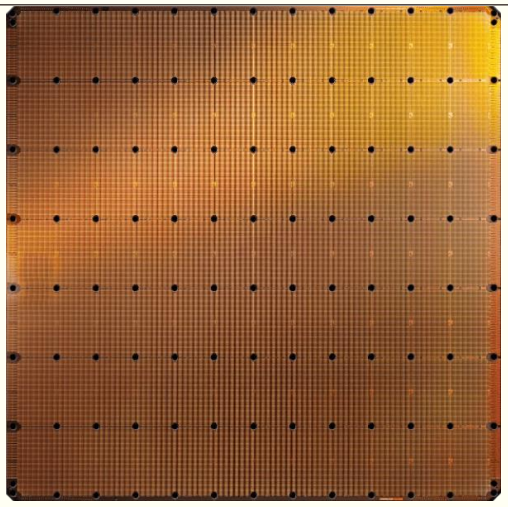
Cerebras WSE
1.2 Trillion transistors
46,225 mm²

Largest GPU
21.1 Billion transistors
815 mm²

https://www.anandtech.com/show/14758/hot-chips-31-live-blogs-cerebras-wafer-scale-deep-learning

https://www.cerebras.net/cerebras-wafer-scale-engine-why-we-need-big-chips-for-deep-learning/

2021: Cerebras Wafer Scale Engine 2



- The largest ML accelerator chip (2021)
- 850,000 cores

Cerebras WSE-2
2.6 Trillion transistors
46,225 mm²

Largest GPU
54.2 Billion transistors
826 mm²

https://www.anandtech.com/show/14758/hot-chips-31-live-blogs-cerebras-wafer-scale-deep-learning
https://www.cerebras.net/cerebras-wafer-scale-engine-why-we-need-big-chips-for-deep-learning/

Transistor Counts Are Growing

Year	Component	Name	Number of MOSFETs (in billions)
2022	Flash memory	Micron's V-NAND chip	5,333 (stacked package of sixteen 232-layer 3D NAND dies)
2020	any processor	Wafer Scale Engine 2	2,600 (wafer-scale design consisting of 84 exposed fields (dies))
2022	microprocessor (commercial)	M1 Ultra	114 (dual-die SoC; entire M1 Ultra is a multi-chip module)
2022	GPU	Nvidia H100	80
2020	DLP	Colossus Mk2 GC200	59.4

In terms of computer systems that consist of numerous integrated circuits, the supercomputer with the highest transistor count as of 2016 is the Chinese-designed Sunway TaihuLight, which has for all CPUs/nodes combined "about 400 trillion transistors in the processing part of the hardware" and "the DRAM includes about 12 quadrillion transistors, and that's about 97 percent of all the transistors."[9] To compare, the smallest computer, as of 2018 dwarfed by a grain of rice, has on the order of 100,000 transistors.

Memory chips have orders of magnitude more transistors than computation chips

Source: https://en.wikipedia.org/wiki/Transistor_count

How to Deal with this Complexity?

- Hardware Description Languages
- What we need for hardware design:
  - Ability to describe (specify) complex designs
  - … and to simulate their behavior (functional & timing)
  - … and to synthesize (automatically design) portions of it
    - have an error-free path to implementation
- Hardware Description Languages enable all of the above
  - Languages designed to describe hardware
  - There are similarly-featured HDLs (e.g., **Verilog**, VHDL, ...)
    - if you learn one, it is not hard to learn another
    - mapping between languages is typically mechanical, especially for the commonly used subset

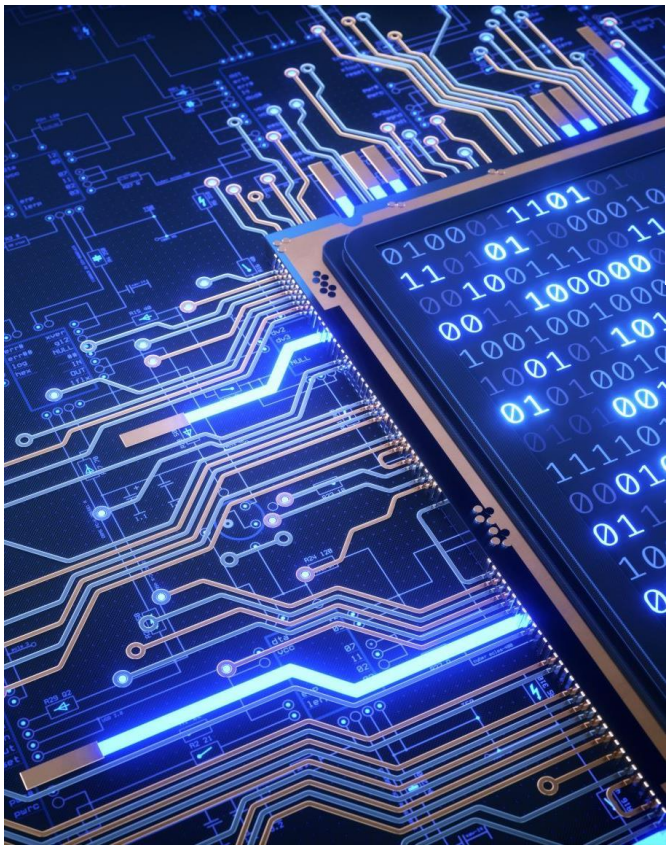
Hardware Description Languages

- **Two well-known hardware description languages**

- **Verilog**
  - Developed in 1984 by Gateway Design Automation
  - Became an IEEE standard (1364) in 1995
  - More popular in US

- **VHDL (VHSIC Hardware Description Language)**
  - Developed in 1981 by the US Department of Defense
  - Became an IEEE standard (1076) in 1987
  - More popular in Europe

- We will use Verilog in this course



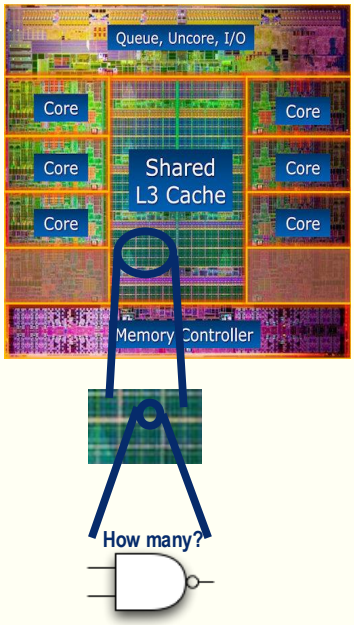
Why Specialized Languages for Hardware?

- HDLs enable easy description of hardware structures
  - Wires, gates, registers, flip-flops, clock, rising/falling edge, …
  - Combinational and sequential logic elements
- HDLs enable seamless expression of parallelism inherent in hardware
  - All hardware logic operates concurrently
- Both of the above ease **specification, simulation & synthesis**

Hardware Design Using HDL

Key Design Principle: Hierarchical Design

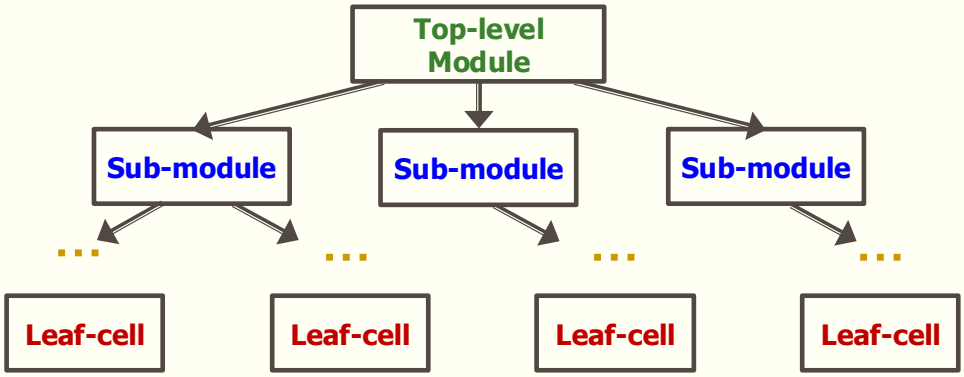
- **Design a hierarchy of modules**
  - Predefined "primitive" gates (AND, OR, …)
  - Simple modules are built by instantiating these gates (e.g., components like MUXes)
  - Complex modules are built by instantiating simple modules, …
- **Hierarchy controls complexity**
  - Analogous to the use of function/method abstraction in programming
- **Complexity is a BIG deal**
  - In real world, how big is the size of a module (that is described in HDL and then synthesized to gates)?



https://techreport.com/review/21987/intel-core-i7-3960x-processor

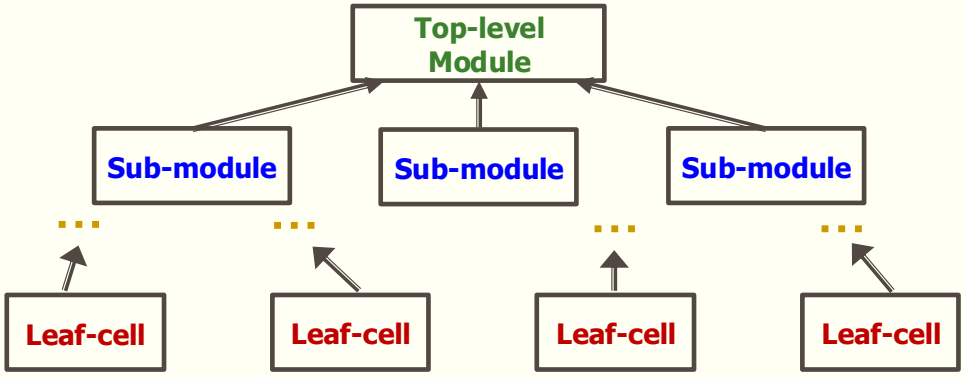
Top Down Design Methodology

- We define the top-level module and identify the sub-modules necessary to build the top-level module
- Subdivide the sub-modules until we come to leaf cells
  - Leaf cell: circuit components that cannot further be divided (e.g., *logic gates, primitive cell library elements*)



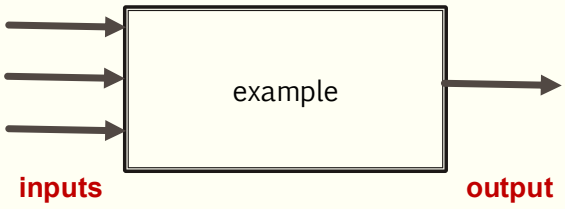
Bottom Up Design Methodology

- We first identify the building blocks that are available to us
- Build bigger modules, using these building blocks
- These modules are then used for higher-level modules until we build the top-level module in the design

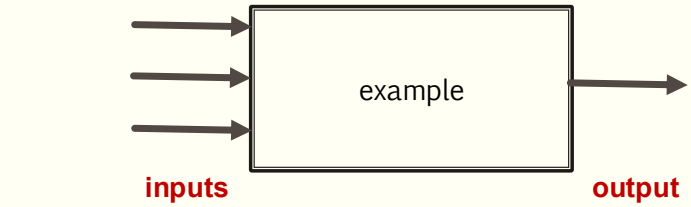


Defining a Module in Verilog

- A module is the main building block in Verilog
- We first need to define:
  - Name of the module
  - Directions of its ports (e.g., input, output)
  - Names of its ports
- Then:
  - Describe the functionality of the module



Implementing a Module in Verilog



name of module

Port list (inputs and outputs)

```
module example (a, b, c, y);
        input a;
        input b;
        input c;
        output y;

// here comes the circuit description

endmodule
```

ports have a declared type

a module definition

A Question of Style (and Consistency)

- **The following two codes are functionally identical**

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

endmodule
```

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

endmodule
```

port name and direction declaration
can be combined

What If We Have Multi-bit Input/Output?

- **You can also define multi-bit Input/Output (Bus)**
  - [range_end : range_start]
  - **Number of bits:** range_end – range_start + 1
- **Example**:

```
input  [31:0] a;   // a[31], a[30] .. a[0]
output [15:8] b1;  // b1[15], b1[14] .. b1[8]
output [7:0]  b2;  // b2[7], b2[6] .. b2[0]
input         c;   // single signal
```

- **a** represents a 32-bit value, so we prefer to define it as: [31:0] a
- It is preferred over [0:31] a which resembles *array* definition
- It is good practice to be consistent with the representation of multi-bit signals, i.e., always [31:0] or always [0:31]

Manipulating Bits

- Bit Slicing
- Concatenation
- Duplication

```
// You can assign partial buses
   wire [15:0] longbus;
   wire [7:0] shortbus;
   assign shortbus = longbus[12:5];

// Concatenating is by {}
   assign y = {a[2],a[1],a[0],a[0]};

// Possible to define multiple copies
   assign x = {a[0], a[0], a[0], a[0]}
   assign y = { 4{a[0]} }
```

Basic Syntax

- Verilog is case sensitive
  - SomeName and somename are not the same!
- Names cannot start with numbers:
  - 2good is not a valid name
- Whitespaces are ignored

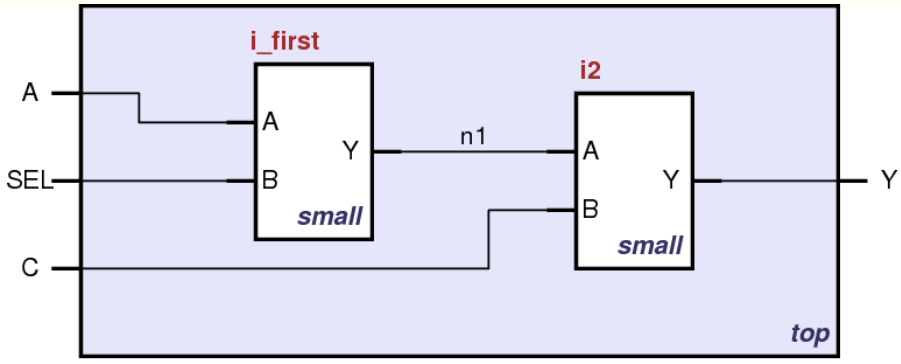
```
// Single line comments start with a //

/* Multiline comments
   are defined like this */
```

Two Main Style of HDL Implementation

- **Structural (Gate-Level)**
  - The module body contains gate-level description of the circuit
  - Describe how modules are interconnected
  - Each module contains other modules (instances)
  - … and interconnections between those modules
  - Describes a hierarchy of modules defined as gates
- **Behavioral**
  - The module body contains functional description of the circuit
  - Contains logical and mathematical operators
  - **Level of abstraction is higher than gate-level**
    - Many possible gate-level realizations of a behavioral description
- **Many practical designs use a combination of both**

Structural (Gate-Level) HDL

Structural HDL: Instantiating a Module

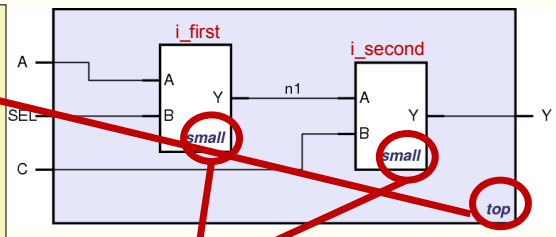


Schematic of module "top" that is built from two instances of module "small"

Structural HDL Example

- **Module Definitions in Verilog**

```
module top (A, SEL, C, Y);
  input A, SEL, C;
  output Y;
  wire n1;









endmodule
```

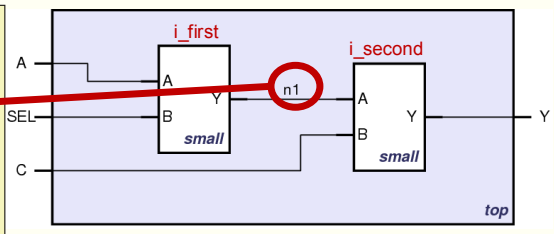



```
module small (A, B, Y);
  input A;
  input B;
  output Y;

// description of small

endmodule
```

Structural HDL Example

- **Defining wires (module interconnections)**

```
module top (A, SEL, C, Y);
  input A, SEL, C;
  output Y;
  wire n1;




endmodule
```

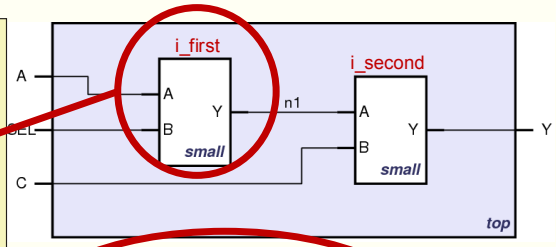



```
module small (A, B, Y);
  input A;
  input B;
  output Y;

// description of small

endmodule
```

Structural HDL Example

- **The first instantiation of the "small" module**

```
module top (A, SEL, C, Y);
  input A, SEL, C;
  output Y;
  wire n1;

// instantiate small once
small i_first ( .A(A),
                .B(SEL),
                .Y(n1)   );



endmodule
```

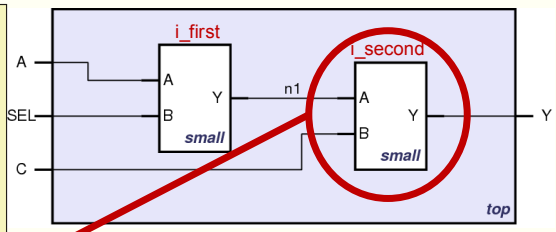



```
module small (A, B, Y);
  input A;
  input B;
  output Y;

// description of small

endmodule
```

Structural HDL Example

- **The second instantiation of the "small" module**

```
module top (A, SEL, C, Y);
  input A, SEL, C;
  output Y;
  wire n1;

// instantiate small once
small i_first ( .A(A),
                .B(SEL),
                .Y(n1)   );

// instantiate small second time
small i_second ( .A(n1),
           .B(C),
           .Y(Y) );

endmodule
```

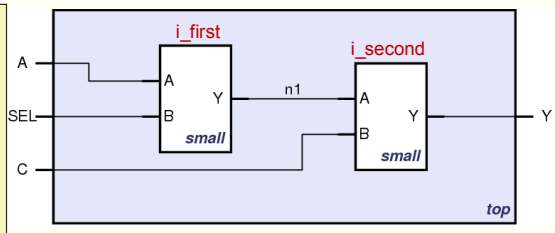



```
module small (A, B, Y);
  input A;
  input B;
  output Y;

// description of small

endmodule
```

Structural HDL Example

- **Short form of module instantiation**

```
module top (A, SEL, C, Y);
  input A, SEL, C;
  output Y;
  wire n1;

// alternative short form
small i_first ( A, SEL, n1 );

/* In the short form above,
   pin order very important */

// safer choice; any pin order
small i_second ( .B(C),
           .Y(Y),
           .A(n1) );

endmodule
```




```
module small (A, B, Y);
  input A;
  input B;
  output Y;

// description of small

endmodule
```

Short form is not good practice
as it reduces code maintainability

Structural HDL Example (II)

- Verilog supports basic logic gates as predefined *primitives*
  - These primitives are instantiated like modules except that they are predefined in Verilog and *do not need a module definition*

```
module mux2(input d0, d1,
            input s,
            output y);
  wire ns, y1, y2;

  not g1 (ns, s);
  and  g2 (y1, d0, ns);
   and  g3 (y2, d1, s);
   or   g4 (y, y1, y2);

endmodule
```