EECS 31L: INTRODUCTION TO DIGITAL DESIGN LAB
LECTURE 1

Salma Elmalaki
salma.elmalaki@uci.edu

The Henry Samueli School of Engineering
Electrical Engineering and Computer Science
University of California, Irvine

Some Slides are compiled from JL Gaudiot, Frank Vahid, and P. Yaghini

LECTURE 1: OVERVIEW

- 31L Team
  - Instructor
  - TAs

- Course administration
  - Course goals and outlines
  - Reference book and outline
  - Course website and policies
  - Tools

- Introduction to logic design
  - Hardware description
  - Structure of HDL models
  - Hardware description types
  - Number and character representation in HDL
  - Testbench
  - Lab 1 description

TEAM

- Instructor:
  - Salma Elmalaki
  - https://faculty.sites.uci.edu/elmalaki/
  - OH: Wed 8:30-9:30 via Zoom (check canvas for link)

- TAs:
  - Rozhin Yasaei ryasaei@uci.edu
    - OH: Tu 5:00-6:00p via Zoom (check canvas for link)

  - Maryam SeyyedHosseini mseyyedh@uci.edu
    - OH: Thu 12:00-1:00p via Zoom (check canvas for link)

COURSE GOALS

- To learn hardware modeling concepts
- To learn Verilog/SystemVerilog structures
  - Data types and operators
  - Basic blocks
  - Concurrent and sequential implementations
- To become familiar with CAD tools
  - Xilinx Vivado
  - To become familiar with the components of a simple single cycle processor
- To learn basic concepts of synthesis

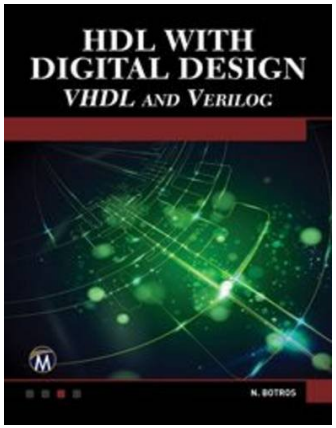
Course outline

- Introduction to logic design
- Data flow description
- Behavioral description
- Structural description
- Tasks, Functions
- Mixed type description
- Advanced HDL description
- Synthesis basics

Textbooks and Material

Reference:

- HDL with Digital Design: VHDL and Verilog
  - Nazeih Botros, Mercury Learning & Information, 2014.



- Lecture notes and discussion
- Lab manuals and lab sessions
- Canvas

Course website

- Course website on Canvas
  - Submission and policies
  - No reconsideration requests more than 1 week after lab grades have been released
- Message board
  - Use it for any questions related to the class

Course policies

- Weekly lab assignments:
  - Should be uploaded to canvas
  - **Do not copy the codes**
    - Plagiarism detection tools are used to detect any duplication
  - Deadline for all labs will be at 11pm(PST) to get full credit. Between 11pm(PST) and 12am(PST), it will be considered a late submission and you will get 50% of the credit. **After 12am(PST), you will get zero points. No exception.**

	Topic	Deadline
Lab 0	Vivado tutorial	No deadline
Lab 1	Basic Logic Blocks in Verilog	See Web site
Lab 2	Arithmetic Operations and ALU	See Web site
Lab 3	Components of Single Cycle Processor	See Web site
Lab 4	Data Path for RISC-V Processor	See Web site
Lab 5	Finalize your 31L Processor Implementation	See Web site

Course policies (cont'd)

- Grading
  - Lab assignments and reports 75%
  - Midterm 20%
  - In class Quizzes 5%
  - +1% Extra credit for instructor evaluation

- Midterm
  - 6th week during class time

- TA Office Hours and Lab sessions:
  - **Make sure ask all your technical questions every week**

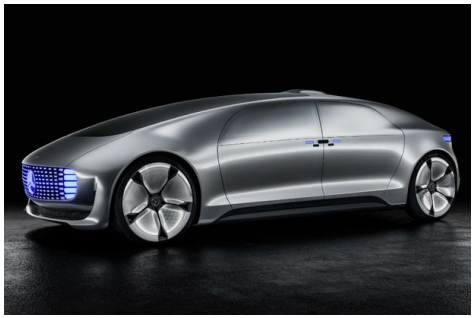
Introduction to logic design

- Digital circuits applications
  - Healthcare devices
  - Electrical automotive (self-driving)
  - Smart homes (Internet of Things)
  - Personal digital assistant (PDA)
  - Home appliance devices

http://blog.boombotix.com/


http://www.pcworld.com/


http://www.eahp.eu/

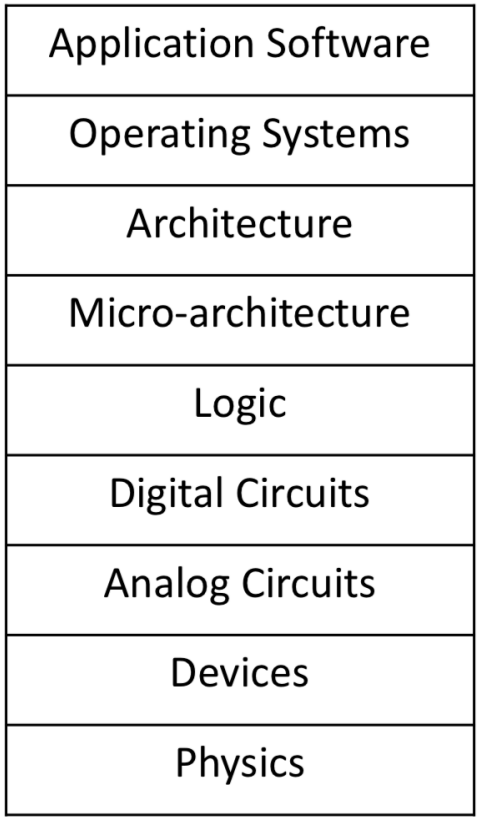

http://www.mirror.co.uk/


http://www.megaleecher.net/

©2015 Ashkan Eghbal

Abstraction

- How an electronic system works?



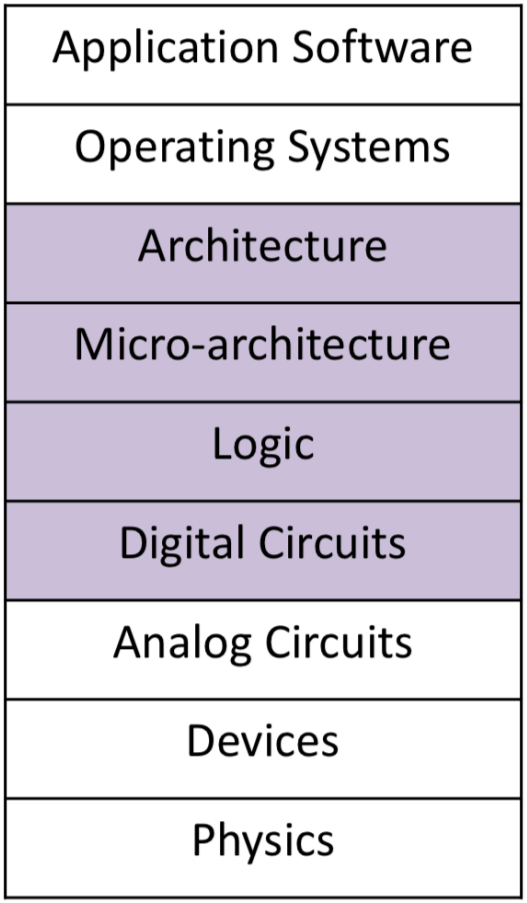
Abstraction

- How an electronic system works?

Programs	Application Software
Drivers	Operating Systems
ISA, Registers	Architecture
Datapath, Control unit	Micro-architecture
ALU, Memories	Logic
AND, OR gates	Digital Circuits
Analog Circuits	Analog Circuits
Devices	Devices
Physics	Physics

Abstraction

- How an electronic system works?



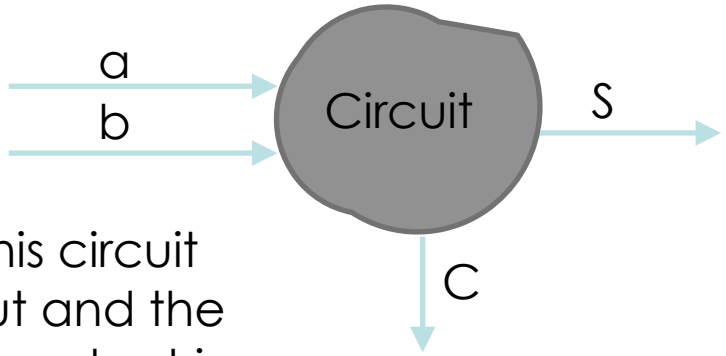
Abstraction

Design Specification:
Half Adder

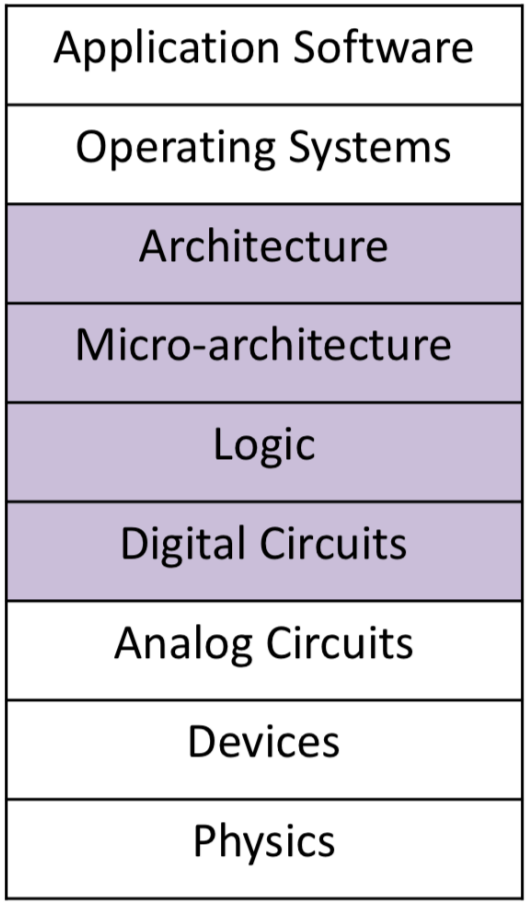
$$S = \bar{a}b + a\bar{b} = a \oplus b$$

$$C_{out} = ab$$

Input		Output	
a	b	S	C
0	0	0	0
1	0	1	0
0	1	1	0
1	1	0	1

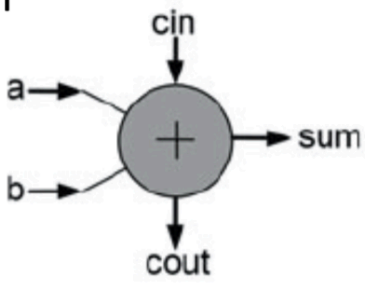


I want to abstract this circuit by defining the input and the output and how the output is a function of input

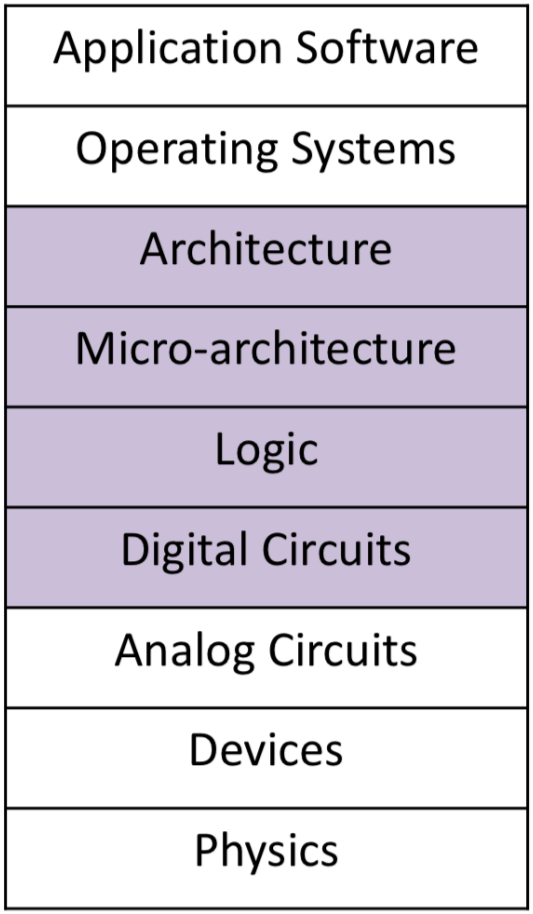


Abstraction

Design Specification

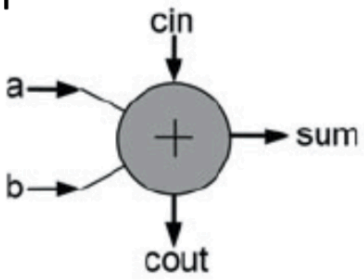


a	b	cin	sum	cout
0	0	0	0	0
0	1	0	1	0
1	0	0	1	0
1	1	0	0	1
0	0	1	1	0
0	1	1	0	1
1	0	1	0	1
1	1	1	1	1



Abstraction

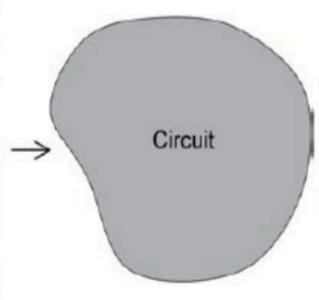
Design Specification



a	b	cin	sum	cout
0	0	0	0	0
0	1	0	1	0
1	0	0	1	0
1	1	0	0	1
0	0	1	1	0
0	1	1	0	1
1	0	1	0	1
1	1	1	1	1

HDL Coding

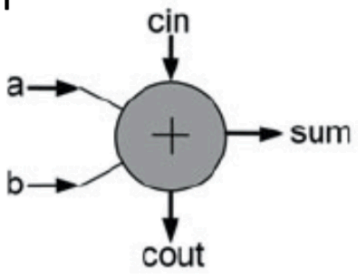
```
module full_adder(
  input a, b, c_in,
  output s, c_out);
  wire net1, net2, net3;
  xor (net1, a, b);
  xor (s, net1, c_in);
  and (net2, net1, c_in);
  and (net3, a, b);
  or (c_out, net2, net3);
endmodule
```



Application Software
Operating Systems
Architecture
Micro-architecture
Logic
Digital Circuits
Analog Circuits
Devices
Physics

Abstraction

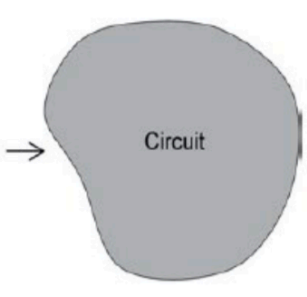
Design Specification



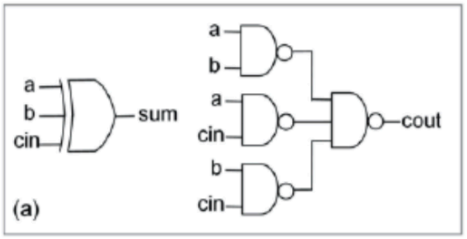
a	b	cin	sum	cout
0	0	0	0	0
0	1	0	1	0
1	0	0	1	0
1	1	0	0	1
0	0	1	1	0
0	1	1	0	1
1	0	1	0	1
1	1	1	1	1

HDL Coding

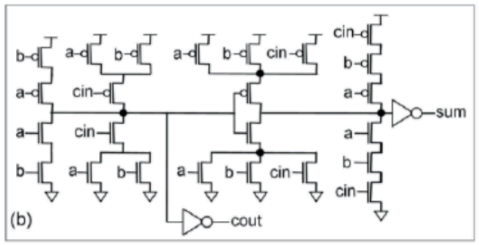
```
module full_adder(
  input a, b, c_in,
  output s, c_out);
  wire net1, net2, net3;
  xor (net1, a, b);
  xor (s, net1, c_in);
  and (net2, net1, c_in);
  and (net3, a, b);
  or (c_out, net2, net3);
endmodule
```



Gate-level code



Circuit-level code

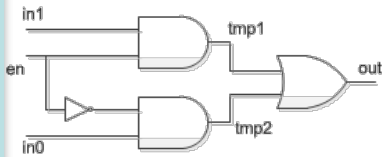


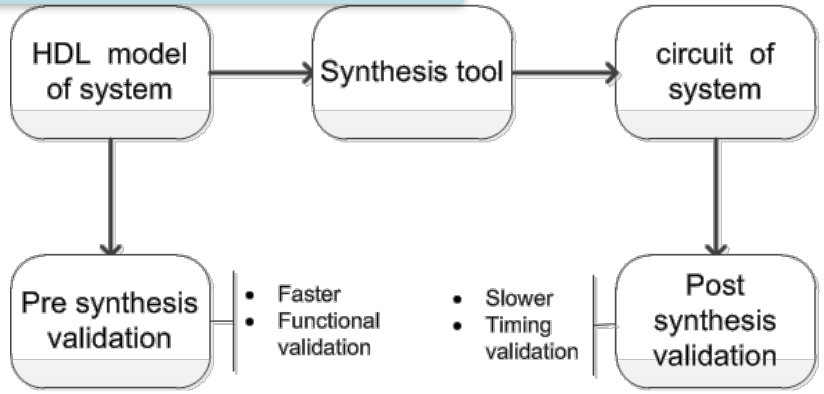
Application Software
Operating Systems
Architecture
Micro-architecture
Logic
Digital Circuits
Analog Circuits
Devices
Physics

Hardware description

- HDL
  - Hardware Description language
    - VHDL and Verilog
    - SystemVerilog and SystemC
- Why HDL?
  - Simulate at higher level
    - Complexity of design
    - Verify the design before manufacturing
  - Direct implementation from HDL code
    - Enhance designer productivity

```
module mux(in1, in2, en, out);
input in1, in2, en;
output out;
// details not shown
endmodule
```





HDL to Gates

- **Simulation**

– Input values are applied to the circuit

– Outputs checked for correctness

– Millions of dollars saved by debugging in simulation instead of hardware

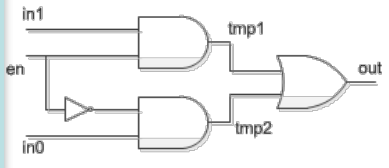
- **Synthesis**

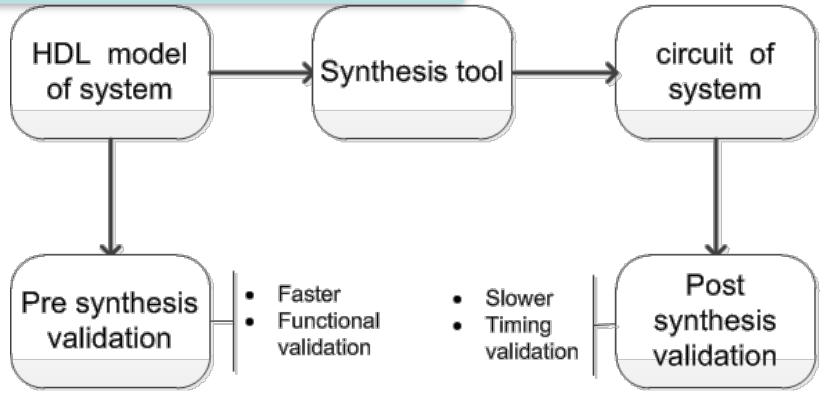
– Transforms HDL code into a netlist describing the hardware (i.e., a list of gates and the wires connecting them)

IMPORTANT:

When describing circuits using an HDL, it's critical to think of the hardware the code should produce.

```
module mux(in1, in2, en, out);
input in1, in2, en;
output out;
// details not shown
endmodule
```





Hardware Design and Modeling Keywords

- **Simulation**
  -- Is the behavior prediction of a design
- EDA Tools (Simulation)
  - Altera: Quartus II
  - Xilinx : ISE, **Vivado**
  - Mentor Graphics : Modelsim (QuestaSim)
  - Synopsys : VCS
  - Cadence: Incisive toolset (ncsim)
  - …
- **Synthesis**
  -- Is the process of converting a design into absolute gate-level netlist, describing the exact structure of the design

  ✓ In SystemVerilog not all statements are synthesizable.
- EDA Tools (Synthesis)
  - Mentor Graphics : leonardo
  - Synopsys: Synplify (FPGA) and Design Compiler (ASIC)
  - Xilinx : ISE / **Vivado (FPGA)**
  - …

Hardware description

- HDL types
  - Many different HDLs are available
    - VHDL
    - Verilog
    - Altera Hardware Description Language (AHDL)
    - THDL++ (Templated HDL inspired by C++)
    - SystemVerilog (Object-oriented for verification)
    - …
- VHDL and Verilog standards
  - VHDL
    - IEEE standard 1076-1993
    - IEEE 1076-2008 (87,93,2000,2002,2008)
  - Verilog
    - IEEE standard 1364-1995
    - IEEE 1800-2005

Structure of the HDL models

- Verilog **module** structure
  - Declaration
    - Input/output signals declarations
  - Body
    - the relation between input and output

```
module Half_adder(a,b,S,C);
        input a,b;
        output S, C;
// Blank lines are allowed



assign S = a ^ b;  // statement1
assign C = a & b; // statement2
endmodule
```

*Blue words are reserved words in Verilog also known as keywords.

Structure of the HDL models

basic types of Modules:

– Data flow:
  describe how the signals flow from inputs to outputs

– Behavioral:
  describe what a module does

– Structural:
  describe how a module is built from simpler modules

Hardware description types

- Data flow description
- Behavioral description
- Structural description
- Switch-level description
- Mixed-type
- Mixed-languages description
- Selection decision
  - Available information of the system
  - The complexity of the system

Hardware description types
Data flow

- System's flow from input to output
- Description by the Boolean description
  - Truth table is needed, or
  - Boolean functions such as sum output in Half_adder
    - `sum = a ^ b`
- Statements are concurrent
- Statement execution controlled by events
- There is no keyword to detect this type
- More on it in Chapter 2

Data flow description example

Verilog example

```
module Half_adder(a,b,S,C);
        input a,b;
        output S, C;
 // Blank lines are allowed

 assign S = a ^ b;
 //statement 1
 assign C= a & b;      //statement 2
 endmodule
```

Concurrent execution

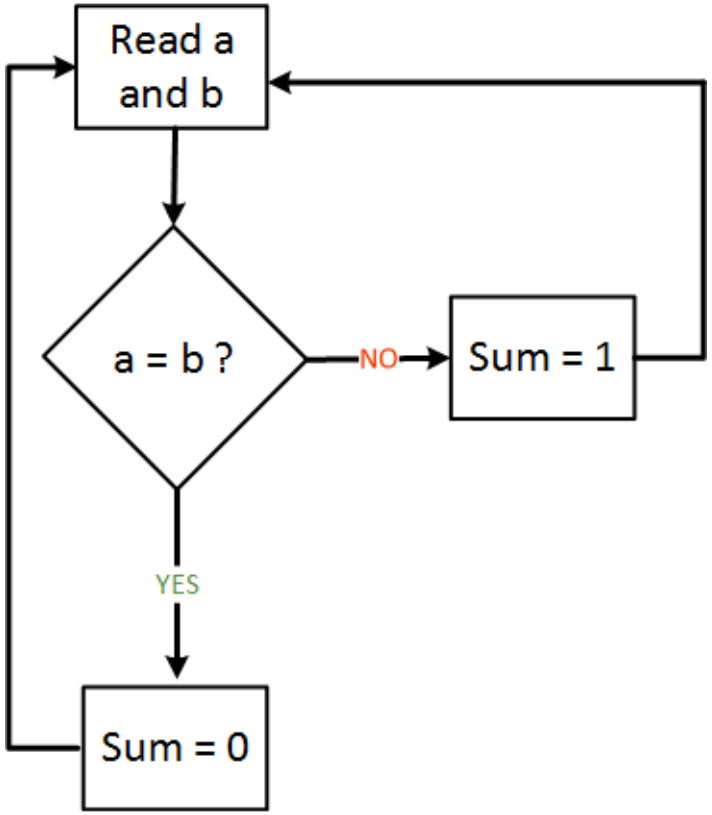
- Boolean functions (Truth table)

Hardware description types
Behavioral

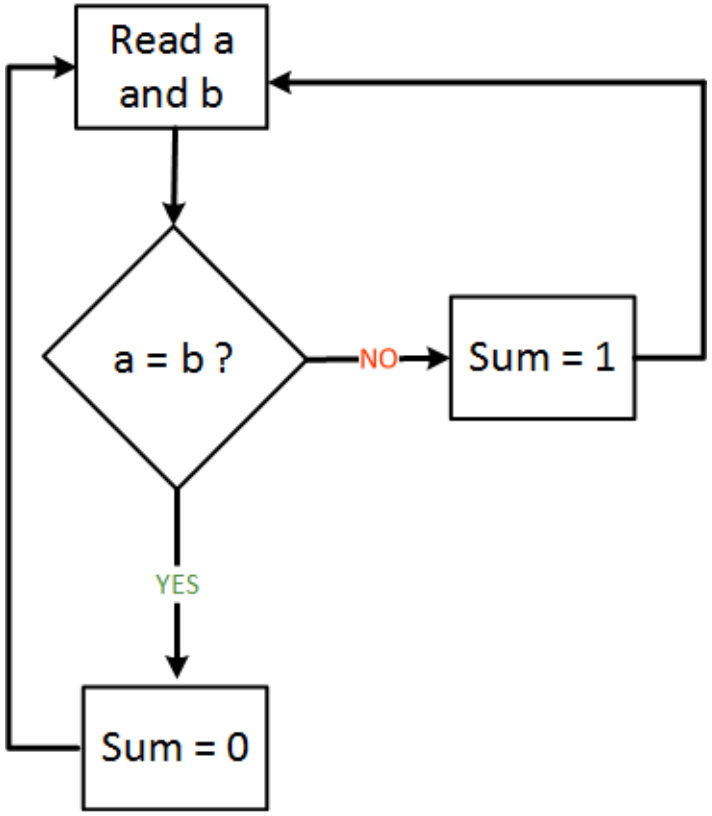
- How the output behaves with the input signal values
- It can be shown by a flowchart
- Usually used when it is hard to obtain either of
  - The Boolean function
  - Digital logic of the system
- The keyword for detecting this model is
  - Verilog: `always` or `initial` inside module
- More on it in Chapter 3

Behavioral description example

- Half adder flow chart
  - If a= b
    - 0 + 0
    - 1 + 1
  - Else a ≠ b
    - 0 + 1
    - 1 + 0



Behavioral description example (cont'd)



Verilog example

```
module Half_adder(a,b,S,C);
input a,b;
output S, C;
reg S;

always @ (a,b)
begin
        if (a != b)
        begin
                S = 1'b1;
        end
        else
        begin
                S = 1'b0;
        end
end
endmodule
```

Hardware description types
Structural

- Models the system as components or gates
- Basic components are designed
  - Define any instances of them as needed
  - For example
    - Half_adder can be implemented by defining different instances of
      - And gates
      - Or gates
      - Xor gates
- It is identified by
  - Verilog: gates constructs such as `and, or, not`
- More on it in Chapter 4

Structural description example (cont'd)

Behavioral

```
module Half_adder(a,b,S,C);
input a,b;
output S, C;
reg S,C;

always @ (a,b)
begin
        if (a != b)
        begin
                S = 1'b1; C = 1'b0;
        end
        else
        begin
                S = 1'b0; C = 1'b0;
                if (a==1'b1)
                  C = 1'b1;
        end
end
endmodule
```

Structural

```
module Half_adder1(a,b,S,C);

input a, b;
output S,C;

and a1(C,a,b);
//The above statement is AND gate

xor x1(S,a,b);
//The above statement is EXCLUSIVE-OR gate

endmodule
```

Data Flow

```
module Half_adder(a,b,S,C);
        input a,b;
        output S, C;
assign S = a ^ b;
assign C= a & b;
endmodule
```

Mixed-type and mixed-languages description

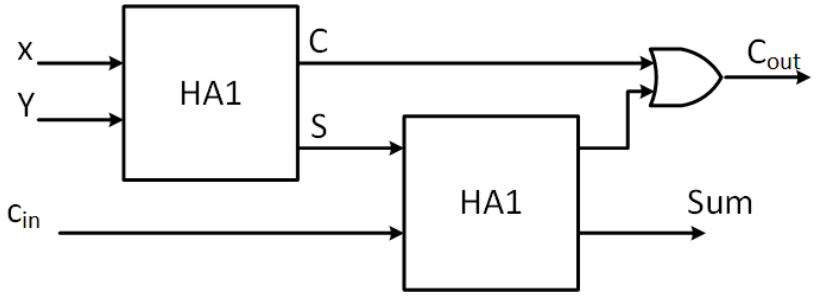
- Mixed-type also called mixed-style
  - Use more than one of the types already discussed
  - Popular in most large systems

- Mixed-languages
  - Newly added to HDL description
  - Write a module in VHDL or Verilog
    - Invoke or import a construct written in other language
    - The construct is a module (Verilog)

Mixed-languages example (cont'd)

- Full_adder implementation with half_adder

X	Y	C_{in}	S	C_{out}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1



Mixed-languages example (cont'd)

Full_adder (verilog)

```
module Full_Adder1 ( x,y, cin, sum,
carry);
        input x,y,cin;
        output sum, carry;
        wire c0, c1, s0;

/* Description of HA is
written in VHDL with HA
entity name
*/

HA H1 (x, y, s0,c0);
HA H2 (s0, cin, sum,c1);
assign carry = c0 | c1;

endmodule
```

Half_adder (VHDL)

```
library IEEE;
use ieee.std_logic_1164.all;

entity HA is
--The entity has to be named here as

port (a, b : in std_logic; s, c: out
std_logic);
end HA;

architecture HA_Dtflw of HA is
begin
        s <= a xor b;
        c <= a and b;
end HA_Dtflw;
```

Number and character representation in Verilog

- Integer numbers:
  - As a decimal number (signed or unsigned)
    - 253, -56
  - as binary, octal, decimal, or hexadecimal form
    - sign
      - Optional
      - Allowed only with decimal numbers
    - Bit_width
      - The (decimal) number in this field corresponds to the width of the number (number of bits)
    - ‘number_base
      - Do not forget the ‘ apostrophe
      - The base of number: **d** (decimal), **b** (binary),**o** (octal),**h** (hexadecimal)
    - Value
      - Corresponds to the value of the number

sign	bit_width	‘number_base	value

From Elsevier

Number and character representation in Verilog

Format: N'Bvalue

N = number of bits, B = base

N'B is optional but recommended (default is decimal)

Number	# Bits	Base	Decimal Equivalent	Stored
3'b101	3	binary	5	101
'b11	unsized	binary	3	00…0011
8'b11	8	binary	3	00000011
8'b1010_1011	8	binary	171	10101011
3'd6	3	decimal	6	110
6'o42	6	octal	34	100010
8'hAB	8	hexadecimal	171	10101011
42	Unsized	decimal	42	00…0101010

Number representation in Verilog (cont'd)

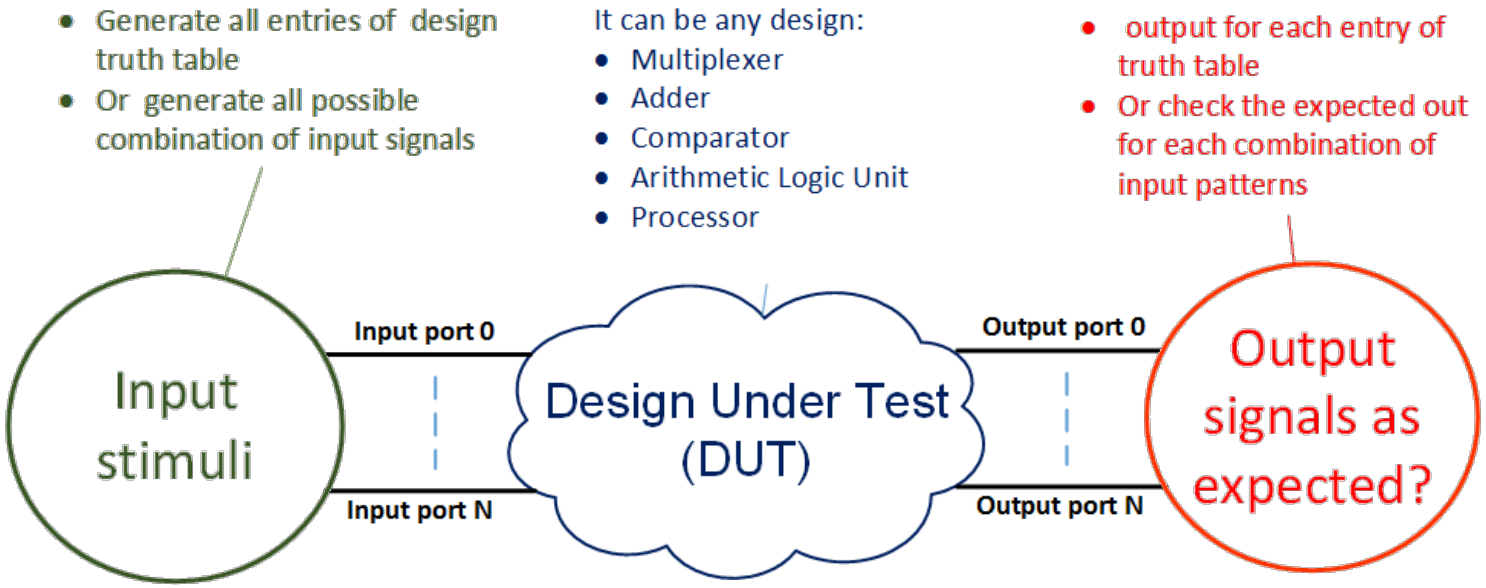
- Some notes about the second representation
  - The base number characters or the magnitude can be either case
    - Although verilog is case sensitive
      - A,B,C,D,E,F,O,X,H,Z are the same as a,b,c,d,e,f,o,x,h,z
  - The underscore character can be used anywhere after the first character.
    - It adds to the readability. It is normally ignored.
  - If the number size is smaller than the size specified
    - The size is made up by padding 0's to the left
    - If the leftmost bit is a **x** or **z**, the same is padded to the left

Number representation in Verilog (cont'd)

- Real numbers
  - Decimal notation
    - -a.b
    - a and b must be present in decimal number
    - No limit with the number of digits on either side of decimal point
  - Scientific notation
    - There is only one digit on the left side of decimal point
    - The number after e shows the power of 10
    - For example 4.3e2 = $4.3 \times 10^2 = 430$

Testbench

- Verify the corrected implemented design
  - Instance of Design Under Test
  - Input stimuli
  - Output signals as expected?



Testbench (cont'd)

- No port definition inside module
- Module is self-contained
  - require no inputs
  - generate no outputs
- An instance of DUT should be defined
- For each port of DUT components
  - Signals connected to the DUT inputs (regs) should be defined
  - Signals connected to the DUT outputs (wire) should be defined
- If DUT is sensitive to clk
  - Clk generation implemented through a separate process
  - Input ports of DUT component should be generated at the beginning of each clock

Testbench for half adder (verilog)

DUT

```
module Half_adder(a,b,S,C);
            input a,b;
            output S, C;
 // Blank lines are allowed

 assign S = a ^ b;
 //statement 1
 assign C= a & b;      //statement 2
 endmodule
```

TB

```
module testbench();
reg in0_tb,in1_tb;
wire s_tb,c_tb;

Half_adder instant
(.a(in0_tb),.b(in1_tb),.S(s_tb),.C(c_tb));

initial
begin
            in0_tb=1'b0;
            in1_tb=1'b0;
end
always
begin      in0_tb=1'b0;
            in1_tb=1'b0;
            #20
            in0_tb=1'b0;
            in1_tb=1'b1;
            #10
            in0_tb=1'b1;
            in1_tb=1'b0;
            #10
            in0_tb=1'b1;
            in1_tb=1'b1;

end
endmodule
```

Lab1 description

- Lab 1 will be assigned next week
- Familiarize yourself with Verilog
- Design logic components like half adders, full adders, subtractors, … etc
  - Design some basic logic blocks in Verilog
  - Verify your designs to see if they are working as expected
  - Run the simulation
  - Report the results