CPEG 422/622
EMBEDDED SYSTEMS DESIGN

Chengmo Yang

chengmo@udel.edu

Evans 201C

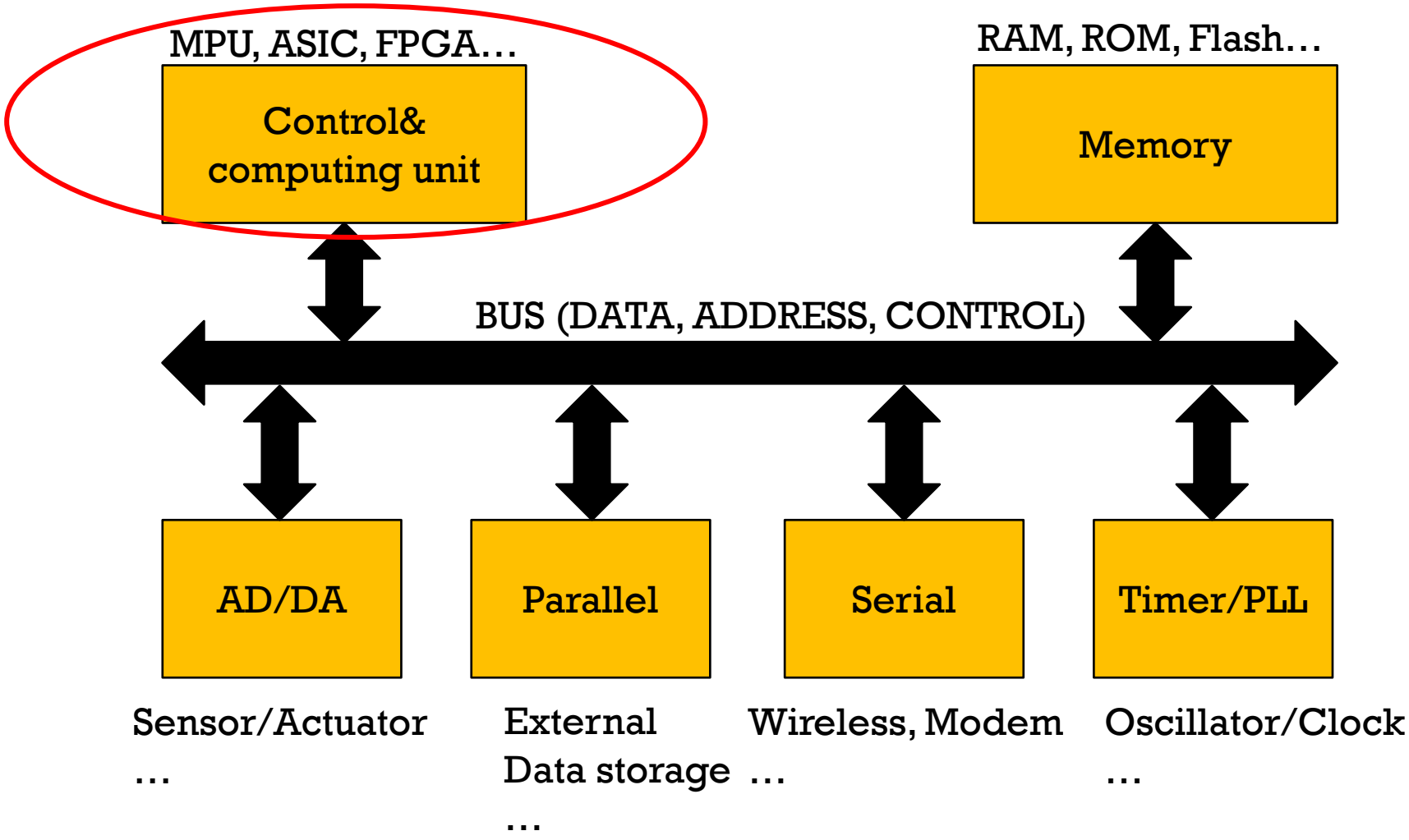
LECTURE 2
FPGA & VHDL INTRODUCTION

WHERE ARE WE?

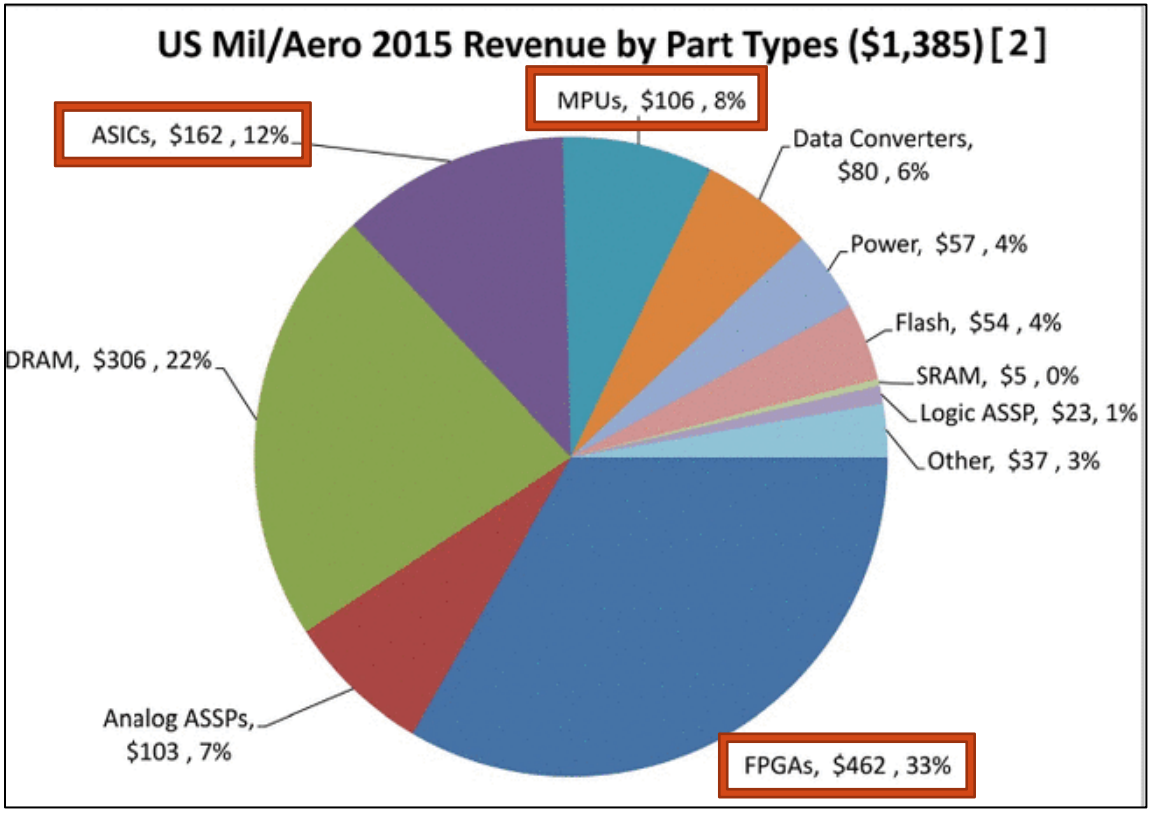
- Last lecture we introduced:
  - Embedded systems, FPGAs, VHDL
  - Course content

- This lecture we will discuss:
  - FPGA structure and programming
  - VHDL design structure

EMBEDDED SYSTEM ARCHITECTURE



FPGAS ARE POPULAR!

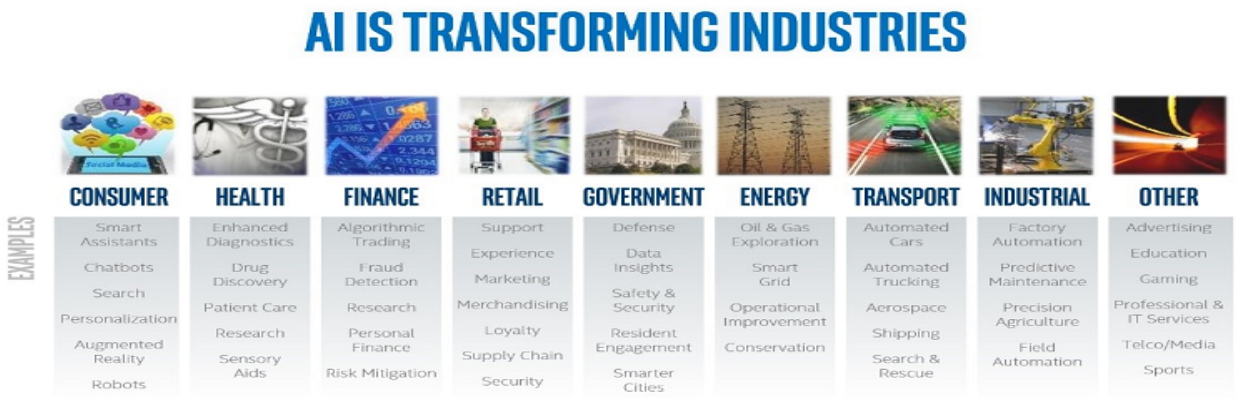


Microelectronics in Department of Defense 2015

Sources: IDA assessment and dataBeans

FPGAS ARE WIDELY USED!

FPGAs in AI era

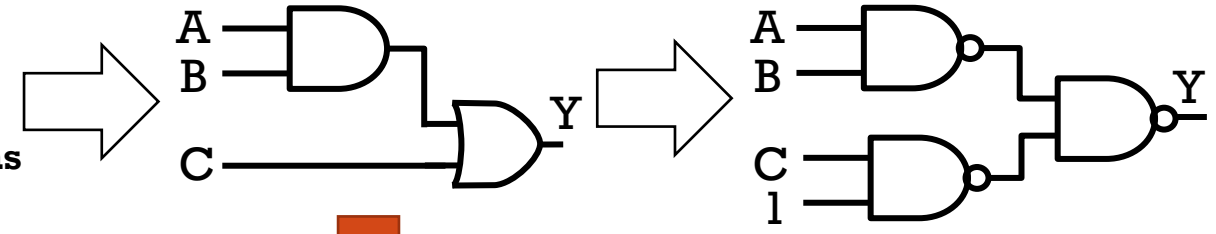


Sources: Intel

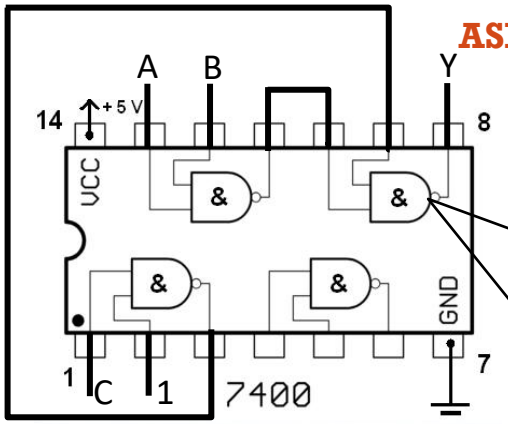
ASIC – FIXED LOGIC

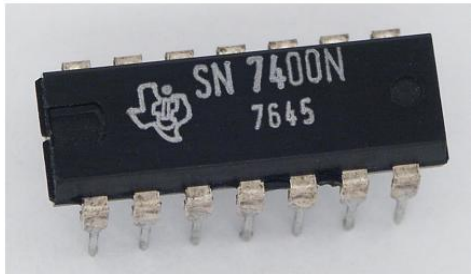
$Y = (A\&B)\,|\,C$

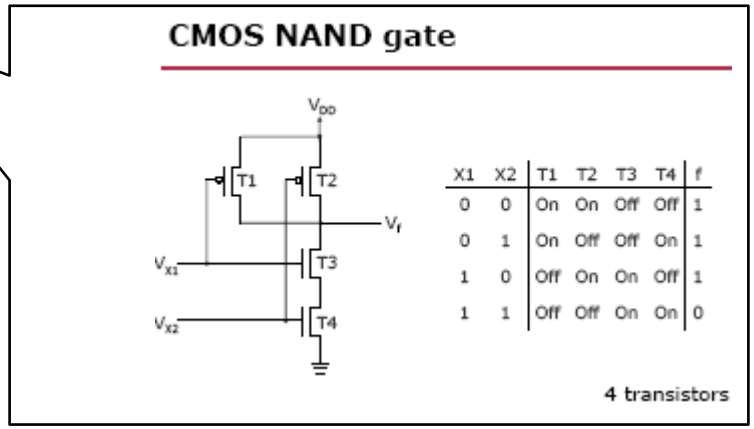
Boolean Expressions



ASIC implementation



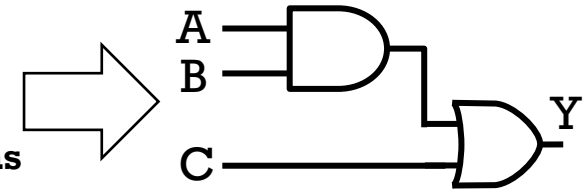




FPGA – REPROGRAMMABLE LOGIC

$Y = (A\&B)\,|\,C$

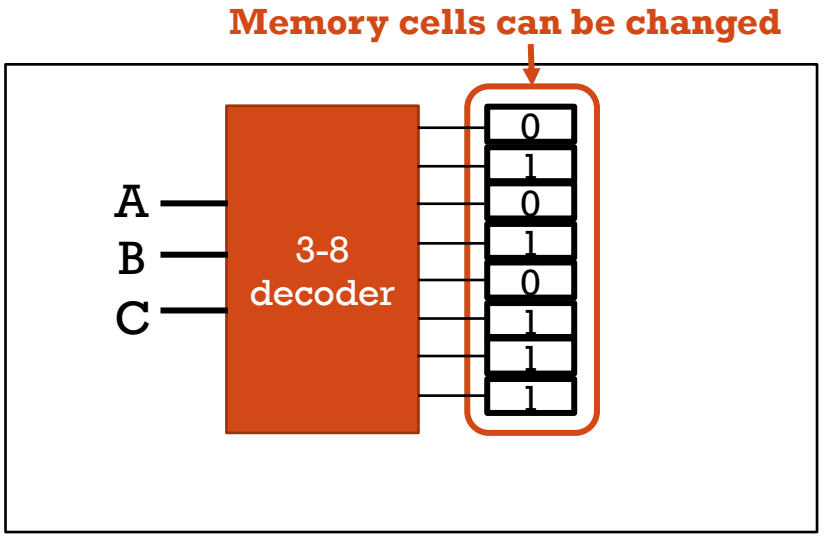
Boolean Expressions



FPGA implementation

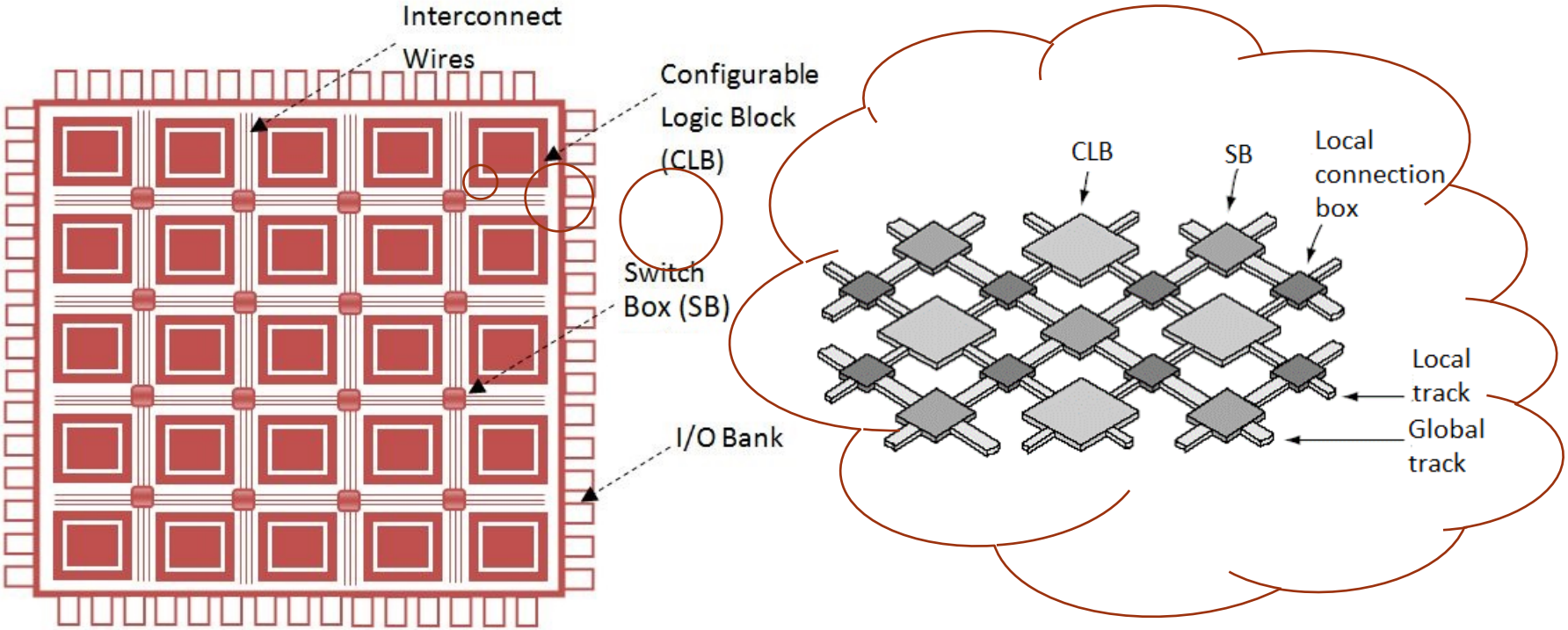
Truth Table

A	B	C	Y
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

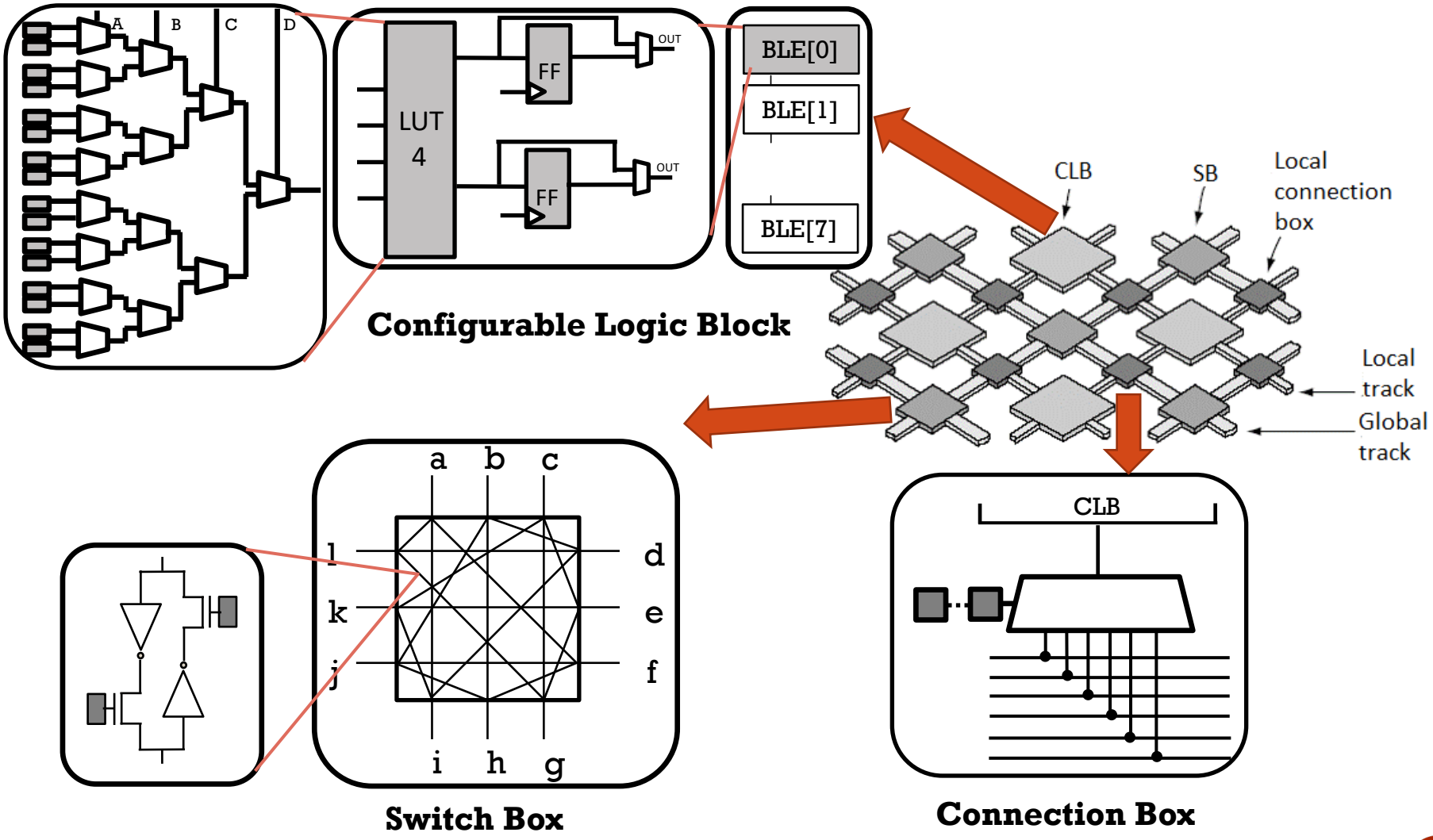


3 Inputs in LUT -> 8 Memory Cells

FPGA STRUCTURE



FPGA STRUCTURE

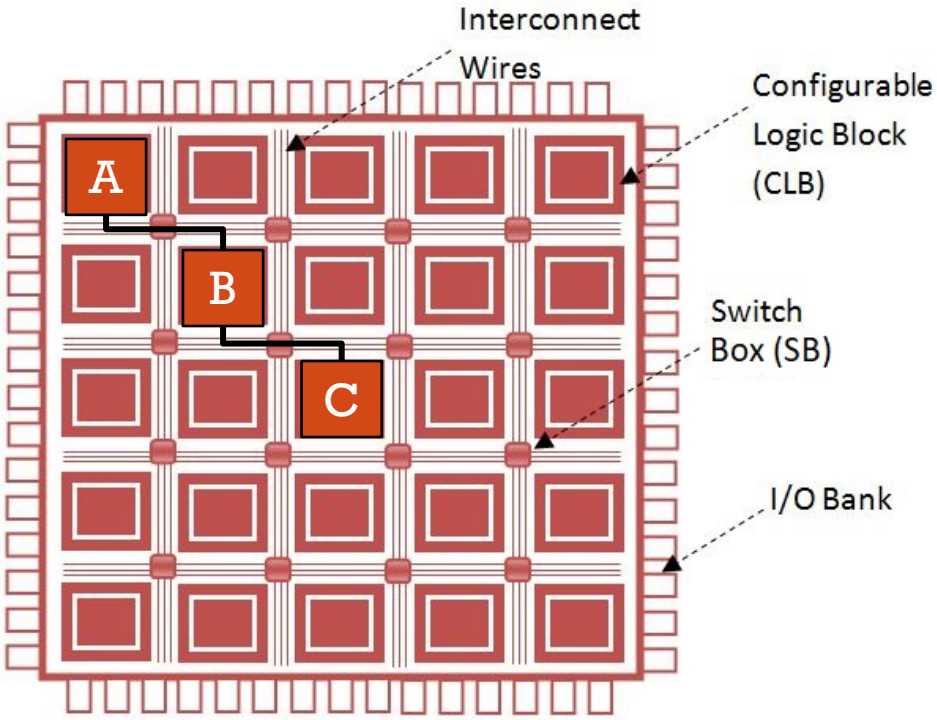


FPGA CONFIGURATION

Configuration means:

- CLBs are programmed to implement logic.
- CBs and SBs are properly routed to connect CLBs.

} **via VHDL**



TODAYS LECTURE

- FPGA structure and programming
- **VHDL design structure**

WHAT IS VHDL?

VHDL stands for

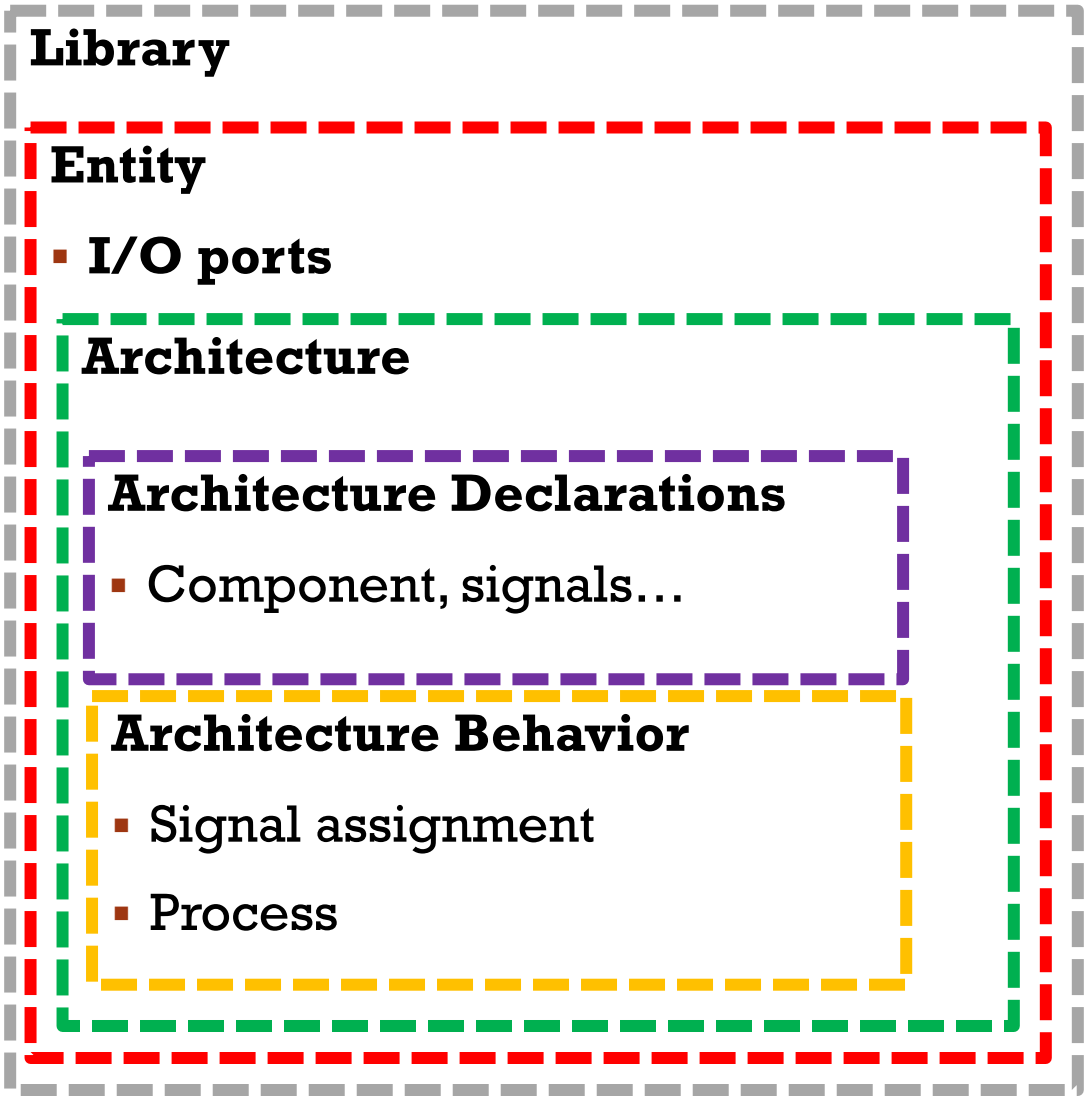
- **V**HSIC (Very High Speed Integrated Circuits)
- **H**ardware
- **D**escription
- **L**anguage

It follows IEEE Standard 1076-1993

A hardware description language is **inherently parallel**

- Commands, which correspond to logic gates, are executed (computed) in parallel, as soon as a new input arrives.

VHDL DESIGN STRUCTURE



LIBRARY

Library is the tool set.

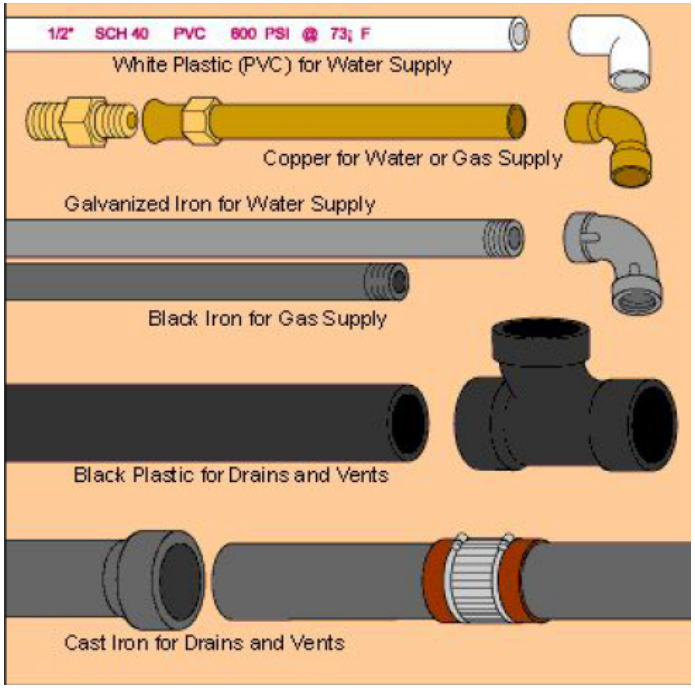
Syntax:

Library *<library_name>;*

Use *< library_name >.<package_name>.[all | part];*

Example:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
```



IEEE LIBRARIES

std_logic_1164	defines standard datatypes such as logic and vector
std_logic_arith	provides arithmetic, conversion and comparison functions for the signed, unsigned, integer, std_ulogic, std_logic and std_logic_vector types
std_logic_unsigned	provides unsigned numerical computation on type std_logic_vector
numeric_std	provides arithmetic functions for vectors for both signed and unsigned
std_logic_textio	provides user defined input/output stream, such as printing on screen

More library packages and more explanation can be found at:
https://www.csee.umbc.edu/portal/help/VHDL/stdpkg.html

ENTITY

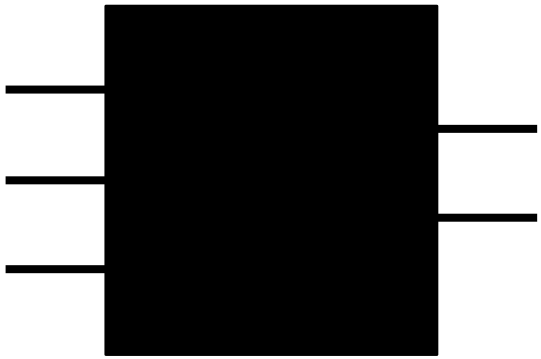
Entity is a black box with inputs and outputs.

Syntax:

Entity *<entity_name> is;*

Port *[Port_list](<name>:<in|out|inout> <type>);*

End <entity_name>;



Example*:*

```
entity full_adder is
 port (
      A : in STD_LOGIC;
      B : in STD_LOGIC;
      Cin : in STD_LOGIC;
      S : out STD_LOGIC;
     Cout : out STD_LOGIC
      );
end full_adder;
```

IDENTIFIERS – EXAMPLE

- Purple and unbold words are keywords.
  - List of VHDL keywords: https://www.csee.umbc.edu/portal/help/VHDL/reserved.html
- Purple and bold words are **datatypes** defined in the library.
- Black words are user-defined identifiers.
- Green words are names of I/O ports, also user-defined identifiers.

Example:

```
entity full_adder is
 port (
      A : in STD_LOGIC;
      B : in STD_LOGIC;
      Cin : in STD_LOGIC;
      S : out STD_LOGIC;
     Cout : out STD_LOGIC
      );
end full_adder;
```

IDENTIFIERS – GENERAL RULE

- Keywords and user-defined identifiers are **case insensitive**.
- May contain only alpha-numeric characters (A to Z, a to z, 0-9) and the underscore (_) character.
- The first character must be a letter and the last one cannot be an underscore.
- An identifier cannot include two consecutive underscores.
  - Lines with comments start with two adjacent hyphens (--) and will be ignored by the compiler.

ARCHITECTURE

Architecture defines function and behavior inside.

Syntax:

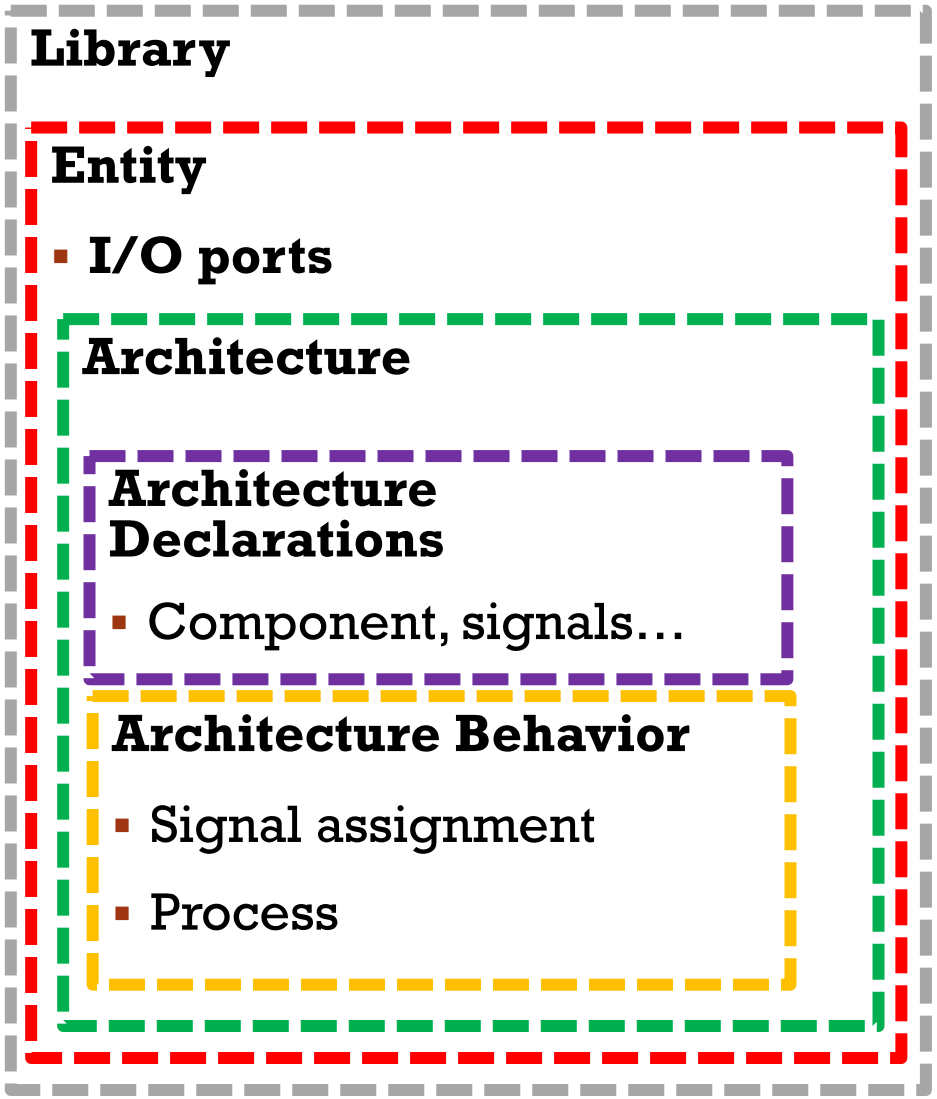
```
architecure <architecture_name> of
<entity_name> is
[
  component declarations (detail)
  function declarations (detail)
  signal declarations (detail)
  constant declarations (detail)
  variable declarations (detail)
  type declarations (detail)
  ...
  -- there are others, but these are what
  you'll need
  -- for this course
]
begin
[
  combinatorial statements
  sequential statements
  ...
]
end architecture;
```

Architecture declaration

- Component
- Signal…

Architecture behavior

EXAMPLE-FULL ADDER




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

entity full_adder is
  port
  (
    A : in STD_LOGIC;
    B : in STD_LOGIC;
    Cin : in STD_LOGIC;
    S : out STD_LOGIC;
    Cout : out STD_LOGIC
  );
end full_adder;

architecture full_adder of full_adder is
begin

  S <= A XOR B XOR Cin ;
  Cout <= (A AND B) OR (Cin AND A) OR (Cin AND B) ;

end full_adder;
```

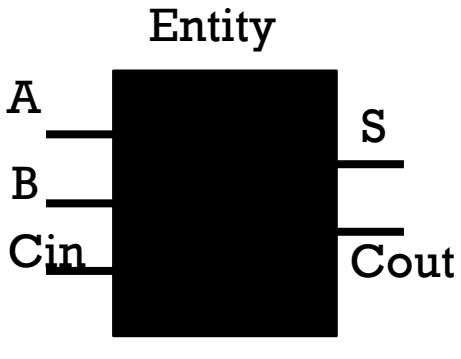
Name match!

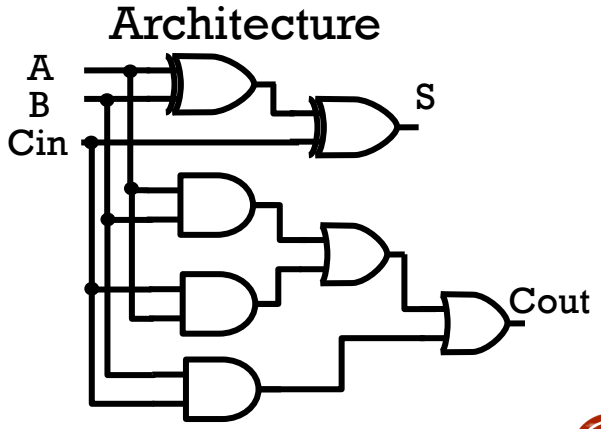
ARCHITECTURE 1

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

```
entity full_adder is
  port
  (
    A : in STD_LOGIC;
    B : in STD_LOGIC;
    Cin : in STD_LOGIC;
    S : out STD_LOGIC;
    Cout : out STD_LOGIC
  );
end full_adder;
```

```
architecture full_adder of full_adder is
begin
  S <= A XOR B XOR Cin ;
  Cout <= (A AND B) OR (Cin AND A) OR (Cin AND B) ;

end full_adder;
```

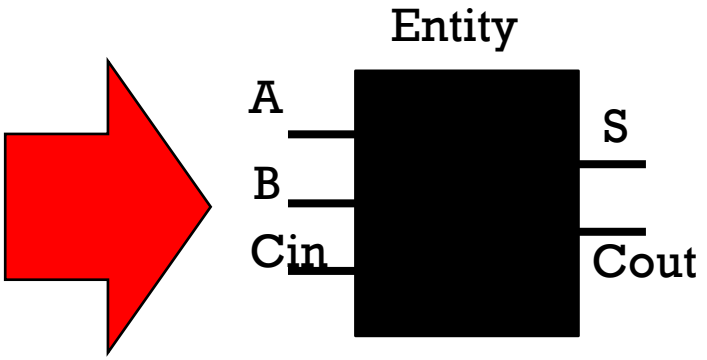


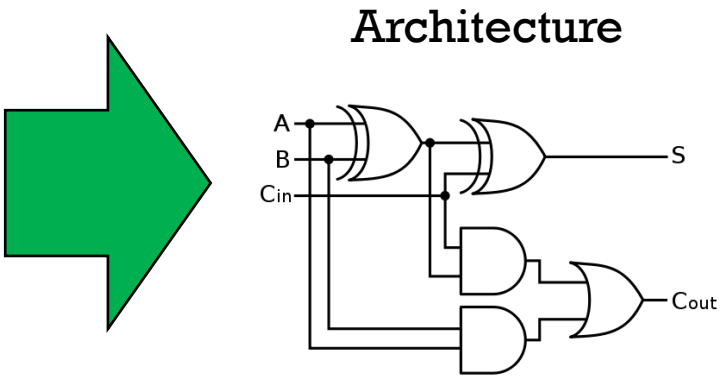


ARCHITECTURE 2

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

entity full_adder is
Port (
   A : in STD_LOGIC;
   B : in STD_LOGIC;
   Cin : in STD_LOGIC;
   S : out STD_LOGIC;
   Cout : out STD_LOGIC
    );
end full_adder;

architecture full_adder of full_adder is
begin

S <= A XOR B XOR Cin ;
Cout <= (A AND B) OR (Cin AND (A XOR B)) ;

end full_adder;
```

A different implementation with fewer gates but a bit slower!





MORE ABOUT SIGNALS

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

entity full_adder is
Port (
   A : in STD_LOGIC;
   B : in STD_LOGIC;
   Cin : in STD_LOGIC;
   S : out STD_LOGIC;
   Cout : out STD_LOGIC
    );
end full_adder;

architecture full_adder of full_adder is
begin

 S <= A XOR B XOR Cin ;
 Cout <= (A AND B) OR (Cin AND (A XOR B)) ;

end full_adder;
```

- I/O ports are signals by default.
- Signal assignments are through the operator "<=".
- Outputs (e.g., S, Cout) are write-only
  - They cannot appear on the right side of "<=
- Inputs (e.g., A, B, Cin) are read-only
  - They cannot appear on the left side of "<="
- Signal assignments are **concurrent**
  - Statements are executed when one or more signals on the right side change their value.
  - The order of these statements does not matter. Reordering S and Cout does not change the outcome.

Q: What if you need more signals than ports?
A: Declare them as part of the architecture.

SIGNAL DECLARATION

Syntax:

```
signal <signal_name> : type;

-- you can add an initial value, but these are only
supported
-- in simulation and not for synthesis
```

Signal type

- **std_logic:** single bit signal
  - '1'-logic 1
  - '0'-logic 0
  - 'X'-unkown
  - 'U'-uninitialized
  - '-'-don't care
  - …

The std_logic types can have nine values, define in:
library ieee; use ieee.std_logic_1164.all;

- **std_logic_vector:** multiple bits bus (vector, array) can be recognized as:
  - N-bit integer or N-bit data serial
  - N independent bits or Boolean values

SIGNAL EXAMPLE 1

Example :

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity ripple_adder is
  Port
      (
       X : in STD_LOGIC_VECTOR (3 downto 0);
       Y : in STD_LOGIC_VECTOR (3 downto 0);
       Carry_in : in STD_LOGIC;
       Sum : out STD_LOGIC_VECTOR (3 downto 0);
       Carry_out : out STD_LOGIC
      );
end ripple_adder;

architecture ripple_adder of ripple_adder is
-- Full Adder VHDL Code Component Decalaration
    component full_adder
      Port (
            A : in STD_LOGIC;
            B : in STD_LOGIC;
            Cin : in STD_LOGIC;
            S : out STD_LOGIC;
            Cout : out STD_LOGIC
            );
    end component;
    signal c1,c2,c3: STD_LOGIC;
begin
-- Port Mapping Full Adder 4 times
  FA1: full_adder port map( A=>X(0), B=>Y(0), Cin=>Carry_in, S=>Sum(0), Cout=>c1);
  FA2: full_adder port map( A=>X(1), B=>Y(1), Cin=>c1, S=>Sum(1), Cout=>c2);
  FA3: full_adder port map( X(2), Y(2), c2, Sum(2), c3);
  FA4: full_adder port map( X(3), Y(3), c3, Sum(3), Carry_out);

end ripple_adder;
```

Signal declaration
Should appear after "architecture", before "begin"

Signals for connections

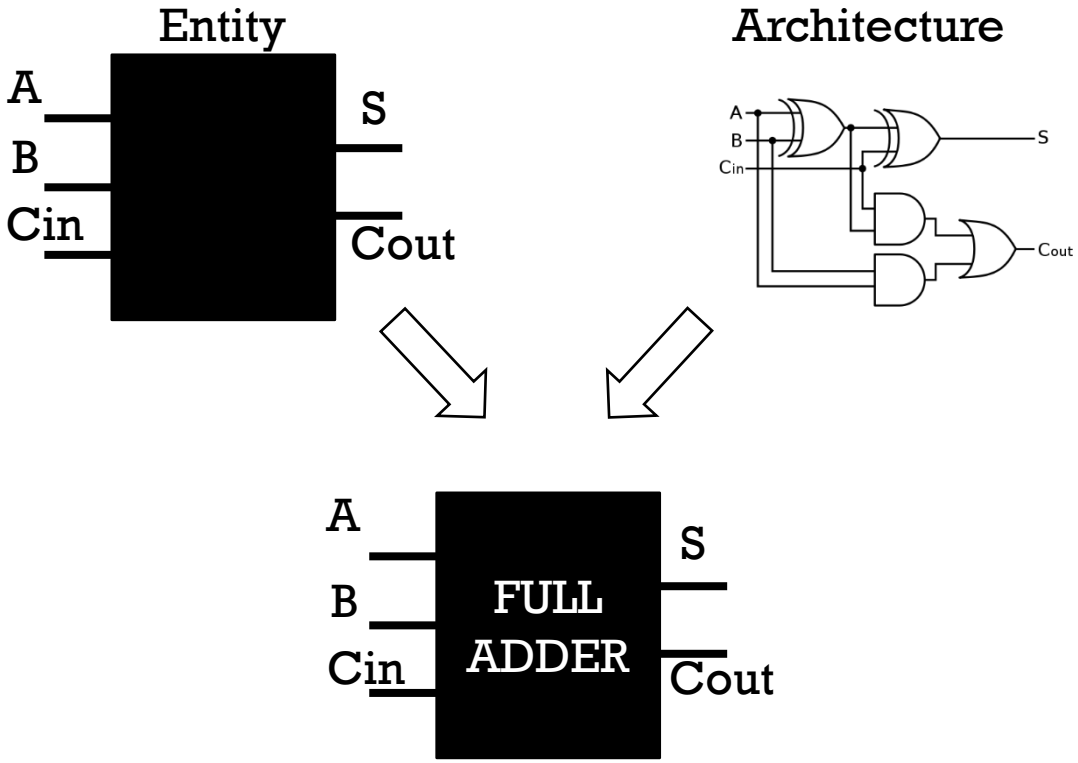
SIGNAL EXAMPLE 2

Example :

```
architecture counter of counter is

signal temp: std_logic_vector(2 downto 0);

begin
   process(Clock, Reset)
   begin
      if Reset='1' then
         temp <= "000";
      elsif(rising_edge(Clock)) then
         if Enable='1' then
           if temp="111" then
              temp<="000";
           else
              temp <= temp + 1;
           end if;
         end if;
     end if;
   end process;

   Output <= temp;

end counter;
```

Signal declaration
Should appear after "architecture", before "begin"

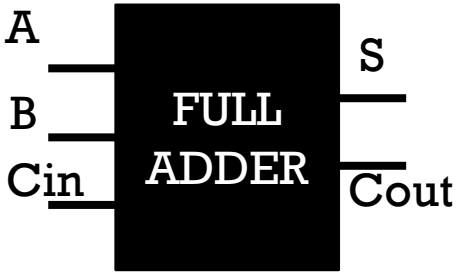
Signals for internal process and storing



we wrap the black box and we get…

COMPONENT

A **component** is a wrapped function unit with I/O.

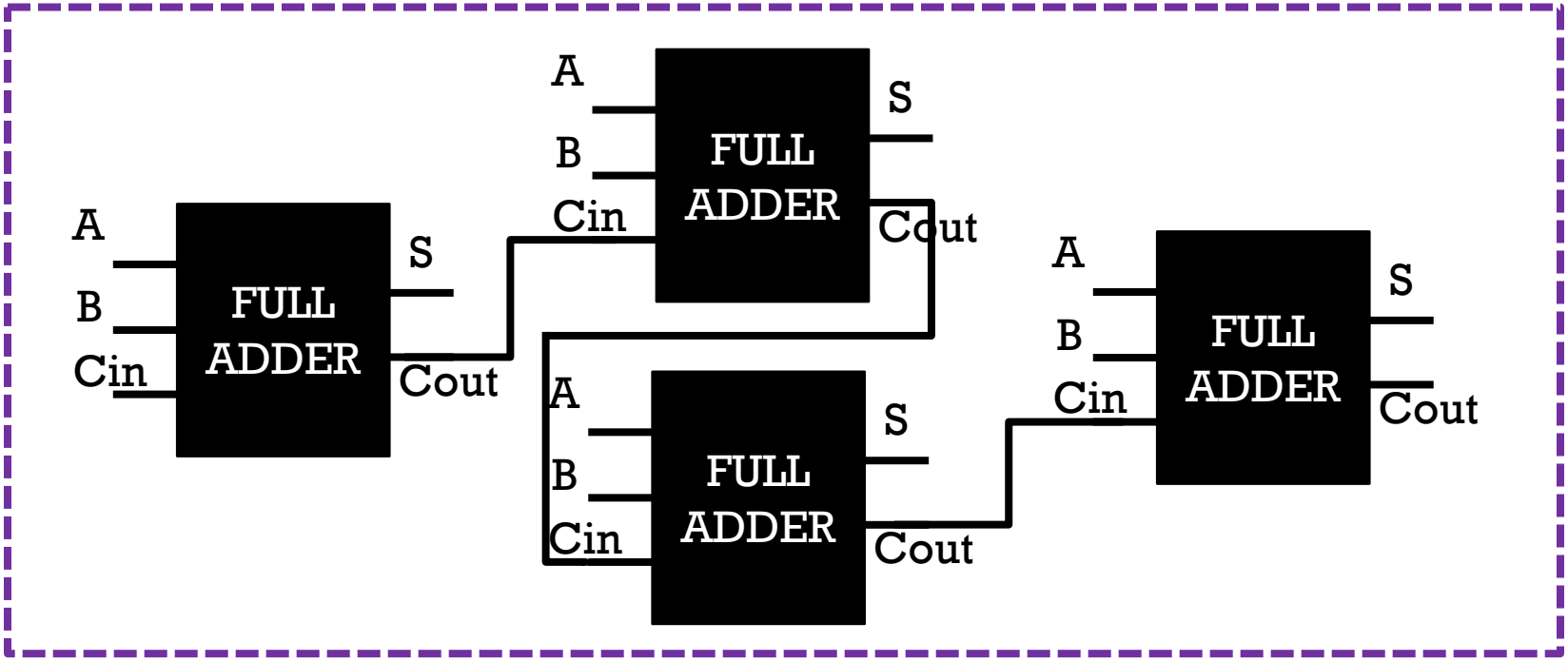


What can we do with components?

Connect them together!

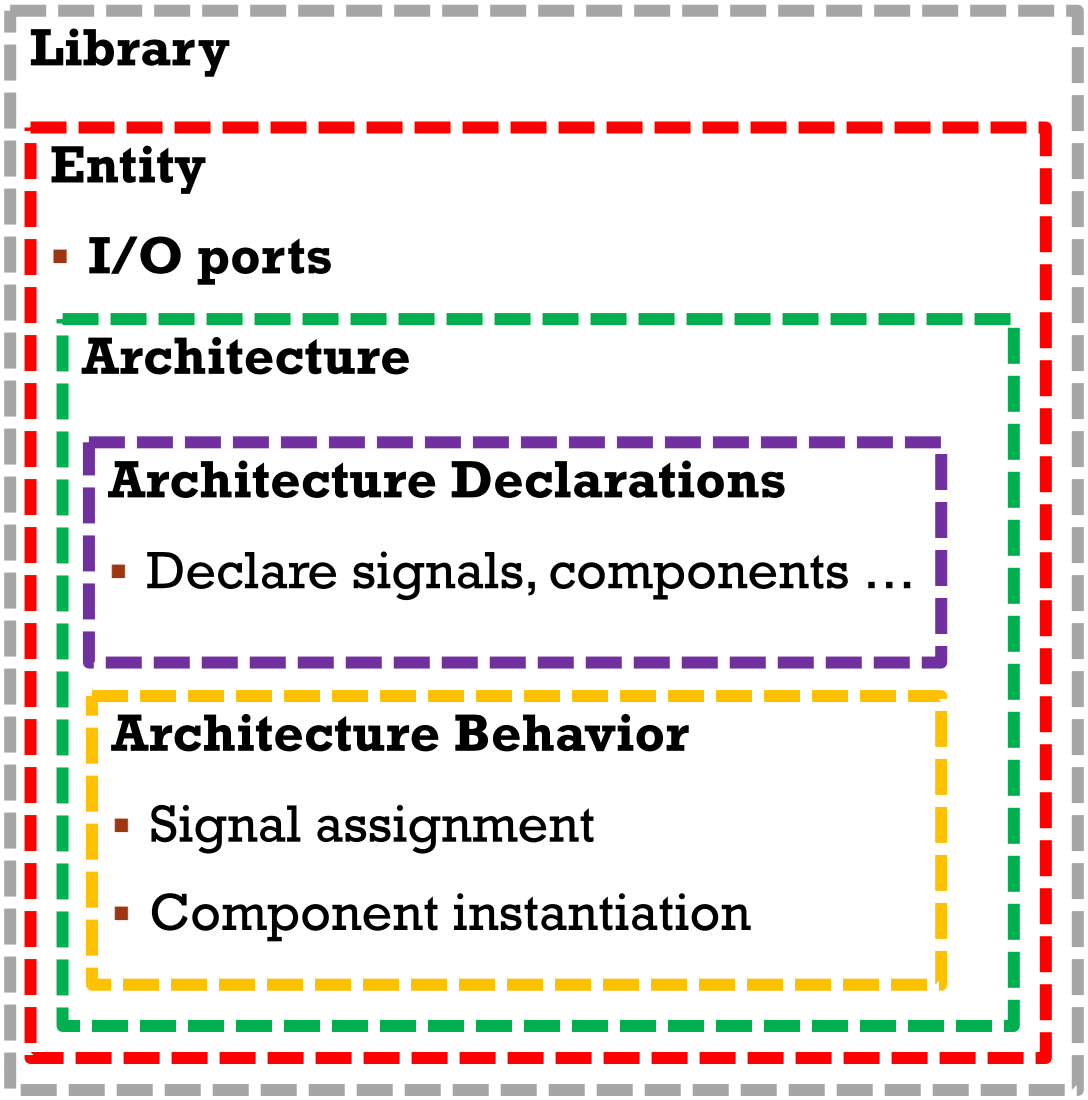
CONNECT THEM TOGETHER

Components are used for structural and hierarchical design.



4-bit carry-ripple adder

VHDL DESIGN STRUCTURE



ARCHITECTURE DECLARATION

Example：

Syntax:

```
architecure <architecture_name> of
<entity_name> is
[
  component declarations (detail)
  function declarations (detail)
  signal declarations (detail)
  constant declarations (detail)
  variable declarations (detail)
  type declarations (detail)
  ...
  -- there are others, but these are what
  you'll need
  -- for this course

]
```

Only major cases are covered at this moment, please check full VHDL syntax after class.

COMPONENT DECLARATION

- The **component name** refers to either the name of an entity defined in a library or an entity explicitly defined in the VHDL file
- The **component declaration** can be done in two ways:
- In the architecture body, after keyword "architecture" and before keyword "begin";
- In the package declaration (not covered in this course)

COMPONENT EXAMPLE

Example :

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity ripple_adder is
  Port
      (
       X : in STD_LOGIC_VECTOR (3 downto 0);
       Y : in STD_LOGIC_VECTOR (3 downto 0);
       Carry_in : in STD_LOGIC;
       Sum : out STD_LOGIC_VECTOR (3 downto 0);
       Carry_out : out STD_LOGIC
      );
end ripple_adder;
architecture ripple_adder of ripple_adder is
-- Full Adder VHDL Code Component Decalaration
    component full_adder
      Port (
            A : in STD_LOGIC;
            B : in STD_LOGIC;
            Cin : in STD_LOGIC;
            S : out STD_LOGIC;
            Cout : out STD_LOGIC
            );
   end component;
-- Intermediate Carry declaration
    signal c1, c2, c3: STD_LOGIC;

begin
-- Port Mapping Full Adder 4 times
  FA1: full_adder port map( A=>X(0), B=>Y(0), Cin=>Carry_in, S=>Sum(0), Cout=>c1);
  FA2: full_adder port map( A=>X(1), B=>Y(1), Cin=>c1, S=>Sum(1), Cout=>c2);
  FA3: full_adder port map( X(2), Y(2), c2, Sum(2), c3);
  FA4: full_adder port map( X(3), Y(3), c3, Sum(3), Carry_out);

end ripple_adder;
```

Component is used for hierarchical design. To use it, there must be a design of full_adder (entity & architecture) defined before.

Component can be directly used with proper input/output.

COMPONENT EXAMPLE

Example :

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity ripple_adder is
  Port
      (
       X : in STD_LOGIC_VECTOR (3 downto 0);
       Y : in STD_LOGIC_VECTOR (3 downto 0);
       Carry_in : in STD_LOGIC;
       Sum : out STD_LOGIC_VECTOR (3 downto 0);
       Carry_out : out STD_LOGIC
      );
end ripple_adder;
architecture ripple_adder of ripple_adder is
-- Full Adder VHDL Code Component Decalaration
    component full_adder
      Port (
            A : in STD_LOGIC;
            B : in STD_LOGIC;
            Cin : in STD_LOGIC;
            S : out STD_LOGIC;
            Cout : out STD_LOGIC
            );
   end component;
-- Intermediate Carry declaration
    signal c1,c2,c3: STD_LOGIC;

begin
-- Port Mapping Full Adder 4 times
FA1: full_adder port map( A=>X(0), B=>Y(0), Cin=>Carry_in, S=>Sum(0), Cout=>c1);
FA2: full_adder port map( A=>X(1), B=>Y(1), Cin=>c1, S=>Sum(1), Cout=>c2);
FA3: full_adder port map( X(2), Y(2), c2, Sum(2), c3);
FA4: full_adder port map( X(3), Y(3), c3, Sum(3), Carry_out);

end ripple_adder;
```

Explicit map

Default map

Default map order is the same as Component I/O definition order.

REFERENCES

A plumber game
http://www.freeonlinegames.com/game/plumber

More information on Architecture Declarations Visit:
http://webdocs.cs.ualberta.ca/~amaral/courses/329/labs/VHDL_Reference.html#function_dec

Books:

- Bebop to the Boolean Algebra (Clive Maxfield, Published by Newnes)
- The Design Warrior's Guide to FPGAs (Clive Maxfield, Published by Newnes)
- Fundamentals of Digital Logic with VHDL(Stephen Brown, Zvonko Vranesic, Published by McGraw Hill)

HOMEWORK

- Get your Zybo board from Mr. Thomas Lum in Evans 127
- Install VM
- Install Vivado

SOFTWARE SETUP

- First, download VM player from https://udeploy.udel.edu/
  - Get Vmware Workstation Pro 15 for Windows or Vmware Fusion 11 for Mac. Older versions will have compatibility issues.

- Then, download Vivado VM from https://drive.google.com/file/d/1alv_qIQMW_c-3vbbGWuvI_nDFHcbYCu0/view?usp=sharing

- If you have a Windows based system you will need to install Digilent Adept 2: https://mautic.digilentinc.com/adept-for-windows-system-landing-page (You will need to supply an email address)

HOMEWORK

Familiar yourself with VHDL!

- A good tutorial can be found at https://www.seas.upenn.edu/~ese171/vhdl/vhdl_primer.html
- Or you can use Google to find other resources!

NEXT LECTURE

- A tutorial about Vivado