Introduction to VHDL Programming & Basics of CPLD

॥ सा विद्या या विमुक्तये ॥

ಭಾರತೀಯ ತಂತ್ರ ಜ್ಞಾ ನ ಸಂಸ್ಥೆ ಧಾರವಾಡ
भारतीय प्रौद्योगिकी संस्थान धारवाड़
Indian Institute of Technology Dharwad

Presented by
Mayur Shivamurthy
Research Scholar
Dept. of EE , IIT Dharwad

Hardware Description Language

What is the need for HDL?

- Advances in IC technology have enabled manufactures to develop complex digital electronic circuits.
- Complex digital circuit designs require more time for development, synthesis, simulation and debugging
- HDL solve the problem by allowing hardware software co-design and model the complex circuit at subsystem levels.
- A **synthesizable** HDL code results to an equivalent hardware.
- It is found to be an excellent language for different programmable devices like FPGAs and CPLDs.

Hardware Description Language

How it is different from C and other programming languages?

HDL	C
Can handle sequential and concurrent instructions	Can handle only sequential instructions
Knowledge of the hardware circuits	written with pure logical or algorithmic thinking.
Limited memory and other logic elements in programmable devices must be taken into account	Resource usages are not critical

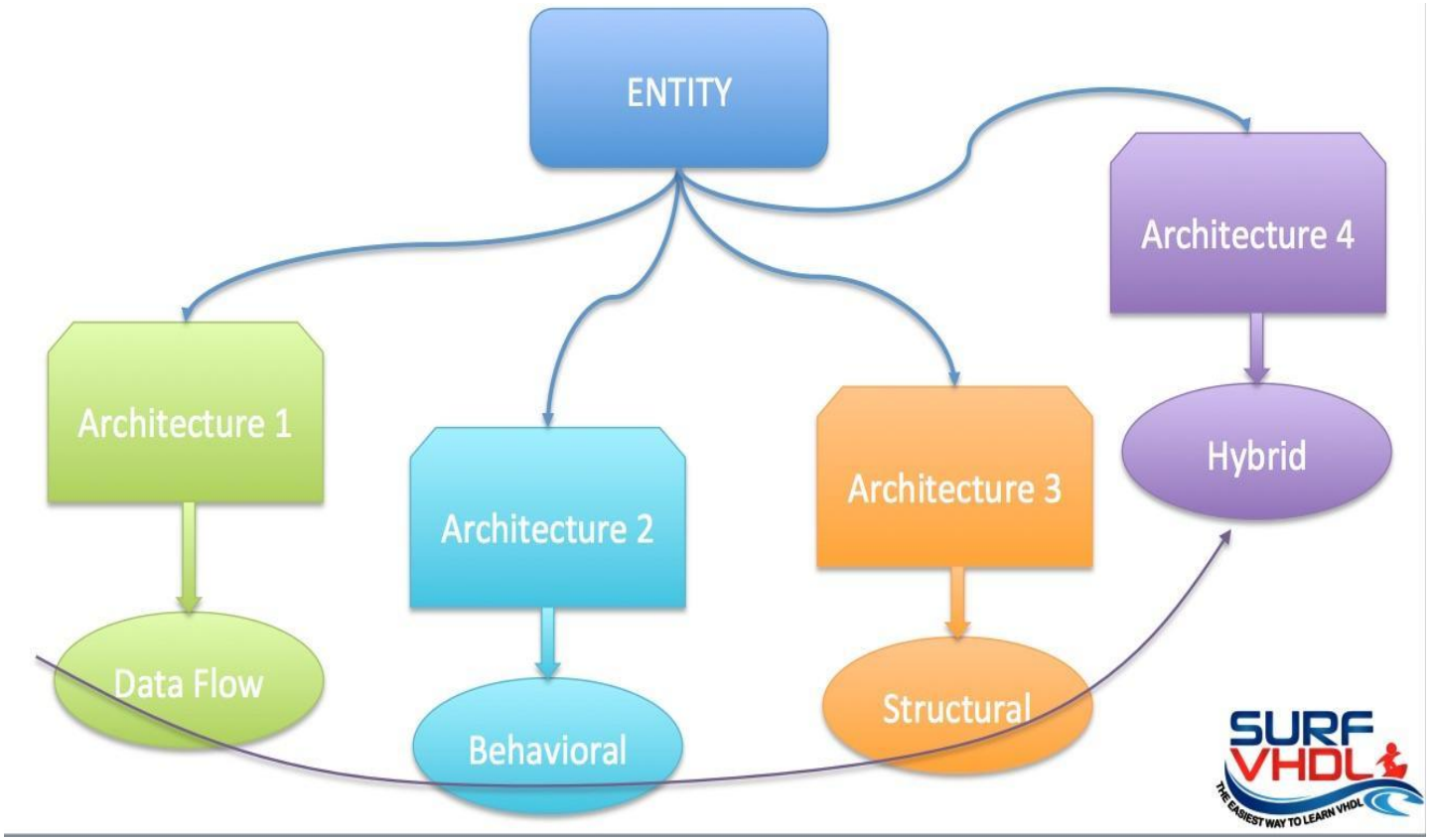
Classification of HDL

VHDL (VHSIC-HDL)	Verilog
It is strongly typed	It is weakly typed
High verbosity	Low verbosity
Case insensitive	Case sensitive
Non C like	More C like
Upto gate level implementation only	Transistor level implementation is possible

Features of VHDL

- Supports concurrent and sequential execution of statements.
- It possesses IEEE standards.
- Like C, VHDL also supports different data types, conditional, relational, logical, arithmetic operators.
- Supports 3 different modelling styles
  1. Dataflow
  2. Behavioural
  3. Structural

Different Modelling Style



https://surf-vhdl.com/wp/wp-content/uploads/2015/07/CodingStyle1.jpg

Data types : Bits and Vectors in Port

port (a : i**n std_logic**; -- signal comes in to port a from outside
b : **out std_logic**; -- signal is sent out to the port b
c : **inout std_logic**; -- bidirectional port
x : in std_logic_vector(7 **downto** 0); -- 8-bit input vector
y : out std_logic_vector(0 **downto** 7) --
);

x[7] -> MSB bit, x[0]-> LSB bit
y[0] -> MSB bit, y[7]-> LSB bit

'U' : uninitialized **'X'** : unknown

'0' : logic 0. '1' : logic 1 **'Z'** : High Impedance.

Package to be added : `ieee.std_logic_1164.all`

Signals Declaration and assignment

Signals are declared without direction.

Example:

```
signal s1, s2 : std_logic;
signal X, Y : std_logic_vector (31 downto 0);
```

Signal Assignment

```
signal resetSR : std_logic_vector(3 downto 0);

resetSR <= "1111";    resetSR <= temp(2 downto 0) & '0';
```

Other Datatypes

SYNTHESIZABLE	*NON SYNTHESIZABLE*
integer,	real
natural,	time
positive,	
integer_vector	
character,	
string	

Package to be added : `ieee.std_logic_1164.all`

Logical Operators

not *highest precedence*

and

or

nand

nor

xor

xnor *lowest precedence*

ARITHMETIC:

- + addition
- - subtraction
- * multiplication
- / division
- ABS absolute value
- MOD modulus
- REM remainder
- ** exponent

Package :
use ieee.NUMERIC_STD.all;

Comparison Operators:

- = equal to
- /= not equal to
- < less than
- > greater than
- <= less than or equal to
- >= greater than or equal to

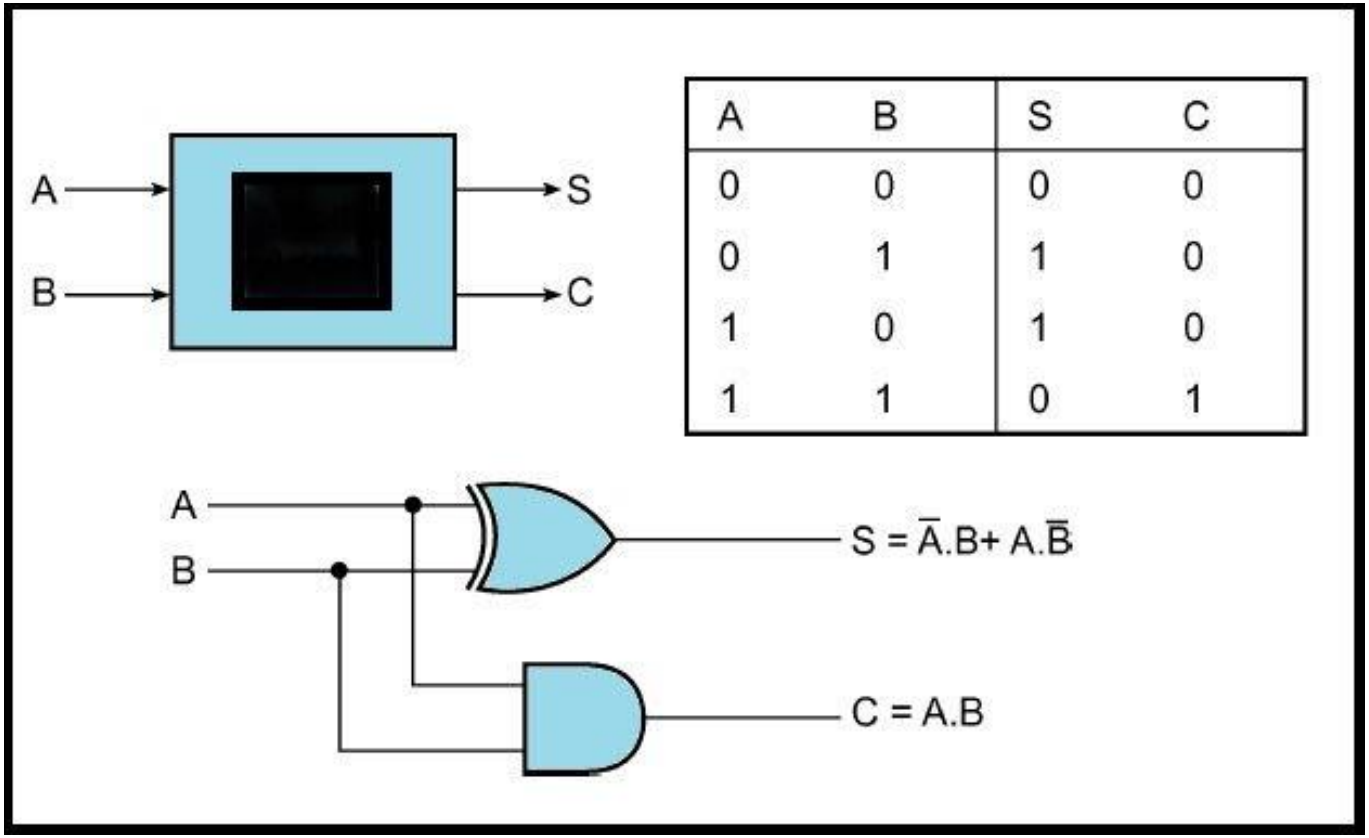
Branching Statements:

```
if condition_1 then
   sequential statements
elsif condition2 then
   sequential statements
else
   sequential statements
end if;
```

Branching Statements:

```
case expression is
    when choice =>
        sequential statements
    when choice =>
        sequential statements
end case;
```

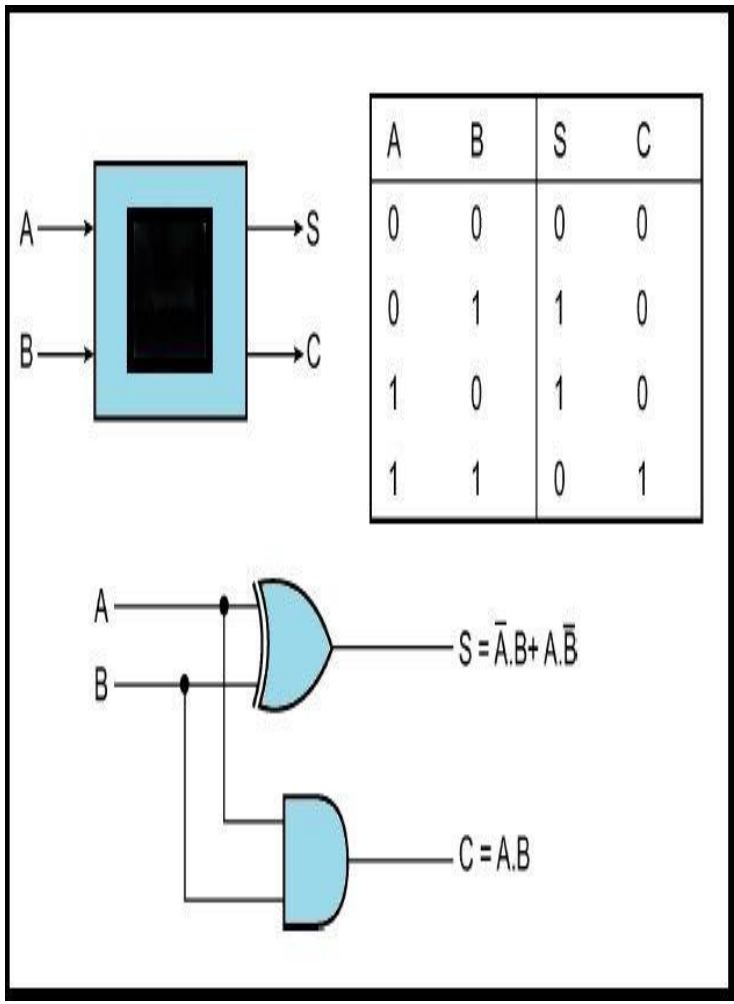
Circuit 1



A	B	S	C
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

VHDL Dataflow Modelling Style

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

entity circuit_1 is
  port (A, B: in
std_logic;
    S, C: out std_logic);
  end circuit_1;

architecture dataflow of
circuit_1 is
  Begin
  S <= A xor B;
  C <= a and b;
```



https://buzztech.in/vhdl-modelling-styles-behavioral-dataflow-structural/

VHDL Dataflow Modelling Style

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

entity circuit_1 is
  port (A, B: in std_logic;
    S, C: out std_logic);
  end circuit_1;

architecture dataflow of circuit_1 is
 Begin
  S <= A xor B;
  C <= a and b;
end dataflow;
```


https://buzztech.in/vhdl-modelling-styles-behavioral-dataflow-structural/

VHDL Dataflow Modelling Style

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

entity circuit_1 is
  port (A, B: in std_logic;
    S, C: out std_logic);
  end circuit_1;

architecture dataflow of circuit_1 is
 Begin
 S <= A xor B;
 C <= a and b;
end dataflow;
```


https://buzztech.in/vhdl-modelling-styles-behavioral-dataflow-structural/

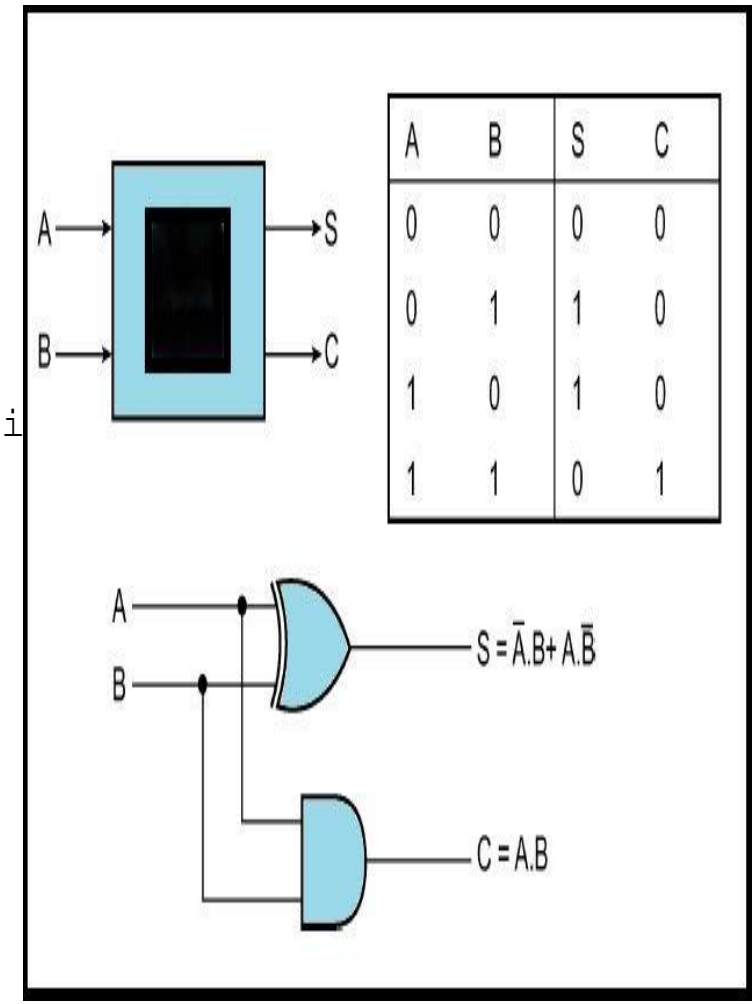
VHDL Dataflow Modelling Style

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

entity circuit_1 is
  port (A, B: in std_logic;
    S, C: out std_logic);
  end circuit_1;

architecture dataflow of circuit_1 is
 Begin
 S <= A xor B;
 C <= a and b;
end dataflow;
```


https://buzztech.in/vhdl-modelling-styles-behavioral-dataflow-structural/

VHDL Behavioral Modelling Style

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

entity circuit_1 is
  port (A, B: in std_logic;
    S, C: out std_logic);
  end circuit_1;

architecture behavior of circuit_1 i
  Begin
    c1: process (A,B)
    Begin
      if A = '1' then
        S <= not B;
        C <= B;
      Else
        S <= B;
        C <= '0';
      end if;
    end process c1;
end behavior;
```

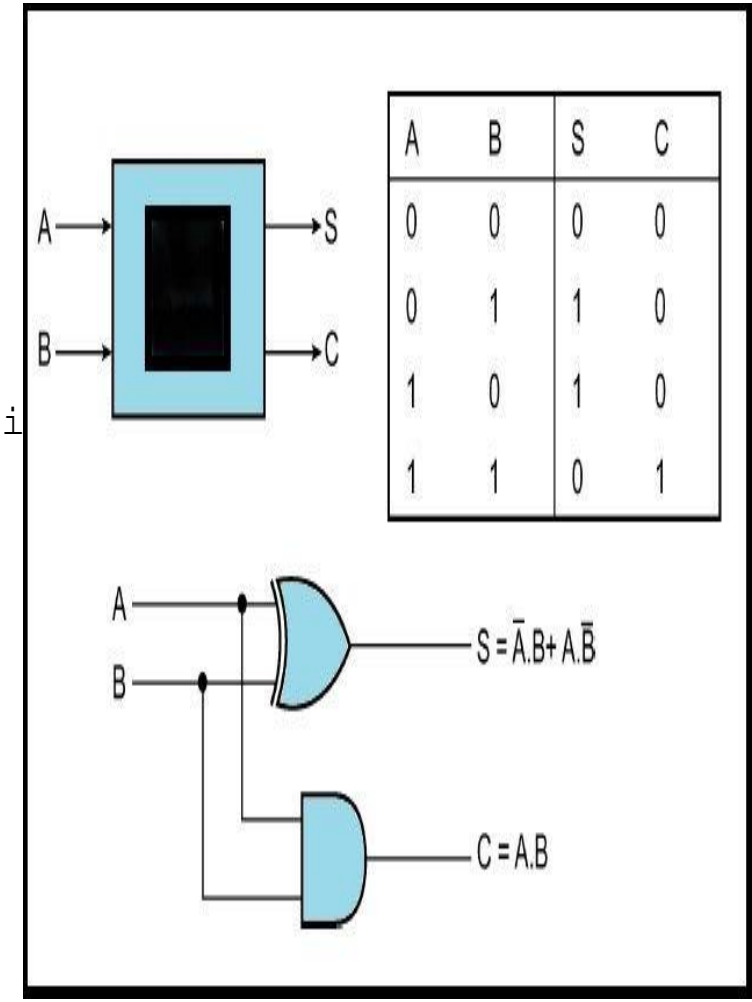


A	B	S	C
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

https://buzztech.in/vhdl-modelling-styles-behavioral-dataflow-structural/

VHDL Behavioral Modelling Style

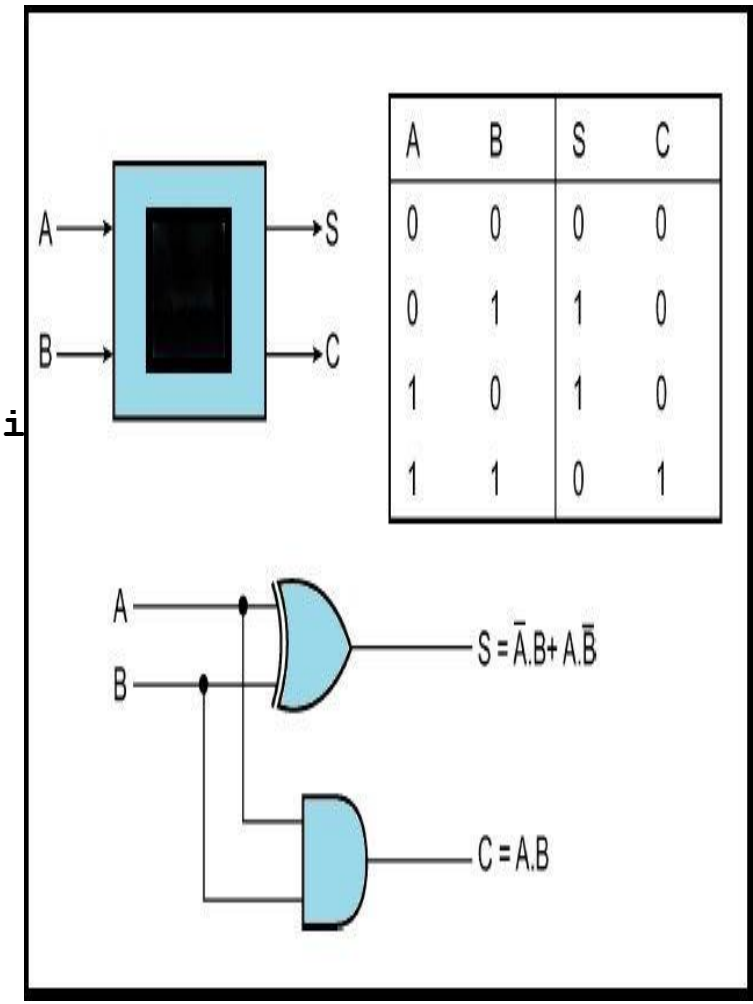
```
library ieee;
use ieee.std_logic_1164.all;

entity circuit_1 is
  port (A, B: in std_logic;
    S, C: out std_logic);
  end circuit_1;

architecture behavior of circuit_1 i
  Begin
    c1: process (A,B)
    Begin
      if A = '1' then
        S <= not B;
        C <= B;
      Else
        S <= B;
        C <= '0';
      end if;
    end process c1;
end behavior;
```



https://buzztech.in/vhdl-modelling-styles-behavioral-dataflow-structural/

VHDL Behavioral Modelling Style

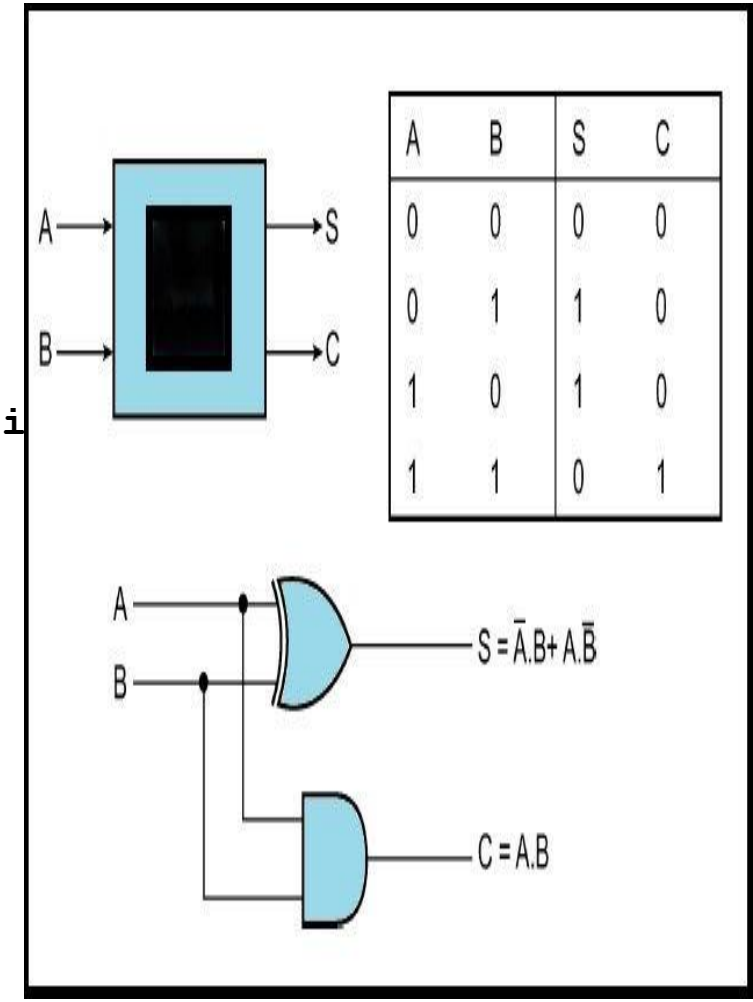
```
library ieee;
use ieee.std_logic_1164.all;

entity circuit_1 is
  port (A, B: in std_logic;
    S, C: out std_logic);
  end circuit_1;

architecture behavior of circuit_1 i
  Begin
    c1: process (A,B)
    Begin
      if A = '1' then
        S <= not B;
        C <= B;
      Else
        S <= B;
        C <= '0';
      end if;
    end process c1;
end behavior;
```



https://buzztech.in/vhdl-modelling-styles-behavioral-dataflow-structural/

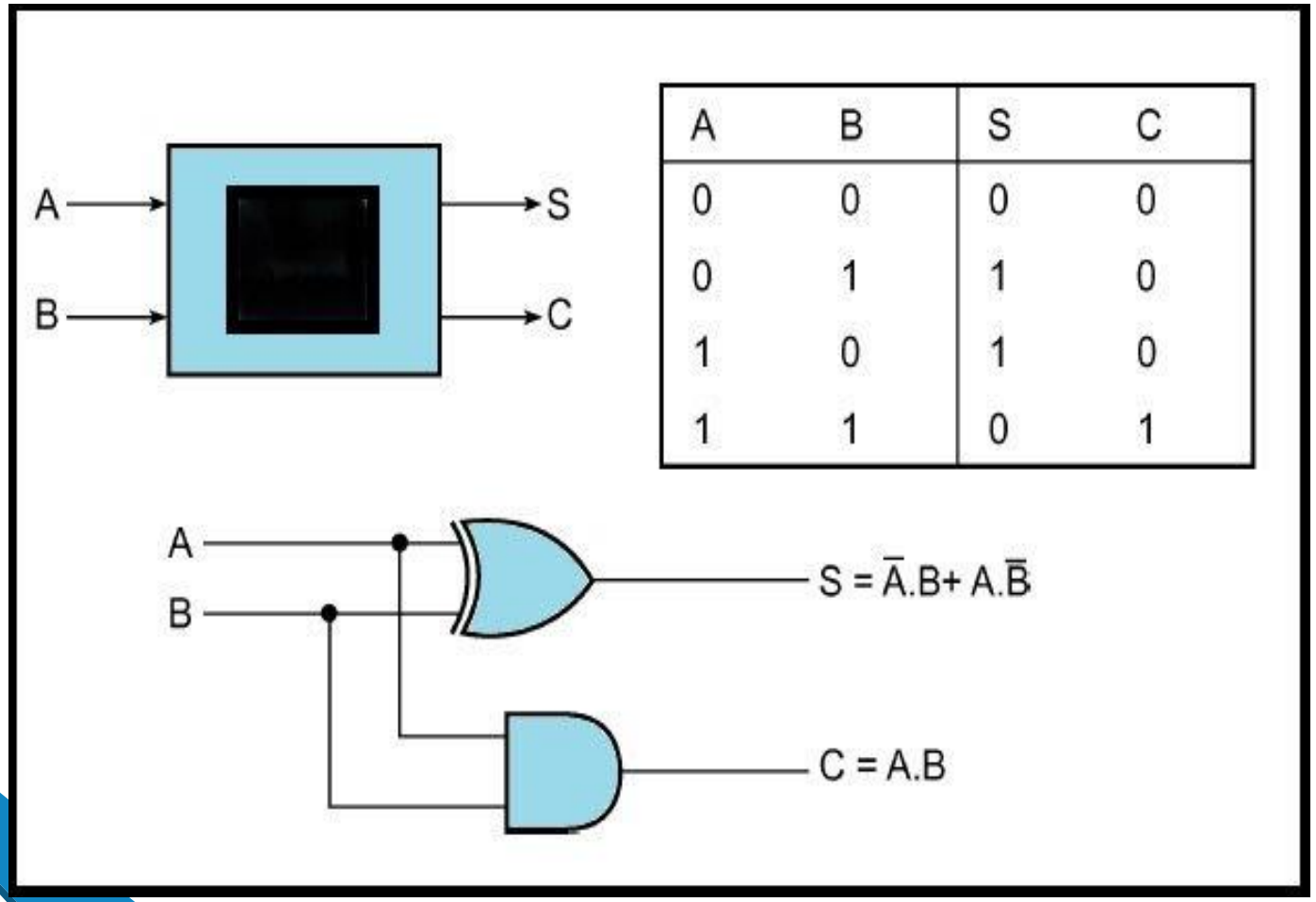
VHDL Behavioral Modelling Style

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

entity circuit_1 is
  port (A, B: in std_logic;
    S, C: out std_logic);
  end circuit_1;

architecture behavior of circuit_1 i
  Begin
    c1: process (A,B)
    Begin
      if A = '1' then
        S <= not B;
        C <= B;
      Else
        S <= B;
        C <= '0';
      end if;
    end process c1;
end behavior;
```



https://buzztech.in/vhdl-modelling-styles-behavioral-dataflow-structural/

VHDL Strucural Modelling Style



A	B	S	C
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

--Functionality for XOR gate in data flow modelling style

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

entity xor_gate is
  port (i1, i2: in std_logic;
   o1: out std_logic);
  end circuit_1;

architecture dataflow of xor_gate is
  Begin
  o1 <= i1 xor i2;
 end dataflow;
```

--Functionality for AND gate in data flow modelling style

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

entity and_gate is
  port (i3, i4: in std_logic;
   o2: out std_logic);
  end circuit_1;

architecture dataflow of and_gate is
  Begin
  o2 <= i3 and i4;
 end dataflow;
```

VHDL Structural Modelling Style

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

entity circuit_1 is                    -- Entity declaration for circuit_1
port (A, B: in std_logic;
   S, C: out std_logic);
end circuit_1;

architecture structure of circuit_1 is  -- Architecture body for circuit_1
begin




end structure;
```


https://buzztech.in/vhdl-modelling-styles-behavioral-dataflow-structural/

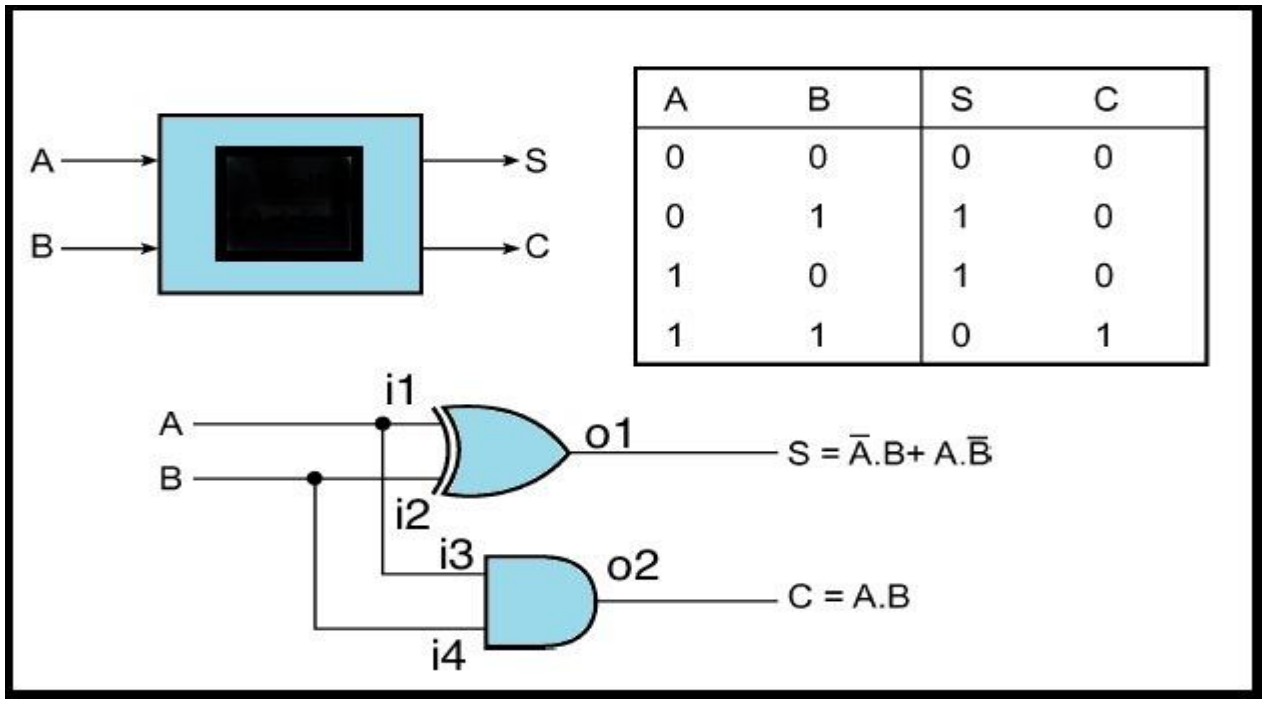
VHDL Structural Modelling Style

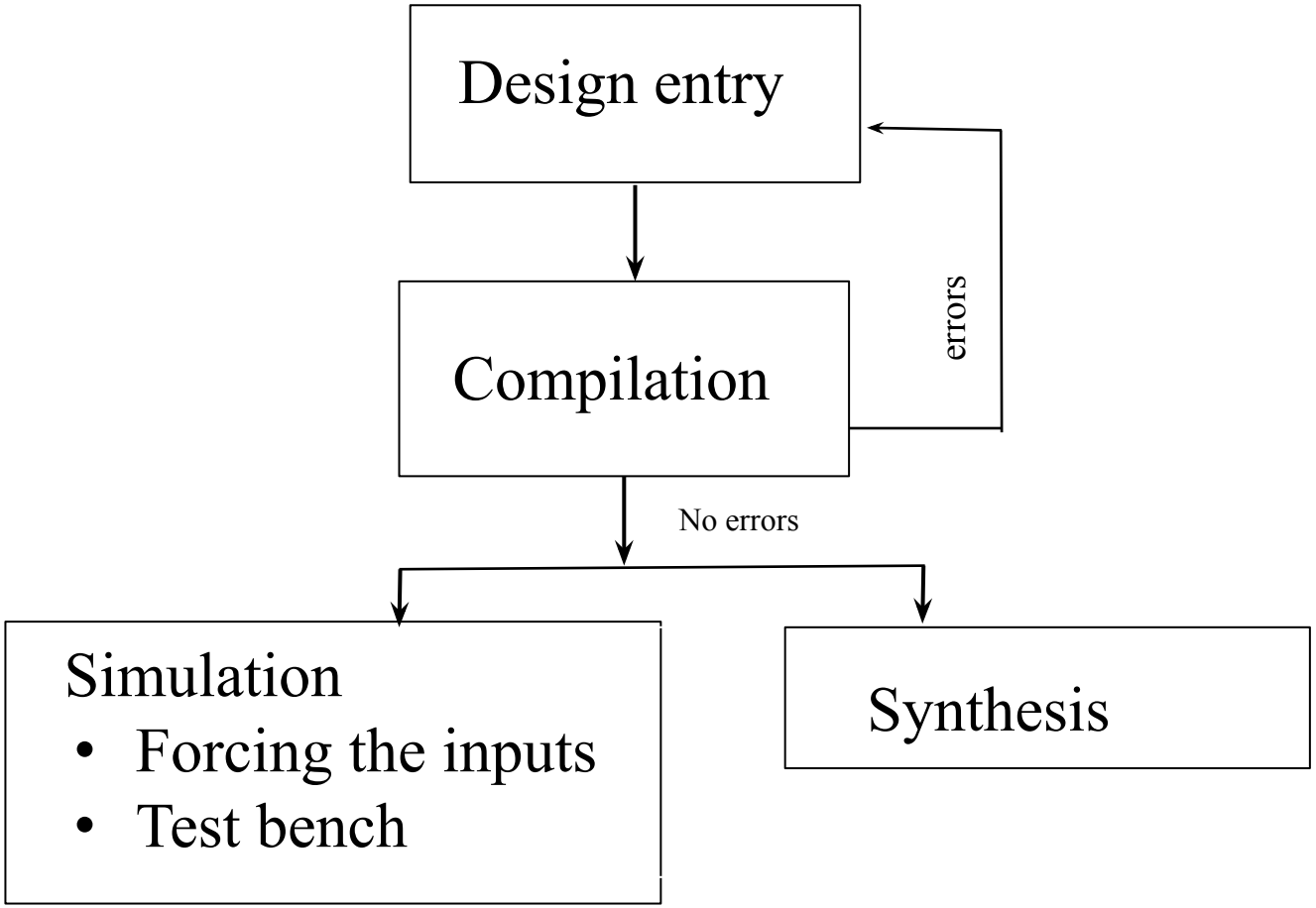
```
library ieee;
use ieee.std_logic_1164.all;

entity circuit_1 is                          -- Entity declaration for circuit_1
port (A, B: in std_logic;
    S, C: out std_logic);
end circuit_1;

architecture structure of circuit_1 is  -- Architecture body for circuit_1
component xor_gate                          -- xor component declaration
   port (i1, i2: in std_logic;
     o1: out std_logic);
 end component;

 component and_gate                         -- and component declaration
   port (i3, i4: in std_logic;
     o2: out std_logic);
 end component;
```


https://buzztech.in/vhdl-modelling-styles-behavioral-dataflow-structural/

VHDL Structural Modelling Style

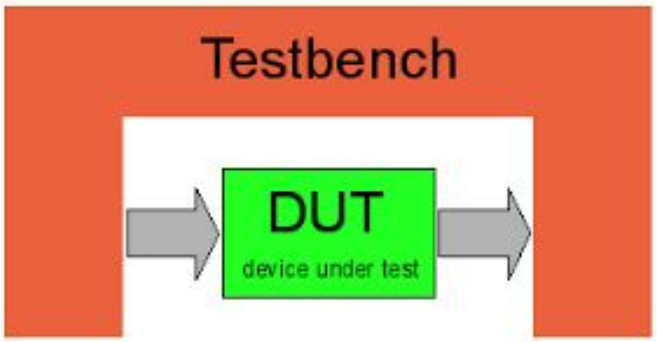
```
begin
   u1: xor_gate port map (i1 => A, i2 => B, o1 => S);
   u2: and_gate port map (i3 => a, i4 => b, o2 => C);
-- We can also use Positional Association
--      => u1: xor_gate port map (a, b, s);
--      => u2: and_gate port map (a, b, c);
end structure;
```




Design entry
Compilation
errors
No errors
Simulation
• Forcing the inputs
• Test bench
Synthesis

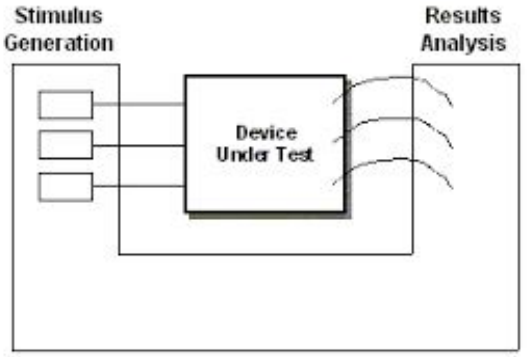
Test Bench

- Used for verification of the design
- HDL code with no port list and non synthesizable constructs.
- The main objectives of Test Bench is to:
- 1.Instantiate the design under test (DUT)
- 2.Generate stimulus waveforms for DUT
- 3.Generate reference outputs and compare them with the outputs of DUT
- 4.Automatically provide a pass or fail indication



VHDL Test Bench for Circuit 1

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

ENTITY circuit_1_tb IS
END circuit_1_tb;

ARCHITECTURE circuit_1_tb OF circuit_1_tb IS

COMPONENT circuit_1
    PORT(
         A : IN  std_logic;
         B : IN  std_logic;
         S : OUT  std_logic;
         C : OUT  std_logic
        );
    END COMPONENT;
```



-- input test signals and resulting output signals from DUT to be declared

```
signal a_tb : std_logic := '0';
signal b_tb :std_logic  := '0';
signal s_tb :std_logic  ;
signal c_tb :std_logic  ;
```

```
BEGIN

uut: circuit_1 PORT MAP
       (
          A => a_tb,
          B => b_tb,          -- instantaite the DUT with
          S => s_tb,          -- structual modelling style
          C => c_tb
        );

Process                        --apply the stimulus with
  Begin                        --behavioural modelling style
  a_tb <= '0';  b_tb <= '0';
  wait for 100 ps;
  a_tb <= '0';  b_tb <= '1';
      wait for 200 ps;
 a_tb<= '1';   b_tb <= '0';
      wait for 500 ps;
 a_tb <= '1';   b_tb <= '1';
      wait;
end process;

END circuit_1_tb ;
```

References:

VHDL Primer by J. Bhaskar.

HDL with Digital design by Botros.

Introduction to CPLDs

References : http://studytronics.weebly.com/programmable-logic-devices.html

Programmable Logic Devices

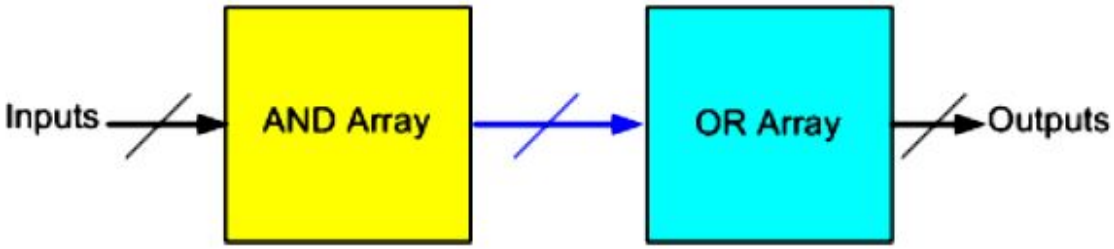
An IC that contains large numbers of gates, flip-flops, etc. that can be configured by the user to perform different functions.

The internal logic gates and/or connections of PLDs can be changed/configured by a programming process.

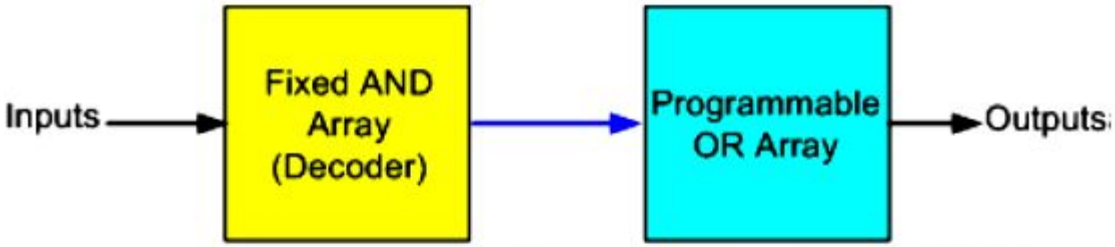
Programming the device involves blowing those fuses along the paths to obtain particular configuration of the desired logic function.

PLDs are typically built with an array of AND gates (AND-array) and an array of OR gates (OR-array).

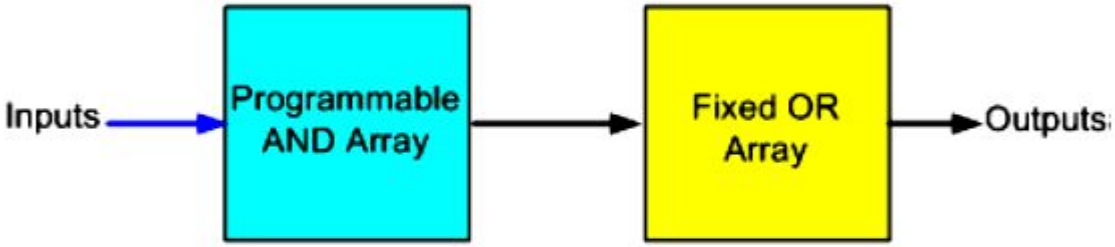
PLDs are mainly classified as (a) PROM (b) PLA (c) PAL



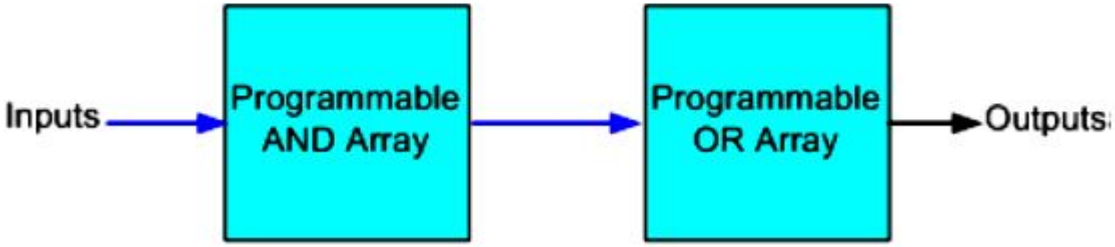
PROM, PAL, PLA



(a) Programmable Read Only Memory (PROM)

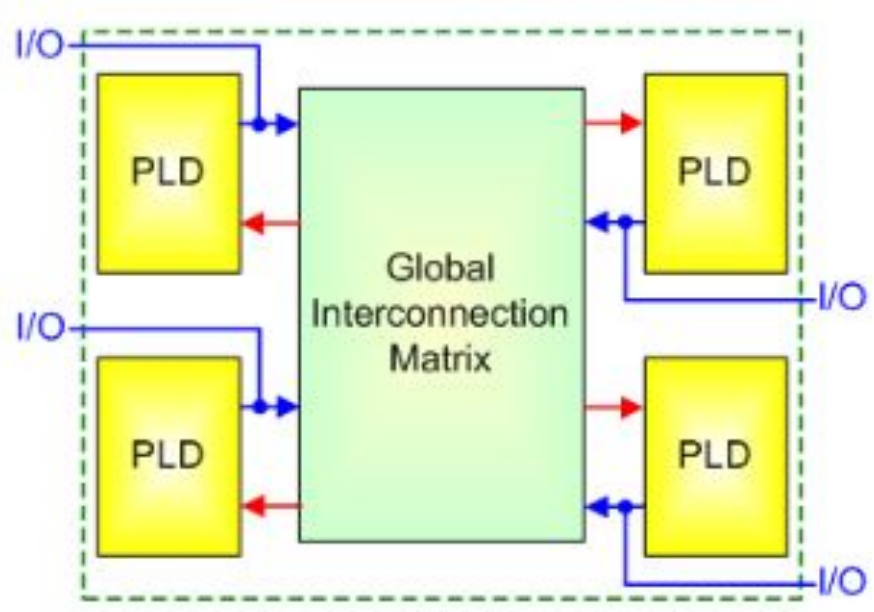


(b) Programmable Array Logic (PAL) Device



(c) Programmable Logic Array (PLA) Device

CPLDs

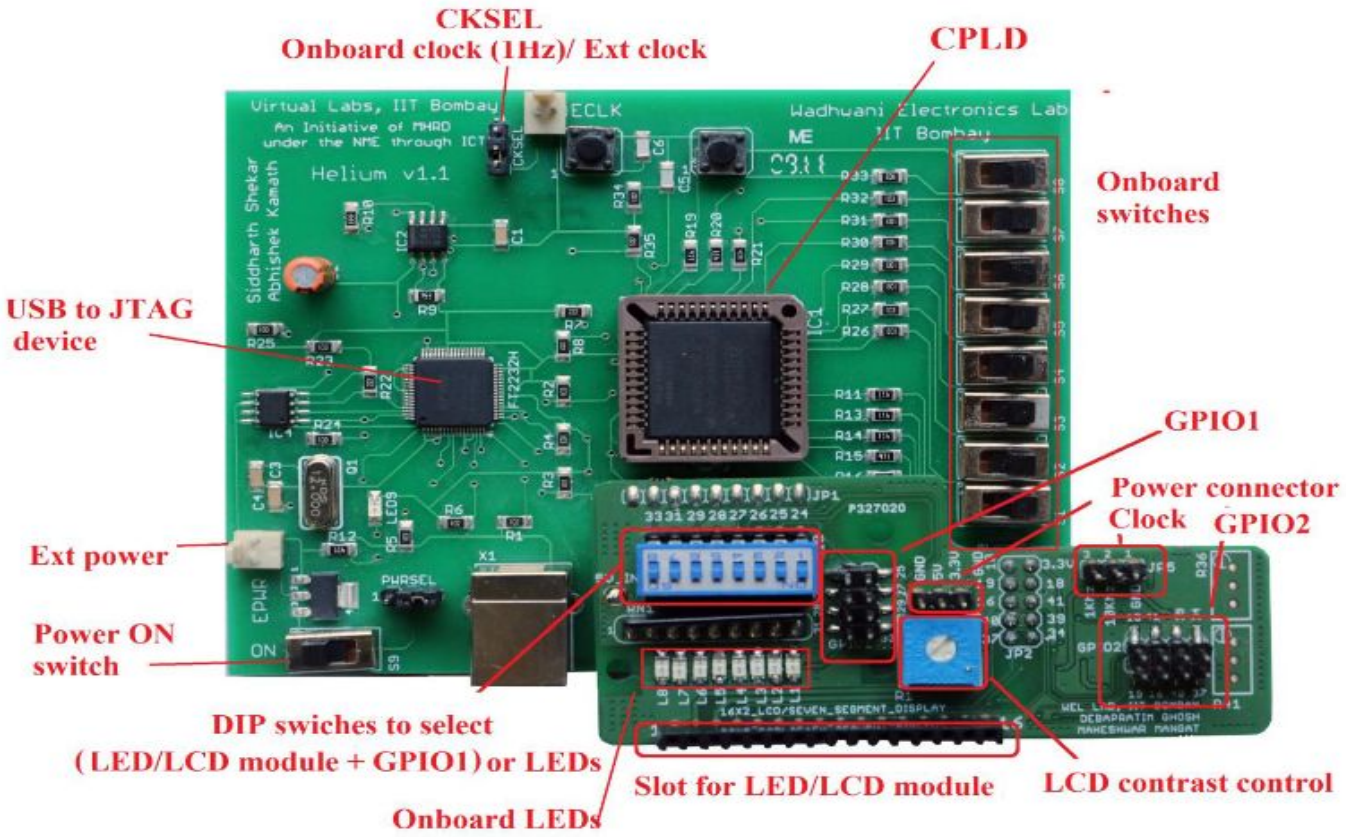


A CPLD contains a bunch of PLD blocks whose inputs and outputs are connected together by a global interconnection matrix.

CPLD has two levels of programmability:

1. Each PLD block can be programmed
2. The interconnections between the PLDs can be programmed.

Board details:



1. Based on Altera MAX 3000 architecture.
2. Powered and Programmed through USB.
3. 8 inputs (switches) and 8 outputs(LEDs) onboard.
4. 8 user configurable GPIOs available onboard.
5. The 8 LEDs output pins can be made available as GPIO1 pins if the DIP switches are turned OFF.
6. Onboard clock: 1Hz, 1kHz, and 10kHz available.
7. Provision for connecting external clock signal.

INPUT AND OUTPUT PINS DETAILS OF MAX 3000A

Inputs	Pin No.
SW1	4
SW2	5
SW3	6
SW4	8
SW5	9
SW6	11
SW7	12
SW8	14

Outputs	Pin No.
LED1	24
LED 2	25
LED 3	26
LED 4	27
LED 5	28
LED 6	29
LED 7	31
LED 8	33

Configuring Helium Board using JTAG

1. Open terminal -> jtag

2. cable ft2232 vid=0x0403 pid=0x6010

3. detect

4.svf “file path.svf”