```
y1 <= '1';
y0 <= '0'; y2 <= '0'; y3 <= '0';
elsif (A = '1' and B = '0') then
    y2 <= '1';
y0 <= '0'; y1 <= '0'; y3 <= '0';
elsif (A = '1' and B = '1') then
    y3 <= '1';
    y0 <= '0'; y1 <= '0'; y2 <= '0';
end if;
end process P1;
end Behavioral;
```

8:1 mux Using Case Statement

```
library IEEE;
use IEEE.std_logic_vector(7 downto 0);
EN: in STD_LOGIC;
SEL: in STD_LOGIC_Vector(2 down 0);
y: out STD_LOGIC);
end mux 8:1;
Architecture
use IEEE.Std_logic_1164.all;
use IEEE.Std_logic_arith.all;
use IEEE.Std_logic_unsigned.all;

entity mux 8:1 is
    Port (D: in STD_LOGIC_Vector
        (7 downto 0);
    EN: in STD_LOGIC;
    SEL: in STD_LOGIC_Vector
        (2 down to 0);
```

y : out STD_LOGIC);

end mux 8:1;

Architecture ~~mux8bitea~~ Behavioral of mux 8:1 is

Begin

Process (EN, SEL, D)

Begin

if (EN = '1') then

y <= '0';

else

Case SEL is

when "000" => y <= D(0);

when "001" => y <= D(1);

when "010" => y <= D(2);

when "011" => y <= D(3);

when "100" => y <= D(4);

when "101" => y <= D(5);

when "110" => y <= D(6);

when other => y <= D(7);

end Case;

end Process;

end behavioral;

2:1 MUX

- Architecture Behavioral of mux 2:1 is

Begin

Process (a, b, s)

```
begin
case S is
when '0' => y <= a;
when '1' => y <= b;
when other => y <= z;
end case;
end process;
```

default statement

```
entity mux 2:1 is
port (a, b, S : in STD_LOGIC;
      y : out STD_LOGIC);
end mux 2:1 is;
```

__/__/__

Pros & Cons of VHDL →

VHDL or VHSIC Hardware Description language has advantages & disadvantages for describing digital logic circuits.

Advantages →

1) Describes Complex Systems → VHDL can describe Complex systems as simpler components which helps designers create accurate models of electronic circuits.

2) Simulation models → VHDL Specifications can be used to create simulation models to verify a part's operation in a system.

3) Modelling Styles → VHDL supports three modelling styles : Behavioral, Structural, & Data flow.

4) Design entry, documentation & Verification → VHDL can be used for design entry, documentation & verification.

Disadvantages →

Learning Curve → VHDL can be difficult

to learn & write, especially if you're not familiar with Strongly typed languages.

Syntax → VHDL has a Complex Syntax that requires declarations, statements & Configuration.

Support → VHDL is less Popular & widely supported than other tools, libraries & Communities.

Standardization → VHDL is not yet standardized for analog design.

2) Various Design approches in VHDL →

In VHDL, the Primary design approches are data flow modelling, Structural modeling & behavioral modelling:

Each representing a different level of abstraction when describing a circuit's functionality, with data flow focusing on Signal flow, Structural on Compment Connections, & behavioral on algorithmic descriptions of the circuit behavior.

Explanation of each approach →

__/__/__

Data flow modelling →

* **focus →** Represents how data flows through a circuit by directly assigning values to signals based on i/p signals using signal assignment. Statements.

Key aspects →

1) uses operators like AND, OR, XOR etc.
2) Best for describing simple logic circuits where data flow is straight forward.

Example → o/p <= input 1 AND input 2 ;

2) Structural modelling →

focus → Describe the circuit architecture by connecting different components (like predefined logic gates or custom designed entities) together.

Key Aspects →

1) Uses Component instantiation to create instances of other entities.
2) Provides a hierarchical approach to design complex systems by breaking down into smaller blocks.
3) **Example :** V1 : entity - name port map (- - - -) ;

Behavioral modelling :

focus → Describe the circuit functionality using Procedural statements like -

"if - then - else", "Case", loops, & sequential statements, allowing for a more algorithmic approach.

Key aspects →

1) Can model Complex logic with State dependent behavior.
2) most flexible approach for Complex designs.

Example — if (Condition) then
output <= value 1;
else
output <= value 2;
end if;

mixed modeling → often, designers Combine different modelling styles within a single design, using data flow for simple logic & behavioral for Complex Control logic.

Abstraction level → The choice of modelling approach depends on the desired level of abstraction, with data flow offering

the lowest level & behavioral Providing the highest.

Abstraction level of VHDL → In VLSI Design :

in VHDL VLSI design, "abstraction level" refers to the level of detail used to describe a circuit, with the most common levels being behavioral (algorithmic), register transfer level (RTL) and gate level; where behavioral is the highest level of abstraction, focus on the overall functionality, while gate level is the lowest, detailing individual logic gates & their connections.

Explanation of the abstraction levels :

Behavioral / Algorithmic level :

This is the highest level of abstraction, where the design is described using high-level constructs like loops, conditional statements, & mathematical operations, focusing on the desired behavior of the circuit without specifying the exact hardware implementation.

__/__/__

Register Transfer Level (RTL)

Considered the most common level for design implementation, RTL describes the data flow between registers, specifying how data is transferred & manipulated b/w different parts of the circuit using logic gates & registers, while still maintaining a relatively high level of abstraction.

Gate level → This is the lowest level of abstraction, where the circuit is described in terms of individual logic gates like AND, OR, NOT etc., providing a detailed representation of the hardware structure, often generated automatically by synthesis tools from the RTL code.

It could be defined as the amount of information an entity is hiding within it.

level of abstraction :-

System level
↓
chip level
↓
Register level
↓
Gate level
↓
Circuit level → layout level

less detailed

more detailed

HDL → Hardware description language

is a Specialized Computer language used to describe the Structure & behavior of electronic circuits, most Commonly to design Asic's & Program FPGA.
A hardware description language enables a Precise, formal description of an electronic circuit that allows for the automated analysis & simulation of an HDL description into a Netlist (A specification of Physical electronic Components & how they are Connected together) which Can then be Placed and routed to produce the set of masks used to create an Integrated circuit.

HDL's form an integral Part of electronic design automation (EDA) Systems. especially for Complex circuits, such as application - specific integrated circuits

microprocessor & Programmable Logic devices.

High Level language →

High Level language are Programming language that are used for writing program

or Software that can be understood by humans & Computers.
High level language are easier to understand for humans because they use a lot of Symbols letters phrases to represent logic & instruction in a Program it Contains a high level of abstraction Compared to low level language.

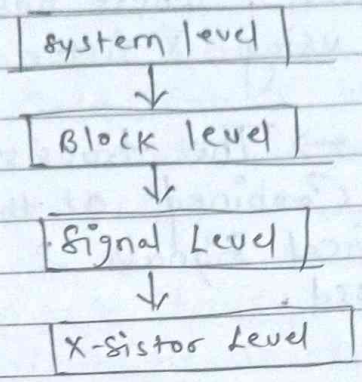
Python, Java, C++

Difference between HDL & High Level language →

	HDL	HLL
(1)	A specialized Computer language used to describe the structure & behavior of electronic circuit most Commonly digital logic circuits.	1) A Computer language used to write a set of instruction to allow the CPU to program a specific task.
(2)	more Complex	2) Not as Complex.
(3)	Verilog & VHDL are Common examples.	(3) Java, C, C++ Python, PHP etc.
(4)	Describe the behavior of digital ckt.	(4) Helps to develop various application.

Top-Down Approach →

The top-down approach focuses on the overall system Architecture.



① System level → The design starts at the highest level where the overall system architecture & functions are defined.

② Block level → The system is then divided into smaller blocks. each of which is designed & developed separately.

③ Signal Level → Next, the blocks are designed at the signal level, where electrical signals & logic gates are used.

④ Transistor Level → finally, the signal level design is implemented at the x-sistor level, where actual transistor & wiring are used.

Bottom-up Approach →

① Transistor level → The design starts at the lowest level, where basic logic gates are built using x-sistor & wiring.

② Signal Level → The transistor level design is then Combined at the signal level. where electrical signals & logic gates are used.

③ Block level → Next the signal level design is Combined at the block level, where smaller blocks are created.

④ System level → finally the block level design is Combined at the system level.

While the Bottom-up approach focuses on the detailed implementation of individual Components.

PLD (Programmable logic devices)

→ PLD Based design flow typically involves the following steps.

__/__/__

① Design Entry → Describe the desired behavior using a hardware description language (HDL) like VHDL or Verilog.

② Synthesis → Synthesis means Convert the HDL Code into a netlist, a ckt description that can be used to Program the PLD.

③ Simulation → Verify the netlist's functionality through Simulation to ensure it matches the desired behavior.

④ Place & Route → Map the netlist to the PLD's Architecture, Placing logic elements & Routing Connections.

⑤ Timing Analysis → Verify the design meets timing requirements, ensuring signal Propogation within the PLD meets Specifications.

⑥ Programming → load the final Configuration into the PLD, making it operational.

⑦ Testing → Validate the PLD's functionality in the target System. This flow may vary depending on the Specific

PLD & design tools used.

Synthesis → Synthesis is a crucial step in the PLD design flow. it involves converting the HLL of the desired behavior (written in HDL) into a netlist that can be used to program the PLD.

During Synthesis the HDL Code is analyzed & optimized to create a digital ckt. that implements the desired behavior. These Process involves.

1. **Parsing** → Breaking Down the HDL Code into Smaller Components.
2. **optimization** → Applying Algorithm to reduce the No. of logic elements & improve Performance.
3. **Mapping** → Converting the optimized design into a Netlist that can be used to program the PLD.
4. **Verification** → checking the Netlist to ensure it meets the desired behavior & timing requirements.

The o/p of the Synthesis step is a netlist. that can be used for