CHAPTER 1

Introduction to Hardware Discription Languages (HDLs)

1.1 Digital Systems

Digital systems are designed using digital ICs. A digital system accepts digital inputs in the form of binary and after processing the inputs it generates digital outputs in the form of binary. The digital systems can be broadly classified into combinational systems and sequential systems.

Combinational Digital Systems

In the combinational systems, the outputs are generated based on present inputs. The outputs are not feedback to inputs and hence the present outputs do not depend on past output and therefore, the combinational systems do not require memory to store past outputs. The combinational systems are developed using logic gates, decoders, multiplexers, adders, etc.

Sequential Digital Systems

In the sequential systems, the outputs are generated based on present inputs and past outputs. The outputs are feedback to inputs and hence the present outputs depend on past output and therefore, the sequential systems require memory to store past outputs. The flip-flops are the fundamental memory device to store binary bits and hence flip-flops are employed in sequential systems to store the past outputs and fed back to input.

The sequential systems are again classified into synchronous and asynchronous sequential systems. The synchronous sequential systems are controlled by a clock whereas the asynchronous sequential systems do not have a clock.

In synchronous sequential systems the inputs and past outputs are recognized to generate outputs only when the clock signal is active. The time period (or frequency) of clock is selected such that the one time period of clock is sufficient for the sequential circuit to settle at a stable value for a change in input.

The asynchronous sequential circuits are not governed by clock and hence, in an asynchronous sequential circuit the change in input at any time will generate a change in output. Therefore, in asynchronous sequential circuit any change in input should be made only after the circuit settle at a stable value for previous input.

1.1.1 Analog and Digital Electronics

The digital electronics refer to devices and circuits that process digital signals which are coded in binary. The real world signals are analog and hence the field of electronics was dominated by analog devices and circuits before the invention of digital devices. The real world analog signals are converted to digital signals and processed by digital systems and after processing the digital signals are used either directly or converted back to analog for real world applications. The ADCs (**A**nalog-to-**D**igital **C**onverter**s**) are used for converting analog signals to digital signals and DACs (**D**igital-to-**A**nalog **C**onverter**s**) are used to convert digital signals to analog signals.

The block diagram of a typical system with ADC and DAC is shown in Fig. 1.1. A system will have a number of control variables that can be modified or processed to achieve a desired task. The control variables are converted to electrical analog voltage signals by sensors and the analog voltage signal is converted to digital signal by ADC.

The digital system process or modify the digital signal and output the processed digital signal which are converted back to analog by DAC. The processed analog signal is converted to physical variables by actuators which are used by the system to achieve the desired task.

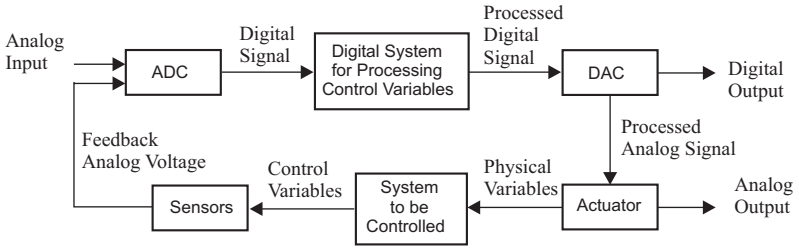


Fig. 1.1: *A physical system interfaced to digital system using ADC and DAC.*

The history of digital electronics dates back to 1900s during that period the digital circuits were made using mechanical relays and later replaced by vacuum tubes. The invention of semiconductor transistors in 1960s led to a revolution by replacing bulky vacuum tubes with transistors. The invention of **I**ntegrated **C**ircuit**s** (ICs) again led to minimize the size of digital devices.

The invention of programmable ICs completely turns the world to digital era. In modern technology the processing and control of signals and systems are made digital.

Advantages of Digital Systems

- Digital hardware is compact, reliable, less expensive and programmable.
- Digital systems are programmable, the performance of the system can be easily upgraded/modified.
- High speed, sophisticated digital hardware leads to higher precision in processing of signals.
- Digital signals can be permanently stored in magnetic media so that they are transportable and can be processed in non-real-time or off-line.

1.1.2 Digital System Design

Primitive Method of Combinational Digital System Design

The primitive method of combinational digital system design involve design using simplified Boolean equations obtained from K-map and implement the degital system using combinational devices and logic gates.

The design starts with problem specifications. The procedure to design a combinational digital system is given below:

- Determine the required inputs and outputs from the problem specifications.
- Assign a symbol to each input and output.
- Derive the truth table.
- Draw K-map for every output and form the prime implicants.
- Determine the simplified Boolean function for every output from the prime implicants.
- Draw the logic diagram using Boolean functions that relate the inputs and outputs.
- Implement the logic digram as a digital circuit using logic gate and combinational devices.

Primitive Method of Sequential Digital System Design

The primitive method of sequential digital system design involve design using state diagram and simplified Boolean equations obtained from K-map and implement the degital system using flip-flops and logic gates.

The design starts with problem specifications. The procedure to design a combinational digital system is given below:

- Draw the state diagram from specifications.
- Determine the state table from state diagram.
- If necssary, reduce the number of states and obtain the reduced state table.
- Assign binary values or codes to states. If there are m state variable and n inputs then 2^{m+n} binary codes are needed to code the state table.
- Form the binary coded state table.
- Choose the type of flip-flop to be used in design (either D or JK or T). The number of flip-flops is equal to number of state variables.
- When D flip-flops are used for design the flip-flops inputs are same as next state outputs. In order to design using JK and T flip-flops determine the flip-flop inputs using excitation table.
- Form the expanded state table with flip-flop inputs.
- Derive the simplified flip-flop input and output equations using K-map.
- Draw the logic diagram using the input and output equations.
- Implement the logic digram as a digital circuit using flip-flops and logic gates.

Modern Method of Digital System Design

The modern method of combinational and sequential digital sysstem design involve design using **H**ardware **D**escription **L**anguage**s** (HDLs) and implement the digital system using programmable and reconfigurable ICs like CPLD (**C**omplex **P**rogrammable **L**ogic **D**evices) and FPGA (**F**ield **P**rogrammable **G**ate **A**rrays).

Hardware **D**escription **L**anguage**s** (HDLs) are modelling languages specially developed to describe digital hardware structures and to study the behavior of logic circuits. It can be used to implement logic circuits, truth tables, Boolean functions and state tables/diagrams as computer programs for simulation and verification. Digital system design simulation, verification and hardware implementation are made easier by HDLs.

VHDL and Verilog are the two popular IEEE standardized HDLs. VHDL is an acronym for **V**HSIC(**V**ery **H**igh **S**peed **I**ntegrated **C**ircuits) **H**ardware **D**escription **L**anguage. The name Verilog is the combination of two words "verification" and "logic".

1.1.3 Digital Integrated Circuits (ICs)

Modern digital systems are built using digital **I**ntegrated **C**ircuits (ICs). The digital ICs are ICs that can accept only digital signals in the form of binary and process the digital signal and output the digital signals in the form of binary.

Classification of Digital ICs Based on Number of Transistors

The fundamental electronic device in a digital IC is transistor. One way of classification of digital ICs are based on density of fundamental transistors used to fabricate an IC.

Based on number of transistors packed in a single IC, the digital ICs are classified as SSI, MSI, LSI, VLSI and UVLSI integrated circuits.

SSI (Small Scale Integration): SSI refers to integration of few (typically 4 to 10) transistors on a chip. It is the beginning of IC technology. This technology is used to fabricate logic gates AND, OR, NOT, etc., as ICs.

MSI (Medium Scale Integration): MSI refers to integration of 10 to 500 transistors on a single chip. This technology is used to fabricate small digital devices like decoders, encoders, multiplexers, adders, etc.

LSI (Large Scale Integration): LSI refers to integration of 500 to 20000 transistors on a single chip. This technology leads to fabrication of early version of programmable ICs and microprocessors.

VLSI (Very Large Scale Integration): VLSI refers to integration of 20000 to 1 Million transistors on a single chip. This technology is used to fabricate complex digital ICs and microprocessors and microcontrollers with advanced features

UVLSI (Ultra Very Large Scale Integration): Integration of more than 1 Million transistors on a single chip is called UVLSI or ULSI. This technology is currently used to fabricate modern processors with billions and trillions of transistors on a single chip.

Programmable and Nonprogrammable Digital ICs

In another way of classification, the digital ICs are classified into programmable and nonprogrammable ICs. The nonprogrammable ICs are designed to perform fixed functions. Examples of nonprogrammable

digital ICs are decoders, encoders, multiplexers, adders, etc. The programmable digital ICs are programmable by the user to perform desired functions. The programmable ICs are again classified into one time programmable, multi-time programmable and field programmable or reconfigurable ICs.

The one time programmable ICs are used to implement small digital functions or systems by the user. The user can program these ICs one time to implement the digital system and once programmed the content or function of the IC cannot be altered. Examples of one time programmable digital ICs are PROM (**P**rogrammable **R**ead **O**nly **M**emory), PAL (**P**rogrammable **A**rray **L**ogic) and PLA (**P**rogrammable **L**ogic **A**rray).

The multi-time programmable ICs will have a set of instructions to write a program to implement the desired digital systems. These ICs will have various internals devices with programmable features whose functions can be altered by a program. The microprocessors and microcontrollers are examples of multi-time programmable ICs.

The CPLD (**C**omplex **P**rogrammable **L**ogic **D**evice) and FPGA (**F**ield **P**rogrammable **G**ate **A**rrays) are reconfigurable ICs. These devices will have programmable logic resources in the form of gate arrays which can be programmed to implement any digital systems. The logic resources are open resources without any internal connections. The programming of these devices will create internal connections and hence the process of program is called configuration. The internal connections made by a program can be altered by reprogramming the device and hence these ICs are referred to as reconfigurable ICs.

1.2 Introduction to Programmable ICs

The digital ICs (**I**ntegrated **C**ircuit**s**) can be broadly classified into fixed function ICs and programmable ICs. The fixed function ICs are manufactured to perform a fixed logical function. Examples of fixed function ICs are flip-flops, decoders, multiplexers, counters, etc. The programmable ICs are designed to be programmed by the user to implement a variety of logical functions.

Some of the programmable logical devices are,

- PROM (**P**rogrammable **ROM**)
- PLA (**P**rogrammable **L**ogic **A**rray)
- PAL (**P**rogrammable **A**rray **L**ogic)
- SPLD (**S**equential **P**rogrammable **L**ogic **D**evice)
- CPLD (**C**omplex **P**rogrammable **L**ogic **D**evice)
- FPGA (**F**ield **P**rogrammable **G**ate **A**rray)

The PROM, PAL and PLA are one-time programmable and consists of programmable logic gate arrays and can be used to implement only combinational logic functions. The SPLD, CPLD and FPGA devices are reprogrammable and consists of programmable logic gate arrays and flip-flops and so can be used to implement both combinational and sequential logic functions.

The advantages of using PLDs to implement a logical function are,

- PLDs require smaller chip and board area to implement a logic circuit.
- Logical functions can be easily altered or modified without rewiring and changing the board.
- Implementation is much faster when compared to fixed function ICs.

1.2.1 Complex Programmable Logic Device (CPLD)

A CPLD integrates a number of PLDs in a single device so that large complex combinational and sequential digital systems can be implemented in a CPLD.

The CPLDs of different manufactures differ in architecture and functionality but have some common features. The common basic architecture of a CPLD is shown in Fig. 1.2.

A CPLD consists of three main components: programmable logic blocks, programmable interconnect channel and IO block. Each programmable logic block of a CPLD consists of a fully programmable AND-OR array similar to a PAL/PLA and a bank of macrocells. A simplified block diagram of a typical programmable logic block is shown in Fig. 1.3.

The AND/OR array is reprogrammable and can perform a large number of logic functions. Macrocells are functional blocks with flip-flop, data selector and other control logic circuits that can perform combinatorial or sequential logic. Macrocells also have the added flexibility for true or complement output, along with feedback paths. Each macrocell has multiple configurations and each macrocell can be used in cascade, so that more complex combinational and sequential logic functions can be realized.

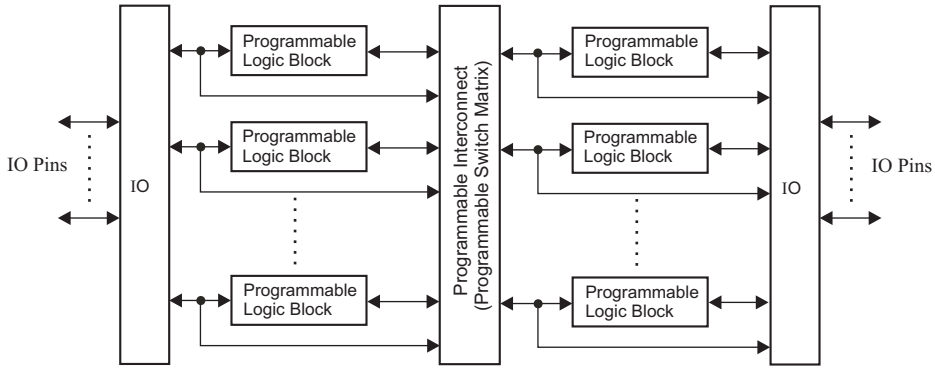


Fig. 1.2: *Basic architecture of a CPLD.*

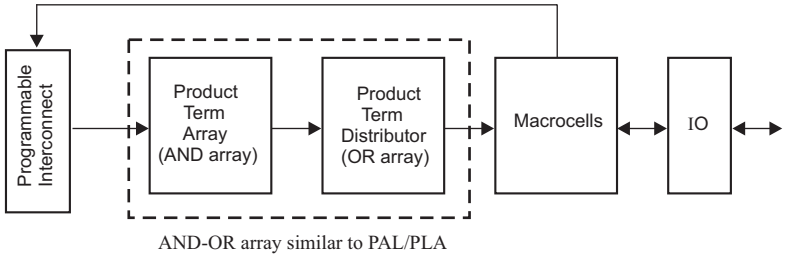


Fig. 1.3: *Simplified block diagram of a programmable logic block in a CPLD.*

Programmable interconnect channels provide interconnection networks between logic blocks, macrocells and input/output pins. The input/output blocks provide the interface between the internal logic to the IO pins of the device. In advanced version of CPLDs, embedded array blocks with on-chip RAM/ROM are usually provided.

1.2.2 Field Programmable Gate Array (FPGA)

R.H.Freeman invented the first practical FPGA in 1985. FPGA is a programmable semiconductor logic device consisting of large number of logic gates which can be configured by the user to implement any kind of digital circuit or system. FPGAs are manufactured with thousands to millions of logic gates. The programming of logic gates of FPGA is called configuration and FPGAs are reprogrammable (or reconfigurable).

The configuration of the FPGA is generally performed using a **H**ardware **D**escription **L**anguage (HDL). The popular HDLs are Verilog and VHDL.

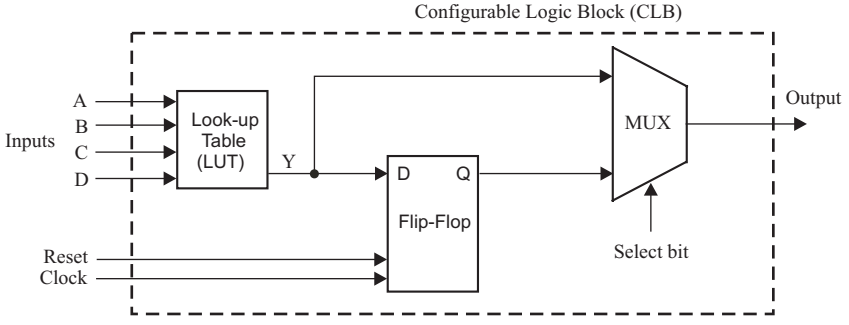
FPGAs store their customized configuration data in SRAM-type internal latches. Depending on the device size and user-design implementation options the number of configuration bits varies from 1 Mb to 33 Mb or even more in modern FPGAs.

The internal SRAM is volatile and so the configurations bits will be erased whenever power is turned OFF. Therefore, when an FPGA is powered up the configuration storage must be reloaded. Usually, the modern FPGA based systems will have serial EEPROM as boot loader, so that the configuration bits are permanently stored in boot loader and whenever power is turned ON the configuration bits are serially transmitted from EEPROM to configuration SRAM in FPGA.

Xilinx and Altera are two major manufacturers of FPGA holding nearly 90% of market share. Xilinx was acquired by AMD in the year 2022 and rebranding Xilinx FPGAs in phased manner. The discussions in this section are with reference to Xilinx FPGAs. Altera FPGAs may also have resources similar to Xilinx FPGAs but there will be differences in architecture and functionality.

Basic Architecture of FPGA

FPGAs are evolved from gate arrays and early versions of FPGA basically have simple **C**onfigurable **L**ogic **B**locks (CLB) and small number of inputs and outputs as shown in Fig. 1.4. Each CLB will consist of a **L**ook-**U**p **T**able (LUT) and a **F**lip-**F**lop (FF). The basic idea of FPGA's is to inter-connect small "truth tables" to emulate complex digital circuits. In an FPGA, these tables are called "**L**ook-**U**p **T**able" (LUT).



LUT	
Inputs A B C D	**Output** Y
0 0 0 0	0
0 0 0 1	1
0 0 1 0	1
⋮	⋮
1 1 1 1	0

Fig. 1.4: *Simple CLB architecture.*

The basic architecture of an FPGA is shown in Fig. 1.5.

The FPGA architecture consists of four major components

- Programmable/Configurable Logic Blocks
- Programmable switch blocks
- Programmable Routing (interconnect lines)
- IO blocks

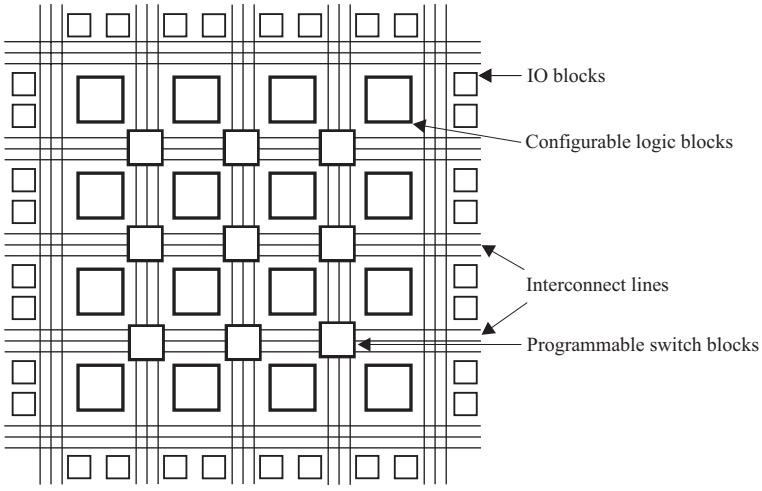


Fig. 1.5: *Basic architecture of FPGA.*

A **C**onfigurable **L**ogic **B**lock (CLB) is the basic logic resource on an FPGA. The **C**onfigurable **L**ogic **B**lock (CLB) includes logic and look-up tables that can be configured to implement combinational logic circuits. When CLBs are linked together by routing resources (carry chain), the components in CLBs execute complex logic functions, implement memory functions and synchronize code on the FPGA. The block diagram of a typical CLB in an FPGA is shown in Fig. 1.6.

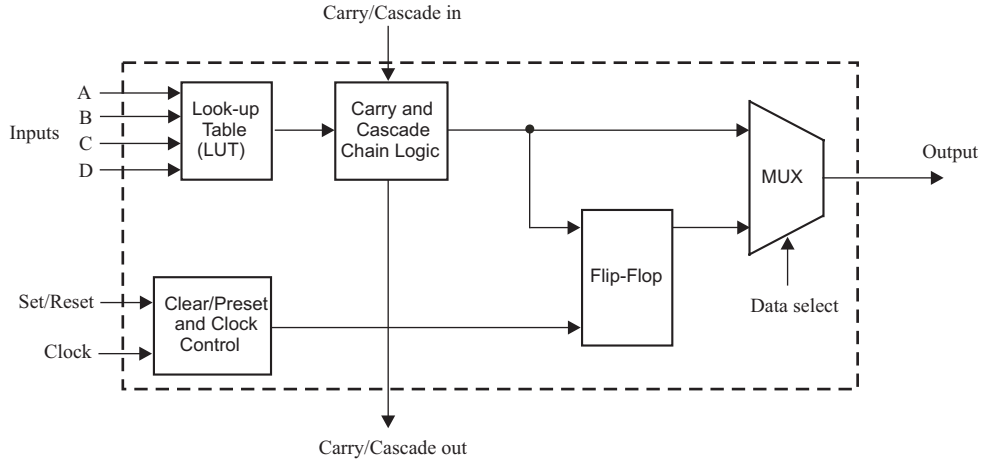


Fig. 1.6: *Block diagram of a typical CLB in an FPGA.*

The CLB shown in Fig. 1.6 contain look-up-table, carry logic, flip-flop and multiplexer. It also has a clear/preset and clock control unit. A LUT can speed up processing by providing the output for a given input without logical processing. A flip-flop is a primitive storage device that can store a single bit of information.

Carry look-ahead logic is implemented in each CLB with a combination of dedicated multiplexer and XOR gates that are used within the carry/cascade chain. The carry/cascade chain logic in an individual CLB is the circuit that directly connects a column of CLBs together. It is possible to cascade a carry chain across multiple CLBs in order to quickly implement addition, subtraction, and multiplication operations on operands that are too large to be processed by a single CLB.

Advantages of FPGA

- FPGAs can implement faster and parallel processing.
- FPGAs are programmable IC, programmed by software and can be re-programmed or reused any number of times.
- Since FPGA ICs are programmable, the solution is available faster to the market.
- FPGA development is cheaper due to non-recurring expenses.
- In FPGA, software takes care of routing, placement and timing and so manual intervention is lesser.

Disadvantages of FPGA

- Programming the FPGA requires the knowledge of VHDL/Verilog programming languages and digital system fundamentals.
- Programmers do not have any control on power optimization.
- Development resources are more specific for specific FPGA.
- FPGAs are costly for large quantity production.

Comparison of CPLD and FPGA

- FPGA is more suitable for structures with large number of flip-flops, while CPLD is more suitable for structures with limited flip-flops and rich product terms.
- The continuous wiring structure of CPLD determines that its timing delays are uniform and predictable, while the segmented wiring structure of FPGA determines the unpredictability of its delays.
- CPLD is programmed by modifying the logic function with fixed interconnect circuit, while FPGA is programmed by changing the wiring of internal wires.
- FPGA can be programmed under the logic gate, while CPLD is programmed under the logic block. Hence, CPLDs are faster than FPGAs and have greater time predictability.
- FPGAs are more integrated than CPLDs and have more complex wiring structures and logic implementations.
- CPLD is programmed using internal EEPROM or FASTFLASH technology, no external memory chip is needed. While the FPGA programming information needs to be stored in external memory, and transferred to internal RAM during switch ON.
- CPLD is mainly based on EEPROM or FLASH memory programming, which can be programmed up to 10,000 times, and has the advantage that the programming information is not lost even when the system is powered off.

1.3 Hardware Description Language (HDL)

The **H**ardware **D**escription **L**anguage (HDL) is a specialized computer language used to describe the structure and behaviour of digital electronic circuits, usually to design **A**pplication-**S**pecific **I**ntegrated **C**ircuits (ASICs). The ASIC can be implemented in **F**ield-**P**rogrammable **G**ate **A**rrays (FPGA).

In the history of digital system design, various notations have been developed for capturing the logical behaviour of digital circuits at different levels of abstraction. Examples of such notations include Boolean equations, timing charts, state transition tables, schematics and hardware description languages.

The HDLprovides a simple method for description of an electronic circuit that allows for the automated analysis and simulation of the circuit. HDL also allows for the synthesis of an HDL description into a netlist. A netlist is a file format that describes the components, connectivity and the placement and routing of an electronic circuit which can used be to produce the set of masks used to fabricate an integrated circuit.

A hardware description language looks much like a programming language such as C, C++, Java, Python, etc. HDL is a textual description of digital logic consisting of expressions, statements and control structures. One important difference between most programming languages and HDLs is that HDLs explicity include the notion of time.

Advantages of HDLs

Circuit Design: HDLs provides a simple way to design digital circuits that meet the required specifications.

Simulation: HDLs helps the designer to test and verify the digital circuit before it is built.

Verification: HDLs provides designer to verify the functionality of the digital circuit by testing it against different inputs and ensuring that the circuit functionality is correct.

Synthesis: HDLs can be used to synthesize digital circuits. Synthesis is a process of automatically generating circuits from HDL code.

Timing Analysis: HDLs provides desiginers to analyse the timing behaviour of digital circuits and ensure that the circuits meet the timing requirements.

Design Reusability: HDLs provides a way to design reusable components that can be used for multiple circuits design and reduce time and effort, which improves overall design quality.

Optimization: HDLs provides a way to optimise the design of digital circuits for performance and power efficiency.

1.3.1 History of HDL

The concept of a HDL as a medium for design capture was first introduced in the 1960s, but wide adoption by the design community started after 1985. Historically, the development of software programming languages supports the evolution of HDLs.

The first hardware description language came in late 1960, which looks like a traditional language. The first HDL was called "**D**iscription **L**anguage for **H**ardware" (DLH) and was developed by IBM in the late 1960s. However, it was not widely used because of its complexity and difficulty to use. In the subsequent years, various other HDLs came into existence such as ABEL and PALASM. Some of popular HDLs are listed in the Table 1.1.

In the mid-1980s VHDL was introduced by the Department of Defence, USA and later standardised by IEEE. VHDL was designed to describe digital circuits, which aimed to develop high performance digital circuits for military applications. In the same year, Verilog was introduced by Phil Moorby and Prabhu Goel owned by Gateway Design Automation. Verilog was originally intended to be used for verification, but it became popular as HDL.

Table 1.1: List of Popular HDLs

Time Period	Year (Approx)	HDL	Developed by (Company/Organization)
Early Academic and Research HDLs (1960s - 1970s)	1967	ISP (Instruction Set Processor)	Carnegie Mellon University
	1969	AHPL (A Hardware Programming Language)	University of Utah
	1970	CDL (Computer Description Language)	University of Cambridge, UK
	1972	APDL (A Processor Description Language)	Carnegie Mellon University
Early Industry/ Simulation HDLs (1970s - early 1980s)	1972	AHDL (A Hardware Description Language)	Intel Corporation
	1974	HILO (High-Level Input / Output)	GenRad (General Radio Corporation)
	1975	CONLAN (Console Language)	Carnegie Mellon University
	1976	SILAGE (Signal Language)	KU Leuven, Belgium
PLD-Oriented HDLs (late 1970s - 1980s)	1978	PALASAM (Programmable Array Logic Assembler)	Monolithic Memories Inc.
	1980	ABEL (Advanced Boolean Expression Language)	Data I/O Corporation
	1982	CUPL (Compiler for Universal Programmable Lagic)	Logical Devices Inc.
Other Notable Pre-Standard HDLs (1970s - early 1980s)	1977	DABL (Design Automation Boolean Language)	IBM Corporation
	1978	ELLA (Electronic Logic Language)	Royal Signals and Radar Establishment. UK
	1980	ZEUS	Stanford University
Standard and Industry-Diminat HDLs (mid 1970s onwards)	1983	VHDL	U.S. Department of Defence
	1984	Verilog	Gateway Design Automation, Inc.
Modern Extensions High - Level HDLs (1990s - present)	1990	System Verilog	Accellera Systems Initiative
	1993	VHDL-AMS	VHDL International/IEEE
	1994	Verilog-AMS	Accellera Systems Initiative
	1996	SystemC	Open SystemC Intiative
	2003	Bluespec System Verilog	Bluespec Inc.
	2011	Chisel (Scala HDL)	University of California, Berkeley
	2012	MyHDL	Open-source community (Independent)
	2014	SpinalHDL	SpinalHDL community (Open-source)
	2016	nMigen / Amaranth	Amaranth Community (Open-source)

1.3.2 IDE Tools for FPGA Based System Design

The manufacturers of FPGA ICs provide IDE (**I**ntegrated **D**evelopment **E**nviroment) tool for designing FPGA based digital systems. IDE tools integrate HDL coding, synthesis, simulation and implementation into a single enviroment. IDE tools are vendor-specific IDEs tied to their FPGA hardware.

Some of the popular IDE tools are,

- **AMD/Xilinx Vivado Design Suite:** The standard for AMD/Xilinx FPGAs (Virtex, Kintex, Artix, Zynq), offering synthesis, implementation and system-level integration.
- **Vitis Unified Software Platform:** Used alongside Vivado for AI, HLS (High-Level Synthesis) and embedded software development.
- **Intel Quartus Prime (Pro/Standard/Lite):** Primary enviroment for Intel/Altera FPGAs (Stratix, Arria, Cyclone).
- **Lattice Diamond / Lattice Radiant:** Specialized IDE for Lattice low-power FPGAs.
- **Microchip Libero Soc:** Design enviroment for Microchip Polarfire, RTG4, SmartFusion2 and IGLOO2 FPGAs.

Some of the popular code editors and front-end IDEs are,

- **VS Code + TerosHDL:** A popular, modern open-source combination for linting, synthesis visualization and documentation.
- **Sigasi Studio:** An intelligent IDE for HDL (VHDL, Verilog, System-Verilog) that acts as a language-aware editor.
- **Synopsys Euclide:** Specialized IDE aimed at debugging and design analysis for ASIC/FPGA.

Some of the simulation and verification tools are,

- **Siemens ModelSim/Questa:** Industry-standard simulators for HDL functional verification.
- **Aldec Riviera-PRO / Active-HDL:** High-performance simulation tools, often used for mixed-language designs (VHDL-2019 support).

Some of the open-source and specialized tools are,

- **Icestudio:** Visual editor tailored for open FPGA boards, particularly Lattice iCE40.
- **F4PGA (Free FPGA):** An open-source tool chain for FPGA synthesis and implementation.
- **OpenFPGAloader:** A universal utility for loading bit streams onto FPGAs.
- **Verible:** A suite of System-Verilog developer tools including a linter and formatter.

1.4 Digital System Design using Vivado IDE

Digital system design and analysis features include logic simulation, IO and clock planning, power analysis, contraint definition and timing analysis, design rule checks, visualization of design logic, analysis and modification of implementation results, programming and debugging.

The entire solution in integrated within a **G**raphical **U**ser **I**nterface (GUI) known as the Vivado **I**ntegrated **D**esign **E**nviroment (IDE). The Vivado IDE provides an interface to assemble,implement and validate the design and the IP. In addition, all flows van be run using Tcl commands.

The Vivado design suite is developed by Xilinx for FPGA based digital system design which offers multiple ways to accomplish the tasks involved in Xilinx FPGA based digital system design, implementation and verification.

The traditional design method using Vivado IDE is **R**egister **T**ransfer **L**evel (RTL)-to-bitstream FPGA design flow. Vivado IDE offers system level integration flows as alternative to RTL-to-bitstream design flows that focus on **I**ntellectual **P**roperty (IP)-centric design and C-based design.

The Vivado Design Suite supports the following established industry design standards:

- Tcl
- AXI4, IP-XACT
- **S**ynopsys **D**esign **C**onstraints (SDC)
- Verilog, VHDL, VHDL-2008, SYSTEM vERILOG
- Syatem C, C, C++

General Design Flows

The various steps in Vivado IDE Design flows are:

- HDL based system design
- Logic simulation
- Assign logical and physical constraints
- Logic synthesis
- Implementation
- Timing closure and design analysis
- Generate bitstream, programming and debug

RTL-to-Bitstream Design Flows

The various steps in RTL-to-bitstream design flows are,

RTL Design: Specify RTL source files to create a project and use these sources for RTL code development, analysis, synthesis and implementation. Xilinx supplies a library of recommended RTL and constraint templates to ensure RTL and XDC are formed optimally for use with the Vivado design suite. Vivado synthesis and implementation suport multiple source file types, including Verilog, VHDL, System-Verilog and XDC.

IO and Clock Planning: The Vivado IDE provides an IO pin planning enviroment that enables IO port assignment either onto specific devices package pins or onto internal die pads and provides tables to design and analyse package and IO related data. Memory interface can be assigned interactively into specific IO banks for optimal data flow.

Xilinx Platform Board Support: In the Vivado design suite, an existing Xilinx evaluation platform board can be used as atarget for the design. In the platform board flow, all of the IP interfaces implemented on the target board are exposed to enable quick selection and configuration of the IP used in the design.

Synthesis: Vivado synthesis performs a top-down synthesis of the overall RTL design. However by default, the Vivado design suite used an **O**ut-**O**f-**C**ontext (OOC) design flow to synthesize IP cores from the Xilinx IP catalogue and block design from the Vivado IP integrator. Specific modules of a hierarchical RTL design can be synthesized as OOC modules. The OOC synthesized netlist is stored and used during top-level implementation to preserve results and reduce runtime.

Design Analysis and Simulation: The Vivado design suite used to perform analysis, verification and modification of the design at each stage of the design process. This analysis can be run after RTL elaboration, synthesis and implementation.

Placement and Routing: When the synthesized netlist is available, Vivado implementation provides all the features necessary to optimize, place and route the netlist onto the availabe device resources of the target FPGA.

Hardware Debug and Validation: After implementation, the FPGA device can be programmed and the analyzed with the Vivado logic analyser or within the standalone Vivado lab edition enviroment. Debug signals can be identified in the RTL design or inserted after synthesis and are processed throughout the flow.