Introduction To HDL / VHDL

Digital Logic Design (CC131)

BSCS 2nd Semester Spring-2026

Lecture: Week-8-a

By

Syed Zulfiqar Ali Shah,

Associate Professor (CS) / HoD

Govt. College of Management Sciences, Kohat

www.linkedin.com/in/syedzulfiqaralishah

Hardware Description Languages (HDL)

- A Hardware Description Language is a specialized programming language used to **describe the structure and behavior of electronic circuits and digital systems** — rather than writing software that runs on hardware, you're writing code that *defines* what the hardware itself looks like and how it behaves.
- Unlike software languages (Python, C), which describe a sequence of instructions executed over time, HDLs describe things that exist **simultaneously in parallel** — because real hardware runs many operations at the same time.
- HDLs serve two primary purposes: **simulation** (testing how a design behaves before building it) and **synthesis** (automatically converting code into actual gate-level circuits for FPGAs).

Why HDLs are Used?

Modern digital chips (CPUs, GPUs, FPGAs) contain **billions of transistors**. Drawing or manually specifying these is impossible.
HDLs solve this by:

- Letting engineers describe hardware at a **high level of abstraction** (behavioral, register-transfer, or gate level).
- Enabling **automated synthesis tools** to convert descriptions into physical circuits.
- Supporting **simulation and verification** before expensive fabrication.
- Making designs **portable and reusable** across different target technologies.

Classical HDLs

Verilog (1984) — the most widely used HDL globally. C-like syntax, concise, and directly supported by almost every FPGA and ASIC toolchain. Used by companies like Intel, NVIDIA, and AMD for chip design.

verilog

```verilog
module half_adder(input a, b, output sum, carry);
  assign sum   = a ^ b;   // XOR
  assign carry = a & b;   // AND
endmodule
```

Classical HDLs

VHDL (1987) — Very High Speed Integrated Circuit HDL. Ada-inspired, verbose, and strongly typed. Preferred in defense, aerospace, and European academia. More explicit than Verilog — every signal type must be declared precisely.

vhdl

```vhdl
entity half_adder is
  port (a, b : in  std_logic;
        sum, carry : out std_logic);
end entity;
architecture rtl of half_adder is
begin
  sum   <= a xor b;
  carry <= a and b;
end architecture;
```

Classical HDLs

SystemVerilog (2005, IEEE 1800) — a superset of Verilog that adds object-oriented features, assertions, interfaces, and powerful verification constructs. It is now the dominant language for both RTL design and testbench writing in professional chip companies.

Modern / High Level HDLs

These embed hardware description inside general-purpose languages, enabling more powerful abstractions and software-style tooling.

- **Chisel** (Constructing Hardware in a Scala Embedded Language) — developed at UC Berkeley. Used to design the RISC-V open-source processor family. Generates Verilog as output, so it is compatible with all standard tools.

- **SpinalHDL** — also Scala-based, popular in open-source FPGA projects. Offers very clean, readable hardware descriptions with strong type safety.

- **MyHDL** — Python-based. You write hardware logic in Python, simulate it with Python tools, and can convert it to Verilog or VHDL for synthesis. Great for rapid prototyping.

Key Concepts in Verilog

Modules are the building blocks — like functions in software, each module describes a hardware component:

```verilog
module AND_gate (
  input  wire A,
  input  wire B,
  output wire Y
);
  assign Y = A & B;   // concurrent assignment
endmodule
```

Key Concepts in Verilog

Always blocks describe sequential or combinational logic that reacts to signals:

```verilog
always @(posedge clk) begin   // triggered on clock edge
  Q <= D;                      // flip-flop behavior
end
```

Key difference from software:
All always blocks and assign statements run **simultaneously**, not one after another.

Editors Used

Verilog doesn't require a special editor — any text editor works — but popular choices include:

- **VS Code** with the "Verilog HDL" or "Verilog-HDL/SystemVerilog" extension (syntax highlighting, linting)
- **Vivado IDE** by AMD/Xilinx — full suite for FPGA design targeting Xilinx FPGAs
- **Intel Quartus Prime** — for Intel/Altera FPGAs, with integrated editor and synthesis
- **Cadence Xcelium** / **Synopsys VCS** environments — used in professional ASIC design
- **EDA Playground** — browser-based, great for learning

What Is the Output?

HDLs produce very different outputs depending on the stage:

From simulation → a **waveform file** (VCD — Value Change Dump) showing how every signal changes over time.

From synthesis → a **gate-level netlist** — a description of the circuit in terms of AND, OR, NOT gates, flip-flops, and their interconnections.

From place-and-route (FPGA) → a **bitstream file** (.bit or .bin) that is literally loaded onto the FPGA chip to configure its programmable logic cells. This is the final "program" for the hardware.

From place-and-route (ASIC) → a **GDSII file** — the geometric layout of every transistor and metal wire, sent to a semiconductor foundry (like TSMC) for physical fabrication. This is how real chips like CPUs are made.

Thank You