Computer Organization and Embedded System

Hardware descripts language and IC technology (ACtE0406)

Ishwar Kumar Singh
Computer Engineer
Government of Nepal

4.6 Hardware descripts language and IC technology (ACtE0406)

- VHDL Overview,
- Overflow and data representation using VHDL,
- Design of combinational logic using VHDL,
- Design of sequential logic using VHDL,
- Pipelining using VHDL

VHDL Overview

- VHDL stands for VHSIC Hardware Description Language.
- VHSIC stands for Very High-Speed Integrated Circuit.
- VHDL is a hardware description language used for **describing** and **simulating digital** and **hardware systems**.
- It provides a means for designing, documenting, and testing digital circuits such as **combinational and sequential logic**.
- VHDL is used widely in **Application-Specific Integrated Circuits** (ASICs) and **Field-Programmable Gate Arrays** (FPGAs) design.

VHDL Overview

Key components of VHDL include**:**

- **Entities** The entity describes the **interface** of a module.
- **Architectures**: The architecture defines the **behaviour** of a module.
- **Processes**: Used to model **sequential behaviour** and can include **sensitivity lists** for event-driven simulation.
- **Data Types**:
  - Scalar Type such as Bit, Boolean, Integer, Character, Real, etc.
  - Composite Type such as Array, Record etc.

VHDL Features

- **Concurrency:** VHDL is designed to describe hardware systems, which are **inherently concurrent**. This means that **multiple processes** can execute simultaneously, reflecting the parallel nature of hardware components.

- **Modularity:** VHDL supports the design of **modular systems** through the use of **entities** and **architectures** which allows for better design abstraction and reuse.

- **Strongly Typed Language:** VHDL enforces strong typing, meaning that **data types are rigorously checked** during compilation. This helps prevent errors by ensuring that operations are only performed on compatible types.

VHDL Features

- **Simulatability and Synthesizability**: VHDL is both a simulation language and a synthesis language i.e. it can be used to **simulate** the **behaviour of the design** as well as to **generate the hardware implementation**.

- **Support for Hierarchical Design**: VHDL supports hierarchical designs, allowing **complex systems** to be **broken down** into smaller, more **manageable modules**.

- **Timing Specification**: VHDL allows for precise timing specifications, making it suitable for designing systems that need **accurate control** of **time-based operations**.

- Prepared by Er. Ishwar Kumar Singh

Overflow and Data Representation Using VHDL

- Overflow occurs when a result **exceeds** the **maximum value** that can be represented by a given data type.
- In VHDL, handling overflow requires careful **attention** to **data types**, especially when performing arithmetic operations.
- **Data Representation**: VHDL supports a variety of data representations, including binary, hexadecimal, and decimal.
- **Fixed and Floating Point**: VHDL provides packages such as fixed_pkg and float_pkg for fixed and floating-point arithmetic.
- **Handling Overflow**: It can be managed using **conditional statements** (e.g., IF-THEN) or by setting **constraints** on the signal ranges.

Design of Combinational Logic Using VHDL

- Combinational logic is a type of digital logic in which the **output** is purely a **function of the current inputs**.
- VHDL allows the modelling of combinational logic using constructs such as:
    - **IF-ELSE Statements**: Commonly used for implementing conditional logic.
    - **CASE Statements**: Useful for selecting one output from multiple inputs, such as in a multiplexer.
    - **WHEN-ELSE**: Another method of conditional assignment in concurrent statements.

Design of Combinational Logic Using VHDL

Important Considerations for Combinational Logic:

- **Sensitivity List**:
  - All that signals that affect the output needs to be included in the sensitivity list to avoid latches or unintended sequential behaviour.
- **No Memory Elements**:
  - Combinational logic should not contain memory elements (e.g., no use of flip-flops or registers).
  - If memory elements are needed, the logic is sequential, not combinational.

Design of Sequential Logic Using VHDL

- Sequential logic depends not only on the **current inputs** but also on **past inputs** or states.
- VHDL is capable of modelling sequential circuits such as flip-flops, counters, and finite state machines (FSMs).
  - **Processes with Clock Signals**: Sequential logic is typically modelled using processes that are sensitive to clock edges.
  - **Flip-Flops**: D, T, and JK flip-flops are commonly described using processes with clock and reset signals.

Design of Sequential Logic Using VHDL

- **Finite State Machines (FSMs)**: FSMs can be implemented using ENUM (Enumerated) types for state encoding and CASE statements for transitions between states.
    - Using an ENUM type for state encoding means that each state in the FSM is assigned a unique name, making the code more readable and maintainable.
- **Counters and Shift Registers**: Sequential counters and registers are modelled using clocked processes that update on clock edges.

Design of Sequential Logic Using VHDL

Important Considerations for Sequential Logic:

- **Clock and Reset Management:** Proper management of clock and reset signals is crucial in sequential designs to ensure **correct initialization and timing**.
- **Avoiding Race Conditions:** Ensure that all processes sensitive to a clock edge have **consistent timing** to avoid race conditions that could lead to unpredictable behaviour.
- **Timing Constraints:** Sequential circuits must adhere to **timing constraints** such as setup and hold times, which can be enforced during synthesis.

Pipelining using VHDL

- Pipelining is a technique which involves breaking a complex operation into multiple stages, where each stage performs a portion of the task.
- In VHDL, pipelining is implemented by dividing the **logic into stages** and using **registers** (flip-flops) to store the **intermediate results** between stages.
- These registers are **triggered** by a clock signal, ensuring that each stage of the pipeline advances on each clock cycle.

Key Concepts in Pipelining

- **Pipeline Stages**:
  - The operation is divided into several smaller tasks, each of which is handled in a separate pipeline stage.
  - Each stage operates concurrently on different data.
- **Registers**:
  - Flip-flops or registers are placed between the pipeline stages to hold intermediate results.
  - These registers are clocked to synchronize data transfer between stages.

Key Concepts in Pipelining

- **Latency vs Throughput**:
  - Pipelining increases the throughput (the number of operations completed per unit of time) but adds latency (the time taken for a single piece of data to pass through the entire pipeline).
- **Balanced Pipeline**:
  - For maximum efficiency, the stages should be balanced, meaning that each stage should take approximately the same amount of time to complete its task.
  - If one stage is slower, it can become a bottleneck for the entire pipeline.

Thank You.