Comprehensive VHDL

Formation de 3 à 5 jours

Présentation

Cette formation modulaire (3 jours + 2 jours) permet de maîtriser le langage VHDL pour la conception et la vérification de circuits numériques : CPLD, FPGA et ASIC.

Elle se compose de deux modules :

- **VHDL pour Conception FPGA – 3 Jours – 21 heures effectives**
- **VHDL Avancé (Jours 4 et 5) - 2 Jours – 14 heures effectives**

pour un total de 5 jours, soit 35 heures effectives

Objectifs pédagogiques

- Comprendre et maîtriser parfaitement l'essentiel du langage VHDL.
- Savoir utiliser au mieux le langage VHDL pour concevoir des projets CPLD, FPGA ou ASIC et obtenir des circuits plus compacts, plus performants et plus fiables.
- Maîtriser l'utilisation du langage VHDL pour la Vérification (modélisation comportementale de base et bancs de Tests sophistiqués).

Connaissances requises

La participation à cette formation demande une connaissance des principes de base de l'électronique numérique (comme dispensé lors de notre formation *Essential Digital Design Techniques* par exemple).

Programme de la Formation

Le Langage VHDL pour la Conception FPGA (Jours 1 à 3)

Introduction

◆ The scope and application of VHDL ◆ Design and tool flow ◆ FPGAs ◆ The VHDL world

Getting Started

◆ The basic VHDL language constructs ◆ VHDL source files and libraries ◆ The compilation procedure
◆ Synchronous design and timing constraints

FPGA Design Flow (Practical exercises using a hardware board)

◆ Simulation ◆ Synthesis ◆ Place-and-Route ◆ Device programming

Design Entities

◆ Entities and Architectures ◆ Std_logic ◆ Signals and Ports ◆ Concurrent assignments
◆ Instantiation and Port Maps ◆ The Context Clause

Processes

◆ The Process statement ◆ Sensitivity list versus Wait ◆ Signal assignments and delta delays
◆ Register transfers ◆ Default assignment ◆ Simple Testbenches

Synthesising Combinational Logic

◆ If statements ◆ Conditional signal assignments and Equivalent process
◆ Transparent latches ◆ Case statements ◆ Synthesis of combinational logic

Types

◆ VHDL types ◆ Standard packages ◆ Integer subtypes ◆ Std_logic and std_logic_vector
◆ Slices and concatenation ◆ Integer and vector values

Synthesis of Arithmetic

◆ Arithmetic operator overloading ◆ Arithmetic packages ◆ Mixing integers and vectors
◆ Resizing vectors ◆ Resource sharing

Synthesising Sequential Logic

◆ RISING_EDGE ◆ Asynchronous set or reset ◆ Synchronous inputs and clock enables
◆ Synthesisable process templates ◆ Implying registers

FSM Synthesis

◆ Enumeration types ◆ VHDL coding styles for FSMs ◆ State encoding
◆ Unreachable states and input hazards

Memories

◆ Array types ◆ Modelling memories ◆ IP Generators
◆ Instantiating generated components ◆ Implementing ROMs

Basic TEXTIO

◆ TEXTIO ◆ READ and WRITE ◆ Using TEXTIO for testbench stimulus and outputs ◆ STD_LOGIC_TEXTIO

Formation Avancée sur le Langage VHDL (Jours 4 et 5)

More About Types

◆ Variables ◆ Loops ◆ Std_logic and resolution ◆ Array and integer subtypes ◆ Aggregates

Managing Hierarchical Designs

Hierarchical design flow ◆ Library name mapping ◆ Component declaration
◆ Configuration ◆ Hierarchical configurations ◆ Compilation order

Parameterised Design Entities

◆ Array and type attributes ◆ Port Maps ◆ Generics and Generic Maps
◆ Generate statement ◆ Generics and generate

Procedural Testbenches

◆ Subprograms ◆ Procedures ◆ Functions ◆ Parameters and Parameter Association
◆ Package declarations ◆ Package bodies ◆ Subprograms in packages ◆ Subprogram overloading
◆ Operator overloading ◆ Qualified expressions ◆ RTL Procedures

Text-File-Based Testbenches

◆ Assertions ◆ Opening and closing files ◆ Catching TEXTIO errors
◆ Converting between VHDL types and strings
◆ Checking simulation results ◆ Initialising memories ◆ Foreign bodies

Gate Level Simulation

◆ Rationale for gate level simulation ◆ VITAL tool flow
◆ Reuse of RTL testbench at gate level
◆ Comparison of RTL and gate level results ◆ Behavioural modelling