A.L.S.E.
http://www.alse-fr.com

Formation Pratique

Expert VHDL

(sont inclus OSVVM & UVVM)

DOULOS

Présentation

Cette **formation modulaire de cinq jours** (2 jours + 3 jours) permet d'approfondir la connaissance du langage VHDL et d'en maîtriser les aspects les plus avancés afin de pouvoir concevoir et vérifier les circuits numériques (CPLD, FPGA et ASIC) et les projets les plus complexes. Elle se compose de **deux modules** :

- **Expert VHDL Conception** (Jours 1 et 2) : axé sur l'utilisation avancée du langage pour la synthèse (RTL), cette partie permet d'améliorer la productivité, la maintenabilité et la ré-utilisation. Une introduction à la vérification par assertions est incluse.
- **Expert VHDL Vérification** (Jours 3 à 5) : met à profit une utilisation avancée du langage pour la modélisation comportementale, la création de bancs de tests sophistiqués et le principe de la vérification par assertions, c'est-à-dire la méthodologie indispensable à la vérification des systèmes actuels hautement complexes.
  Cette partie comprend désormais une introduction aux **librairies open source de vérification : OSVVM et UVVM.**

Ces modules s'enchaînent naturellement, mais ils peuvent être suivis indépendamment.

Ce cours s'inscrit dans le cadre d'une action d'acquisition ou de perfectionnement des connaissances.

Les participants peuvent utiliser au choix différents outils de Conception, Synthèse et Simulation (FPGA ou ASIC) durant les **exercices pratiques**, qui occupent environ **50 % du temps de la formation.**
Ces exercices, éléments clés du succès de ces formations, sont indispensables pour un réel apprentissage.
Ils **assurent** également un **contrôle continu de l'acquisition des compétences.**

Les **Instructeurs ALSE** sont aussi et surtout des **Experts en Conception** qui utilisent à journée entière les Langages et les Méthodes de Conception qu'ils enseignent, pour concevoir et vérifier des systèmes complexes et performants. Ils savent partager leur savoir-faire avec passion et sont particulièrement appréciés des participants.

Pré-requis

La participation à cette formation demande une **connaissance et une pratique préalable du langage VHDL**. Elle peut faire suite à notre formation de base *Comprehensive VHDL* après une période de mise en pratique recommandée de 6 mois au moins.

Durée

Deux (2) jours, ou **trois (3) jours,** ou **cinq (5) jours**
soit **14 heures**, ou **21 heures** ou **35 heures effectives** de cours ;
Typiquement 9h30 → 18H, le premier jour et 9H → 17h30 les jours suivants.

Lieu

dans les locaux d'ALSE, **Paris XIII^{ème}**

NB : à partir de quatre ou cinq stagiaires, une formation sur site est envisageable.

ALSE est un Organisme de Formation Professionnelle Continue déclaré auprès de la DIRECCTE depuis 1996 sous le numéro 26.21.01281.21 et enregistrée dans le Datadock.

Objectifs pédagogiques

- Utilisation avancée du langage **VHDL pour la synthèse** (code RTL) et tirer parti de VHDL 2008.
- Comprendre, résoudre et **éviter les problèmes d'aléas**, et les erreurs usuelles de conception.
- Savoir écrire du code RTL plus fiable, plus efficace, mieux ré-utilisable en tirant le meilleur parti des outils de conception actuels.
- Savoir gérer les problèmes de complexité (**multiples domaines d'horloges** par exemple).
- Utiliser les concepts avancés du langage pour concevoir des **modèles comportementaux** (*Behavioural Modeling*).
- Maîtriser la construction de bancs de tests évolués et la **Vérification** de systèmes complexes.
- Découvrir les **nouvelles librairies alternatives de Vérification : OSVVM et UVVM**.
- Comprendre les principes de la **Vérification par Assertions** (ABV).

Contenu des modules de la Formation

Expert VHDL Conception (jours 1 et 2)

- Techniques de Codage RTL. Fiabilité et de qualité de résultat après synthèse. VHDL 2008.
- Optimisation des performances et timings, Gestion des domaines d'horloges multiples.
- Codage et implémentation des Machines d'États (FSMs).
- Introduction aux Propriétés et aux Assertions (ABV).

Expert VHDL Vérification (jours 3 à 5)

- Concepts avancés du langage VHDL et application à la vérification fonctionnelle.
- Utilisation du langage pour la construction d'un environnement de vérification puissant et versatile.
- Sous-programmes, E/S Fichiers, Transaction Level, BFMs, vérification temporelle.
- Random et couverture fonctionnelle.
- Introduction à OSVVM.
- Introduction à UVVM.

Programme

Expert VHDL Design (Jours 1 et 2)

RTL Synthesis and Synchronisation

• Synthesis of Combinational Logic • Synthesis of Sequential Logic
• Combinational and Sequential Together • Variables in Clocked Processes • How Many Registers?
• Resolution Functions and Drivers • Unresolved Types • Synchronous Design
• Synchronous Design Rules • Non-Synchronous Features • How Clever is Your Synthesis Tool?
• Resource Sharing • Synthesizing Arithmetic - WYSIWYG • Eliminating the Mux • Shift Left/Right
• Forcing an Implementation • Synthesis Attributes • Inference or Instantiation? • Static Timing Analysis
• Timing Analysis Exceptions • Re-timing and Pipelining • Timing Analysis Exceptions
• Hierarchy and Optimization • Registered Ports • Asynchronous Inputs • Input Hazard • Metastability
• Synchronizer in VHDL • Multiple Clock Domains • Transferring Data Between Clock Domains
• Using a FIFO for Synchronisation • Synchronising Reset

Writing Readable Designs

• VHDL Features for Abstraction • Records • Using Record Fields • Aggregates • Using Records for Wiring
• Connecting Records • Inside a Peripheral • Multiplexed Bus Structure • Multiplexing Records
• Representing Register Maps • Accessing Individual Registers • Accessing the Status Register
• Collecting Registers Together • Alias • Using Aliases with Vectors • Using Alias with a Bus
• Other Uses of Alias • Named Ranges • Using Named Ranges • Summary
.../...

Writing for Re-Use

• Language-Level Reuse • Reusing Code Fragments • Standard Component Reuse • Procuring IP
• Writing Reusable RTL IP • Reuse Tradeoffs • Example of a Regular Structure • Arrays of Arrays
• Generics • Type Generics • Array Attributes • Generate Statement • More on Generate Statements
• VHDL-2008 Generate Statements • Implementing the Behavior • Unconstrained Array Ports
• Parameterizing the Data Width • VHDL-2008 Arrays • Multidimensional Arrays • Flattening Matrices
• Multidimensional Arrays of Ports • Getting and Setting Rows • Architecture Using 2D Arrays
• Instancing the Design • Optional Ports • Default on Component

Advanced Coding Styles

• Subprograms • Procedures • Parameter Class, Mode, and Type • Subprogram Overloading
• Signal Parameters • Synthesizing Procedures • Destructive Register Reads • Read Procedure
• Procedure in FSM • Functions • Recursion • Recursive Function Example • Creating a Generic Counter
• Creating a log2 Circuit • Recursive Instantiation • Recursive Component Declaration
• Solving the Problem • Solving the Problem (VHDL-2008) • Synthesis Results

Finite State Machine Synthesis

• State Transition Diagrams • Finite State Machines • A Simple Two Process Style
• Another Two Process Style • One Process Style • State Machine Architecture
• Comparison of Coding Styles • Easier Registered Outputs • Registered Outputs Using Local Variables
• State Encoding • No Output Decoding • Unreachable States • Controlling Unreachable States
• Explicit One Hot Style • Hand Optimised One Hot Style • Collision Signal
• Too Many States? • FSM Summary

Packages & Configurations

• Packages • Subprograms in Packages • Deferred Constants • VHDL-2008 Packages
• Generics on a Package • Float Package • Fixed Package • Fixed Point Example • Use Clause Scope
• Component Declaration and Instantiation • Component Declarations in a Package • Configuration
• Default Binding Rules • Hierarchical Configurations • Nested Configurations • Port Name Changes
• Type Conversions • Setting Generics • Configurations with Generate Statements

Properties & Assertions

• Property versus Assertion • Applying Properties • Who Writes Properties? • Observability
• A Simple Assertion • Simulation Checker or Monitor • Properties and the Specification
• The Story so Far... • OVL • Using OVL with VHDL • OVL Packages • OVL Configuration
• Instancing an Assertion • OVL VHDL Assertions • Example Assertions
• PSL • The Elements of a Property • PSL Basics • The Structure of PSL • The Boolean Layer
• Clocks and Default Clocks • Holds and Implication • next • The Temporal Layer • The Verification Layer
• Verification Units • Modelling Layer • Using PSL with an HDL • Simulation of Temporal Properties
• Summary of Benefits

Expert VHDL Verification (Jours 3 à 5)

Functional Verification Methodology

• What Is Verification? • Approaches to Verification • Verification Strategy • What to Verify?
• Towards a Verification Plan • Don't Plan Everything • Identify Testcases • Verification Metrics
• Coverage • Including Functional Coverage • Coverage Driven Verification • From Features to Tests
• Checking • Verification Planning Revisited • The Basic Testbench • Verification Environment
• Verification Methodologies • VHDL Methodology • Design for Verification • Glass Box Testing
• Analysis to Choose Tests • Boundary Conditions & Corner Cases • Black Box Testing
• Regression Testing • Stress Testing • Different Sorts of Stimulus

SubPrograms and Protected Types

• Using Abstraction in a Testbench • Subprograms • Procedures • Parameter Class, Mode, and Type
• Subprogram Overloading • Signal Parameters • Functions • Subprograms in Packages
• Impure functions • Protected types • Protected type body • Declaring a Shared Variable

More on File IO

• The Basic Testbench • TEXTIO Output • Procedure READ • When READ Goes Wrong
• Converting Values to Text Strings • Opening and Closing Files • Testing the File Open Status
• Managing Text Files • Package STD_LOGIC_TEXTIO • Using Built-in Files
• Reading Variable Length Data • Files Without Textio • Binary Files • VHDL-2008 File IO

Transaction Level Verification

• Structuring Testbenches • Build the Complete Testbench • Monolithic Testbenches are Inflexible
• Hide DUT Interface from Testcase • Layered Architecture • Transaction Level Testcase
• Making a Transaction in VHDL • Accessing the Fields • Communicating Transactions
• A Simple Example • What is a (VHDL) Transaction? • Interfacing Without Events
• Generating Transactions • Using Procedures with Signals • A Systematic Approach
• Non-blocking Procedures • Bus Functional Modelling • Bus Functional Model
• Bus Functional Model Using Get • Synchronization • Synchronization Channel • Summary

More on BFMs - Time in Testbenches

• Bus Functional Modelling • Wait Statements • Wait Statements and Time • Inspecting the Event Queue
• Example SRAM Timing • Setup Time Check • Hold Time Check • Combined Setup / Hold Time Check
• Pulse Width Check • Entity Declarations • Passive Processes • Using Vital Packages
• Setup/Hold Check With Vital • What About Transactions? • Concurrent Signal Assignments • Drivers
• How to "See" Drivers • Multiple Driver Issues • Longest Static Prefix • Sequential Signal Assignments
• Inertial Delay • Identical Successive Assignments • Inertial Delays • Inertial and Transport Delays
• Sampling Data - RTL • Postponed Processes

Behavioral Modelling and Checkers

• Checking Results • Variable Latency • Arrays of Records • Queues • VHDL Queue Implementation
• Using Queues • Coping with Out-of-Order Completion • Scoreboarding • Dynamic Memory Allocation
• Access Types • Allocators • Deallocating Memory • Writing to a FIFO • Rrading from a FIFO
• Pointer Problems • Behavioural Modelling Example • Modelling the 2-wire Bus • Two Wire Slave Model
• Protocol Implementation • Data Generation • Slave Procedure • Modelling State

Random Testing and Coverage

• Verification - Reminder • Random Stimulus • Constraining Random Stimulus
• Random Sequence of Valid Actions • Functional Coverage • Where Am I? • Concurrent Procedures
• Coverage Procedure • Calling Coverage Procedures • Why Use Path Parameter?

Other Testbench Features

Completing our Methodology • Monitoring Internal Signals • Monitoring Internal Signals - VHDL-2002
• External Names • Monitoring Internal Signals - VHDL-2008 • Force and Release
• The Objection Mechanism • Implementing Objection • Resolution Functions
• Implementing Custom Objections • The Stop and Finish Procedures • Run-time Configuration
• Implementing Run-time Configuration

Introduction to OSVVM

• What is OSSVVM? • Randomization • Seed Management • Functional Coverage • Sampling
• Specifying Bins • Specifying a Minimum Hit Count Per-Bin • Specifying Cross Bins
• Specifying Ignore Bins • Displaying Coverage • Non-Repeating Randomize • Defining Explicit Weights
• Weight by Coverage Shortfall • Logging • Redirecting to a Log File • Alerts • Stop Count
• Disabling Alerts • Conditional Alerts • Hierarchical Alerts • Other Packages

Introduction to UVVM

• What is UVVM? • Utility Library • Logging and Verbosity Control • Outputting Log Messages
• Controlling Log Messages • Message IDs • Redirecting Log Messages • Alerts • Controlling Alerts
• Reporting Alerts • Checks • Awaits • String Handling • Randomization • Signal Generators
• Synchronization • BFM Common Package • Warnings • UVVM VVC Framework • UVVM Structure
• UVVM Test Harness • UVVM Test Bench • Test Sequencer • UVVM Command Distribution
• Example Command Distribution Methods • UVVM VHDL Network Model • VIP
• OSVVM vs UVVM Feature Comparison