

V2 - VHDL avancé pour les FPGA

Acquérir une solide méthodologie de conception avec le meilleur du VHDL pour la simulation et la synthèse

Objectifs

- Comprendre les différentes possibilités offertes par le langage VHDL
- Être capable de lire et de tester des composants VHDL
- Comprendre les notions de synthèse logique
- Comprendre le problème crucial de l'implémentation des machines à états finis (FSMs) dans le matériel
- Organiser le code en développant des packages et des bibliothèques
- Réutilisation des composants
- Apprendre à écrire des testbenches efficaces pour la simulation
- Connaître les différents styles de rédaction et leur impact sur la qualité des résultats de la synthèse
- Vérifier les timings
- Comment structurer et écrire des environnements de vérification structurés VHDL importants et complexes
- introduction aux méthodologies de vérification VHDL OSVVM et UVVM

Pré-requis

- Connaissance de base du langage VHDL, niveau du cours [oV1 - Les bases du langage VHDL](#)

Environnement du cours

- Cours théorique
 - Support de cours au format PDF (en anglais)
 - Le formateur répond aux questions des stagiaires en direct pendant la formation et fournit une assistance technique et pédagogique
- Activités pratiques
 - Les activités pratiques représentent de 40% à 50% de la durée du cours
 - Elles permettent de valider ou compléter les connaissances acquises pendant le cours théorique.
 - Un PC Linux en ligne par stagiaire pour les activités pratiques
 - Xilinx Vivado IDE
 - Exemples de code, exercices et solutions

Audience visée

- Tout ingénieur ou technicien en systèmes embarqués possédant les prérequis ci-dessus

Course Outline

Premier Jour

Finite State Machine (1st part)

- The Finite State Machine Approach
 - Sequential Circuits and State Machines
 - State Transition Diagram
 - Transition Types
 - Moore-to-Mealy Conversion

- Mealy-to-Moore Conversion
- Exercises
- Hardware Fundamentals
 - Flip-Flops
 - Metastability and Synchronizers
 - Pulse Detection
 - Glitches
 - Pipelined Implementations
 - Exercises
 - Hardware Architectures for State Machines
 - Fundamental Design Technique for Moore Machines
 - Fundamental Design Technique for Mealy Machines
 - Moore versus Mealy Time Behavior
 - State Machine Categories and State-Encoding Options
 - Safe State Machines

Finite State Machine (2nd part)

- Design Steps and Classical Mistakes
 - Classical Problems and Mistakes
 - Design Steps Summary
- Regular State Machines
 - Architectures for Regular Machines
 - Number of Flip-Flops
 - Exercises
- Timed State Machines
 - Architectures for Timed Machines
 - Timer interpretation
 - Transition Types and Timer Usage
 - Timer Control Strategies
 - Time Behavior of Timed Moore and Mealy Machines
 - Examples of Timed Machines

Exercise : Designing a burstable RAM controller

Deuxième Jour

Design Methodology for Synthesis

- Designing for Synthesis
- Metastability
- Memory Synthesis
- Reset Generation
- Crossing Clock domains

Exercise : Metastability

Timing analysis and constraints

- Timing Closure challenges
- A methodology for successful Timing Closure
- Common Timing Closure Issues
- Static Timing Analysis
- Role of Timing Constraints in STA
- Common Issues in STA
- Delay Calculation versus STA
- Timing Path
- Setup and Hold
- Slack

- On-Chip Variation
- Clock
- Port Delays
- Completing Port Constraints
- False Paths
- Multi Cycle Paths
- Combinational Paths
- Xilinx Extensions

Exercise : Design closure

Exercise : Analyzing and Resolving timing violations

Troisième Jour

Introduction to Open Source VHDL Verification Methodology (OSVVM)

- Overview
- Transaction-Level Modeling
- Constrained Random Test Generation
- Functional Coverage
- Intelligent Coverage Randomization Methodology
- Utilities for Testbench Process Synchronization
- Transcript Files
- Error Logging and Reporting: Alerts and Affirmations

Introduction to Universal VHDL Verification Methodology (UVVM)

- Utility Library
- VVC (VHDL Verification Component) Framework
- BFM (Bus Functional Models)
- OSVVM and UVVM

Vivado Debug

- Vivado Integrated Logic Analyzer (ILA)
- Adding debug nets
- Analyzing debug data
- Resources