Modélisation Transactionnelle des Systèmes sur Puce avec SystemC
Phelma 3A — filière SEOC
Grenoble-INP

TLM Avancé & Conclusion

Frédéric Pétrot

frederic.petrot@univ-grenoble-alpes.fr

2022-2023

Ensimag

Planning des séances

05/12/22 (FP) CM1 Introduction : systèmes sur puce et modélisation au niveau transactionnel

14/12/22 (FP) CM2 Introduction au C++ et présentation de SystemC

14/12/22 (FP) CM3 Communications haut-niveau et modélisation TLM en SystemC

04/01/23 (FP) CM4 Utilisations des plateformes TLM et Notions Avancées en SystemC/TLM

04/01/23 (FP) TP1 (1/1): Plateforme matérielle SystemC/TLM

09/01/23 (OM) CM5 Synthèse d'architecture

09/01/23 (OM) TP3 (1/2) : Synthèse de haut niveau et génération de circuits numériques

09/01/23 (OM) TP4 (2/2) : Synthèse de haut niveau et génération de circuits numériques

16/01/23 (FP) CM6 Intervenant extérieur : Jérôme Cornet (STMicroelectronics)

18/01/23 (FP) TP2 (1/2) : Intégration du logiciel embarqué

28/01/23 (FP) TP2 (2/2) : Intégration du logiciel embarqué

Sommaire

Sommaire

Préparer l'examen

- Annales disponibles (répertoire `exam/`)
- Documents interdits
- Une feuille A4 recto-verso manuscrite autorisée

Sujet d'examen

- Questions de cours C++, SystemC/TLM, intervenant extérieur
- 1 exercice sur le temps simulé/wall-clock (cf. années précédentes)
- 1 problème : extension de la plateforme « TP2 ». Cette année (2018-2019) : composant matériel pour calculer le nombre de voisin à 1 d'un point dans une image bitmap

À savoir impérativement

- C++
- Bases de SystemC : `SC_MODULE`, `SC_THREAD`, `SC_METHOD`, `wait`, `notify`, ...
- Les principes de TLM 2.0
- L'API EnsitIm : `read`, `write`, `map`, `bind`
- L'API hal : `hal_read32`, `hal_write32`, `hal_cpu_relax`, `hal_wait_for_irq`.

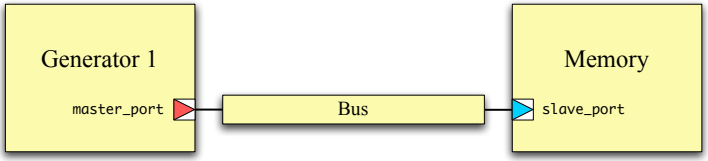
Ce qui énerve le correcteur ...

- Confusion entre hardware et software (e.g. écrire du SystemC dans le soft embarqué, ou utiliser `hal.h` dans le modèle de matériel)
- « Justifiez brièvement » mal lu
- Les erreurs sur les points répétés N fois en TP/cours.

Sommaire

TP n°1, début

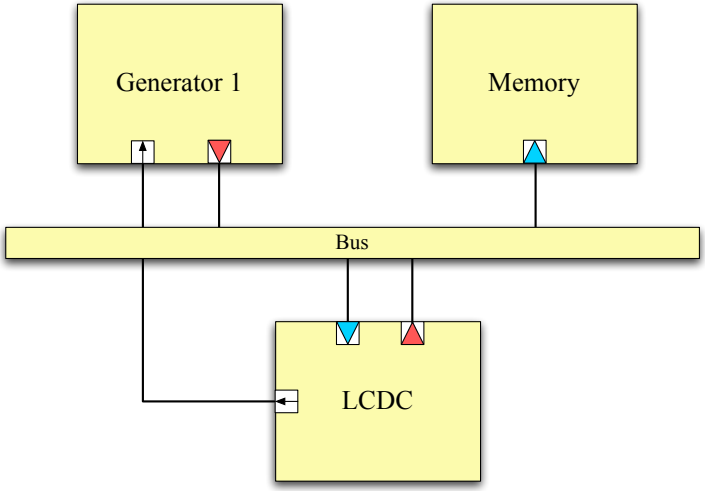
- Prise en main de SystemC/GCC
- Écriture d'un générateur de transactions
  - Outil de test de plateforme
  - Représente les accès que ferait un processeur (par ex)
- Écriture d'une mémoire
  - Mécanisme d'adresse locale (offset)
  - Implémentation du comportement (tableau dynamique C++)
- Comportement global



TP n°1, fin

- Lecture de documentation technique : contrat d'utilisation du LCDC
- Modélisation de registres
  - Utilisation des événements SystemC
  - Correspondance avec la documentation
- Gestion des interruptions
- Fabrication d'images en mémoire, ...

TP n°1 - Figure



TP n°2

- Intégration du logiciel embarqué.
  - Avec ISS
  - En simulation native
- Correspondance entre plateforme physique (FPGA) et TLM
  - Même registres, même addressmap, même comportement
  - RAM programme gérée différemment
  - Protocole de bus non modélisé en TLM
- Logiciel portable via `hal.h`:
  - Une implémentation en simulation native
  - Une implémentation pour RISC-V (ISS ou FPGA – tbc –)

TP n°2: Chaînes de compilation

- Native:
  - `g++`/`gcc`, comme d'habitude
  - `extern "C"` pour faire communiquer le C et le C++ (problème de *mangling* et d'ABI)
  - Édition de liens entre plateforme et logiciel
- Croisée:
  - `riscv64-unknown-elf-{gcc,ld,objdump}` : tourne sur x86_64, génère du code pour RISC-V
  - Logiciel embarqué compilé en un fichier ELF ...
  - ... chargé dynamiquement en RAM par la plateforme
  - `boot.s`: adresse de boot, vecteur d'interruption, ...
  - `it.s`: routine d'interruption (sauvegarde/restauration de registres avant d'appeler une fonction C)
  - `ldscript`: utilisé par `riscv64-unknown-elf-ld` pour décider des adresses des symboles.
  - `printf`: marche sur FPGA via une UART, trivial en simu native, composant UART en simu ISS.

TP n°2 : ce à quoi vous avez échappé...

- Fait pour vous:
  - Écriture des composants TLM (Giovanni Funchal)
  - ISS RISC-V, `boot.s`, `it.s` (votre serviteur, riscv-probe)
- Non géré:
  - `gdb-server`: pour déboguer le logiciel avec `gdb` comme s'il tournait sur une machine physique distante.
  - Temps précis
  - Transaction bloc (entre RAM et VGA en particulier)
  - Conflits sur le bus entre RAM $\leftrightarrow$ VGA et fetch.
  - Contrôleur d'interruption évolué (le notre est essentiellement une porte « ou »)

Sommaire

Réutilisation de composants

Piloté par le consortium Accellera (`https://www.accellera.org`)

- Point de vue d'un industriel :
  - Écriture de modèles TLM réutilisables de composants maisons
  - Modèles TLM de composants d'entreprises tierces ?
- Idée : chaque fabricant de composant fournit plusieurs modèles
  - RTL ou netlist
  - Modèle TLM, etc.
  - etc.
- Problème : mettre tout le monde d'accord sur l'écriture de modèles TLM

Documentation

Piloté par le consortium Accellera (`https://www.accellera.org`)

- Besoin d'informations organisées sur chaque composant
  - Ensemble de registres
  - Nombre de ports
  - Technologies de fabrication supportées
  - Consommation électrique
  - Surface, ...

Documentation

Piloté par le consortium Accellera (`https://www.accellera.org`)

- Besoin d'informations organisées sur chaque composant
  - Ensemble de registres
  - Nombre de ports
  - Technologies de fabrication supportées
  - Consommation électrique
  - Surface, ...
- Création d'un consortium d'industriels pour standardiser les informations associées à un composant
  - Consortium SPIRIT : Structure for Packaging, Integrating and Re-using IP within Tool-flows
    Standard IP-XACT.
  - Exemple de document : fichier XML conforme à un schéma
  - Création d'outils exploitant ces informations

Conclusion

- SystemC
  - « Langage » de modélisation niveau système
  - Utilisation par les industriels
  - Nombre conséquent d'outils
    - ★ Dédiés (CAD Vendors)
    - ★ Provenant de C++ (GCC, gdb, gprof, valgrind, etc.)
- TLM
  - Niveau émergent de modélisation de composants électroniques
  - Utilisation de SystemC
  - Existence d'outils spécifiques TLM (Cadence, Synopsys, GreenSocs, ...)