5se05 : SocLib

Prise en main d'une plateforme virtuelle

Tarik Graba
<tarik.graba@telecom-paris.fr>
Septembre 2026

Plateforme virtuelle de modèles en SystemC

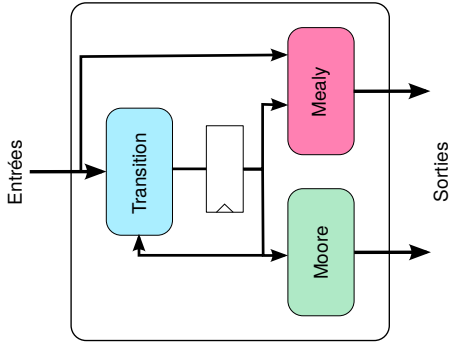
- Des modèles d'IPs
  - Processeurs (ISS: Instruction Set Simulator)
  - Interconnects, caches, RAM
  - UART, Timer, …
- Des utilitaires de simulation:
  - Terminal virtuel
  - Loader (charger une RAM à partir d'un elf)
  - ServerGDB
- Une infrastructure de compilation (`soclib-cc`)

SocLib

Particularités

- Deux types de modèles
  - CABA : Cycle Accurate/Bit Accurate
  - TLM--DT : Transaction Level Modeling with Distributed Time
- Interface VCI des modèles CABA
  - La majorité des ``composants'' utilise l'interface VCI
  - Nous utiliserons l'interface whishbone
- Style de description/codage (que nous ne sommes pas obligé de respecter)

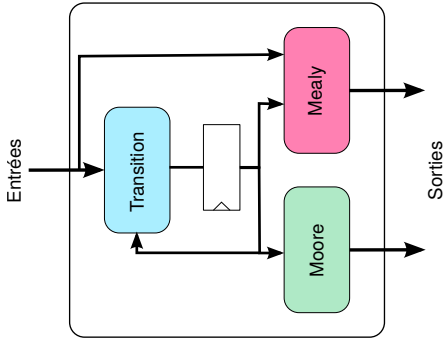
SocLib
Style de codage



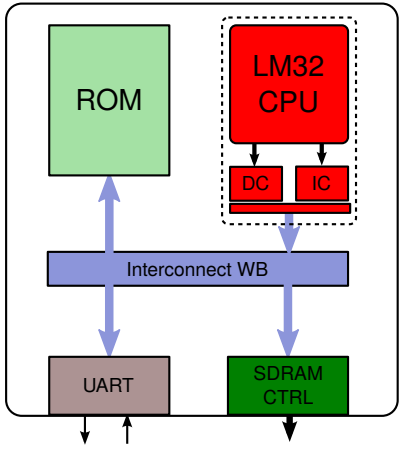
- Modéliser un composant en utilisant le minimum de ressources
  - minimiser le nombre de fonctions/threads
  - minimiser le nombre d'événements

SocLib
Style de codage

- 3 **SC_METHOD** au maximum
- `transition` :
  - Déclenchée sur le front montant de l'horloge
- `genMoore` :
  - Déclenchée sur le front descendant de l'horloge
- `genMealy` :
  - Déclenchée sur le front descendant de l'horloge et les changement des entrées

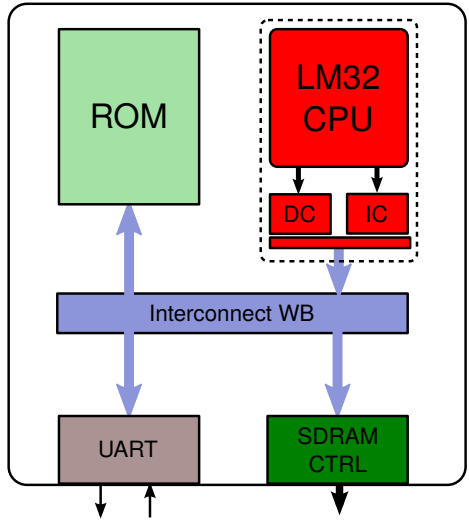


Un SoC de base
Structure de la plateforme



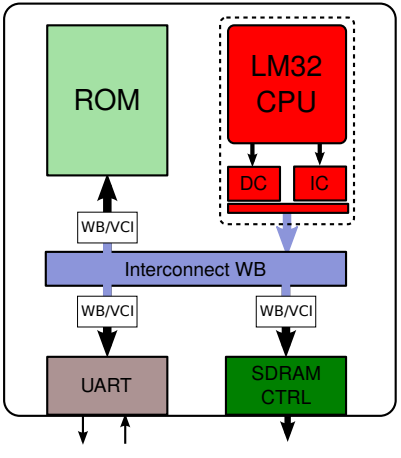
Un SoC de base
Structure de la plateforme

- Un processeur Risc 32bits (LM32)
- Une ROM, une RAM
- Une UART
- Les connecter ensemble



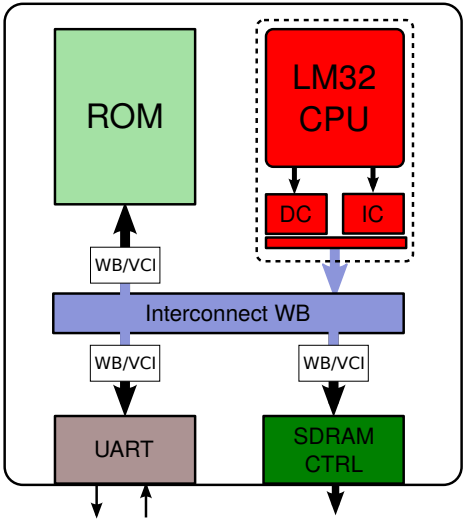
Un SoC de base

Utilisation de modèles SocLib



Un SoC de base
Utilisation de modèles SocLib

- ``wrapper'' WB $\rightarrow$ VCI
- On travaillera en Wishbone
- Les composants SocLib sont VCI



Comment l'utiliser

- configurer le PATH:
  - `export PATH=/comelec/softs/bin:$PATH`
- Définir les variables d'environnement SYSTEMC et SOCLIB
  - `source /comelec/softs/opt/soclib/TPT/env.sh`
- Télécharger l'exemple
  - `git clone git@gitlab.enst.fr:5se05/soclib_hello_world.git`

Contenu du dépôt

```
soclib_hello_world
    IPs
        …
    Makefile
    platform_desc
    segmentation.h
    soft
        lm32_sys.c
        lm32_sys.h
        main.c
        Makefile
        soclib.ld
    top.cpp
```

Hello world
logiciel du LM32

- Contenu du dossier `soft`
- logiciel destiné à être exécuté par le LM32
- Produit un exécutable `soft.elf` chargé par la plateforme

Hello world
platform_desc

- Décrit les la plateforme et les composants qu'elle utilise.
- Utilisé par l'outil `soclib-cc` pour compiler et gérer les dépendances

```
Platform('caba', 'top.cpp',
        uses = [
             Uses('caba:wb_xcache_wrapper',
                 iss_t = 'common:iss2_simhelper',
                 simhelper_iss_t = 'common:lm32'
                 ),
             Uses('caba:vci_ram'),
             Uses('caba:vci_rom'),
             Uses('caba:vci_multi_tty'),
             Uses('caba:wishbone'),
             Uses('caba:wb_interco'),
             Uses('caba:wb_slave_vci_initiator_wrapper'),
             Uses('common:elf_file_loader'),
        ],
.....
```

Hello world
Code SystemC

- `top.cpp`
- Description du contenu du SoC
- Modélise le matériel
- C'est du SystemC!

Hello world
Code SystemC

```
// Mapping table
soclib::common::MappingTable maptab(32, IntTab(8), IntTab(8), 0x80000000);

maptab.add(Segment("rom" , ROM_BASE , ROM_SIZE , IntTab(0), true));
maptab.add(Segment("ram" , RAM_BASE , RAM_SIZE , IntTab(1), true));
maptab.add(Segment("tty" , TTY_BASE , TTY_SIZE , IntTab(2), false));
```

- Décrit la carte mémoire (mapping)
- Permet d'identifier les périphériques
- Précise si les accès à cette zone passent par le cache du processeur

Hello world
Code SystemC

```
// Mapping table
soclib::common::MappingTable maptab(32, IntTab(8), IntTab(8), 0x80000000);

maptab.add(Segment("rom" , ROM_BASE , ROM_SIZE , IntTab(0), true));
maptab.add(Segment("ram" , RAM_BASE , RAM_SIZE , IntTab(1), true));
maptab.add(Segment("tty" , TTY_BASE , TTY_SIZE , IntTab(2), false));
```

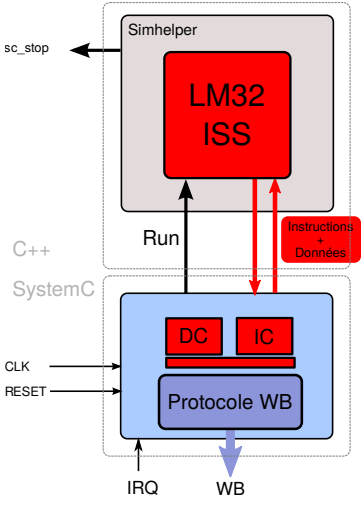
- Le bus d'adresse fait 32 bits
- Les 8 bits de poids fort de l'adresse identifient les périphériques et doivent être différents
- Le bit de poids fort définit les adresses ``cachées''

Hello world
Code SystemC

```
soclib::caba::WbXcacheWrapper
   <wb_param, soclib::common::Iss2Simhelper<soclib::common::LM32Iss <false > > >
      lm32("lm32", 0, maptab,IntTab(0), 2,128,8, 2,128,8);
```

- Les ``wrapper'' d'ISS sont des modules mutualisés dans SocLib
  - Ici vers une interface WishBone
  - Il inclue le cache
- On précise le type de processeur qu'on simule
  - Ici un LM32

Hello world
Code SystemC



- seul le CacheWrapper est un module SystemC
- il génère les transactions sur le bus
- l'iss peut être modifié pour simuler un autre CPU
- le wrapper peut être changer pour utiliser un autre protocole de bus

Hello world
Code SystemC

```
// elf loader
soclib::common::Loader loader("soft/soft.elf");

// memories
soclib::caba::VciRom<vci_param> rom("rom", IntTab(0), maptab, loader);
soclib::caba::VciRam<vci_param> ram("ram", IntTab(1), maptab, loader);
```

- Lit l'éxecutable en début de simulation
- Utilisé pour précharger les mémoires

Hello world
Code SystemC

```
// WB interconnect
//                                              sc_name     maptab  masters slaves
soclib::caba::WbInterco<wb_param> wbinterco("wbinterco",maptab, 1,      3     );

////////////////////////////////////////////////////////
//////////////////// WB Net List ///////////////////////
////////////////////////////////////////////////////////

wbinterco.p_clk(signal_clk);
wbinterco.p_resetn(signal_resetn);

wbinterco.p_from_master[0](signal_wb_lm32);

wbinterco.p_to_slave[0](signal_wb_rom);
wbinterco.p_to_slave[1](signal_wb_ram);
wbinterco.p_to_slave[2](signal_wb_tty);
```

- On précise le nombre de maîtres et d'esclaves
- On lui passe la ``mapping table''
- Les connexions se font en fonction des indices définis dans la mapping table
- L'arbitrage est ``round-robin''

Hello world
Code SystemC

```
// WB interconnect signals
soclib::caba::WbSignal<wb_param> signal_wb_lm32("signal_wb_lm32");
soclib::caba::WbSignal<wb_param> signal_wb_ram("signal_wb_ram");
soclib::caba::WbSignal<wb_param> signal_wb_rom("signal_wb_rom");
soclib::caba::WbSignal<wb_param> signal_wb_tty("signal_wb_tty");
```

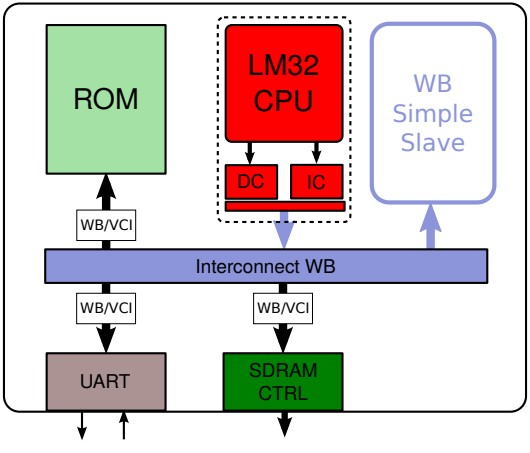
- Encapsule l'ensemble des signaux Wishbone
- On peut les ajouter à une ``simulation trace''
- On peut imprimer leur valeur (`cout << `)

Hello world
Code SystemC

```
sc_signal<typename wb_param::wb_data_t> MWDAT; // masters -> slave
sc_signal<typename wb_param::wb_data_t> MRDAT; // slave -> master
sc_signal<typename wb_param::wb_add_t>  ADR;   // master output address
sc_signal<bool>                        ACK;   // Acknowledge from slave
sc_signal<bool>                        CYC;   // cycle valid
sc_signal<bool>                        ERR;   // error from slave
sc_signal<bool>                        LOCK;  // lock request
sc_signal<bool>                        RTY;   // retry from slave
sc_signal<typename wb_param::wb_sel_t> SEL;   // BE
sc_signal<bool>                        STB;   // commande valid
sc_signal<bool>                        WE;    // write enable
```

- Encapsule l'ensemble des signaux Wishbone
- On peut les ajouter à une ``simulation trace''
- On peut imprimer leur valeur (`cout << `)

Personnaliser
Ajouter un esclave



Personnaliser
Ajouter un maître