École Polytechnique de Montréal

Département de Génie Informatique

INF3610

Automne 2013

Laboratoire #3

Modélisation d'un système embarqué avec SystemC

---

1. Objectif

L'objectif de ce laboratoire est de comprendre la méthodologie de conception pour systèmes embarqués basée sur la librairie SystemC.

Plus précisément, les objectifs du laboratoire sont :

- S'initier à la librairie SystemC
- Se familiariser au développement de SoC avec une méthodologie de conception haut niveau
- Mettre en pratique les étapes de raffinement
- Connaître les différents niveaux d'abstraction

Ce laboratoire consiste au développement et au raffinement d'une application composée de trois modules à l'aide de SystemC. Premièrement, vous allez développer chaque module à un haut niveau d'abstraction *Un-Timed Functional.* Par la suite, vous allez modifier le contenu des modules pour avoir un comportement plus détaillé à un niveau d'abstraction plus bas. Vous allez donc raffiner vos modules au niveau *Approximately Timed.*

2. Mise en contexte

Lors de la conception d'un système embarqué, il y a plusieurs étapes à respecter avant d'obtenir un produit final. Certes, il est possible de directement bâtir l'application sur une puce de développement avec un langage RTL, mais cette avenue est souvent problématique, car la source des problèmes peut provenir d'une multitude de facteurs (i.e mauvaise logique du code applicatif, difficulté de développement à bas niveau, difficulté à changer l'architecture, difficulté à tester les composants matériels, etc.) ce qui rend le débogage ardu. Dans le but d'accélérer les phases de développement des systèmes embarqués, la modélisation à haut niveau du système à l'aide de la librairie SystemC est une étape importante puisqu'elle permet de valider ou d'infirmer les spécifications, de corriger les bogues applicatifs, de faire une vérification fonctionnelle, entre autres. De plus, il est plus facile de déceler et corriger les problèmes à cette étape qu'aux étapes subséquentes.

SystemC est une librairie C++ qui permet la modélisation du matériel à plusieurs niveaux d'abstraction, incluant le RTL (*Register Transfert Level*) à très bas niveau ainsi que le UTF (*Un-Timed Functional*) à très haut niveau. Ceci permet aux développeurs de garder le même langage d'un bout à l'autre du flot de conception. La première étape du flot de conception consiste à décrire les modules sans détails concernant l'architecture, sans horloge et sans les détails de communication, à un haut niveau d'abstraction. Le but de cette étape est de faire une vérification fonctionnelle du système, et ainsi valider l'algorithme de calcul de chaque module. En appliquant successivement les étapes de raffinement, il est ensuite possible d'obtenir un modèle de niveau intermédiaire nommé TF, puis ensuite un modèle bas niveau communément appelé *cycle accurate* ou RTL. Dans ce laboratoire, vous allez travailler sur les modèles UTF et TF.

La modélisation à haut niveau est une étape qui apporte beaucoup d'avantages lors du processus d'élaboration des systèmes embarqués.

3. Conception du système

La première étape de ce laboratoire consiste à implémenter un modèle SystemC au niveau d'abstraction **UTF** qui exécute le jeu du bonhomme pendu.

Le circuit est composé de 4 modules :

- Une mémoire de données
- Un module qui prend en entrée les lettres du joueur

- Un module qui sert de contrôle
- Un module d'affichage

La deuxième étape est de raffiner le module lecteur, le module de recherche de chaînes de caractères ainsi que la communication entre ces deux modules au niveau d'abstraction **AT**.

3.1. Description des modules UTF

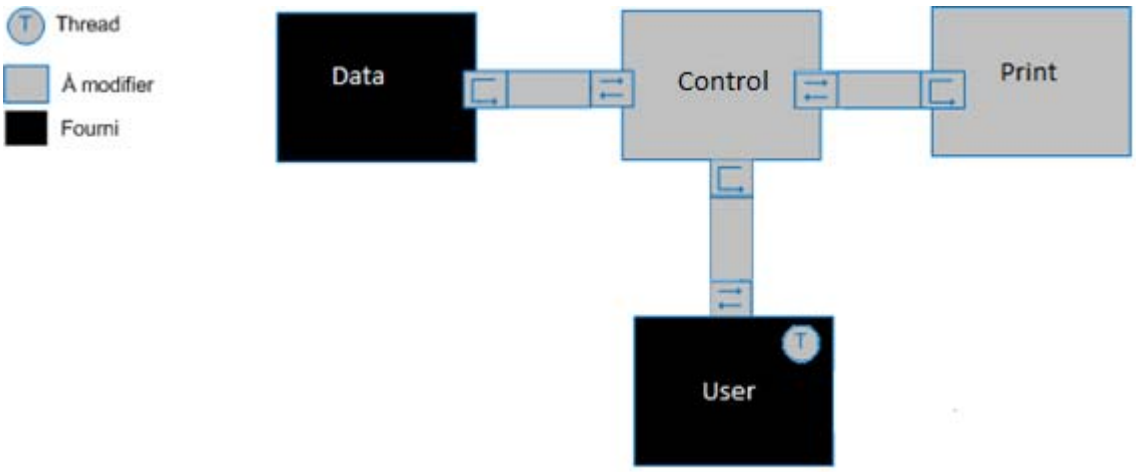

Figure 1 Schéma du circuit pour le jeu bonhomme pendu au niveau *Un-Timed functional*

Module Control

Type	Nom	Description
sc_port<InterfacePrint>	printPort	Port pour le module d'affichage
sc_port<InterfaceData>	dataPort	Port pour la mémoire de données

Le module *Control* gère la communication entre les trois autres modules. Il doit accepter les valeurs en entrée du module Utilisateur puis vérifier que la lettre n'apparaît pas dans le mot recherché. Il doit mettre au courant l'usager en affichant les nouvelles valeurs. C'est grâce aux deux ports *printPort* et *dataPort* que la communication sera possible avec les modules *Print* et *Data*. Ces ports supportent les fonctions décrites dans les interfaces *InterfacePrint* et *InterfaceData*. De plus, ce module hérite de l'interface *InterfaceRead*. Ceci permettra au module *User* de demander des requêtes au module *Control*.

(Voir l'exemple de comment implémenter une interface et de comment faire le branchement d'un sc_port dans l'annexe 1)

- Fonctionnement interne :

  - Initialisation de la partie dans *void StartGame()*
  - Tester si une lettre est présente ou non dans le mot dans *bool TryNewLetter()*

Module Print

Ce module ne sert que d'interface avec l'usager. La communication entre ce module et celui de *Control* se fera par *InterfacePrint*. Il affichera à chaque entrée du joueur, une mise-à-jour du nombre d'erreurs et les lettres trouvées jusqu'à présent. (Voir Spécification Internes pour des exemples d'affichage)

- Fonctionnement interne :

  - Afficher le nombre de caractères que le mot contient
  - Afficher la séquence des lettres trouvées par l'utilisateur
  - Afficher le nombre d'erreurs
  - Appeler les fonctions *PrintWin()* et *PrintLose()* au moment approprié

3.2. Description des modules AT

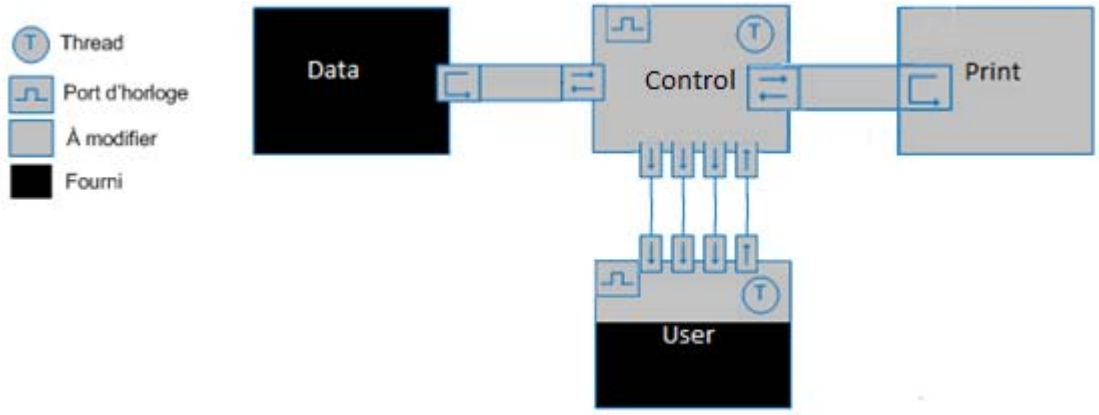

Figure 2 Schéma du circuit du jeu au niveau Approximately Timed

Module Control

Type	Nom	Description
sc_port<InterfacePrint>	printPort	Port pour le module d'affichage
sc_port<InterfaceData>	dataPort	Port pour la mémoire de données
sc_in_clk	clk	Horloge
sc_in<bool>	startGamePort	Commencer une partie
sc_in<bool>	requestLetterPort	Prêt à gérer une nouvelle lettre
sc_in<char>	incomingLetterPort	Lettre provenant du module User
sc_out<bool>	endGamePort	La partie est terminée
sc_out<bool>	requestStartPort	Avertir que l'initialisation a fonctionné
sc_out<bool>	ackPort	Accusé de réception

La logique à l'intérieur du module ne devrait pas changer de beaucoup. Cependant la communication entre les modules *User* et *Control* est plus raffinée. La synchronisation s'effectue grâce au protocole simple de *handshaking*. La communication avec le Module *Print* et Data se fait toujours au travers des interfaces *InterfaceData* et *InterfacePrint*. Une horloge et plusieurs signaux ont été ajoutés dans le module. **(Voir exemple handshaking dans l'annexe 1)**

- Fonctionnement interne :

- Attendre que la commande de départ soit reçue
- Choisir au hasard un mot contenu dans la base de données
- Prendre ce mot caractère par caractère
- Afficher les informations du mot choisis à la console
- Attendre une lettre de l'usager
- Vérifier si la lettre est dans le mot sélectionné
- Réafficher les informations à la console
- Les trois dernières étapes en boucle jusqu'à ce que le mot soit trouvé ou que le maximum d'erreurs soit atteint

Module User

Type	Nom	Description
sc_in_clk	clk	Horloge
sc_out<bool>	startGamePort	Commencer une partie
sc_out<bool>	requestLetterPort	Prêt à gérer une nouvelle lettre
sc_out<char>	incomingLetterPort	Lettre provenant de l'utilisateur
sc_in<bool>	endGamePort	La partie est terminée
sc_in<bool>	requestStartPort	Avertir que l'initialisation a fonctionné
sc_in<bool>	ackPort	Accusé de réception

Le module est à moitié complété. Vous devez modifier la communication en la raffinant au niveau *Approximately Timed*. Vous devez donc synchroniser ce module avec le module *Control*. Donc le fonctionnement interne de ce module sera selon les informations nécessaires pour la bonne exécution du module *Control*.

Module Print

Selon votre implémentation de votre module *Control*, votre module *Print* peut demeurer le même que pour le niveau d'abstraction UTF. La communication entre ce module et celui de *Control* se fera par l'*InterfacePrin*t. Il affichera à chaque entrée du joueur, une mise-à-jour du nombre d'erreur et des lettres trouvées jusqu'à présent.

4. Spécifications du système

Il ne doit pas y avoir de type *string* à l'intérieur des modules *Control* et *Print* ainsi que pour les communications entre les modules. Il y a deux constantes de déclarées dans le main.cpp que vous devez obligatoirement utilisées (`DICTIONARYSIZE et NUMBEROFERRORSMAXIMUM`). Puisque nous utiliserons un autre fichier de database.txt et que nous modifierons le nombre d'erreurs possible, nous ne voulons pas avoir à modifier votre code. Pour simplifier la tâche, on ne peut compléter qu'une partie par exécution. Voici des exemples d'affichages pour le premier mot (aiguille) :

Figure 1- Exemple bon affichage

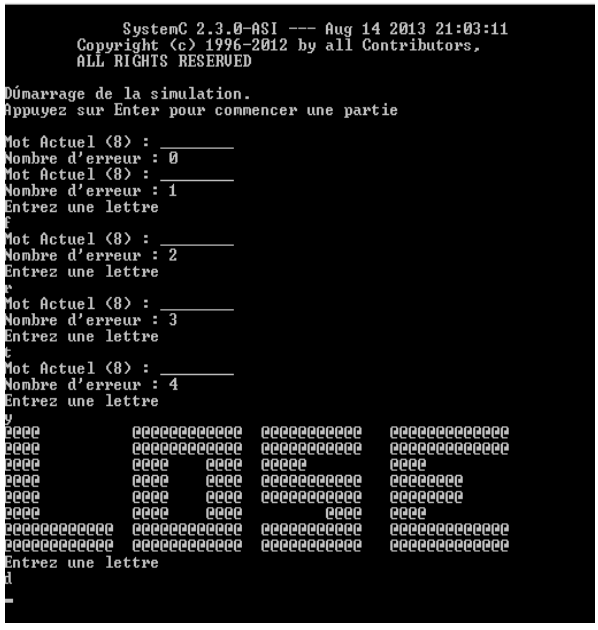

Figure 2-Exemple mauvais affichage

5. Travail à réaliser

Vous devez compléter les fichiers :

- Control.h et Control.cpp
- Print.h et Print.cpp
- Main.cpp
- User.h et User.cpp (Uniquement pour le niveau d'abstraction AT)

Le code dans le répertoire *code/UTF* doit implémenter le niveau d'abstraction UTF. Le code dans le répertoire *code/AT* doit implémenter le niveau d'abstraction AT. Si vous voulez tester votre système avec d'autres valeurs, vous pouvez modifier le fichier « Database.h ».

Dans le but de vous simplifier la tâche, un projet Visual Studio vous est fourni. Cependant, vous devez vous assurer d'être en mode Debug, sinon il y a des risques de rencontrer certains problèmes (Le mode Release n'a pas été configuré pour le projet). Si vous le voulez, vous pouvez utiliser l'ancien compilateur MinGW fournit dans les anciennes versions du laboratoire. Si ce dernier est utilisé, un *makefile* a été ajouté au code de départ pour compiler votre projet. L'outil *MinGW* se trouve dans le répertoire *C:\SpaceCodesign\msys\msys.bat* et lancer simplement la commande *make* dans le répertoire du laboratoire où *makefile* est situé. MinGW est fortement déconseillé puisque la correction se fera avec le projet de départ Visual Studio, mais reste quand même une possibilité.

6. Évaluation

Vous disposez de 2 semaines pour effectuer ce laboratoire (1 séance de laboratoire seulement). Vous devez envoyer votre travail à votre chargé de laboratoire par courriel avant le jeudi 14 novembre 2013 à 23h59. Vous devez remettre un fichier .zip ou .rar clairement identifié contenant uniquement les deux dossiers avec les 5 fichiers mentionnés dans la section Travail à réaliser (AT et UTF). Si vous avez modifié d'autre(s) fichier(s), les inclure et ajoutez un ReadMe.txt qui explique vos modifications supplémentaires. Le nom du dossier compressé doit être présenté sous la forme : Lab3_INF3610_A13_Matricule1_Matricule2.zip. Au début de la première séance du TP4, le chargé posera deux questions que vous devrez répondre individuellement.

7. Barème de correction

• Partie UTF (code + exécution)	/2
• Partie AT (code + exécution)	/2
• Respect des spécifications du système	/1
• Question #1	/2
• Question #2	/2
TOTAL	/10

Bon travail !

Annexe 1

```
// Variable

sc_signal<bool> sEnable;


// On effectue le branchement

Instance_cd.enable(sEnable);

Instance_auto.enable(sEnable);
```

Exemple de branchement

Interface_audio.h	cd.h
…	…
class Interface_audio : public virtual sc_interface	class cd : public sc_module, public Interface_audio
{	{
…	…
private:	private:
virtual void play() = 0;	virtual void play() ;
virtual void stop() = 0;	virtual void stop() ;
}	…
	};

Exemple d'implémentation d'une interface

auto.h	Main.cpp

…
class auto : public sc_module
{
public:
sc_port<Interface_audio> cdPort;
…
}
	…
	main
	{
	…
	Auto instance_auto();
	Cd instance_cd();
	Instance_auto.cdPort (instance_cd);
	…
	};

Exemple branchement sc_port

auto.cpp	cd.cpp

…
// Envoi de l'adresse
address.write(addr);
enable.write(true);
 	
// Synchronisation
do{
wait(clk->posedge_event()
}while(!ack.read());
 	
// Poursuite du traitement
enable.write(false);
…
	…
	do{
	wait(clk->posedge_event()
	}while(!enable.read());
	
	// On lit l'adresse
	addr = address.read();
	
	// Synchronisation
	ack.write(true);
	
	…

Exemple de synchronisation (*handshaking*)