Hes·so VALAIS WALLIS
Haute Ecole Spécialisée de Suisse occidentale
Fachhochschule Westschweiz
University of Applied Sciences Western Switzerland

Domaine Sciences de l'ingénieur
Rte du Rawyl 47
CH- 1950 Sion 2
Tél. +41 27 606 85 11
Fax +41 27 606 85 75
info@hevs.ch

www.hevs.ch

Filière Systèmes industriels

Orientation Infotronics

Diplôme 2012

Charles Papon

Hybrid Systems

Professeur	Medard Rieder
Expert	Rico Steiner

Sion, le 9 juillet 2012

HES-SO Valais

SI	TV
X	X

Données du travail de diplôme
Daten der Diplomarbeit

FO 1.2.02.07.AB
pof/31/01/2009

☒ FSI ☐ FTV	Année académique / *Studienjahr* **2011/2012**	No TD / *Nr. DA* **it/2012/25**
Mandant / *Auftraggeber* ☒ HES—SO Valais ☐ Industrie ☐ Etablissement partenaire *Partnerinstitution*	Etudiant / *Student* **Charles Papon** Professeur / *Dozent* **Medard Rieder**	Lieu d'exécution / *Ausführungsort* ☒ HES—SO Valais ☐ Industrie ☐ Etablissement partenaire *Partnerinstitution*
Travail confidentiel / *vertrauliche Arbeit* ☐ oui / ja [1] ☒ non / nein	Expert / *Experte (données complètes)* **Steiner Rico** Turtig 13 | 3942 Raron	

Titre / *Titel*

Hybrid Systems

Description et Objectifs / *Beschreibung und Ziele*

Des systèmes hybrides (matériel programmable avec un processeur ou un contrôleur) vont jouer un rôle considérable dans le futur des systèmes embarqués. Un terme important autour des systèmes hybrides est le Co-Design. L'institut Systèmes industriels (ISI) a réalisé un nombre de travaux de recherche concernant le Co-Design qui ont même été publiés à un niveau international. Après la mise en place d'une méthodologie de Co-Design pendant le travail de semestre, l'objectif est à présent d'essayer de réaliser un environnement simple de Co-Design pendant le travail de bachelor sur la base du module SmartFusion de la société ACTEL.

Les travaux suivants devront être réalisés :

— Brève révision de la méthodologie selon les critiques obtenues pour le travail de semestre
— Définition des outils pour former l'environnement de Co-Design
— Réalisation d'un prototype de l'environnement de Co-Design
— Conception d'une application pour tester l'environnement de Co-Design
— Réalisation de l'application de test à l'aide de l'environnement
— Etablissement de la documentation technique
— Rédaction d'un rapport final

Délais / *Termine*

Attribution du thème / *Ausgabe des Auftrags:*
14.05.2012

Exposition publique / *Ausstellung Diplomarbeiten:*
31.08.2012

Remise du rapport / *Abgabe des Schlussberichts:*
09.07.2012 | 12h00

Défense orale / *Mündliche Verteidigung:*
Semaine / *Woche 36*

Signature ou visa / *Unterschrift oder Visum*

Responsable de l'orientation
Leiter der Vertiefungsrichtung: .....

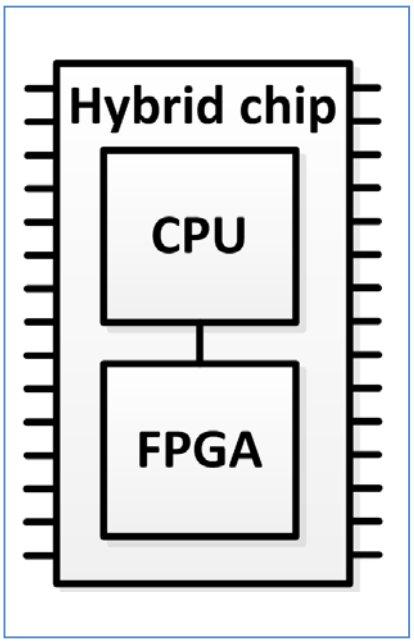
[1] Etudiant/*Student*: .....

[1] *Par sa signature, l'étudiant-e s'engage à respecter strictement la directive et le caractère confidentiel du travail de diplôme qui lui est confié et des informations mises à sa disposition.*
Durch seine Unterschrift verpflichtet sich der Student, die Richtlinie einzuhalten sowie die Vertraulichkeit der Diplomarbeit und der dafür zur Verfügung gestellten Informationen zu wahren.

Rapport reçu le / *Schlussbericht erhalten am* ..... Visa du secrétariat / *Visum des Sekretariats* .....

Hes·so VALAIS WALLIS
Haute Ecole Spécialisée de Suisse occidentale
Fachhochschule Westschweiz
University of Applied Sciences Western Switzerland

Hybrid Systems



Diplômant/e Charles Papon

Objectif du projet

Ce projet de diplôme a pour but de développer une méthodologie et une chaine d'outils pour concevoir et implémenter des systèmes combinant logique programmable et processeur entièrement à partir d'un modèle du système en notation UML. L'objectif final étant d'obtenir une tool-chain complète avec des génératrices de code pour FPGA et microprocesseur.

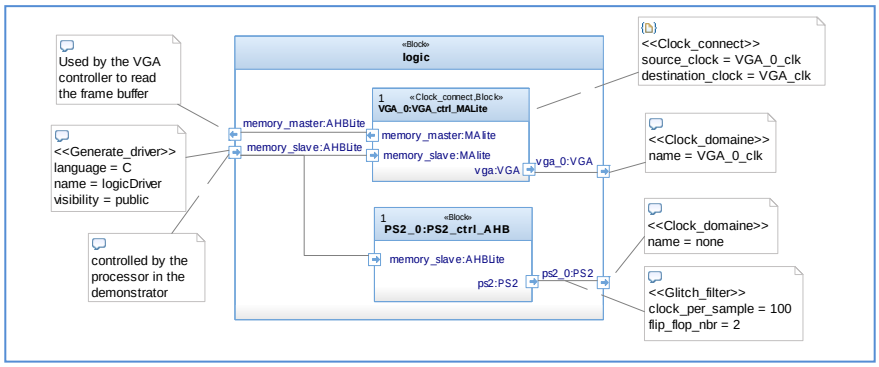
Méthodes | Expériences | Résultats

Ce projet s'est principalement porté sur la façon de modéliser un système complexe pour pouvoir en déduire un modèle pour le software et pour le hardware tout en automatisant en même temps la génération des interfaces entre matériel et logiciel.

A l'aide des règles de modélisation établies durant le projet de semestre, un démonstrateur représentant un ordinateur primitif avec un processeur, un clavier et un écran et exécutant trois taches en parallèle a été développé.

Une bonne partie d'une génératrice de code et interfaces a de même été réalisée.

Au final ce projet a permis de démontrer qu'une approche modèle pour concevoir et implémenter des systèmes hybrides a des grands avantages tels que l'abstraction du bas niveau, l'automatisation du traitement des interfaces entre matériel et logiciel et la réduction des probabilités d'obtenir des codes source comportant des bugs.



La figure ci-dessus montre un exemple du modèle qui a été conçu et implémenté pour le démonstrateur selon les règles de modélisation établies.

Travail de diplôme
| édition 2012 |

Filière
Systèmes industriels

Domaine d'application
Infotronics

Professeur responsable
Medard Rieder
medard.rieder@hevs.ch

Partenaire
HES-SO Valais

HES-SO Valais
Route du Rawyl 47
1950 Sion

Tél. 027 606 85 11
URL www.hevs.ch

SOMMAIRE

1. INTRODUCTION

Ce projet traite de la problématique du développement de systèmes liant une partie software et une partie hardware. Ci-dessous une représentation simplifiée du processus de développement:

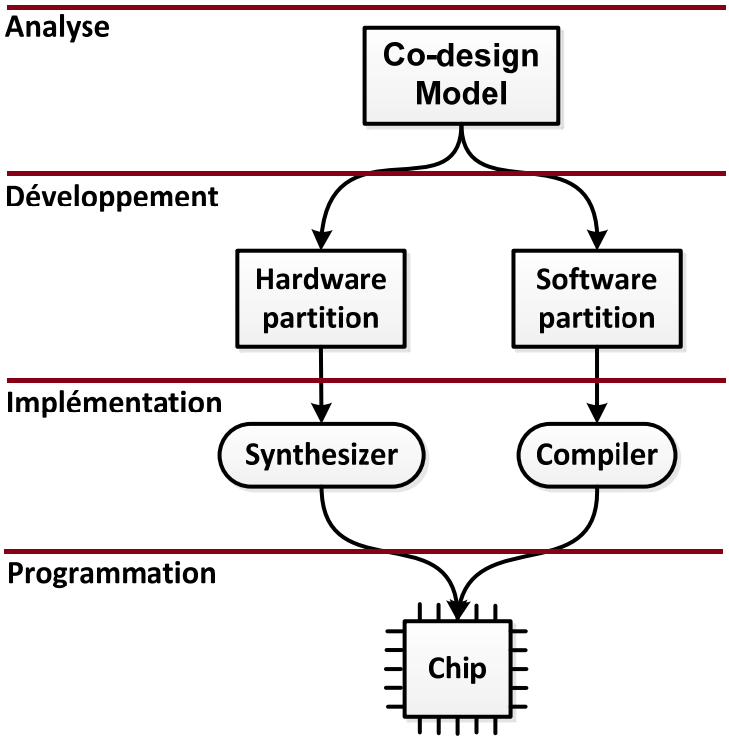


Figure 1: Processus de développement

La phase d'analyse permet d'identifier les majeures problématiques liées au système ainsi que d'identifier si telle ou telle partie trouve sa meilleure place dans une transition analyse-design hardware ou software.

Pendant la phase de conception, les différentes parties du système sont implémentées à la fois pour le hardware et le software.

L'implémentation fait appel aux outils des fabricants pour générer les fichiers bytecode.

Ce projet traite en majeure partie de la conception, plus particulièrement de la partition hardware ainsi que de la communication entre l'interface hardware et software. Une grande partie de l'établissement des règles de modélisation hardware a été établie durant le projet de semestre « Hybrid Systems » [1]. Je vous recommande de le consulter avant la lecture de ce document. Le glossaire résume également les termes les plus importants en faisant des analogies entre le monde hardware et software.

2. TOOLCHAIN

En l'état actuel du projet, les outils nécessaires au développement d'un système co-design à partir d'un modèle sont :

- Un éditeur de modèle UML
- Une génératrice de code C/C++
- Une génératrice de code VHDL
- Un éditeur de texte pour implémenter la logique des blocs VHDL « Leaf » (non composite) et éventuellement pour éditer le code C/C++
- Un synthétiseur VHDL
- Un compilateur C/C++
- Les programmateurs de chip

Ci-dessous, un diagramme représentant la toolchain partant de l'analyse du système jusqu'à la programmation des chips utilisés.

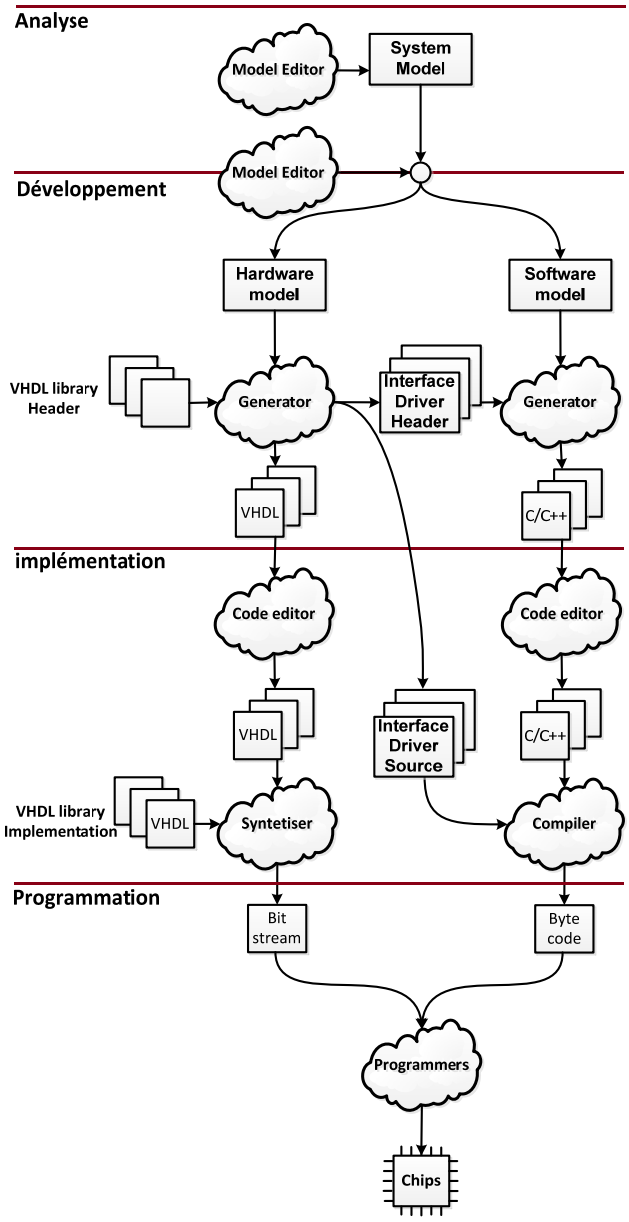


Figure 2: Toolchain co-design

3. RAPIDE RESUME DES REGLES DE MODELISATION

Ce chapitre va brièvement résumer les règles de modélisation majeure du projet. La grande majorité de ces règles provient du projet de semestre « Hybrid Systems » [1] .

3.1 BLOCS ET INSTANCES DE BLOCS

Il y a la possibilité de créer des nouveaux blocs pour définir l'architecture du système, chaque bloc peut contenir plusieurs instances d'autres blocs.

3.2 TYPE DE DONNEE

Les données manipulables peuvent être d'un type de base (bool, data, signed, unsigned, integer) ou des structures de données plus élaborées elles-mêmes composées de sous-données.

3.3 COMMUNICATION

Chaque bloc peut avoir plusieurs ports pour communiquer avec l'extérieur.
Ces ports peuvent être de natures très différentes les unes des autres.

- Data port : Transmission des données en permanence (câblage)
- Stream port : Transmission des données par paquet (flux discontinu)
- Memory access port : Transmission sur des bus mémoires (adresse donnée arbitration)
- Functional port : Transmission à l'aide d'appel de fonction définie par des interfaces

Les ports peuvent être connectés entre eux lorsque cela a une signification, par exemple :

- entre 2 data port (câblage direct)
- entre 2 stream port
- entre 2 Functional port

Les memory access port sont particuliers, car ils peuvent se connecter à n'importe quel autre type de port.
Par exemple si un memory access port représente le bus mémoire d'un processeur et qu'on le connecte à un stream port, le processeur pourra émettre des paquets de données sur le stream port.

3.4 EXEMPLE

Voici plusieurs petits exemples de modélisation utilisant ces règles de modélisation.

Un contrôleur SPI :

- Le port contrôle est un functional port qui réceptionne les commandes
- Le port spi est un data port qui doit être connecté au bus SPI à contrôler

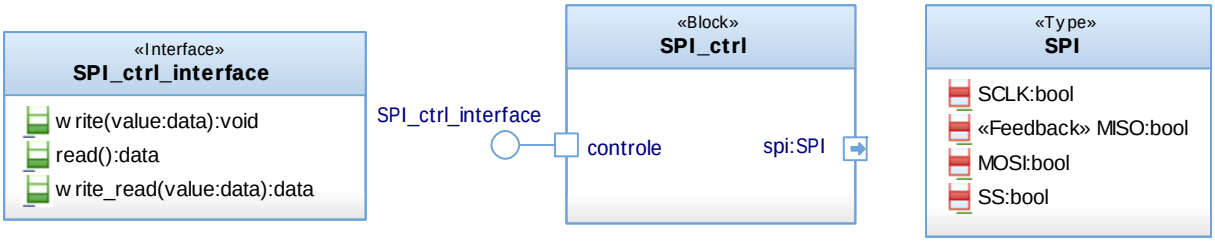


Figure 3: Contrôleur SPI

Un contrôleur I2S accessible par un bus mémoire :

- Le bus qui contrôle le système arrive par le port memory_access
- Ce bus prend le contrôle du port audio_stream
- Audio_stream est un stream port qui réceptionne les samples audio que le contrôleur doit transmettre en I2S
- Les ports I2S_master sont des data port qui transportent les bits d'arbitration et de données de l'interface I2S

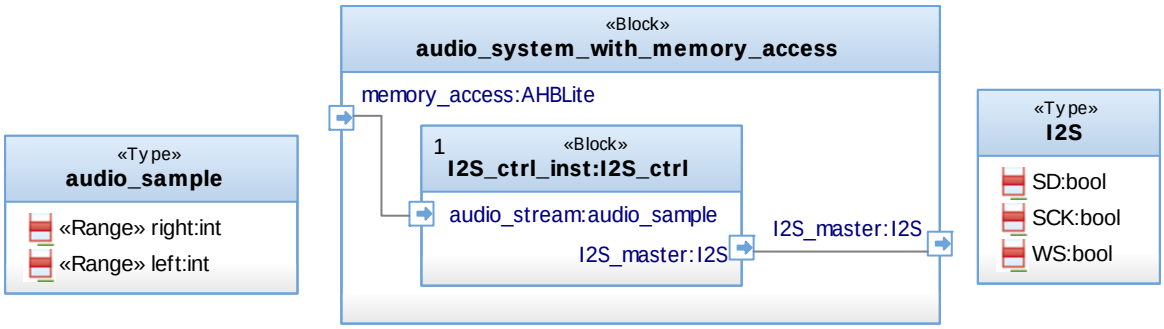


Figure 4: Contrôleur I2S accessible par un bus

4. OPTIMISATION DES REGLES DE MODELISATION EXISTANTES

Dans ce chapitre seront abordées les modifications des règles de modélisation établies durant le travail de semestre [1].

4.1 PROFIL

Le profil s'est enrichi d'une fonctionnalité intéressante qui est de donner accès à toute une gamme de blocs/types de base standardisées.

4.1.1 TYPE ET MEMORY ACCESS

Toute une gamme d'interfaces couramment utilisées pourrait être prédéfinie dans le profil, des interfaces telles que :

- SPI / I2C / UART / i2S / ect …

La même réflexion est également valable pour les bus mémoires :

- AHBLite / APB / Avalon / Wishbone / ect …

La prédéfinition de ces types / bus mémoire permettrait de rajouter automatiquement des fonctions permettant d'émettre des trames / commandes dans les test-bench générés par la génératrice de code.

4.1.2 BLOC (IP)

De base la génératrice a besoin, dans certains cas, de blocs spécifiques pour connecter des éléments du modèle.
Ces blocs de base sont :

- Glitch_filter
- Stream_buffer_single_clock
- Stream_buffer_dual_clock

Ces blocs ne sont pas implémentés par la génératrice étant donné leur bas niveau et les variations qu'ils peuvent subir en fonction de la technologie de FPGA utilisée. Ils doivent donc être implémentés à la main si leur version „universelle" n'est pas correctement interprétée (bloc ram) par le synthétiseur VHDL.

Une implémentation du stream_buffer_single_clock a été effectuée pour le démonstrateur.

Il est également envisageable d'insérer dans le profil toute une gamme de blocs IP couramment utilisés pour simplifier le développement de systèmes et éviter de réinventer la roue. Des blocs tels que :

- Contrôleur d'interface de communication
  - SPI / I2C / UART / Ethernet / I2S / Ect …
- Contrôleur d'interface mémoire
  - SDRAM / SSRAM / Flash serial / Flash parallèle / Ect ..
- Unités arithmétiques
  - Multiplication / Division / Opérateur virgule flottante

Un grand atout dans l'intégration de ces librairies au profil serait que leur interface de communication serait standardisée grâce au stream port / memory access port / functional port dont voici quelques illustrations :

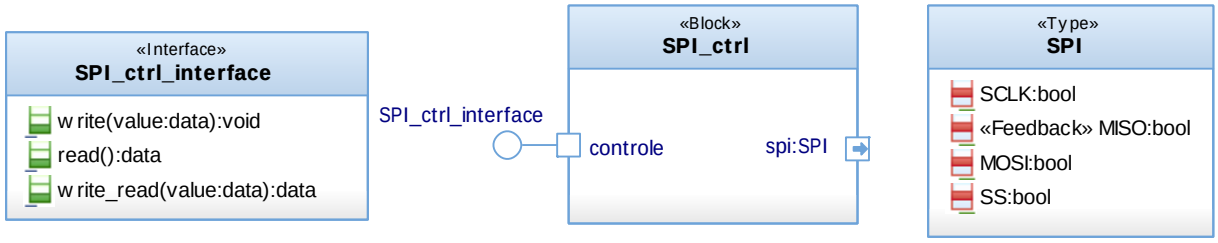


Figure 5: Contrôleur SPI

Ci-dessus un contrôleur SPI fournissant les fonctions de base telle qu'écriture-lecture sur le bus SPI.

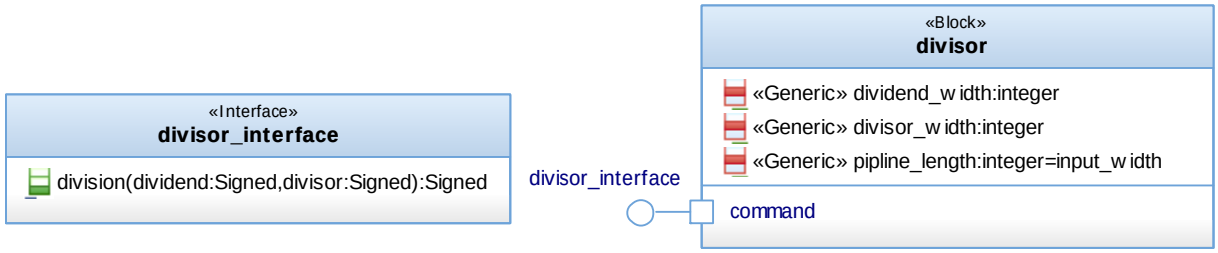


Figure 6: Unité de division

Une unité de division hardware comme celle présentée ci-dessus pourrait par exemple être contrôlée par un processeur en vue d'augmenter ses capacités de calcul. Dans d'autres cas, elle pourrait être directement utilisée par du hardware.

4.2 ÉCRITURE

Lors du projet de semestre [1, p12], le séparateur à utiliser pour accéder à un attribut d'une class en VHDL était '__'.
Un test avait d'ailleurs été effectué sur un outil pour en vérifier le bon fonctionnement.
Malheureusement, la norme pure du VHDL ne supporte pas de nom comportant 2 '-' de suite.
La nouvelle règle pour la remplacer est désormais que :

- Le séparateur pour accéder à un attribut d'une class est '_'.

4.3 FILTRE DE CONNEXION

La méthode retenue durant le projet de semestre [1, p29] pour connecter des attributs d'une structure de données était de configurer le nom de End1 et End2 sur un « link ».
Cette configuration n'étant pas exportée dans le modèle XMI 2.1, il a fallu trouver une alternative.

L'alternative retenue est d'appliquer le stéréotype « Connection_filter » qui possède les deux tags de configuration.
Tag :

- Source : le filtre devant être appliqué sur la source de données
- Destination : le filtre devant être appliqué sur la destination des données

Ci-dessous un exemple ou un filtre de connexion est utilisé pour relier 2 data ports de types différents.

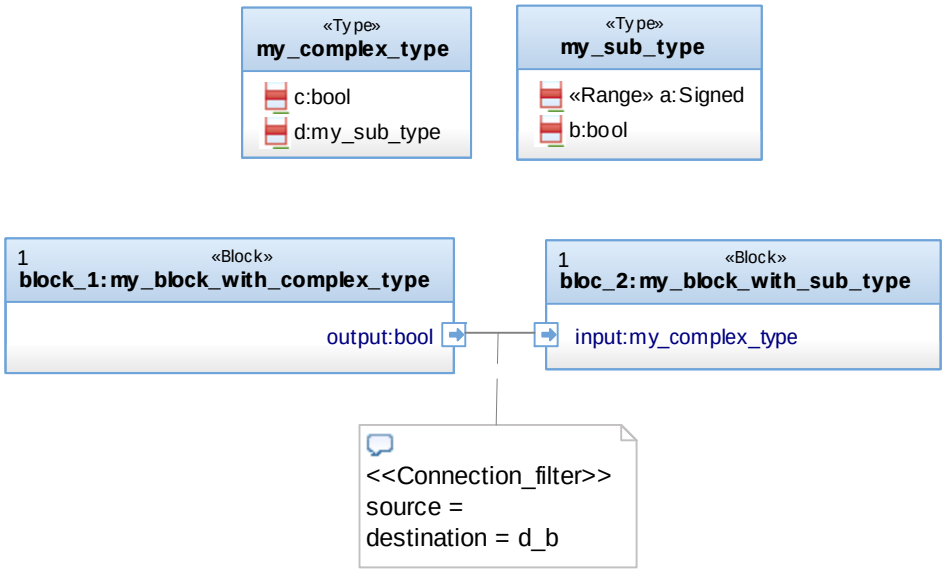


Figure 7: Utilisation de <<Connection_filter>>

L'exemple ci-dessus sera traduit en :

Bloc2_input_d_b <=bloc1_output ;

4.4 GLITCH FILTRE

Dans le rapport de semestre, pour le hardware lorsqu'un lien reliant deux data port provenant de deux clocks de domaines différents, une adaptation de clocks était automatiquement réalisée à l'aide de deux bascules chaînées.

La gestion de ce cas est maintenant améliorée, car il y a désormais la possibilité de stéréotyper le lien avec « Glitch_filter » qui permet de customiser ce filtrage de glitch.

Tag :
Clock_per_sample : Spécifie chaque combien de coup de clocks une mesure est effectuée
Flip_flip_nbr : Spécifie combien de bascules doivent être placées pour éviter des états métastables.
Stable_sample_nbr : Spécifie combien de sample d'affilée doivent être à la même valeur afin de valider cette valeur en sortie du filtre.

4.5 GENERATION DE DRIVER

En réalisant le démonstrateur, il s'est avéré que la méthode de génération de driver choisie durant le projet de semestre devait être corrigée et complétée.

4.5.1 LOW LEVEL

Chaque memory access port stéréotypés avec « Generate_driver » sont retranscrits en une class C++. Lorsque ce port est connecté à un data port, stream port, memory access port aucune nouvelle class n'est créée contrairement à ce qui avait été décidé durant le travail de semestre et l'accès y est donné directement à travers des fonctions dans la class du memory access port « Generate_driver ».

Si un memory access port « Generate_driver » est connecté à un memory access port « Generate_driver » d'un sous-bloc, ce premier fait une instance du driver du second.

Ce changement est dû au fait qu'avec l'ancienne règle de modélisation, un trop grand nombre de class était généré sans grand intérêt.

Il est également possible de donner un nom au driver grâce au tag name de « Generate_driver ».

Ci-dessous l'exemple donné dans le rapport du projet de semestre, mais traduit avec les nouvelles règles et en C.

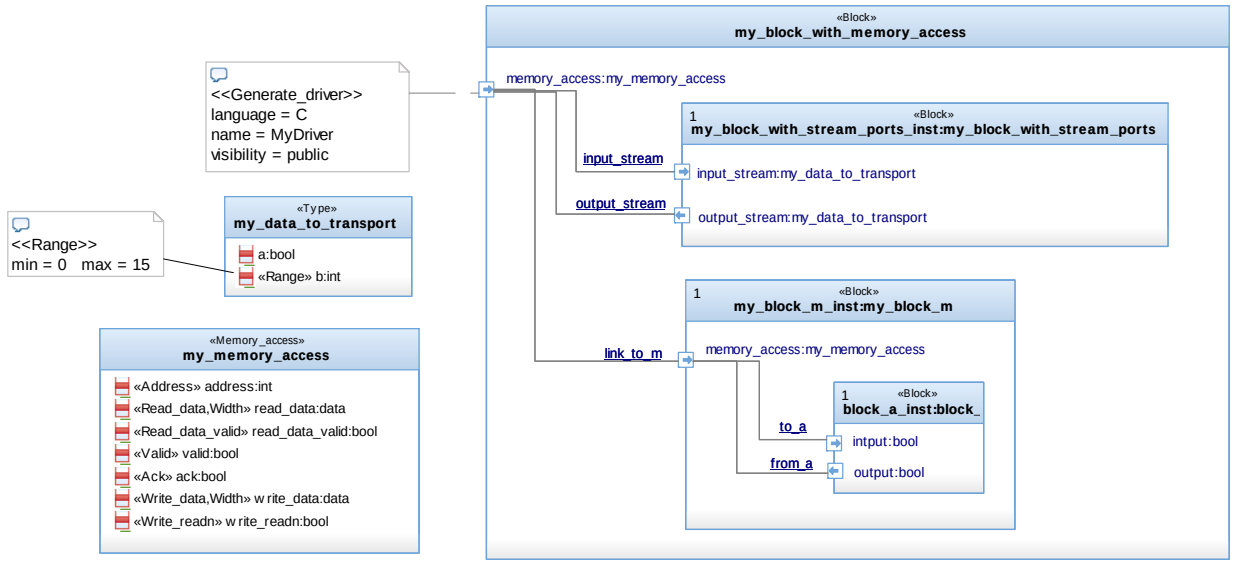


Figure 8: Exemple de génération de driver bas niveau (modèle)

```
1
2  #ifndef MyDriver_H
3  #define MyDriver_H
4
5  #include "typeDriver.h" //Has type dedclared in model
6
7  typedef struct MyDriver
8  {
9    Hardware hardware;   //Hardware address pointer
10 }MyDriver;
11
12
13 void myDriver_init(MyDriver *this,Hardware hardware); //Low level init
14
15 //Input stream
16 UINT32 MyDriver_input_stream_get_state(MyDriver *this);
17 void MyDriver_input_stream_push(MyDriver *this,my_data_to_transport *data);
18
19 //Output stream
20 UINT32 MyDriver_output_stream_get_state(MyDriver *this);
21 my_data_to_transport MyDriver_output_stream_pop(MyDriver *this);
22
23 //To_a
24 void MyDriver_link_to_m_set_to_a(MyDriver *this,UINT32 value);
25
26 //From_a
27 UINT32 MyDriver_link_to_m_get_from_a(MyDriver *this);
28
29 #endif
```

Figure 9: Exemple de génération de driver bas niveau (fichier.h)

4.5.2 HIGH LEVEL

Afin d'améliorer le découplage entre le hardware et le software et de permettre la saisie des fonctions d'accès hardware à partir du software, le système suivant a été imaginé:

En créant une class portant le même nom que le driver et en y mettant le stéréotype « driver », la possibilité de créer un driver haut niveau est donné.
Toutes les fonctions implémentées dans cette class seront rajoutées au driver. Il est également possible d'y ajouter un constructeur.

Afin de restreindre l'utilisateur du driver aux fonctions implémentées dans le modèle et lui interdire l'accès aux fonctions générées automatiquement, il est possible de forcer les fonctions générées automatiquement en privé grâce au tag « visibility » du stéréotype.

Ci-dessous, le driver haut niveau du contrôleur VGA qui permet de fixer une résolution et d'indiquer quelle image doit être affichée.

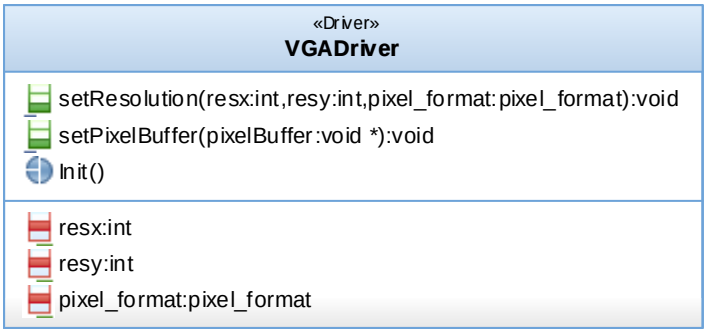


Figure 10: Exemple de génération de driver haut niveau (modèle)

```
16  #ifndef VGADriver_H
17  #define VGADriver_H
18  #include "typeDriver.h"
19
20  typedef struct VGADriver
21  {
22    Hardware hardware;
23
24    UINT32 resx,resy;
25    Pixel_format pixel_format;
26  }VGADriver;
27
28
29
30
31  //Low level init, called bye logicDriver.h
32  void VGADriver_init(VGADriver *this,Hardware hardware);
33
34  //High level init
35  void VGADriver_Init(VGADriver *this);
36
37  //Try to configure the VGA controller with specified resolution,
38  //Use the 640*480 base resolution and stretch the picture
39  UINT32 VGADriver_setResolution(VGADriver *this,UINT32 resx,UINT32 resy,Pixel_format pixel_format);
40
41  //Set the pixel buffer to display. Not wait the vertical syncro of the screen
42  UINT32 VGADriver_setPixelBuffer(VGADriver *this,void *pixel_buffer);
43  #endif
```

Figure 11: Exemple de génération de driver haut niveau (fichier.h)

4.6 BASETYPE

Dans le but de pouvoir être précis dans la définition des données/signaux/port, 5 class stéréotype « Basetype » ont été ajoutés au profil.

- Bool : un boolean standard, en VHDL => std_ulogic
- Data : une donnée, en VHDL => std_ulogic_vector
- Unsigned : Une valeur non signée, en VHDL => unsigned
- Signed : Une valeur signée, en VHDL => signed
- Integer : Une valeur entière, en VHDL => integer

Afin de définir les grandeurs de ces basetype, les méthodes suivantes sont possibles :

- Appliquer le stéréotype « Width » à un data/unsigned/signed pour en définir le nombre de bit;
- Appliquer le stéréotype « Range » à un unsigned/signed/integer pour définir les bornes que la donnée doit pouvoir au minimum exprimer.

4.7 <<BLOCK>>

Les blocs on reçut deux ajustements. Le premier étant de pouvoir définir le nom du clock par défaut du bloc, le second étant une aide à la vérification du système.

4.7.1 CLOCK PAR DEFAULT

Pour définir dans un bloc le nom du clock par défaut, un tag a été rajouté au stéréotype « bloc » pour le spécifier.
Tag :

- Clock_name : nom du clock utilisé par défaut par le bloc

4.7.2 BANC DE TEST

Pour chaque bloc hardware du système, un banc de tests est automatiquement généré. Il contient les éléments suivants :

- Instance du composant à tester
- Génération des clocks et des reset
- Génération de fonction permettant de stimuler les entrées sorties du bloc. Par exemple :
    - Envoyer des paquets sur un stream port
    - Appeler des fonctions sur un functionnal port
    - Lire et écrire des données sur un memory access port

Le dernier point de cette énumération est particulier, car il permettrait la saisie très rapide du test-bench sans devoir auparavant écrire des banalités.

5. REGLES DE MODELISATION SUPPLEMENTAIRES

5.1 INSTANCE PHYSIQUE D'UN SYSTEME HARDWARE

Pour spécifier qu'il existe une instance d'un bloc qui existe physiquement (top level), il faut créer une instance de ce block et lui appliquer le stéréotype « top_level ».

Tag :

- FPGA_fabric : définit qui est le fabricant (xilinx, altera, actel,…)
- FPGA_family : définit la famille du chip (Spartan 3, Cyclone 3, Smartfusion,…)
- FPGA_name : définit le nom exact du chip (A2F200, EP4CE115,…)
- Clock_constraint_format : définit dans quel format de fichier les contraintes des clocks doivent être exportées (sdc , …)
- Pin_constraint_format : définit dans quel format de fichier les contraintes des pins doivent être exportées (pdc,ucf, …)

5.2 DEFINITION DES LIENS ENTRE LE MODELE DU HARDWARE ET LE PACKAGE

Pour permettre la saisie d'information telle que :

- Quelle pin physique est reliée à quel signal
- Fréquence d'un clock
- Adaptation d'une logique négative à une logique positive et vice versa
- Convection d'un tri-state éclaté en 3 signaux (_out,_in_oen) en un tri-state mono signal (inout) sur une pin

La solution suivante a été imaginée :
En créant une instance du bloc qui est le top level du système et en y rajoutant le stéréotype « Top _level », l'utilisateur peut ensuite y connecter des contraintes. Lors de la génération du code, un fichier VHDL portant le nom de l'instance servira de top level et fera l'intermédiaire entre le design et les packages.

Ci-dessous un exemple d'utilisation :

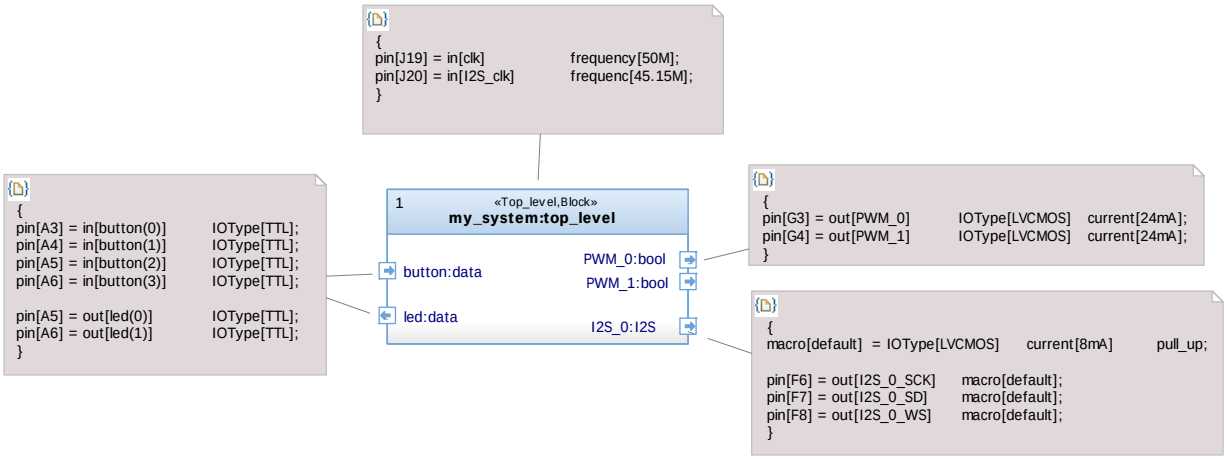


Figure 12: Exemple de définition des liens entre modèle hardware et package

5.2.1 PIN[]

Pour lier des signaux à une pin physique, la syntaxe suivante est à utiliser :

pin[package pin ID] = property[property parameter] property[property parameter] … … …;

Les properties possibles sont les suivantes :

- Out : définit l'expression VHDL qui contrôle la sortie de la pin
- In : définit les signaux qui sont assignés à la valeur de la pin
  - Possibilité d'assigner plusieurs signaux en les séparant par des virgules
  - Possibilité d'inverser la valeur qui sera assignée à un signal avec un « not » devant le nom du signal
- Oen : définit l'expression VHDL qui contrôle le buffer tri-state de la pin
- IOType : définit la technologie de la pin (TTL,LVCMOS,open drain, …)
- Slew : définit la rapidité de commutation d'une entrée (fast, slow).
- Current : définit la limitation de courant maximum de la pin (8mA,16mA, etc.)
- Frequency : définit la fréquence de commutation de la pin (clock)
- Fast_input_register : réduit le délai entre la pin et les bascules qui la lisent
- Fast_output_register : réduit le délai entre la sortie de la bascule et la pin
- Fast_output_enable_register : réduit le délai entre la sortie de la bascule et le output enable de la pin
- Pull_up : ajouter une pull up à la pin
- Pull_down : ajouter une pull down à la pin
- In_GF : permet d'ajouter un glitch filtre entre la pin et les signaux qui la lisent
  - Paramètre 1 : combien de flip flop
  - Parametre 2 : combien de clocks par sample
  - Paramètre 3 : combien de samples identiques pour valider l'état

D'autres properties sont également envisageables, par exemple une gestion des paires différentielles, un duty cycle pour les clocks, ect.

5.2.2 MACRO[]

Il est également possible de définir une macro de la manière suivante :
Macro[macro_name] = property[property parameter] property[property parameter] ... … ;

Lors de la définition d'une pin il est par exemple possible de rappeler une macro de la manière suivante :
pin[package pin ID] = property[property parameter] **macro[macro_name]** … … …;

L'avantage à utiliser des macros est d'éviter de recopier 100 fois les mêmes configurations et de pouvoir les changer pour plusieurs pins en faisant une seule édition à un endroit.

5.3 CLOCK SOURCE

Certains blocs hardware comme les PLL ou encore le MSS du smartfusion se comportent comme des sources de clocks.
Pour spécifier qu'un bloc émet un clock, il faut lui appliquer le stéréotype « Clock_source ».

Tag :

- Name : le nom du clock émis par le bloc.

Une instance d'un tel bloc est sensible au « Clock_connect »

Ci-dessous l'exemple du MSS du smartfusion qui fournit 2 clocks aux blocs, qui l'instancie :

- Sys _clk : est utilisé par tout le système
- Pll_clk : est un clock de sortie d'un module PLL

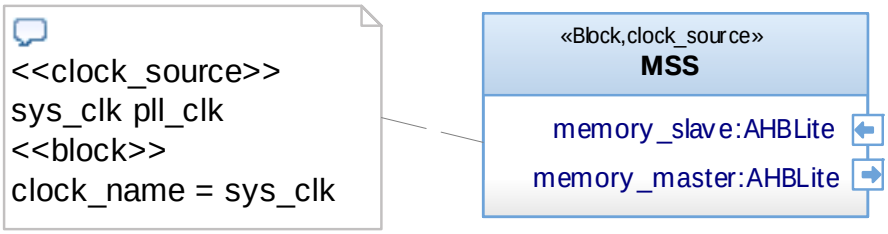


Figure 13: Utilisation de <<clock_source>>

5.4 PULSE PORT

Lorsque <<Pulse>> est appliqué à un port typé avec un booléen, cela indique que l'information transportée par le port est représentée par le fait d'avoir son signal à 1 pendant une période du clock.
C'est une forme d'évent « event » sans paramètre.
Si un lien relie deux pulses port de clocks domaines différents, une logique permettant de transmettre cette pulse est automatiquement ajoutée par la génératrice de code.

5.5 PORTS DE FONCTIONS

Les ports de fonctions offrent la possibilité de faire l'équivalence en hardware des appels de fonction software du monde software avec les functionals ports de UML.

Afin de traduire un port d'appel de fonction, deux streams port sont créés.
Le premier sert à transmettre les appels de fonction avec leurs paramètres.
Le second sert à récupérer le résultat des appels de fonction.

L'exemple ci-dessous met en œuvre un contrôleur SPI (SPI_ctrl_2) qui utilise un functional port pour recevoir des commandes avec son équivalence (SPI_ctrl_2_stream_port_equivalent) en stream port.

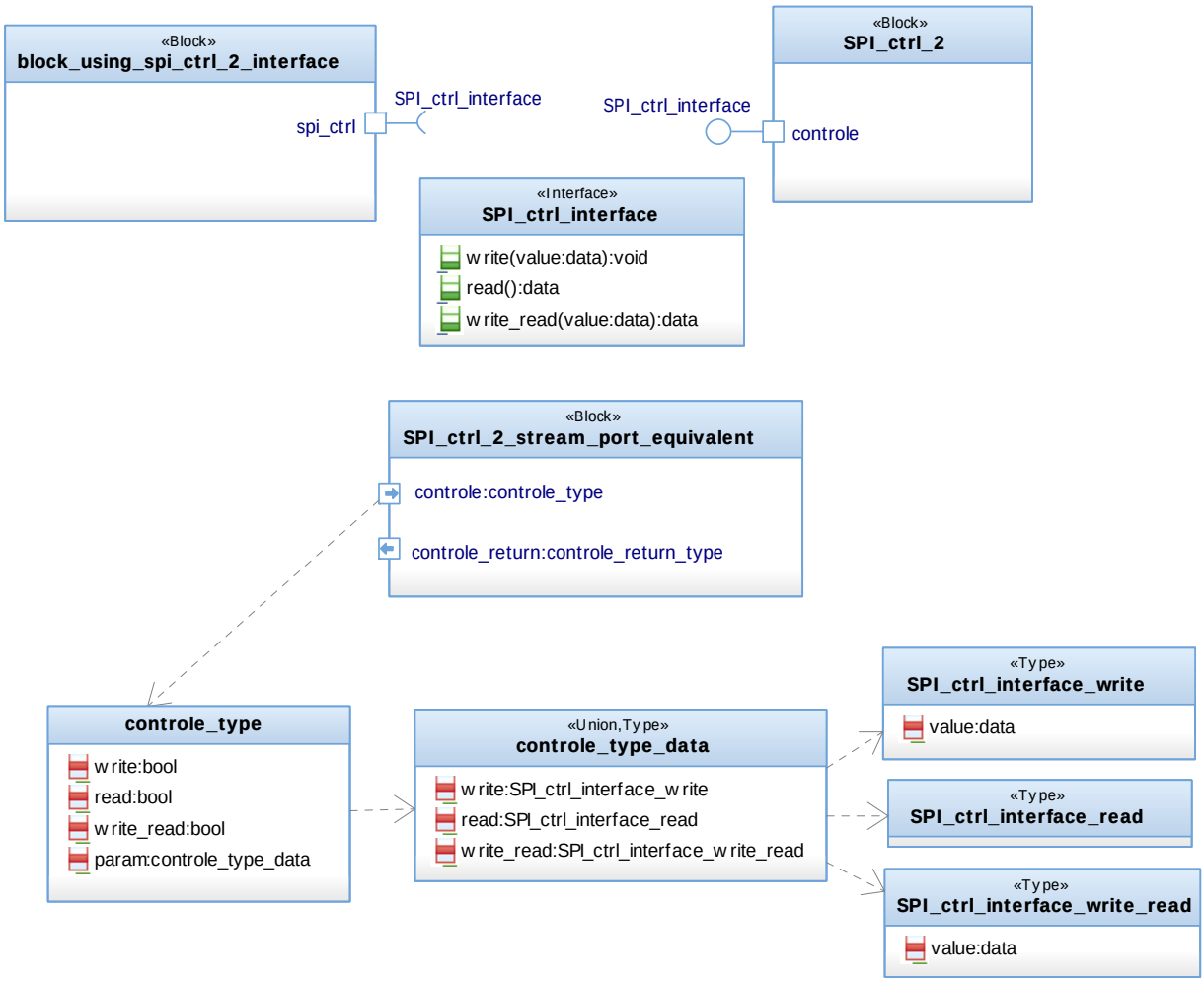


Figure 14: SPI_controller avec functional ports

Il est également possible de connecter un memory access port à un functionnal port pour que, par exemple, un processeur puisse en prendre le contrôle simplement à l'aide d'un driver généré automatiquement.
Si un processeur se connectait via un memory access port au port « contrôle » d'une instance du bloc « SPI_ctrl_2 » (figure précédente), le driver généré contiendrait les fonctions suivantes :

```
//.....
void driverName_linkName_write(Driver driver,UINT8 data);
UINT8 driverName_linkName_read(Driver driver);
UINT8 driverName_linkName_write_read(Driver driver,UINT8 data);
//.....
```

Figure 15: Driver de functional ports

Un problème important est à noter. Avec rhapsody il n'est pas possible de saisir un stéréotype pour l'appliquer à un argument de fonction. Une parade reste à trouver pour contourner ce problème.

5.6 VALEUR PAR DEFAUT

Afin d'établir la valeur par défaut d'un signal contenu dans un port ou une typeInstance, il faut affecter le stéréotype « Default_value » à l'objet en question.

Tag :
Signal_name : le nom du signal qui doit être affecté
Value : valeur par défaut qui doit être appliquée

5.7 ADRESSE FIXE POUR LES LIENS DE MEMORY ACCESS PORT

Le stéréotype <<Fixed_address>> permet de forcer l'adresse de base qui doit être donnée à un lien ainsi que la taille de l'espace réservé.
Tag :

- Base : Adresse forcée
- Size : Taille forcée

6. CO-DESIGN BASE MODELE DU DEMONSTRATEUR

Le choix du démonstrateur est détaillé dans le rapport du travail de semestre [1, p5].

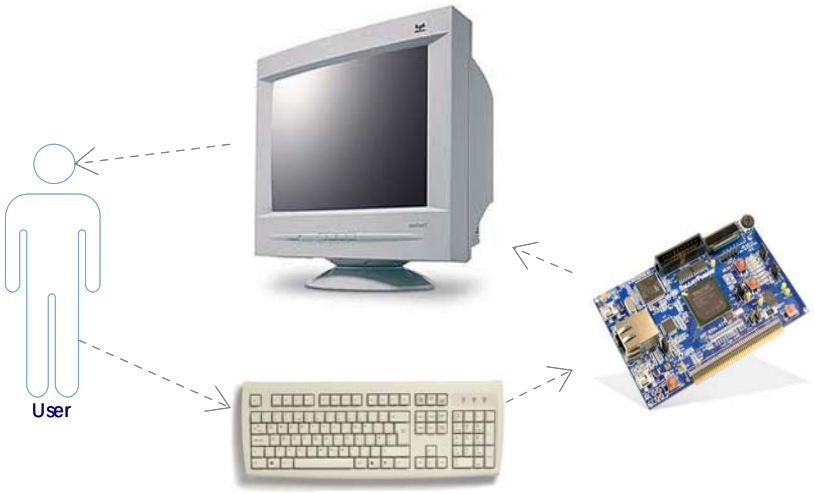


Figure 16: Résumé du démonstrateur

De ce fait, ce chapitre va aborder la vue « top_level » de la logique FPGA du démonstrateur, les blocs nécessaires à l'utilisation de l'interface PS/2 et VGA ainsi que l'application de démonstration qui a été implémenté sur le processeur.

6.1 PHASE D'ANALYSE DU SYSTEME

L'analyse du système sans tenir compte de hardware ou du software est représentée sur le diagramme suivant :

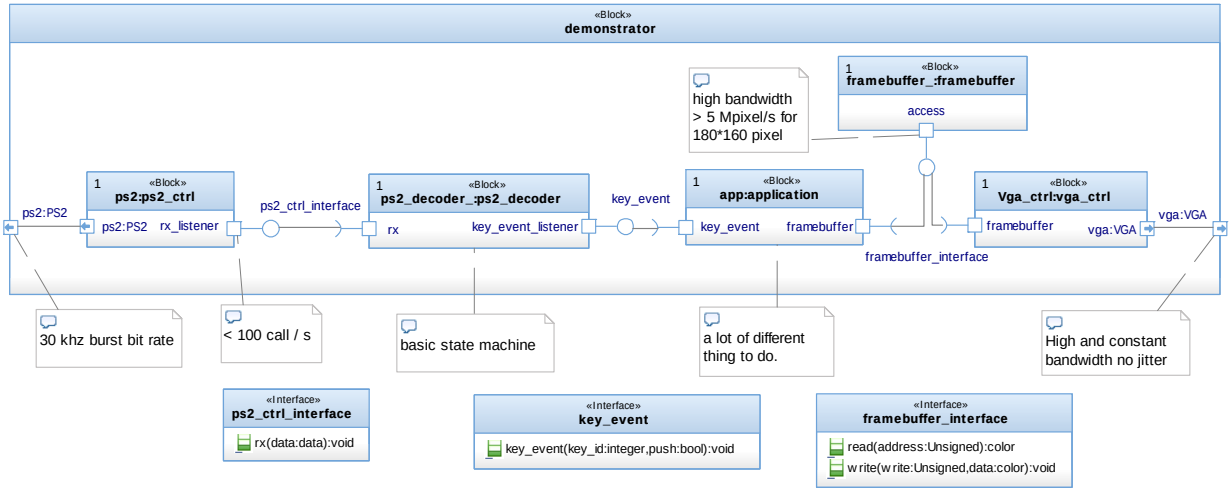


Figure 17: Analyse du démonstrateur

Un clavier émet des trames PS2 qui sont réceptionnées par le contrôleur PS2. Ce contrôleur les décode puis les retranscrit sous forme de byte au décodeur PS2 qui les analyse grâce à une machine d'état pour déterminer les évents qui doivent être émis. Ces évents seront émis vers l'application.

L'application évolue en fonction des évènements qu'elle reçoit et écrit l'image à afficher dans le framebuffer qui sera ensuite lu par le contrôleur VGA pour être traduit en trame VGA.
Trame qui sera transmise à l'écran.

De cette analyse on peut en tirer les choix architecturaux nécessaires à l'obtention d'un système fonctionnel.

- Le contrôleur PS2 doit être mis en hardware, car c'est le clavier qui émet le clock des données (demande de réactivité), et le débit d'entrée est élevé (30Kbit/s).
- Le décodeur des paquets PS2 peut sans problème être mis en software, car il ne nécessite pas une réactivité élevée et est relativement lent.
- L'application doit être mise en software, car une très large palette de choses variées doit y être faite.
- Le contrôleur VGA doit obligatoirement être mis dans du hardware. Car il demande un débit élevé, constant et sans jitter de donnée.
- Le framebuffer ne peut être mis que dans du hardware étant donné l'important débit qui y transite (5Mpixel/s) et le partage de son accès.

6.2 HARDWARE

Dans ce chapitre, les phases de design, implémentation et vérification de la partition hardware vont être abordées.

6.2.1 DESIGN

Dans ce chapitre, seront définies toutes les différentes parties du modèle pour le design hardware.

6.2.1.1 SMARTFUSION_EVALUATION_BOARD

Ce blockInstance représente la fpga et ses connexions avec le package. C'est le „top level" physique.

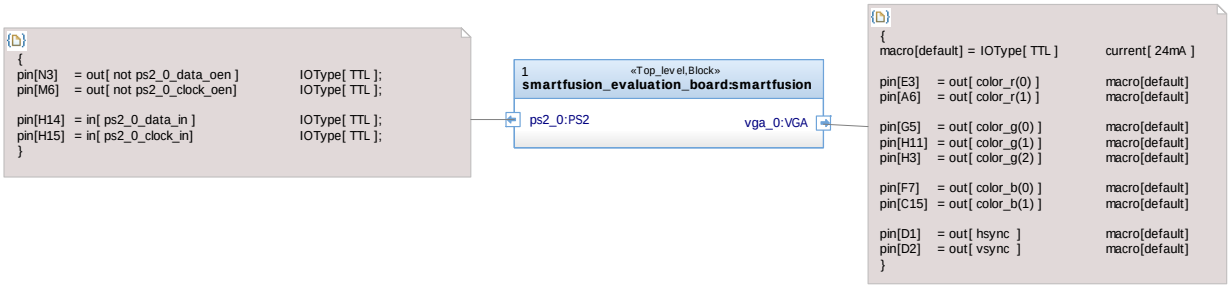


Figure 18: Lien entre smartfusion et le PCB

6.2.1.2 SMARTFUSION

Le bloc smartfusion représente le comportement du „toplevel" du système. Il instancie les deux principaux blocs du design.

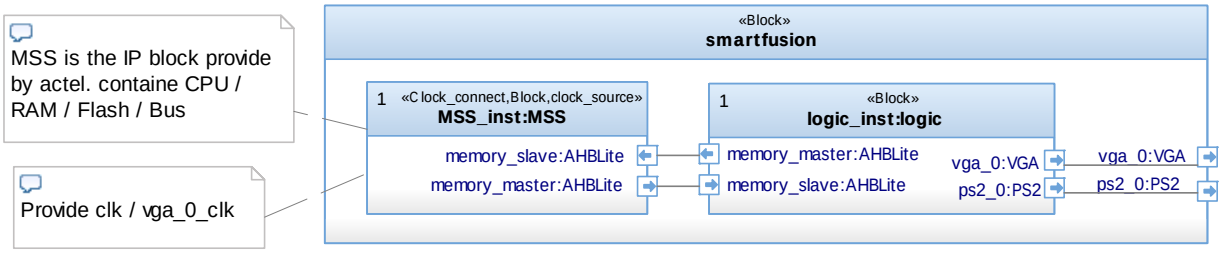


Figure 19: Modèle smartfusion

6.2.1.3 LOGIQUE

Le bloc logique instancie le contrôleur PS2 et VGA et les connecte à un bus de contrôle commun.

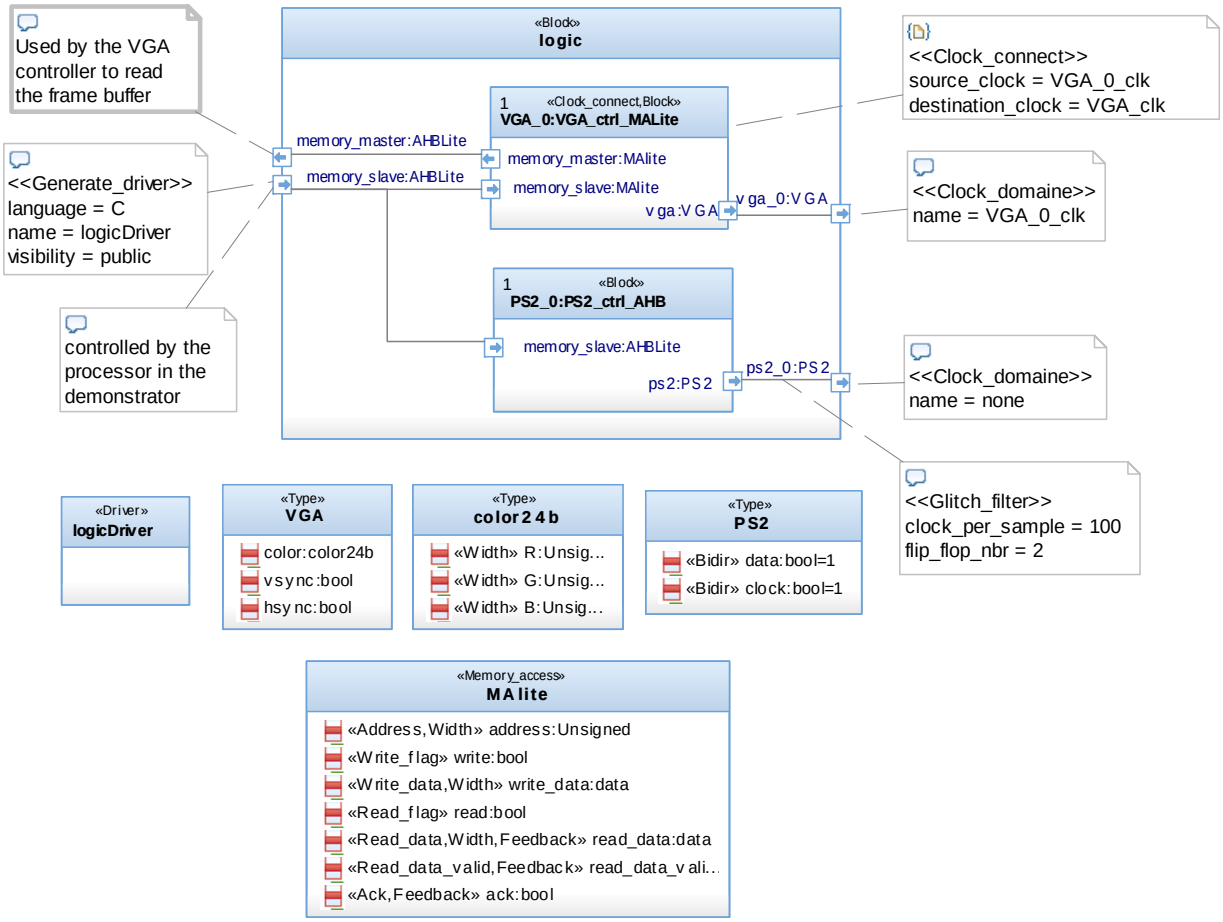


Figure 20: Modèle Logique

Voici une interprétation plus bas niveau des liens et des conversations qu'ils impliquent :

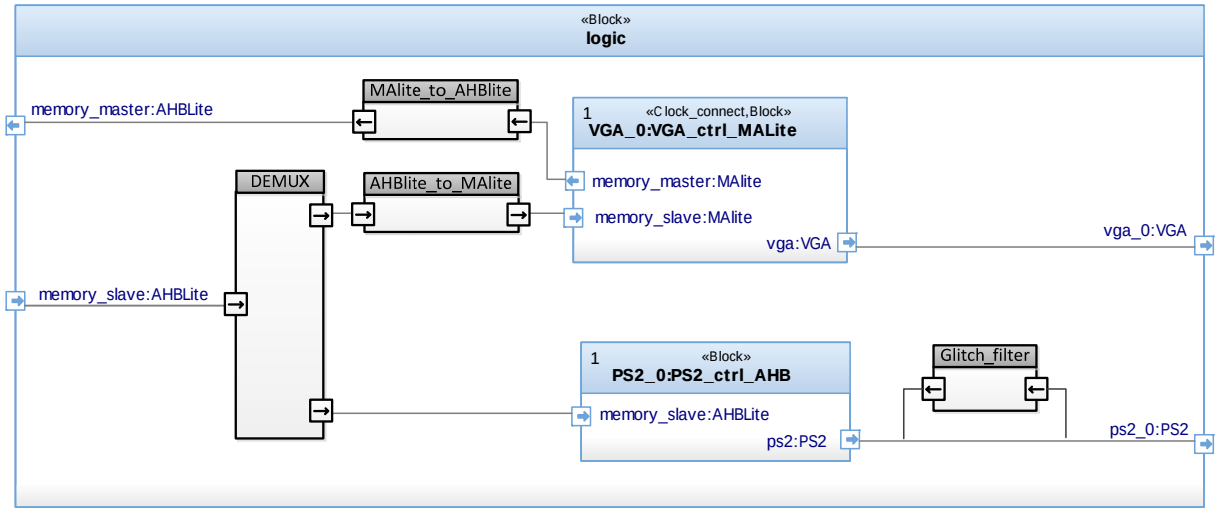


Figure 21: Modèle Logique (link)

6.2.1.4 PS/2 CONTROLLER

Le protocole PS/2 est particulier. Les périphériques (clavier, souris) prennent le contrôle du bus. Lorsque l'unité de contrôle (CPU) veut envoyer une commande/configuration au périphérique, il doit demander l'accès au bus en forçant le clock à zéro puis en attendant que le périphérique fasse de même avec la ligne de data. L'unité centrale devient ainsi maître de la ligne de donnée jusqu'à la fin du transfert.

Voici le modèle du contrôleur PS2 :

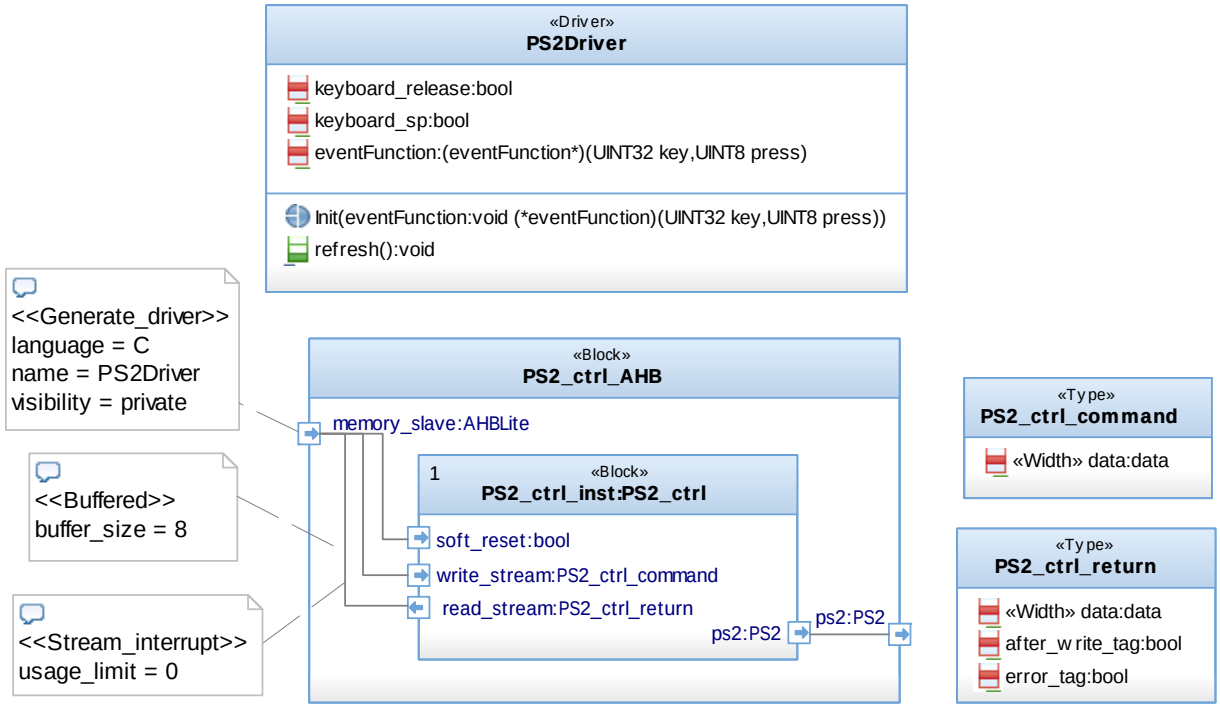


Figure 22: Contrôleur PS2

- Memory_slave port peut demander une écriture PS/2 en envoyant un paquet dans le write_stream.
- Memory_slave port peut réceptionner des paquets envoyés par le périphérique connecté sur le PS/2 en lisant le read_stream.
- Le tag « after_write_tag » du read_stream permet d'indiquer que le paquet lu est le premier après l'envoi d'une commande. Ce flag a été ajouté pour s'assurer de la cohérence des données.

Ci-dessous le fichier.h du driver généré pour le memory_slave port :

```
16  #ifndef PS2Driver_H
17  #define PS2Driver_H
18  #include "typeDriver.h"
19
20  typedef struct PS2Driver
21  {
22    Hardware hardware;
23
24    UINT8 keyboard_release;
25    UINT8 keyboard_sp;
26    void (*eventFunction)(UINT32 key,UINT8 press);
27
28  }PS2Driver;
29
30  //Low level init, called bye logicDriver.h
31  void PS2Driver_init(PS2Driver *this,Hardware hardware);
32
33
34  //High level Init, The eventFunction pointed function is called each time the keyboard
35  //Send event
36  void PS2Driver_Init(PS2Driver *this,void (*eventFunction)(UINT32 key,UINT8 press));
37
38  //Pulling function, read the PS2 read_stream and call eventFunction if necessary
39  void PS2Driver_refresh(PS2Driver *this);
40
41
42  #endif
```

Figure 23: Header du driver PS2

La class PS2Driver contient :

- Un pointeur pointant le hardware (Hardware hardware)
- Des attributs nécessaires au fonctionnement du driver haut niveau (ligne 24,25,26)
- Un constructeur bas niveau pour lui assigner le hardware (ligne 31)
- Un constructeur haut niveau dépendant de l'application (ligne 36)
- Les fonctions du driver haut niveau qui, dans ce cas, est la fonction refresh (ligne 39) qui permet de faire du pulling pour savoir si une donnée a été réceptionnée par le contrôleur.

6.2.1.5 VGA CONTROLLER

Ce bloc a plusieurs fonctions :

- Permettre la configuration du système à partir du memory_slave port
- Instancier le vga_logic.
- Générer un Stream à partir du pixel buffer qui se trouve en mémoire RAM pour alimenter le vga_logic en pixel à afficher.

Le lien transportant le pixel buffer stream est <<buffered>> car:

- les latences mémoires sur le memory_master peuvent être très élevées.

- Le débit de la mémoire sur le memory_master peut être inconstant.
- Le protocole VGA demande un débit élevé et constant.

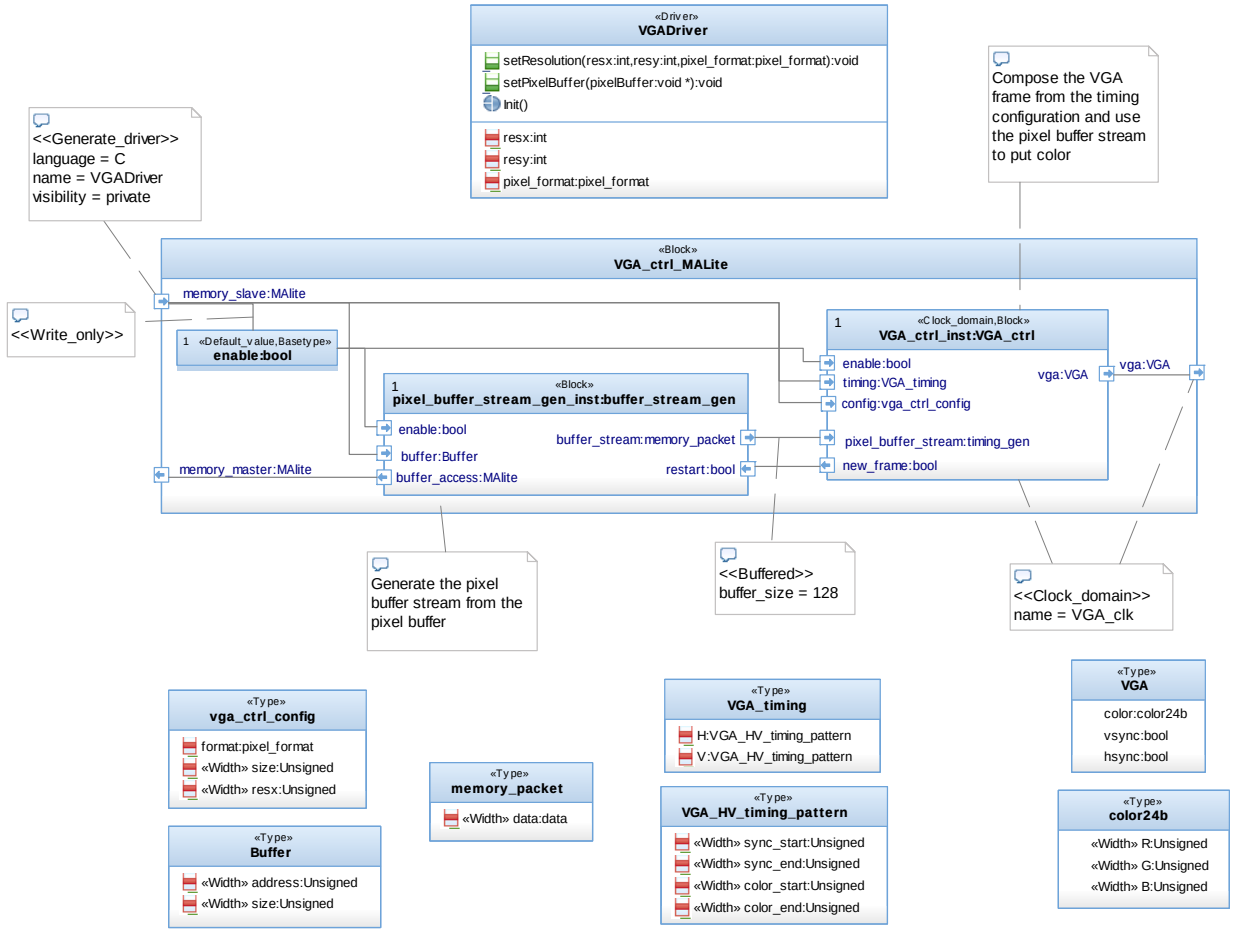


Figure 24: Contrôleur VGA

```
16  #ifndef VGADriver_H
17  #define VGADriver_H
18  #include "typeDriver.h"
19
20  typedef struct VGADriver
21  {
22    Hardware hardware;
23
24    UINT32 resx,resy;
25    Pixel_format pixel_format;
26  }VGADriver;
27
28
29
30
31  //Low level init, called bye logicDriver.h
32  void VGADriver_init(VGADriver *this,Hardware hardware);
33
34  //High level init
35  void VGADriver_Init(VGADriver *this);
36
37  //Try to configure the VGA controller with specified resolution,
38  //Use the 640*480 base resolution and stretch the picture
39  UINT32 VGADriver_setResolution(VGADriver *this,UINT32 resx,UINT32 resy,Pixel_format pixel_format);
40
41  //Set the pixel buffer to display. Not wait the vertical syncro of the screen
42  UINT32 VGADriver_setPixelBuffer(VGADriver *this,void *pixel_buffer);
43  #endif
44
```

Figure 25: Header du driver VGA

La class VGADriver contient :

- Un pointeur pointant le hardware (Hardware hardware)
- Des attributs nécessaires au fonctionnement du driver haut niveau (ligne 24,25)
- Un constructeur bas niveau pour lui assigner le hardware (ligne 32)
- Un constructeur haut niveau dépendant de l'application (ligne 35)
- Les fonctions du driver haut niveau (ligne 39, 42)

6.2.1.6 VGA LOGIC

La mémoire RAM dans laquelle est partagé le frame buffer étant d'une taille limitée (64KB) et la norme VGA ne descendant pas en-dessous d'une résolution de 640*480, il a été décidé d'ajouter une fonction d'upscaler au contrôleur VGA. Le facteur d'agrandissement est l'attribut « size » transmis par le port pixel_info.
L'existence de l'instance de bloc line_cache_inst est directement liée à cette fonction d'upscaler. Cela évite de relire en mémoire plusieurs fois la même ligne.

En comparant avec le design établi durant le projet de semestre, il est à noter que la lecture du line_cache se fait maintenant à l'aide d'un functional port mettant à disposition la fonction read de l'interface line_cache_request. Le grand avantage de cette méthode est que le line_reader n'a pas besoin de connaître la latence du line cache. Tant que cette dernière est stable, le système fonctionne bien.

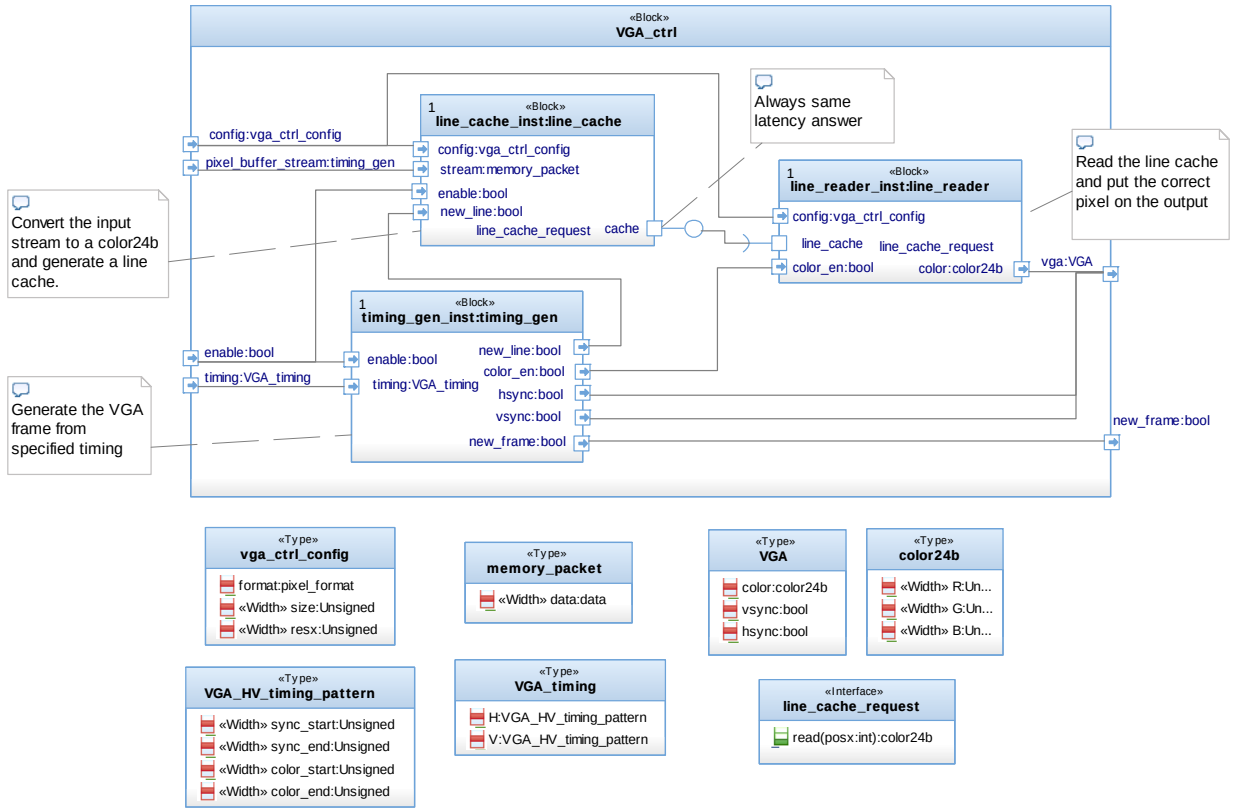


Figure 26: Logique du VGA

6.2.2 IMPLEMENTATION

L'implémentation manuelle du hardware du démonstrateur a été faite selon le modèle qui avait été établi durant le projet de semestre.

Cette implémentation manuelle avait comme majeure fonction d'indiquer les points et les règles de modélisation s'avérant incomplètes ou inopérantes. Ces points ont été corrigés dans le profil et dans le modèle du démonstrateur.
L'arbitration des stream ports s'est relevée tout à fait adaptée et facile à utiliser.
Les memory access port n'ont pas posé de problème particulier, et ont permis d'expérimenter un template de codage « standard » et donc de résoudre les défis qu'ils posent sans effectuer du « cas par cas ».
Ce template de codage peut se résumer à :

- Lecture asynchrone
- Arbitration asynchrone
- Pour les bus tels que l'AHB qui se déroule sur plusieurs cycles => ramener tout au même cycle (shift registre sur l'adresse et l'arbitration) pour obtenir un dénominateur commun à tous les bus.

6.2.3 VERIFICATION

Tous les blocs ont été simulés et testés physiquement. Le système est fonctionnel et stable.

La stabilité demandée par l'interface VGA en termes de jitter a pu être expérimentée :
Initialement le clock VGA était généré par un module RC. La variation de fréquence de cet oscillateur était tellement élevée que l'écran tremblait à en donner la nausée.
Il a finalement fallu la stabilité d'un oscillateur à cristal pour obtenir une image stable.

6.3 SOFTWARE

Pour le démonstrateur il a été décidé d'implémenter un petit OS sur lequel trois applications graphiques s'exécuteront.

- Une animation 3D représentant un cube
- Une simulation du « Game of life »
- Un jeu avec un vaisseau devant traverser un champ d'astéroïdes

Les chapitres qui vont suivre abordent le design l'implémentation et la vérification du software.

6.3.1 DESIGN

Dans ce chapitre le design général du software va être abordé sans entrer dans les détails, car le projet en lui-même ne se concentre pas là-dessus.

6.3.1.1 MODELE

Ci-dessous une figure du modèle UML du software. On peut y voir l'OS instanciant les drivers ainsi que les 3 applications (GameOfLife Scene3D MeteorGame).

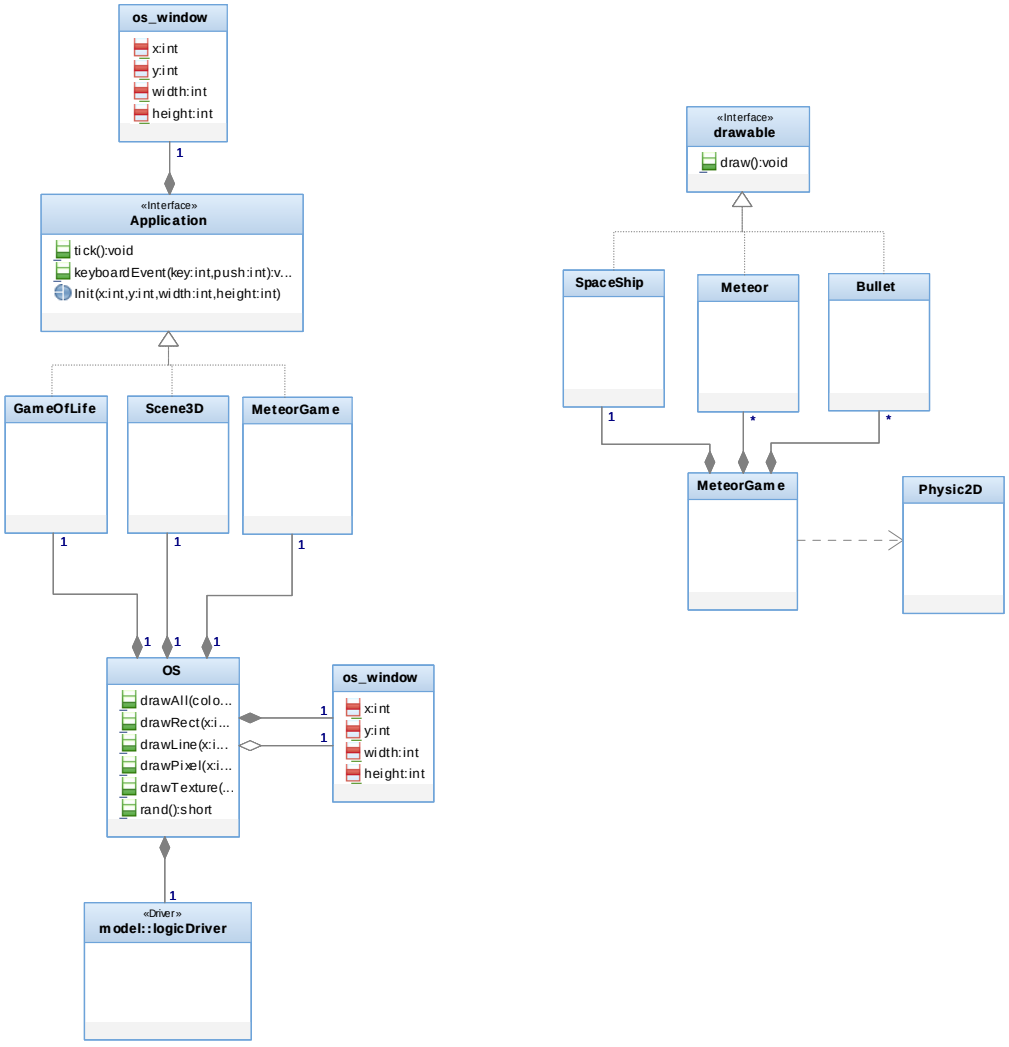


Figure 27: Software modèle

6.3.1.2 FONCTIONNALITE DE L'OS

L'OS offre les fonctionnalités de base suivantes :

- Gestion des frames buffer et des fonctions graphiques
  - drawAll
  - drawPixel
  - drawRect
  - drawTexture
  - drawLine
- La génération de nombres pseudo-aléatoires

- Appels des fonctions tick() des applications pour les rafraîchir
  - Dans cet OS, il n'y a pas de base de temps, dès qu'une application a fini d'exécuter son tick(), l'OS appelle le tick() de l'application suivante.
- Transmission des events du clavier aux applications

Cet OS a un design volontairement primitif. Les raisons de ce choix sont :

- Eviter de prendre trop de temps pour un démonstrateur
- Codage en C

Ci-dessous un diagramme où est représenté le job répété en boucle par l'OS qui donne dans l'ordre :

1. Réception des inputs clavier
2. Rafraîchissement des applications
3. Affichage de la nouvelle image à l'écran

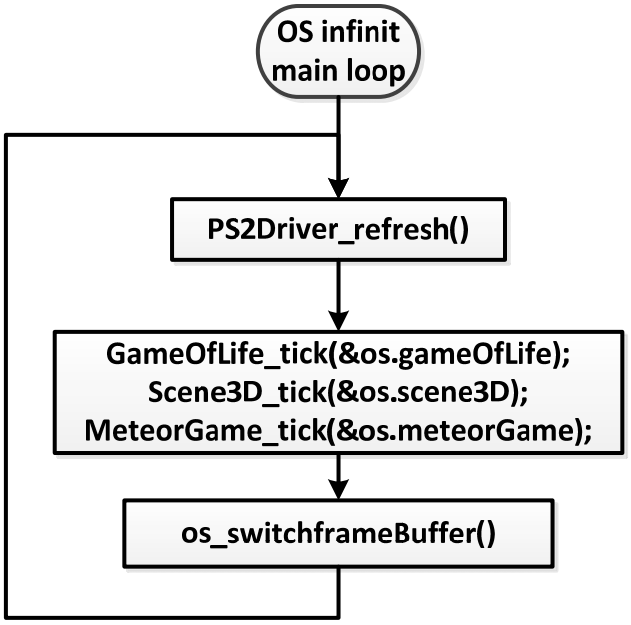


Figure 28: Job de l'os

6.3.2 IMPLEMENTATION

L'implémentation s'est faite en C et non pas en C++ car les outils de développement étaient déjà fonctionnels pour le C.
La résolution utilisée est de 180 * 180 pixels et de 128 couleurs par pixel (RGB_232). Le facteur limitant pour la résolution est la capacité de mémoire RAM du smartfusion de 64 KB .

Voici quelques captures d'écran où l'on peut voir :

- en haut la simulation de « Game of life »,
- en bas à gauche un cube qui tourne sur lui-même en fonction des flèches directionnelles du clavier
- en bas à droite le jeu qui est également contrôlé par les flèches directionnelles du clavier

Le système marche parfaitement en mode debug –o3, néanmoins après la programmation en mode release le système reboot à intervalle régulier d'environ 5 secondes malgré l'utilisation du linker script adapté (production-execute-in-place.ld).

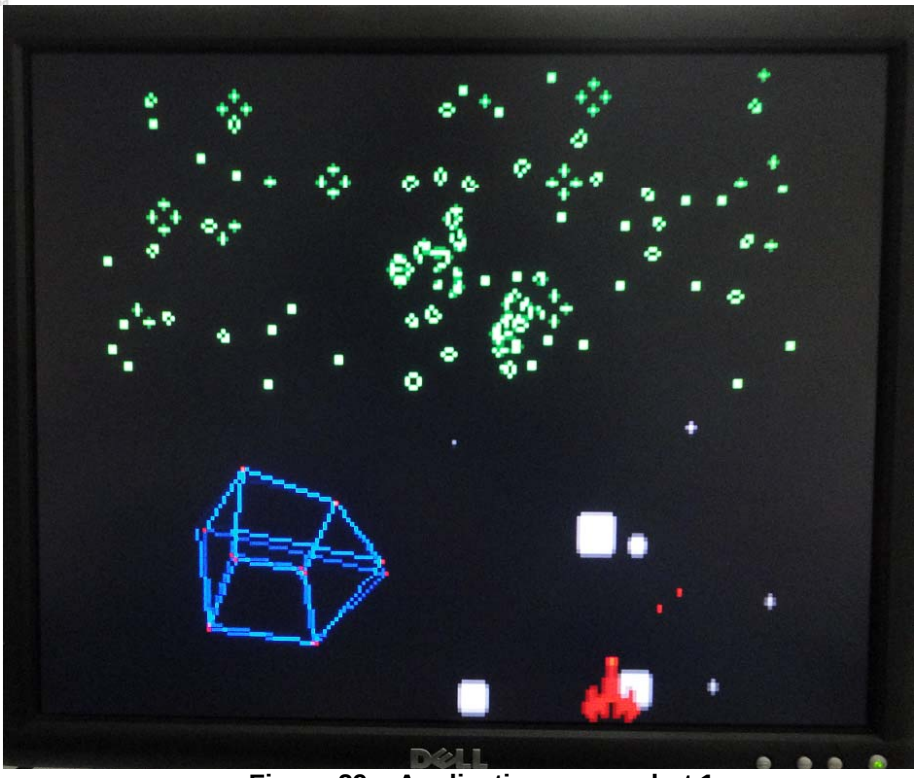


Figure 29: Application screenshot 1

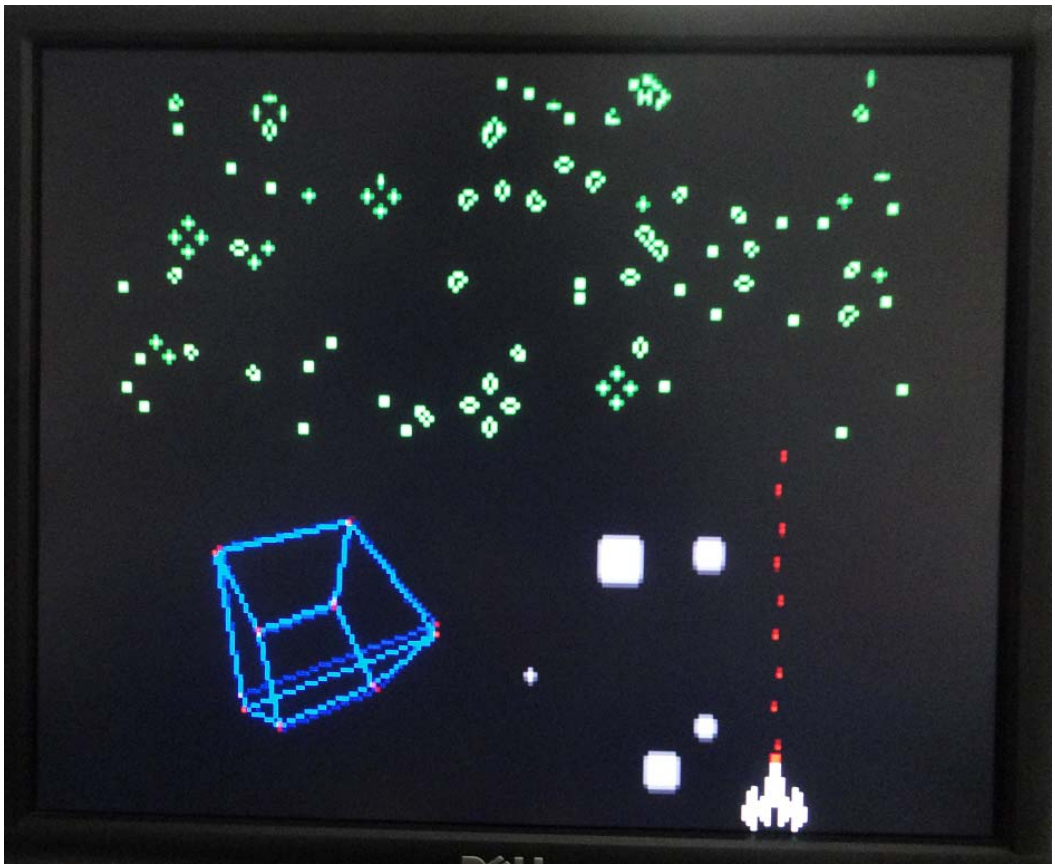


Figure 30: Application screenshot 2

Sur la figure ci-dessus, on voit distinctement le vaisseau qui tire des lasers.

6.3.3 VERIFICATION

Les vérifications du software ont été :

- Contrôle visuel du comportement à l'écran selon les interactions avec le clavier.
- Laisser tourner les programmes pendant 4 heures sans constater de problème.

Les memory leak sont impossibles étant donné qu'aucune allocation dynamique de mémoire n'est faite.

6.4 BOARD D'EXTENSION

Pour le démonstrateur, le kit d'évaluation smartfusion ne possédant ni interface VGA ni PS/2, il a fallu créer une petite board d'extension.

6.4.1 SCHEMATIC

La plaque d'extension comporte une sortie VGA passive 128 couleurs (RGB : 232) ainsi qu'une interface pour clavier PS/2.
L'interface PS/2 étant 5V pull-up et les pins d'entrée sortie de chip smartfusion supportant jusqu'à 3.6V, une adaptation de voltage a été nécessaire.

La schématique se trouve en **Annexe 1.**

Le principe de conversion DA utilisé pour le RGB du VGA est le suivant :

Sachant que l'écran représente une charge de 75 ohm, chaque bit va y être connecté à travers une résistance dont la valeur sera égale à : R/(poids du bit).

$$\frac{VCC*RL}{Uout} - RL = R // R/2 // R/4$$

Vcc = 3.3V
RL = 75 ohm
Uout = 0.7V

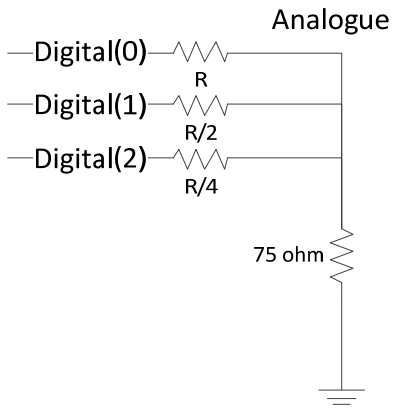


Figure 31: Convertisseur DA vidéo

6.4.2 PCB

Voici une image sur laquelle on peut voir le système monté et connecté :

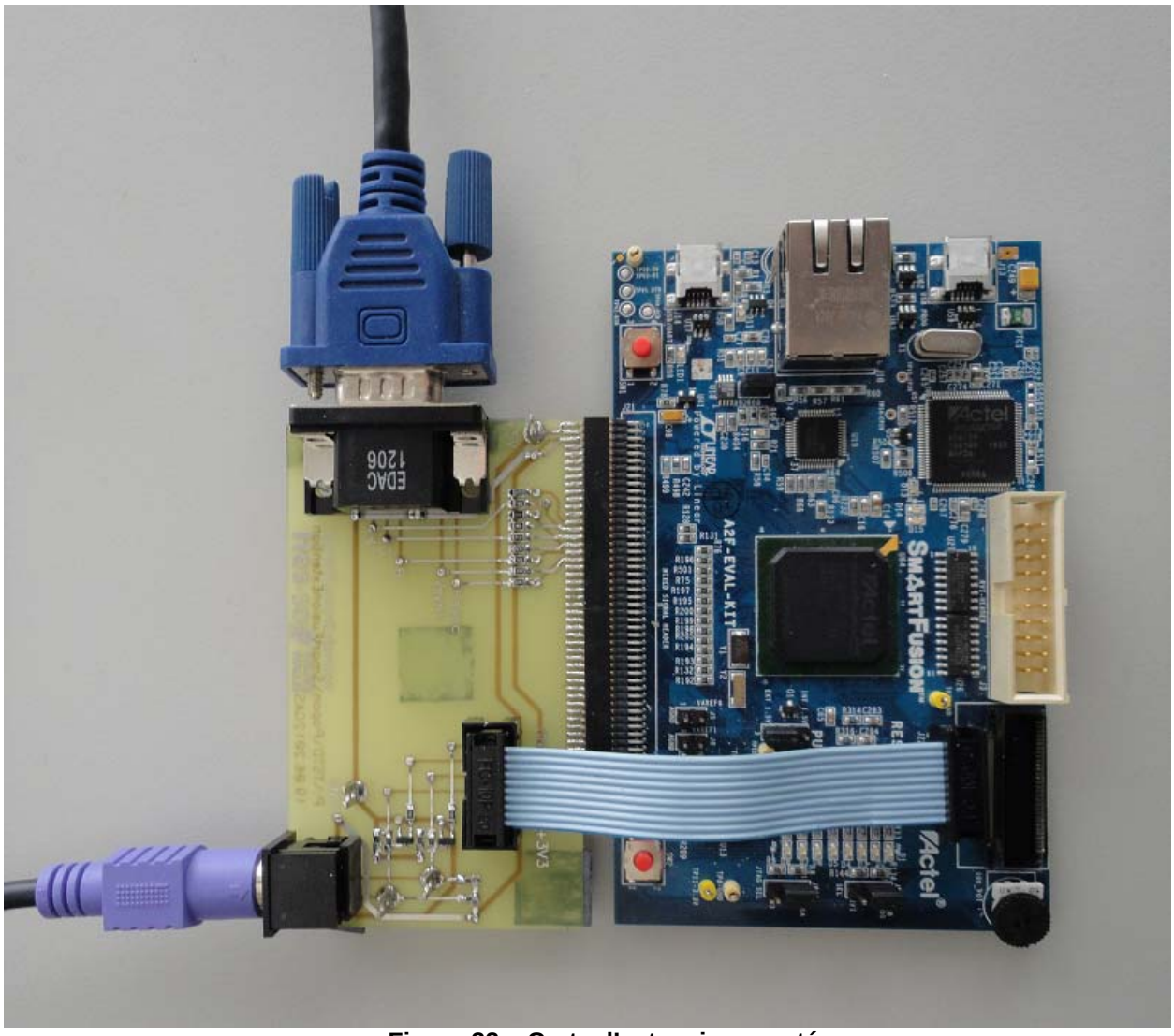


Figure 32: Carte d'extension montée

7. GENERATRICE DE CODE

Dans ce chapitre vont être abordées les questions des outils et de l'implémentation d'une génératrice de code partiel.

7.1 OUTILS

Trois outils différents m'ont été proposés pour implémenter la génératrice :

- VB de rhapsody
- Quex/C++
- MD Workbench

7.1.1 VB DE RHAPSODY

Dans ce cas il s'agit de directement interagir avec l'outil Rhapsody avec du visual basic.

Avantage :

- Interaction directe avec Rhapsody

Inconvénient :

- Uniquement pour Rhapsody

7.1.2 QUEX/C++

Quex est un analyseur lexical qui, en fonction des règles qu'on lui donne, génère un parseur codé en C/C++.
Suite à cela à partir d'un projet C++ il est possible de donner un fichier/stream de caractère au parseur pour qu'il en retourne un « event ,Token» stream.
L'idée est de l'utiliser pour décoder les fichiers XMI que bon nombre d'outils de modélisation UML sont en mesure d'exporter à partir d'un modèle.

Avantage :

- Flexible
- Le standard XMI
- Langage C++ maîtrisé
- Projet interne décodant le XMI en arborescence d'objet en mémoire (Thomas Sterren)

Inconvénient :

- Projet interne compatible UML 1.4. => portage vers UML 2.1 nécessaire

7.1.3 MD WORKBENCH

MD workbench est un outil créé par Sodius dans le but de créer des génératrices/convertisseuses de format/fichier.

Avantage :

- Flexible
- Le standard XMI compatible UML 2.1
- Moteur de recherche d'élément dans le modèle

Inconvénient :

- Langage non maîtrisé

7.1.4 CHOIX DE L'OUTIL

VB de rhapsody a été relativement vite écarté pour son inconvénient qui est de se limiter à rhapsody.
Quex/C++ et MD Workbench sont deux solutions tout à fait fiables. Cependant, MD Workbench a été choisie pour son support d'UML 2.1, son débuggeur efficace et sa prise en main rapide.

7.2 ANALYSE DU MODELE

Ce chapitre va aborder les différentes passes de transformation du modèle en code, ainsi que le méta-modèle intermédiaire à mi-chemin entre le modèle et le VHDL.

7.2.1 VUE GLOBALE

La génératrice demande comme entrée :

- Le modèle hardware au format XMI à traduire en VHDL
- Le profile à utiliser au format XMI

Elle va ensuite analyser et traduire ce modèle en trois passes.

- une première sans intelligence qui s'occupe que d'analyse simple
- la deuxième gérant la complexité des connections
- la troisième générant l'écriture des fichiers sources

Ces 3 phases sont plus précisément décrites dans le chapitre « passe d'analyse du modèle »

En sortie de la génératrice, on trouve :

- Les fichiers VHDL correspondant aux blocs
- Les fichiers VHDL correspondant aux instances de blocs « top_level »
- Les fichiers de contrainte des entrées sorties
- Les drivers d'interface entre hardware et software

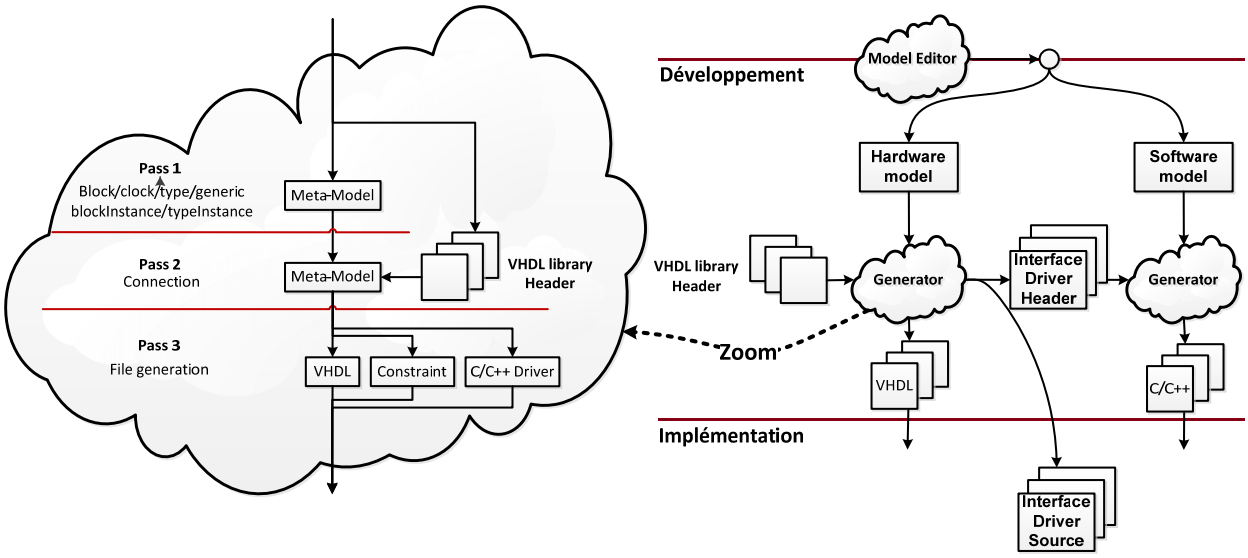


Figure 33: Vue globale de la génératrice de code hardware

7.2.2 META-MODELE INTERMEDIAIRE

Afin de simplifier l'analyse et la traduction en VHDL du modèle, un méta-modèle intermédiaire a été développé.
Ce méta-modèle offre les possibilités suivantes :

- Traduction en VHDL très directe
- Un grand nombre de données utiles à l'analyse du modèle y sont stockées
- Les résultats des analyses demandant de la récursivité sont également stockés

7.2.2.1 TYPE UTILISE PAR LE META-MODELE

Voici un diagramme qui représente les types utilisés :

Figure 34: Signal modèle

Clock représente les données liées à un clock. Pour l'instant seul le nom est stocké, mais à l'avenir il sera possible d'y ajouter la valeur du flanc ainsi que la fréquence.

Signal est une interface de base utilisée pour représenter un signal VHDL, un générique VHDL ou encore un signal d'entrée sortie d'un bloc. Toute class en héritant doit en réimplémenter les trois fonctions de base qui permettent la manipulation des signaux par la génératrice de code.
Ces fonctions sont :

- VHDLTypeDeclaration : qui permet d'extraire du signal la syntaxe VHDL de la déclaration du type
- Split : qui permet à la génératrice, à partir d'un std_ulogic_vector, de recomposer plusieurs signaux (spécifié par une liste) qui y ont été précédemment concaténés.
- Concat : qui permet à la génératrice de concaténer une liste de signaux dans un std_ulogic_vector.

La raison pour laquelle split et concat sont nécessaires est que pour par exemple insérer une stream dans une stream_buffer, il faut pouvoir les regrouper dans une liste de signal dans l'entrée de ce stream_buffer qui est un std_ulogic_vector.

7.2.2.2 STRUCTURE DU META-MODELE

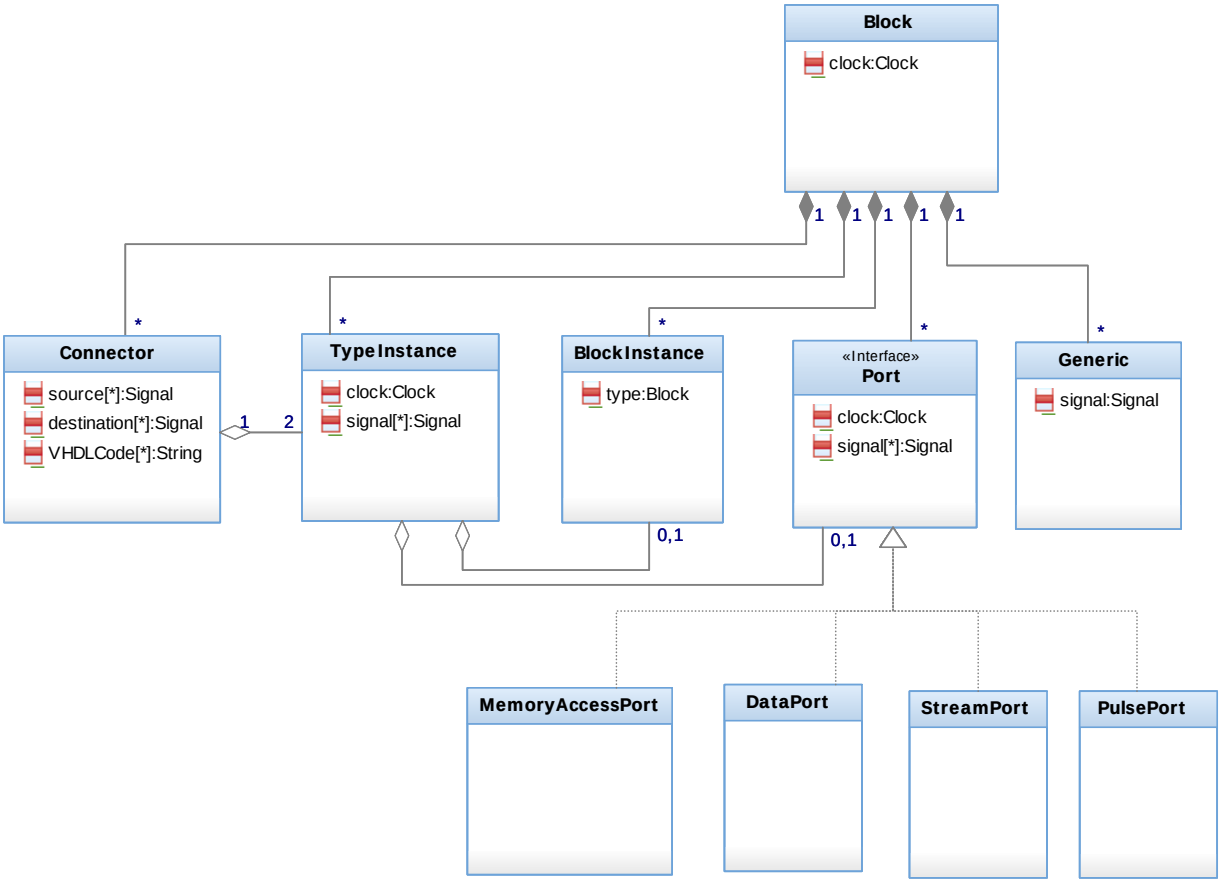


Figure 35: Meta-modèle proche du VHDL

Chaque bloc (entity VHDL + architecture) peut contenir plusieurs:

- Connector (câblage direct VHDL avec possibilité d'insérer du code)
- TypeInstance (~=instance de record VHDL)
- BlockInstance
- Port (Groupement d'entrées sorties d'une entity VHDL ayant une fonctionnalité)
- Generic

Une TypeInstance peut avoir été créée de trois manières différentes :

- Elle existait sur le modèle initial
- Elle a été créée par la génératrice pour y connecter un port d'une instance de bloc
- Elle a été créée pour correspondre à un port du bloc lui-même.

7.2.3 PASSE D'ANALYSE DU MODELE

Comme le présente le diagramme ci-dessous, l'analyse de modèle se fait en trois phases.

La première phase a pour objectif principal de préparer le modèle, de simplifier sa lecture, et de le transformer légèrement pour mieux coller au VHDL.

La deuxième phase va s'appuyer presque entièrement sur les transformations effectuées pendant la première phase pour analyser les connexions entre les éléments du modèle puis les connecter à l'aide de syntaxe proche du VHDL.

La troisième phase est la génération des fichiers sources. Cette partie ne possède aucune intelligence et ne fait que de lire les informations précédemment générées.

Toutes les phases s'appuient sur le squelette du modèle initial.

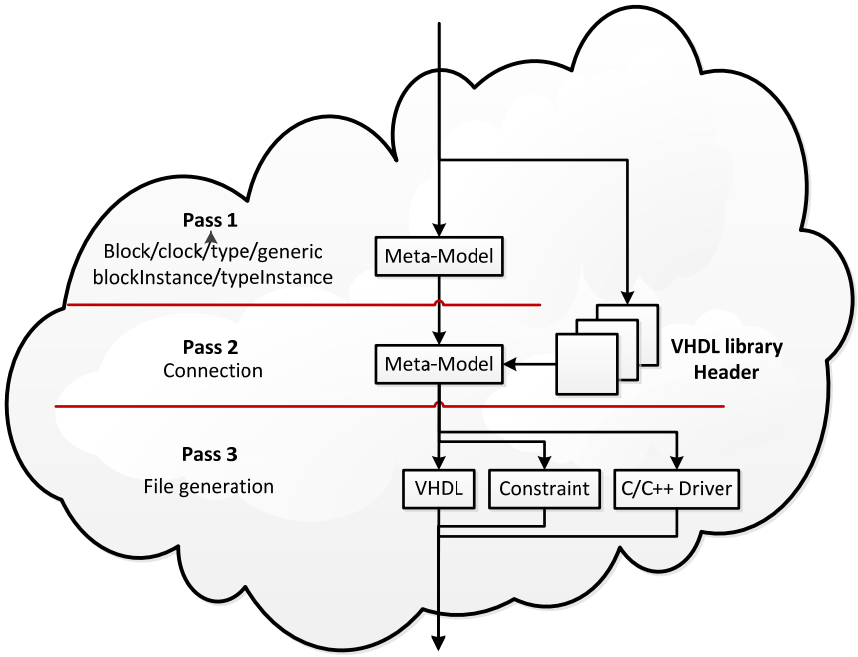


Figure 36: Vue globale de la génératrice de code

7.2.3.1 PASSE 1.1 : CREATION DES TYPEINSTANCE

Cette passe a pour objectif de rapprocher le modèle d'une structure VHDL. Pour ce faire, la génératrice va analyser tous les blocs de la manière suivante :

- Pour chaque port => création d'une typeInstance équivalente
- Pour chaque port appartenant à une blockInstance => création d'une typeInstance équivalente
- Pour chaque typeInstance créée, redirection des connexions pointant le port vers la typeInstance.

7.2.3.2 PASSE 1.2 : ANALYSE DES CLOCKS

Pour les éléments du modèle qui utilisent des clocks (Block / BlockInstance / Port / TypeInstance), des fonctions ont été implémentées pour permettre d'établir une liste de ces clocks.
Ces fonctions sont :

- BlockClockList
- PortClockList
- BlockInstanceClockList
- TypeInstanceClockList

Il est également à noter que :

- Ces fonctions fonctionnent de manière récursive.
- Le résultat du premier appel d'une de ces fonctions sur un objet est automatiquement mis en cache à l'aide des transient links afin de ne plus le recalculer à la prochaine requête.
- Actuellement, ces fonctions ne gèrent par les blocs „Clock_source".

Les chapitres suivants expliquent leurs fonctionnements.

7.2.3.2.1 BlockClockList

Pour identifier quels sont les clocks utilisés par un block il suffit de lister tous les clocks des ports et des instances.
Les TypeInstance n'ont pas besoin d'être pris en compte étant donné qu'elles interagissent forcément avec un port de même clock.

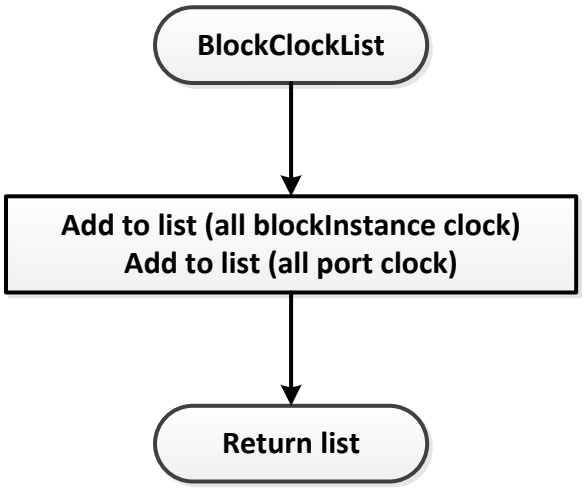


Figure 37: Clock utilisé par un block

7.2.3.2.2 PortClockList

Pour identifier quel sont les clocks utilisés par un port, il suffit de récupérer le nom du clock par défaut du bloc dans lequel le port est déclaré.
Exception toutefois si le stéréotype « Clock_domain » est appliqué.

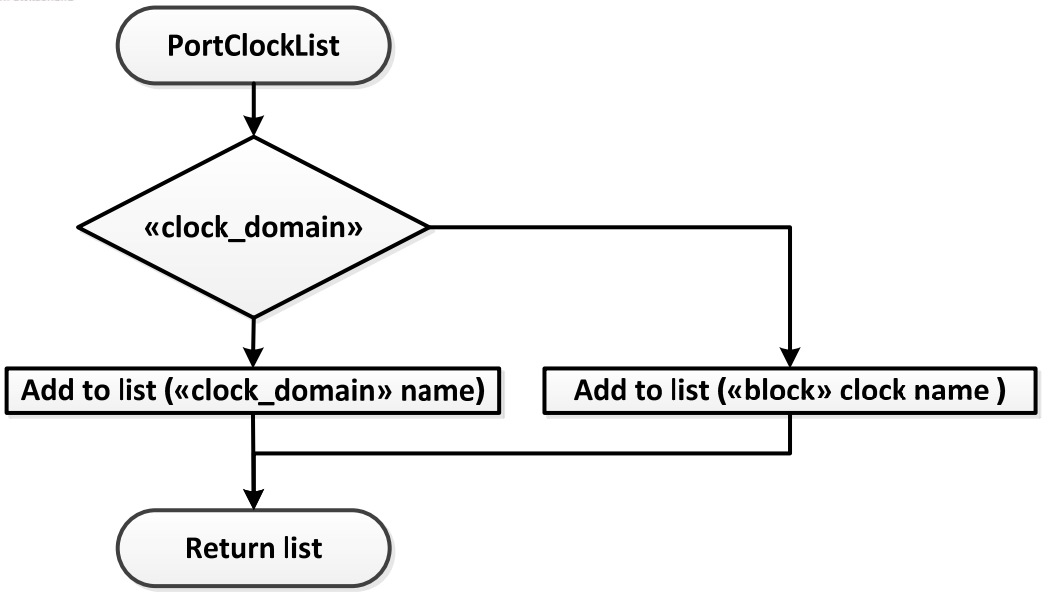


Figure 38: Clocks utilisés par un port

7.2.3.2.3 BlockInstanceClockList

Pour identifier les clocks utilisés par une instance de bloc, il faut récupérer la liste de base des clocks utilisés par le bloc puis y appliquer successivement 2 transformations/redirection. Celle du « clock_domain » puis celle du « clock_connect ».

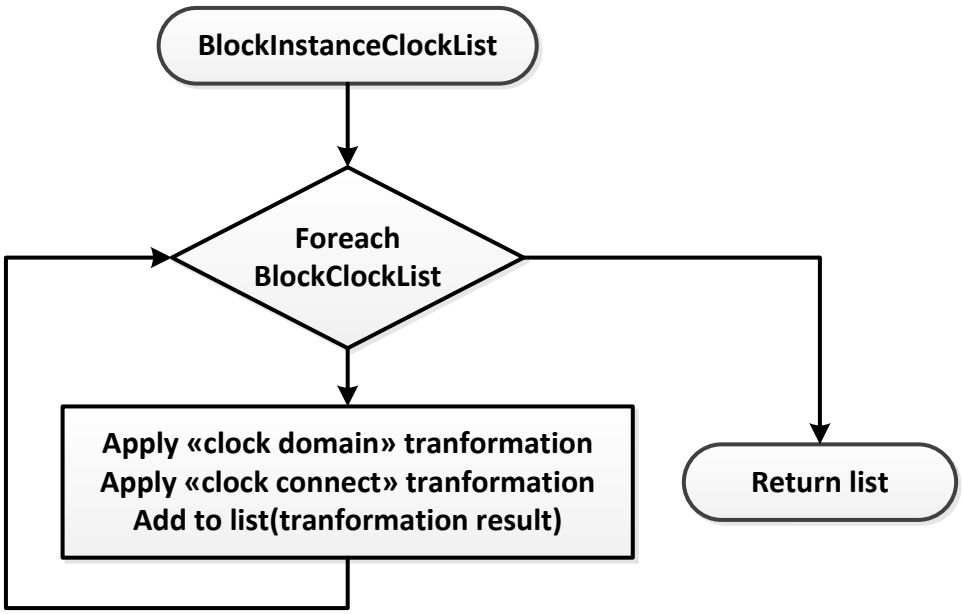


Figure 39: Clocks utilisés par une instance d'un bloc

7.2.3.2.4 TypeInstanceClockList

L'identification des clocks utilisés par une instance de type est la moins trivial des quatre.
Cela dépend avant tout de l'origine de l'instance de type (présente sur le modèle de base ou ajoutée par la génératrice pour correspondre à un port du bloc ou à un port d'une instance d'un bloc) ;

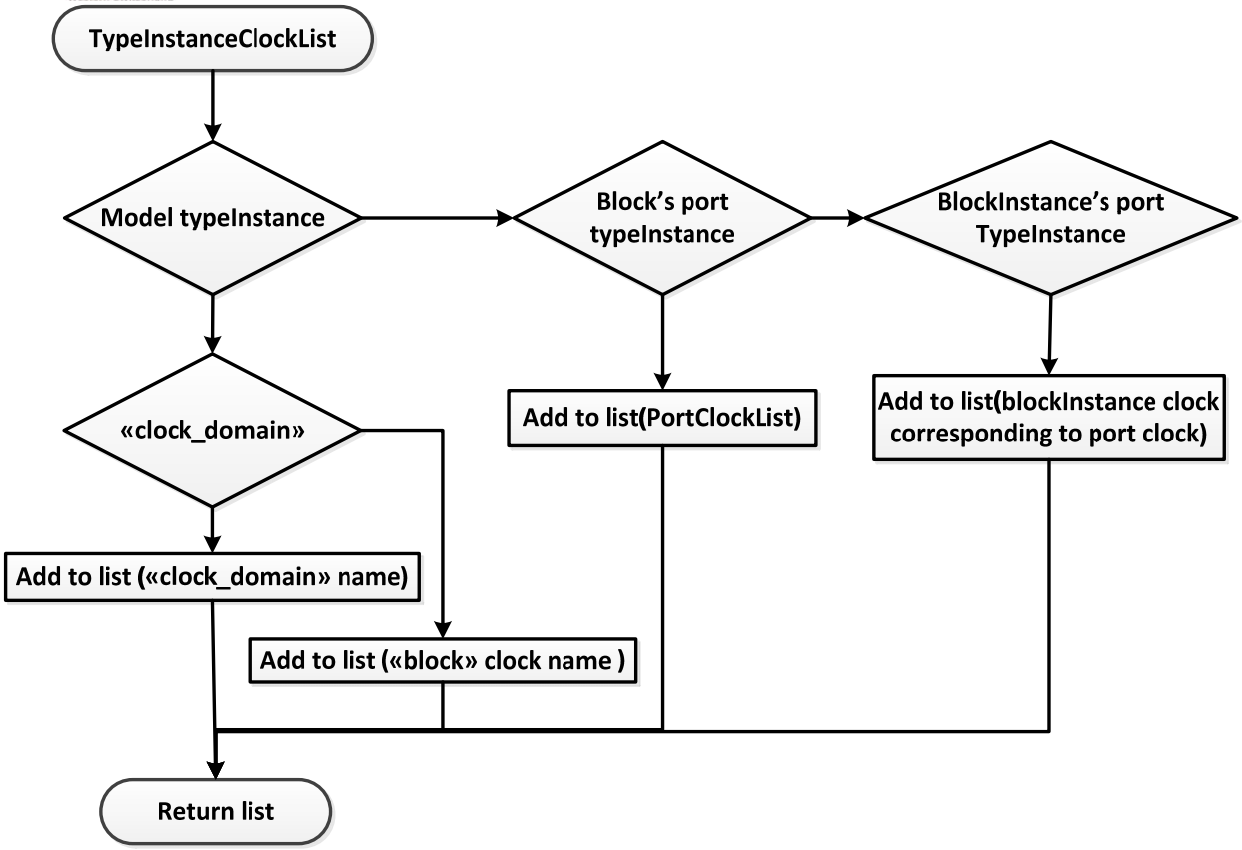


Figure 40: Clocks utilisés par une instance d'un type

7.2.3.3 PASSE 1.3 : CREATION DES SIGNAUX

Cette étape consiste à parser tous les Type / Port / Generic de bloc / TypeInstance pour créer dans chacun une liste de tous les signaux qui y transitent (nom, direction, nombre de bit).

7.2.3.3.1 typeSignalList

Pour l'analyse des signaux contenus par un type il y a deux cas :

- Si le type est un « baseType », on ajoute directement le signal du « baseType » transformé (direction/nom/stéréotype) à la liste de signaux.
- Si le type est un « Type » on interroge tous les attributs de ce type pour construire la liste de signal.

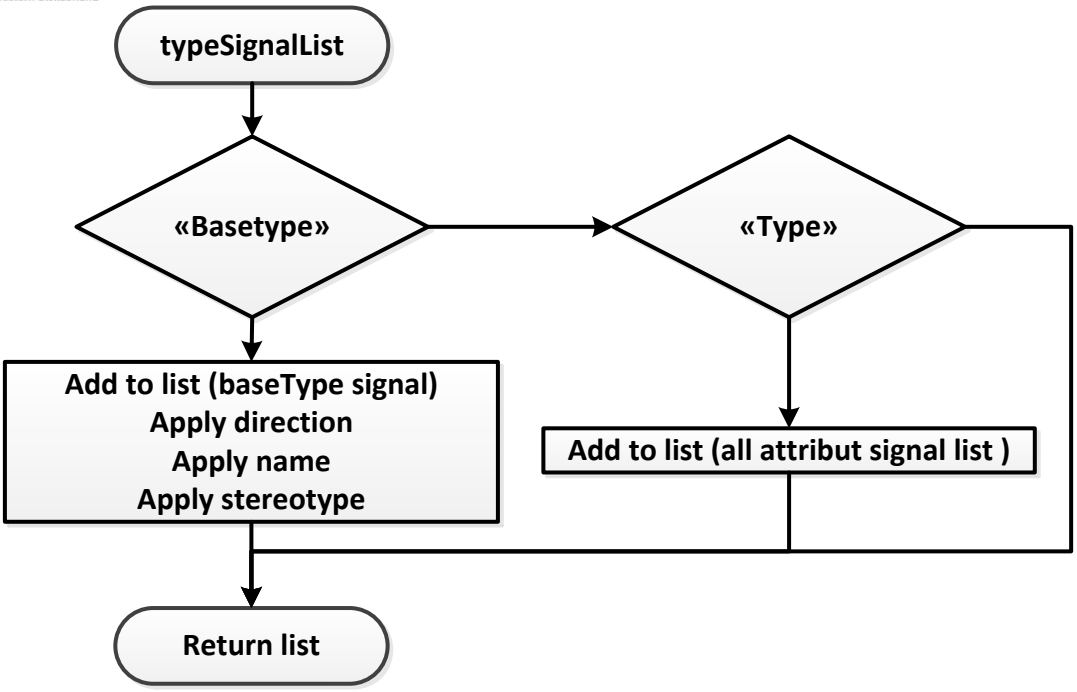


Figure 41: Signaux contenus par un type

7.2.3.3.2 attributSignalList

Pour déterminer les signaux qu'un attribut d'un type apporte, il faut récupérer tous les signaux du type de l'attribut puis les transformer.

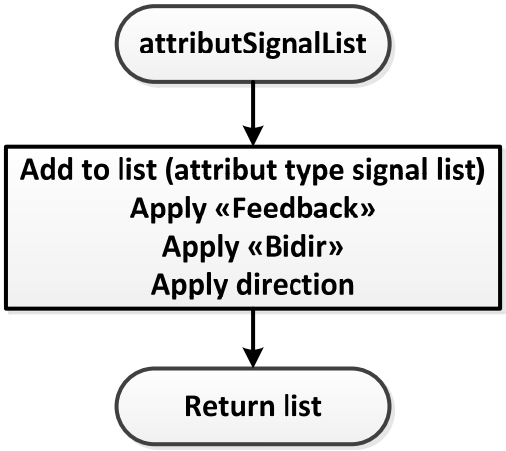


Figure 42: Signaux contenus par un attribut

7.2.3.3.3 portSignalList

Pour déterminer les signaux d'un port, il faut tout d'abord identifier de quel genre est le port (data port, Stream port…), puis, selon le cas, appliquer les bonnes transformations de signaux et ajouts d'arbitrations.

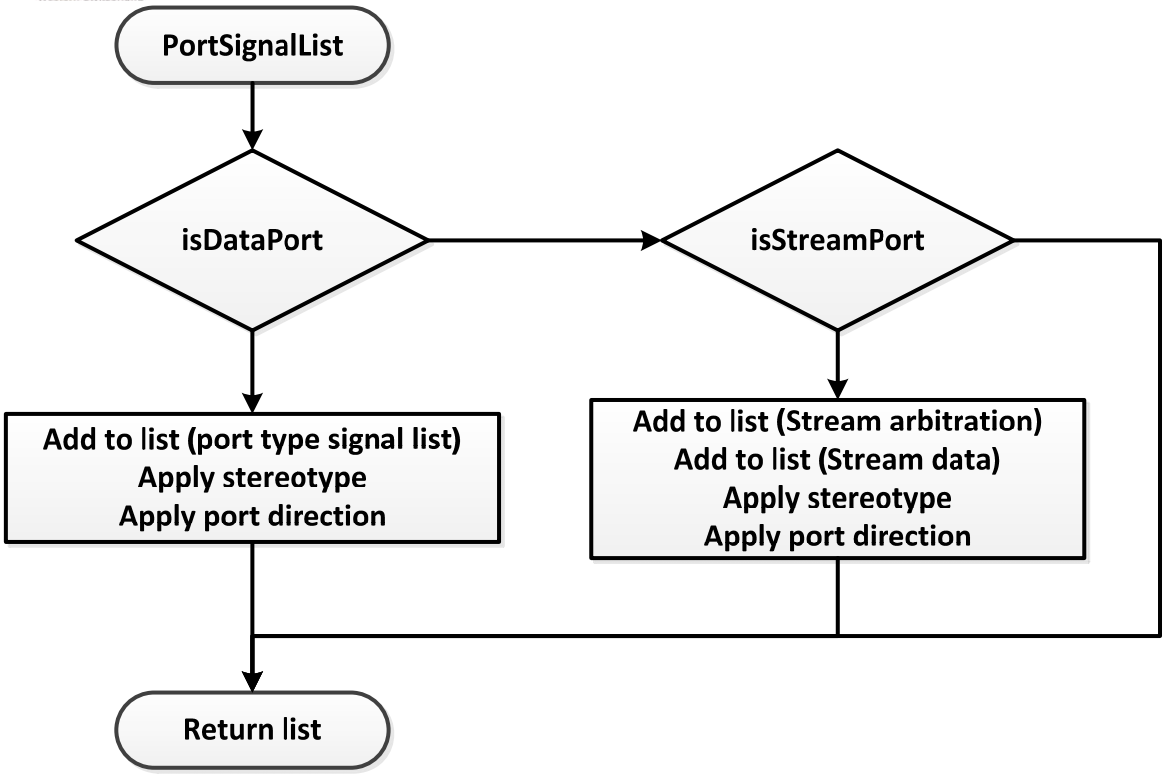


Figure 43: Signaux contenus dans un port

7.2.3.4 PASSE 2 : ANALYSE DES CONNEXIONS

C'est la passe „intelligente" de la génératrice de code.
Cette étape consiste à déterminer la fonction de chaque lien reliant les éléments du modèle et à y rajouter, si besoin est, des blocs/ligne de code/câblage supplémentaires pour effectuer cette fonction.

7.2.3.4.1 Implémentation actuelle

Actuellement, l'algorithme d'analyse des connexions ne gère que les connexions entre stream port ou entre data port. Pour ces deux types de connexions, la question des clocks domaines est gérée.

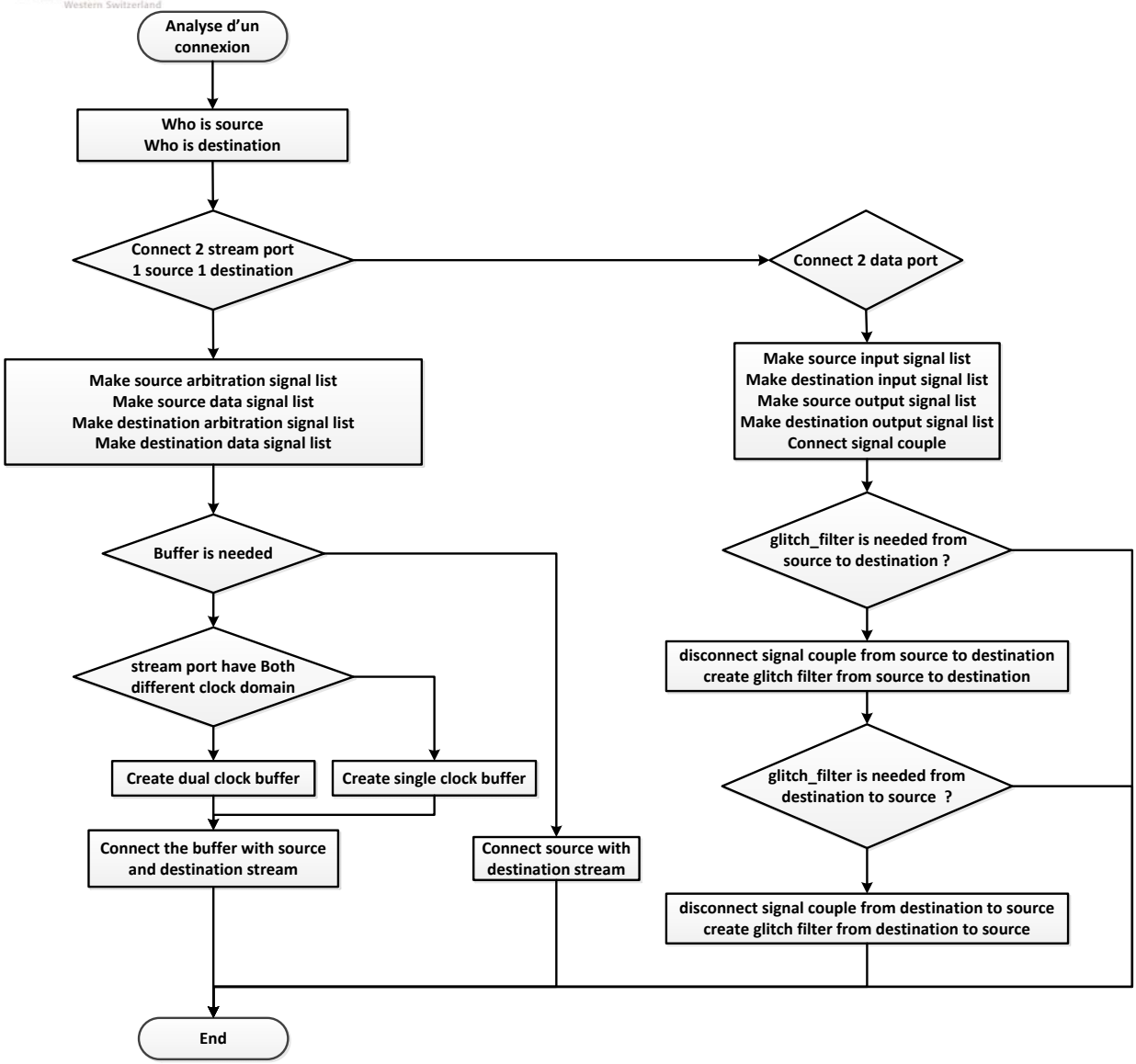


Figure 44: Analyse d'une connexion

7.2.3.4.2 Challenge futur

La passe 2 est la partie complexe de la génératrice, il faut encore y intégrer toute la gestion des memory access port, les functional port ainsi que toutes les interactions entre les ports de types différents.

7.2.3.5 PASSE 3

La génération des fichiers est très simple étant donné que tout le travail d'analyse a été effectué durant la passe 1 et 2.

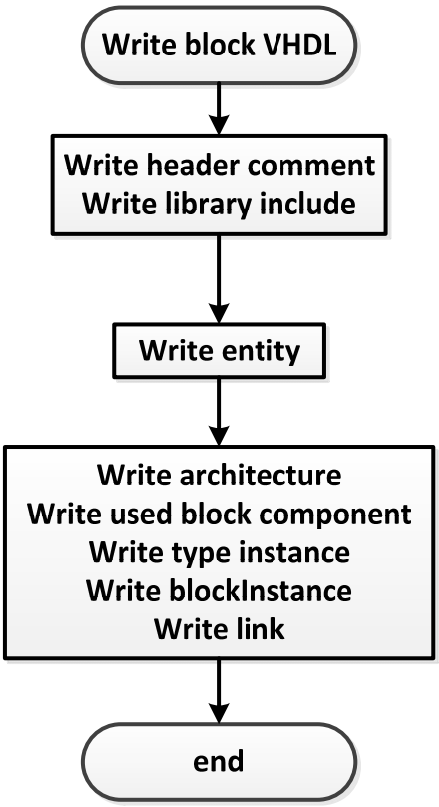


Figure 45: Écriture d'un bloc dans un fichier VHDL

7.3 IMPLEMENTATION

L'implémentation a été, selon les parties faites, en trois différents langages :

- Parcours et l'analyse du modèle : MQL
- Class / Structure de données : Java
- Création de fichiers VHDL : TGL

7.3.1 DIFFICULTES RENCONTREES

MQL et TGL étaient deux langages inconnus. Un certain temps a été requis avant de bien assimiler les possibilités du langage, en particulier celles offertes par les « transients links » qui permettent de lier des objets entre eux, ainsi que les possibilités d'intégrer du Java dans le code.

7.3.2 FONCTIONNALITE ACTUELLE

Actuellement la génératrice de code gère :

- La création de blocs
- Les clocks (domaine/ connect) sur les ports et les instances de bloc.
- Les génériques
- Les data port
- Les types comportant des bool/data/unsigned/signed/integer
- Les stream port
- Les connexions entre data port
    - o Les connexions standards
    - o Les connexions filtrées

  - Les connexions traversant deux clocks domaine
  - Les connexions bidirectionnelles
- Les connexions entre stream port
  - Entre stream port à arbitration différente
  - Entre stream port à clock domaine différent

7.3.3 TODO

- Tout ce qui touche au memory access port et aux functional port
- La génération des fichiers de contraintes
- Les connexions filtrées se connectant à un bit d'un signal
- Gérer les blocs et instances de bloc qui sont des „clock source"
- Gérer les énumérations

7.4 VERIFICATION

En l'état actuel, un petit modèle de test a été créé pour tester la génératrice de code. Ce modèle ne couvre que les fonctionnalités implémentées et il n'est pas impossible que des cas d'utilisation particuliers aient été oubliés.

8. CONCLUSION

L'implémentation du démonstrateur a permis de mettre en évidence certains défauts/manques des règles de modélisation qui ont été établies durant le travail de semestre. Les défauts ont été corrigés et les manques comblés.
Une des fonctionnalités majeures qui a été ajoutées est la possibilité de définir les contraintes des liens entre le design et le package des chips.

L'approche modèle pour établir le design d'un système comportant du hardware et/ou du software a de nombreux atouts :

- Abstraction du bas niveau
- Automatisation du traitement des interfaces entre matériel et logiciel
- Réduction des probabilités d'obtenir des codes sources comportant des bugs
- Création automatisée de bancs de test comprenant des fonctions capables de stimuler l'entrée sortie du bloc à tester (hardware).
- Gain de temps

L'ensemble des règles de modélisation forme un tout à présent cohérent et fonctionnel et c'est pour cette raison qu'une ébauche de génératrice de code a été entreprise.
Les résultats de cette ébauche de génératrice sont encourageants et méritent d'être implémentés plus amplement.

9. BIBLIOGRAPHIE

ALT, O., 2012. *Modellbasierte systementwicklung mit SysML.* s.l.:HANSER.
Berthouzoz, C. et al., 2005. *Synthesized UML, a Practical Approach to map UML to VHDL.* s.l.:s.n.
Green, P., s.d. *UML AS A FRAMEWORK FOR COMBINING DIFFERENT MODELS OF COMPUTATION.* s.l.:s.n.
Gubler, O. A., 2012. *Hardware Execution Framework,* s.l.: s.n.
[1] Papon, C., 2012. *Hybrid Systems,* s.l.: s.n.
Rieder, M. & Steiner, R., 2008. *Baseline for a Proposal of an UML Profile focused on Co-Design.* s.l.:s.n.
Steiner, R., 2004. *Co-Design,* s.l.: s.n.
Zahno, S., 2012. *Pure Digital,* s.l.: s.n.

10. GLOSSAIRE

Ce projet mélange des notions modèles, software, hardware voici un rapide récapitulatif de différents termes avec leur équivalence dans ces trois domaines.

Bloc

VHDL : Définition d'un composant, c'est-à-dire son entity
C++ : Définition d'une class, sa structure (.h)
UML : Class qui en SYSml est stéréotypée avec <<Block>>

Instance d'un bloc

VHDL : Instancier un bloc pour l'utiliser dans un autre bloc.
myBlockInstance ; myBlock(***port map***) ;
C++ : new myClass() ;
UML : Objet d'une class

Port

VHDL : Entrée sortie définie dans l'entity d'un bloc.
C++ : Peut être assimilé à une communication entre deux class via le partage d'une variable, d'appel de fonction d'une interface.
UML : Equivalent au composant Flowport de la norme SYSml.

Type

VHDL : Définition d'une structure de donnée, allant du bit jusqu'à la structure à plusieurs éléments tels que les records

```
type clock_time is record
    hour : integer range 0 to 12 ;
    minute : integer range 0 to 59 ;
    seconde : integer range 0 to 59 ;
end record ;
```

C++ : Définitions d'une struct ou d'une class ne possédant que des attributs.
UML : Définitions d'une class avec le stéréotype <<Type>> ne possédant que des attributs.

Stream

VHDL : C'est un bus qui transporte des données de manière discontinue (data_valid) d'un master à un slave sans signal d'adresse et muni d'une arbitration minimale (ack).
Analogie : Dans une connexion TCP/IP, un socket émet un flux/stream de bytes à un autre socket.

Memory_access

VHDL : C'est un bus mémoire tel que AHB APB AXI Avalon Wishbone
C++ : Le processeur y accède en tant que master.

IP

VHDL : ensemble de fichier VHDL formant une fonctionnalité particulière, librairie.
C++ : librairie pouvant être fournie sous forme de fichier source ou de fichier.a

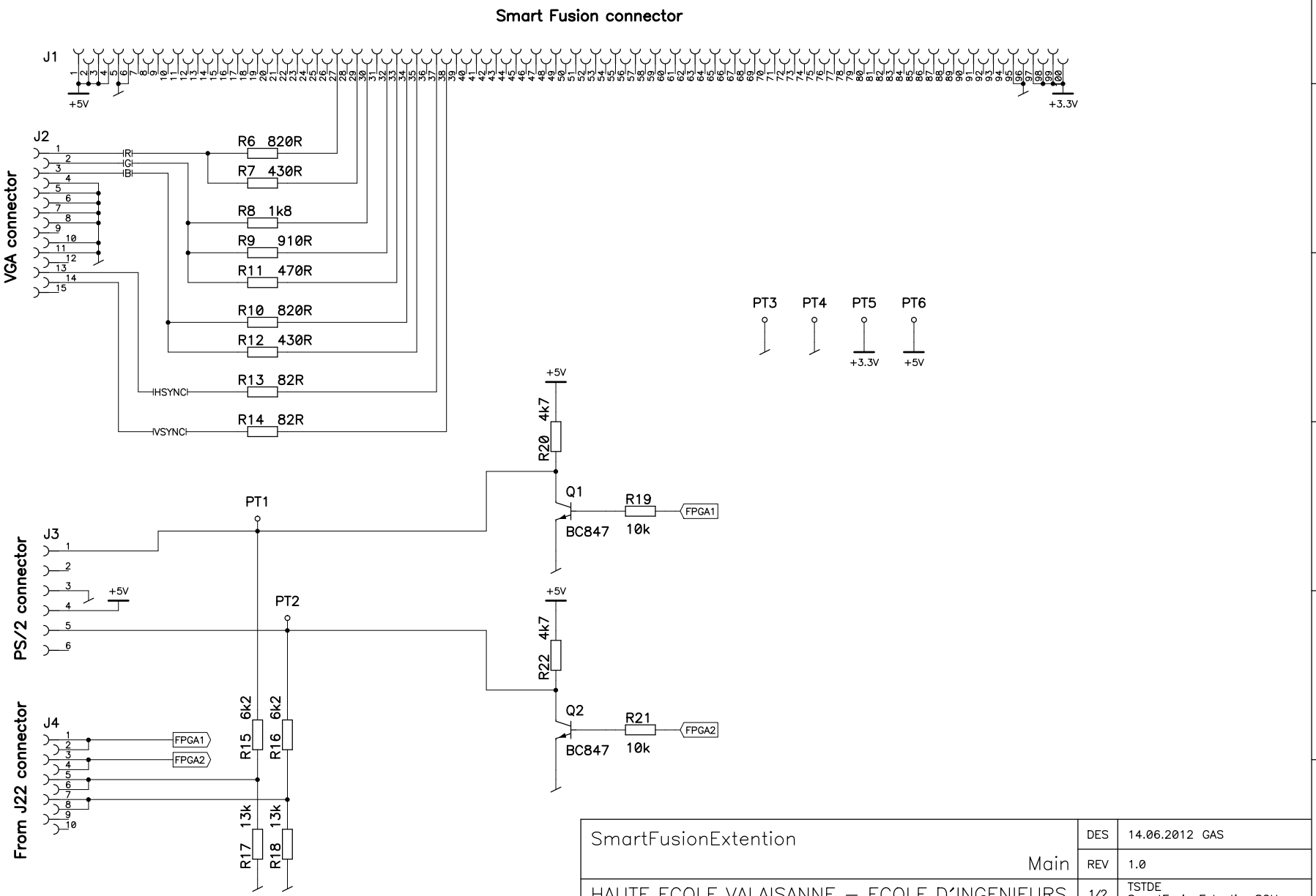
11. TABLE DES FIGURES

12. ANNEXE

Annexe 1 : Schématique board d'extension
Annexe 2 : CD-rom

Tous ces répertoires proviennent de C:\Users\papc\smartfusion (Si des problèmes de lien surviennent ave l'utilisation de ces projets)

- demo_model : Projet Rhapsody du démonstrateur
- demo_model\vhdl : code VHDL du démonstrateur
- AHBlite_try : Projet Libero 10.0 du démonstrateur
- AHBlite_try\SoftConsole : Projet C SoftConsole (eclipse) du démonstrateur
- Doc : Fichier utilisé pour la documentation du projet
- eclipse_workspace : Workspace pour MD Workbench de la génératrice
- profile : Projet Rhapsody du profile développé pour la modélisation
- xmi_gen_for_vhd : Projet Rhapsody utilisé pour tester la génératrice


Smart Fusion connector
J1
+5V
+3.3V
J2
VGA connector
R6 820R
R7 430R
R8 1k8
R9 910R
R11 470R
R10 820R
R12 430R
R13 82R
R14 82R
IHSYNCI
IVSYNCI
PT3
PT4
PT5
PT6
+3.3V
+5V
PT1
PT2
J3
PS/2 connector
+5V
R20 4k7
R22 4k7
Q1
Q2
BC847
BC847
R19 10k
R21 10k
FPGA1
FPGA2
J4
From J22 connector
R15 6k2
R16 6k2
R17 13k
R18 13k
SmartFusionExtention
Main
DES 14.06.2012 GAS
REV 1.0
HAUTE ECOLE VALAISANNE – ECOLE D'INGENIEURS
1/2
TSTDE SmartFusionExtention.SCH