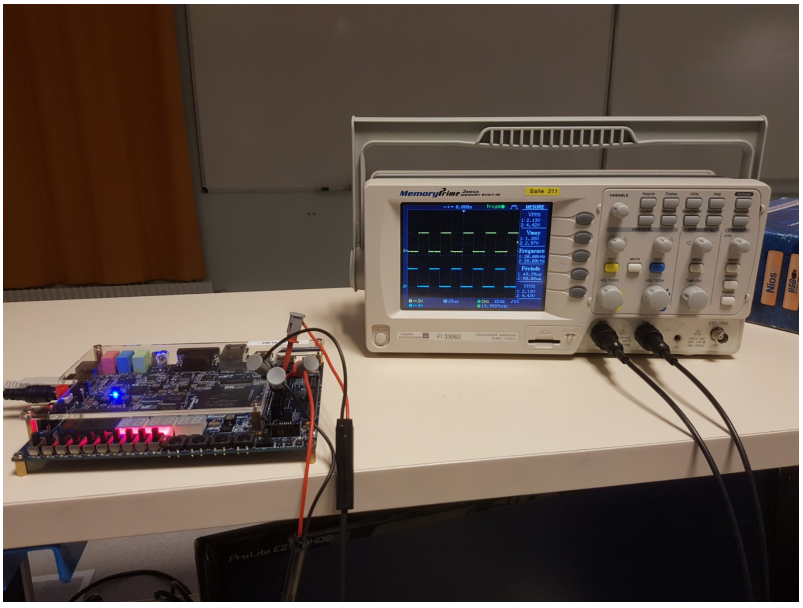
Université Clermont Auvergne

Spécialité Systèmes Embarqués

UE Conception Conjointe

Rapport de TP : Prototypage d'un contrôleur d'onduleur triphasé sur FPGA SoC

Etudiant : Mouhamadou Ndoye



Contents

1

Introduction

L'objectif du TP est de prototyper les différents blocs (matériels et logiciels) d'un contrôleur d'onduleur triphasé sur un FPGA. Il est destiner à piloter un moteur triphasé. Pour réaliser le contrôleur, on mettra en oeuvre une modulation de type vectorielle. L'architecture du système sera valider en simulation ensuite une mesure des performances du système en terme de ressources utiliser et de temps de calcules sera faite. Ces analyses permettront de juger du ratio coût/performance pour une éventuelle gravure sur silicium.

Remarques Importantes :

J'ai utilisé la platine 18 durant mes séances de TP. Les broches GPIO_D1 et GPIO_D3 ne répondent pas sur cette platine. Durant mes phases de test sur l'oscilloscope, ces broches ne se mettaient pas à zéro lors des phases de reset du système. Ils restent au niveau haut quelques soient les valeurs mises en entrées des broches. Pour les testes sur l'oscilloscope, utiliser les broches GPIO_D4 et GPIO_D5 qui correspondent aux sortie high_line_W et low_line_W ou sinon modifier l'assignement des pins afin de tester toutes les sorties du système.

Le projet a été crée et compilé sous quartus lite edition v18.0 si cette version n'est pas disponible, privilégié la version 18.1 afin d'éviter à devoir mettre à jour les IP sous Qsys. Pour charger le projet sur le board utilisé également Intel Program Version 18.0 ou 18.1

2

Spécification du système

Comme dit dans l'introduction, le système mis en place est un contrôleur d'onduleur triphasé destiné à piloter un moteur triphasé. On mettra en place une modulation de type vectorielle pour le contrôle de l'onduleur. Le système mise en place doit avoir les caractéristiques suivantes :

Horloge Bloc	100 MHz
Fréquence signaux de sortie	[5 ; 25] KHz
Résolution rapport cyclique	10 bits minimums
Bande morte	[0 ; 4µs]
Formes d'onde symétriques	OUI
Entrée de mise en sécurité	OUI
Interruption fin de période	OUI
Registres de contrôle	6

Le matériel utilisé est une platine DE10-standard.

NB : L'horloge du bloc doit être cadencée à 100Mhz or les 4 horloges connectées à la partie FPGA du système sont toutes cadencées à 50Mhz. Pour palier à ce problème et répondre au cahier des charges, j'ai rajouté sur Qsys un PLL pour augmenter la fréquence à 100Mhz.

3

Partitionnement matériel/logiciel

Pour ce qui est du partitionnement matériels/logiciel, les blocs de génération de la MLI ainsi que les registres de contrôle sont contenus dans la partie FPGA. le programme principal est contenu dans la partie HPS. Les deux parties HPS et FPGA communiquent grâce au bus avalon. Voir figure ci-dessous pour plus de détails sur le partitionnement.

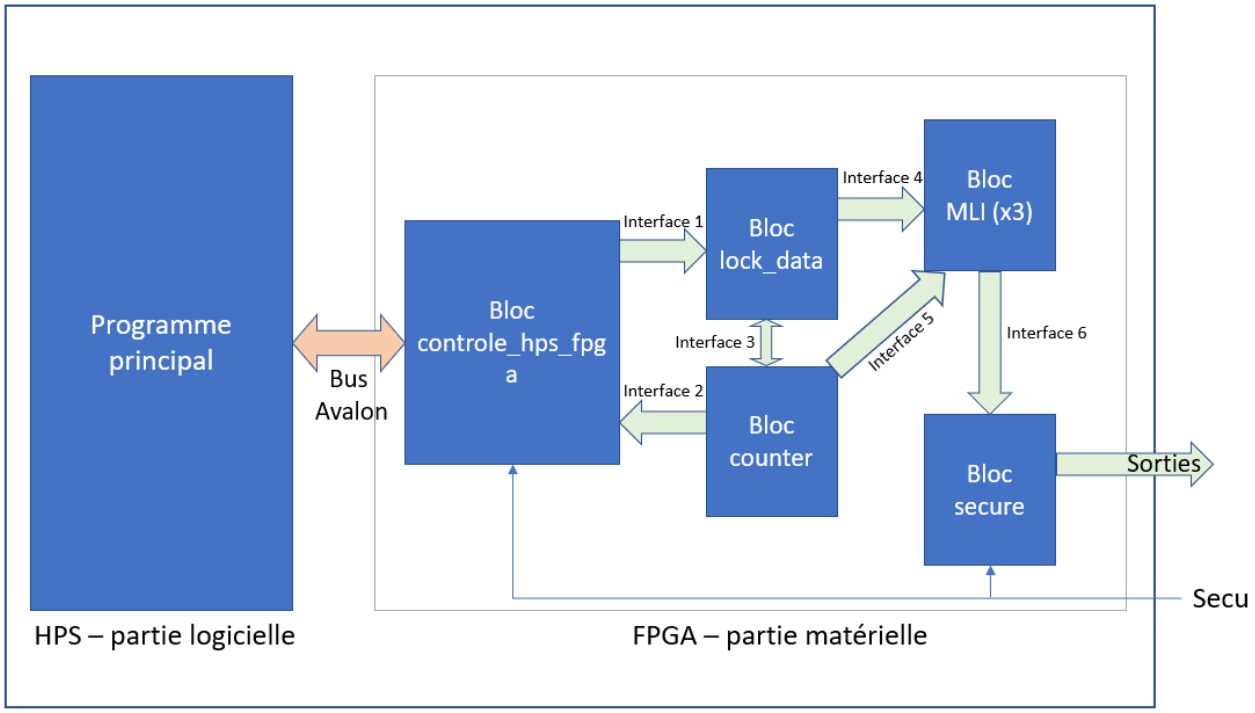


Figure 1: PARTITIONNEMENT MATÉRIEL/LOGICIEL.

4

Étude des interfaces et des blocs matériels

4.1 Bloc secure

C'est le dernier bloc dans la chaîne de génération de la MLI. Il sert à mettre à **"0"** toutes les sorties lorsque l'entrée sécurité passe à **"1"**. Avoir un bloc dédié uniquement à la mise en sécurité du système permet un débogage plus simple en cas de test. ça permet également une modification plus simple de l'architecture du système de sécurité sans pour autant modifier l'architecture du système de génération de la MLI. Les entrées/sorties de ce bloc sont décrites sur la figure 2.

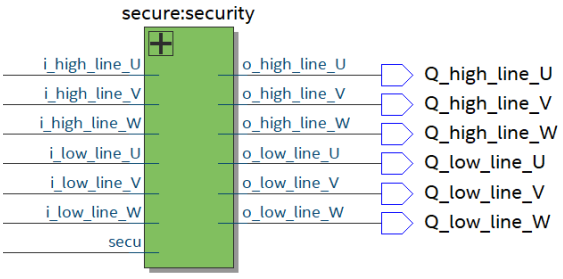


Figure 2: ENTRÉES / SORTIES BLOC SECU.

4.2 Bloc counter + MLI

Ces deux blocs permettent de générer la forme d'onde symétrique des signaux en sorties du système. Le bloc **counter** réalise un compteur/dé-compteur et le bloc **MLI** génère la MLI en fonction de la valeur dans **counter** et du rapport cyclique. Les entrées/sorties de ces blocs sont décrites sur les figures ci-dessous.

.

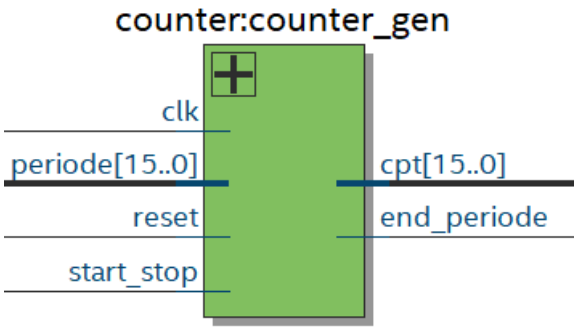


Figure 3: COUNTER

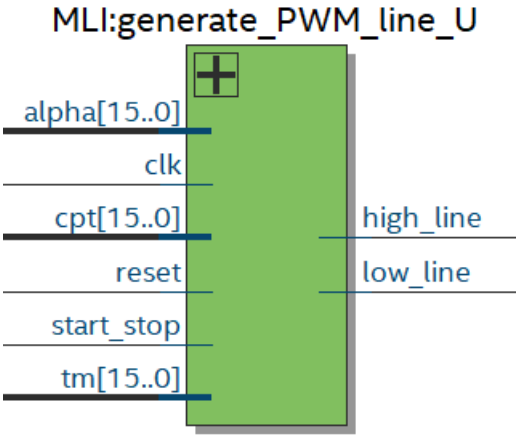


Figure 4: MLI

4.3 Bloc lock_data

IL a pour rôle de verrouiller les valeurs de la période, des rapports cycliques (U, V, W) et du temps mort jusqu'à la fin de la période actuelle. A la fin de chaque période, ce bloc met à jour les sorties avec les nouvelles valeurs envoyer depuis le bloc **controle_hps_fpga**. Les entrées/sorties de ce bloc sont décrites sur la figure 5.

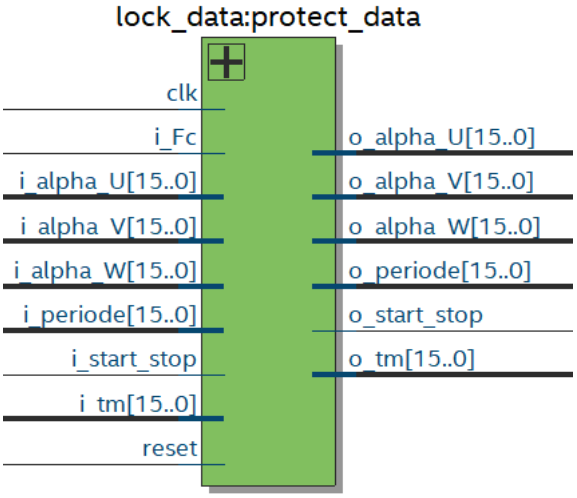


Figure 5: ENTRÉES / SORTIES BLOC LOCK_DATA.

4.4 Bloc controle_hps_fpga

Ce bloc vient écrire sur les 6 registres de la partie matérielle. Il reçoit les valeurs à écrire dans les registres depuis la partie HPS grâce au bus avalon. Les entrées/sorties de ce bloc sont

décrites sur la figure 6.

Fonctionnement : Pour écrire ou lire dans les registres, la partie HPS envoie en même temps l'adresse, byteenable = "11" et le type d'opèration (écriture ou lecture). A la fin de chaque période, ce bloc envoie à la partie HPS une demande d'interruption grâce au signal ins_irq

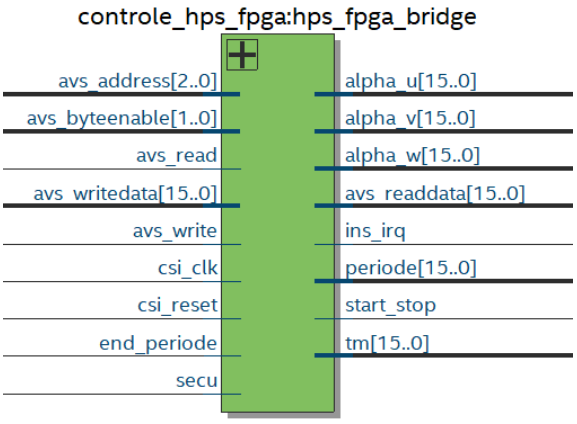


Figure 6: ENTRÉES / SORTIES BLOC CONTRÔLE_HPS_FPGA.

La figure 7 montre le mapping des adresses de chaque registre du côté maître (HPS) et du côté esclave (FPGA).

Adresse maître (HPS)	Adresse esclave (FPGA)	15 11 7 3 0	
0x0003000	000		State_register
0x0003002	001		registre alpha_U
0x0003004	010		Registre alpha_V
0x0003006	011		Registre alpha_W
0x0003008	100		Registre periode
0x000300A	101		Registre temps_mort

Figure 7: MAPPING ADDRESSES.

4.5 Etude du Bus Avalon

Ce bus sert à la communication entre la partie HPS et la partie FPGA. Le principal avantage du bus avalon est la simplicité de son architecture. l'architecture du bus est conçue pour pouvoir dialoguer avec des registres ce qui est le cas dans le système qu'on veut concevoir. Les différentes signaux constituant le bus sont présentés la figure 8.

NB : On remarquera qu'on a pas de signal chip_select car la communication se fait exclusivement entre deux composants (master / slave).
NB : on a pas de signal waitrequest. Pour palier à cela, Je retard la lecture et l'écriture d'un cycle d'horloge après avoir transmis l'adresse.
NB : La partie HPS ne possède pas de bus avalon mais plus tôt un bus AXI. Altera réalise l'interfaçage entre le bus avalon et le port AXI de façon totalement transparent pour nous en tant que utilisateur.

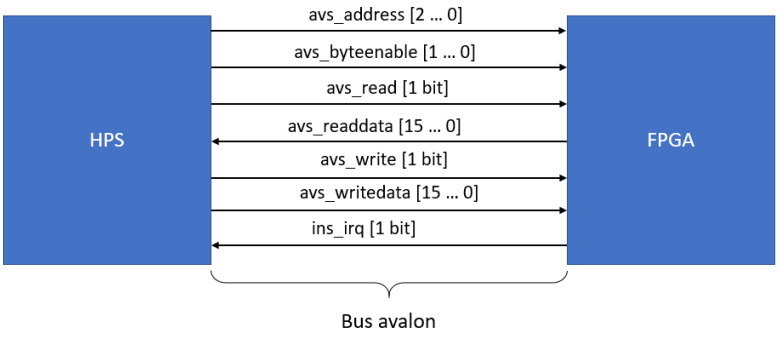


Figure 8: Bus Avalon.

5

Étude des fonctions logiciels

La partie software est contenue dans la partie HPS du système. elle est constitué d'un programme principal qui appelle une routine d'interruption à chaque demande d'interruption envoyée par la partie HPS (partie 4.4) . Dans la routine d'interruption, je configure les différents registres du système en écrivant dans les adresses côté maître (figure 7).

6

Tests et Vérifications

J'ai simulé et validé les différents blocs du système sur modelsim d'abord individuellement (figure 9, 10, 11) ensuite en les intégrant au plus haut niveau (figure 12) avant de créer le composant sur Qsys.

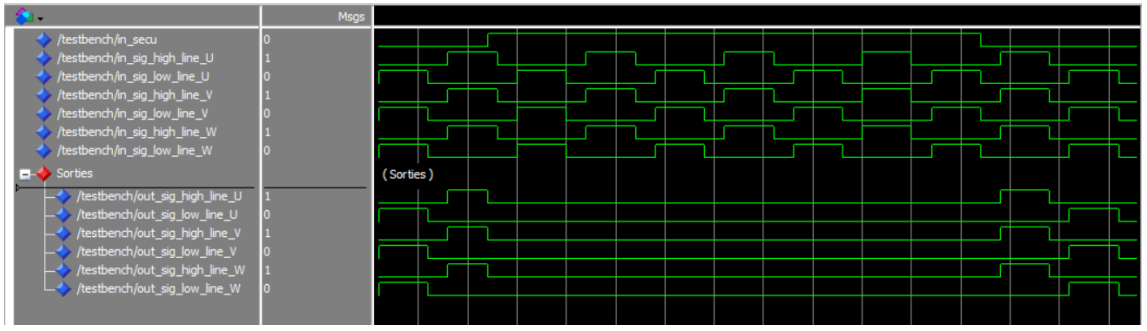


Figure 9: SIMULATION BLOC COUNTER + MLI + SECURE.

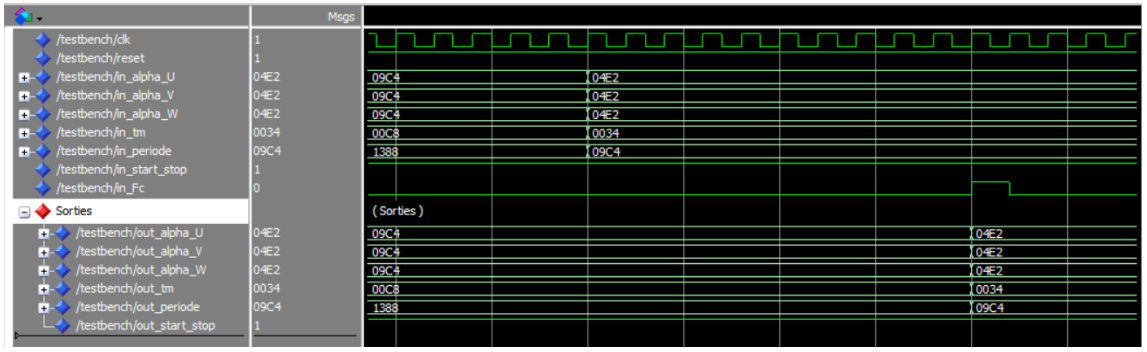


Figure 10: SIMULATION BLOC LOCK_DATA.

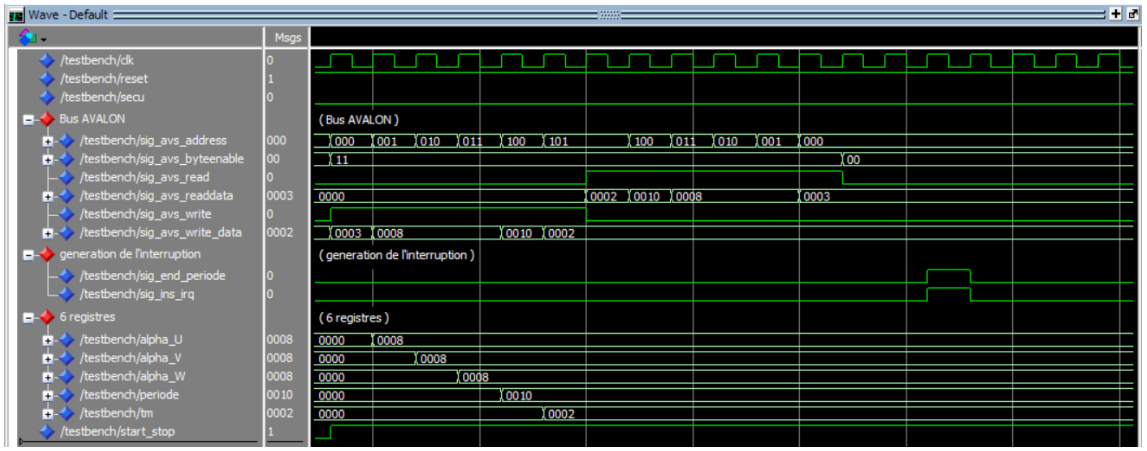


Figure 11: SIMULATION BLOC CONTROLE_HPS_FPGA.

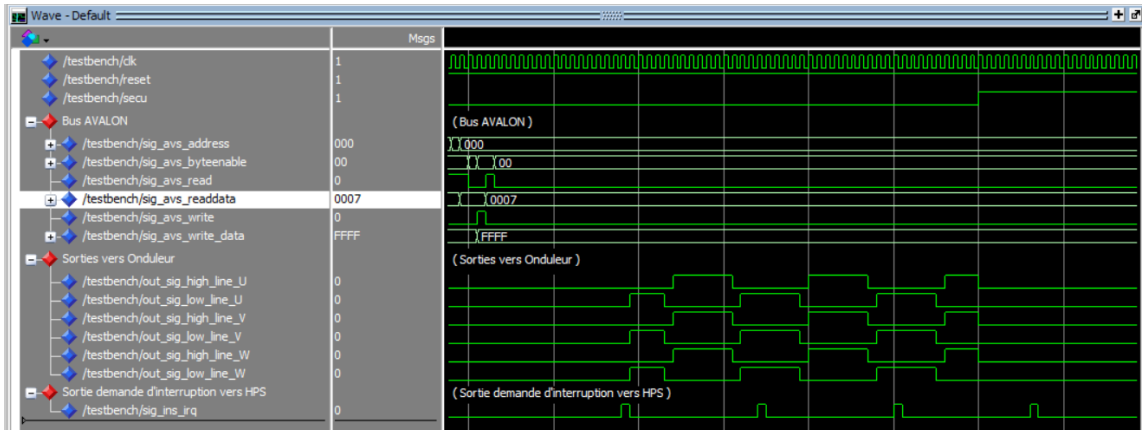


Figure 12: SIMULATION SYSTEM GLOBAL (controle_onduleur).

J'ai ensuite créé le composant sous Qsys et validé le bon fonctionnement de l'ensemble du système d'abord avec **System Console** et ensuite avec **Intel Monitor Program**.

7

Rapport d'implémentation

Le tableau 13 résume l'utilisation des ressources du FPGA après synthèse du projet. Le projet utilise 21.91% des registres disponibles. 512 bits de blocs mémoires liée à l'utilisation du **JTAG_Console.** le projet utilise également 1 DLL et 1 PLL pour fournir les 100Mhz du cahier des charges à partir de l'horloge de 50Mhz fournie de base dans le FPGA. Le rapport de synthèse montre également que les fichiers de contraintes sont appliqués (figure 14) l'entrée d'horloge clk_clk de notre système reçoit bien l'horloge à 50Mhz du FPGA. Je n'ai pas définie les contraintes pour les 3 horloges :
hps_io_hps_io_i2c0_inst_SCL
hps_io_hps_io_i2c1_inst_SCL
hps_io_hps_io_usb1_inst_CLK
provenant de la partie HPS raison pour laquelle elles sont signalées en rouge mais cela ne gêne pas dans le fonctionnement du système.

Ressources	Utilisé	Disponible	pourcentage %
Flip-Flop	2221	110K	21.91
Block memory bits	512	5530K	< 1.00
PLLs	1	15	7.00
DLLs	1	4	25.00

Figure 13: RAPPORT SUR LES RESSOURCES UTILISEES.

Clock Status Summary

<<Filter>>

	Target	Clock	Type	Status
1	altera_reserved_tck	altera_reserved_tck	Base	Constrained
2	clk_clk	clk_50MHz	Base	Constrained
3	core_hps_0:hps_0|core_hps_0_hp...s_enable_ctrl~DQSENABLEOUT_DFF	hps_0|hps_io|border|hps_sdram_inst|hps_sdram_p0_sampling_clock	Generated	Constrained
4	core_hps_0:hps_0|core_hps_0_hp...s_enable_ctrl~DQSENABLEOUT_DFF	hps_0|hps_io|border|hps_sdram_inst|hps_sdram_p0_sampling_clock	Generated	Constrained
5	core_hps_0:hps_0|core_hps_0_hp...s_enable_ctrl~DQSENABLEOUT_DFF	hps_0|hps_io|border|hps_sdram_inst|hps_sdram_p0_sampling_clock	Generated	Constrained
6	core_hps_0:hps_0|core_hps_0_hp...s_enable_ctrl~DQSENABLEOUT_DFF	hps_0|hps_io|border|hps_sdram_inst|hps_sdram_p0_sampling_clock	Generated	Constrained
7	core_hps_0:hps_0|core_hps_0_hps_i...am_inst|hps_sdram_pll:pll|afi_clk	core_hps_0:hps_0|core_hps_0_hps_io...ps_sdram_pll:pll|afi_clk_write_clk	Base	Constrained
8	core_hps_0:hps_0|core_hps_0_hps_io...st|hps_sdram_pll:pll|pll_write_clk	core_hps_0:hps_0|core_hps_0_hps_io:..._pll:pll|pll_write_clk_dq_write_clk	Base	Constrained
9	hps_io_hps_io_i2c0_inst_SCL		Base	Unconstrained
10	hps_io_hps_io_i2c1_inst_SCL		Base	Unconstrained
11	hps_io_hps_io_usb1_inst_CLK		Base	Unconstrained
12	memory_mem_ck	memory_mem_ck	Generated	Constrained
13	memory_mem_ck_n	memory_mem_ck_n	Generated	Constrained
14	memory_mem_dqs[0]	memory_mem_dqs[0]_IN	Base	Constrained
15	memory_mem_dqs[0]	memory_mem_dqs[0]_OUT	Generated	Constrained
16	memory_mem_dqs[1]	memory_mem_dqs[1]_IN	Base	Constrained
17	memory_mem_dqs[1]	memory_mem_dqs[1]_OUT	Generated	Constrained
18	memory_mem_dqs[2]	memory_mem_dqs[2]_IN	Base	Constrained
19	memory_mem_dqs[2]	memory_mem_dqs[2]_OUT	Generated	Constrained
20	memory_mem_dqs[3]	memory_mem_dqs[3]_IN	Base	Constrained
21	memory_mem_dqs[3]	memory_mem_dqs[3]_OUT	Generated	Constrained

Figure 14: RAPPORT CONTRAINTES HORLOGE SYSTÈME.

La boucle principale de mon programme C s'exécute en 32 cycle d'horloge.

8

Conclusion

En terme de spécification client, j'ai rempli le cahier des charges demandé néanmoins le système reste améliorable en terme d'architecture du code. On pourrait également rajouter d'autres fonctionnalités au système par exemples activer la demande d'interruption dés que le bouton sécu est activé sans attendre la fin du cycle en cours ou encore utilisé un bus AXI4-lite à la place du bus AVALON. On pourrait également rendre le système configuration des registres un peu plus simple pour le client en lui faisant juste entrer la période souhaiter sans prendre en compte la fréquence d'horloge du système.