Compte rendu de projet de circuits programmables FPGA sous la direction de Jean Vuillemin et Mark Shand

Antoine Miné

3 juin 1999

Abstract

This document shows an application example of reprogrammable circuits — Xilinx FPGA — as a student project, using a Pamette card. The goal of this project is to drive a TGA video card with an FPGA in order to draw a life-like game automaton on a screen. The automaton is provided by Denis Oddoux as a logical circuit emitting a flow of pixels we have to store in the frame-buffer of the video card.

This project is based on an existing PCI Bridge due to Mark Shand allowing transparent communication between the CPU and any PCI device thru a Pamette card. We show here how to implement a hardware derivation in the Bridge in order to drive directly the video card while maintaining the direct communication between the CPU and the video card.

1 Introduction

1.1 But du projet

Le but de ce projet est de piloter une carte graphique TGA à partir de circuits reprogrammables FPGA.

Ce projet est complémentaire du projet de Denis Oddoux intitulé *simulation du jeu de la vie amélioré avec un* FPGA. Le but est d'utiliser le pilote graphique développé dans mon projet pour visualiser les images générées par le projet de Denis.

1.2 Matériel utilisé

La partie électronique reprogrammable est constituée d'une carte Pamette comportant quatre circuits Xilinx 4020e. Le simulateur du jeu de la vie de Denis Oddoux occupe un de ces circuits.

La carte écran pilotée est une carte PCI TGA EBVXG 8-bit dotée de 1Mo de mémoire vidéo.. Elle comporte un processeur graphique DECchip 21030 (référence Digital EC-N0683-72) et un ramdac Brooktree bt485a.

La carte Pamette est dotée de deux connecteurs PCI. Le premier sert à placer la carte Pamette dans le slot PCI d'une station de travail DEC Alpha. La carte TGA est connectée au second.

En plus des quatre circuits, la Pamette comporte un circuit, le *PIF*, dont le programme – fixé une fois pour toute – permet de décoder les commandes PCI venant de la station de travail. Un des quatre circuits programmables est chargé avec un programme, symétrique du PIF, permettant de communiquer avec la carte TGA en utilisant le protocole PCI. Ces deux PIFs ont été programmés par Mark Shand.

Entre les deux PIFs, un programme – également du a Mark Shand – transmet les commandes PCI ; c'est le *Bridge*.

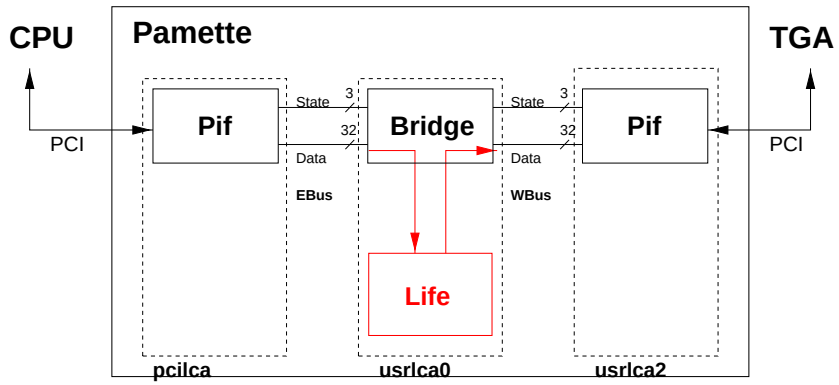
L'outil principal de développement utilisé est la bibliothèque C++ PamDC qui s'interface avec les outils de conception de circuit de Xilinx. Le développement s'est fait sur une station de travail Alpha tournant sous OSF1.

1.3 Travail effectué

Programmée avec le Bridge et les deux PIFs, la carte Pamette se comporte de manière transparente aux requêtes PCI de la part de l'ordinateur.

Le but premier du projet est de poser une *dérivation* pour permettre au circuit de Denis d'écrire dans la mémoire écran de la carte TGA sans perturber la communication entre la station de travail et la carte TGA.

J'ai donc modifié le circuit Bridge de Mark Shand pour poser cette dérivation et l'interfacer avec le circuit de Denis.



Pour obtenir une image sur un écran, il est également indispensable d'initialiser la carte graphique. Il s'agit d'une opération assez complexe mais qu'il n'est nécessaire de faire qu'une seule fois. C'est pourquoi elle est faite par le soft et non par la Pamette.

2 Schéma de fonctionnement du Bridge

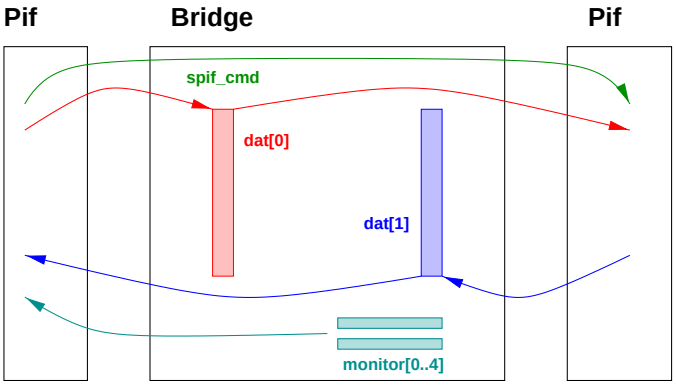
Cette section présente le fonctionnement du Bridge en détail ainsi que les modifications apportées par la dérivation.

2.1 Fonctionnement général

Lors d'un transfert de mémoire entre le CPU et la carte TGA, le Bridge sert de *relais*.

Le schéma du Bridge est très symétrique : les deux PIFs utilisent le même protocole de communication avec le Bridge. Le terme *hôte* caractérisera aussi bien le CPU que la carte TGA lors d'une transaction.

Chacun des hôtes dialogue avec le Bridge indépendemment de l'autre hôte. L'hôte est maître et initie des *requêtes*. Chaque *transfert* de mémoire entre les deux hôtes consiste en deux étapes : une requête du premier hôte écrit des données qui sont stockées dans le Bridge, puis une requête du deuxième hôte lit ces données.



Chaque requête d'un hôte est décomposée en *cycles* de 30 ns chacun (33 MHz) qui suivent un certain protocole. Ce protocole est interne au Bridge et aux PIFs et diffère du protocole PCI.

2.1.1 Détail d'une requête entre un hôte et le Bridge

Le Bridge est en communication avec le PIF du CPU par le `EBus` ainsi que le PIF de la carte TGA par le `WBus`. Chacun de ces bus est en fait composé de deux parties : le premier bus porte *32 bits de données* `Bus<0..31>` et le second bus porte *3 bits d'état* `Bus<32..35>`.

Le bus d'état est contrôlé exclusivement par le PIF tandis que le bus de données est tantôt contrôlé par le PIF, tantôt par le Bridge, selon la direction du transfert de donnée. La valeur de l'état sert à se repérer lors d'un transfert. Cela permet de savoir dans quel sens fonctionne le bus de données lors de ce cycle et si celui-ci présente une adresse, un mot de donnée ou une commande PCI.

Un exemple de requête est donné dans le tableau suivant :

Cycle	État	Données
0	AUI	—
1	DW	0x12345678
2	DW	—
3	DV	—
4	DW	0x11111111
5	DV	—
6	DW	0x22222222
7	DW	—
8	DV	—
9	IDLE	0x33333333
10	IDLE	—

Cette requête correspond à l'écriture des trois mots 0x11111111, 0x22222222 et 0x33333333 aux adresses 0x12345678, 0x1234567C et 0x12345680.

Le Bridge détecte le début de cette requête quand l'état du bus passe de IDLE à AUI. L'hôte présente l'adresse de la première écriture le cycle suivant le passage à l'état AUI. Les mots à écrire apparaîssent ensuite sur le bus de données un cycle après chaque état DV (Data Valid). La fin de la requête est marquée par le retour à l'état IDLE.

2.1.2 Détail d'un transfert entre hôtes

Lors d'un transfert entre le CPU et la carte TGA, les mots de données sont stockés dans une RAM double-port. Le terme *double-port* signifie qu'il est possible, pendant un même cycle, de lire un bit à une adresse et d'écrire un autre bit à une adresse différente. Ceci assure l'indépendance entre une requête d'écriture d'un hôte et une requête de lecture de l'autre hôte.

Cette RAM double-port a une capacité de 16 mots de 32 bits dans chacun des sens CPU vers carte TGA et carte TGA vers CPU ; elles se nomment `dat[0]` et `dat[1]`.

Lors d'une écriture, les 4 bits de poids faible de l'adresse servent d'offset dans la RAM double-port. Cela permet à un hôte de stocker 16 mots consécutifs avant que l'autre hôte ne commence à lire ces mots.

En plus des transfert mémoires entre un hôte et la RAM double-port du Bridge, un mécanisme permet au CPU d'envoyer directement un mot de commande PCI à la carte TGA. Lorsque le CPU demande à écrire un mot à l'adresse spéciale 0x800000, le Bridge envoie directement ce mot au bus de données `WBus<0..31>` du PIF connecté à la carte TGA via le chemin `spif_cmd` plutôt que de le stocker dans `dat[0]`.

Par ce chemin, le CPU envoie d'abord une adresse, puis une commande PCI qui sont interprétées par le PIF de la carte TGA comme un appel de

requête. La carte TGA répond par une demande de requête via le processus décrit dans la section précédente. L'adresse et le mot de commande suffisent à eux seuls a définir complètement la requête puisque le mot de commande contient la taille et la direction (lecture/écriture) du transfert.

D'autre part, le Bridge gère en interne cinq registres de statu appelés `monitor[0..4]`. Ceux-ci sont accessibles au CPU par une lecture aux adresse spéciales 0x800000, 0x900000, 0xA00000, 0xB00000 et 0xC00000.

Ces registres contiennent l'adresse de la dernière requête ainsi qu'un compteur du nombre de requêtes de chacun des hôtes et dans chaque sens. Ils permettent au logiciel de savoir si la carte TGA a terminé la dernière requête demandée via `spif_cmd`. Ils permettent également de savoir quand le `WBus` (PIF de la carte TGA) est libre et qu'il est possible d'envoyer une commande PCI ou une adresse via `spif_cmd`.

Ainsi les protocoles de transfert entre le CPU et la carte TGA sont les suivants :

Le CPU écrit dans la carte TGA
Le CPU remplit `dat[0]`
Le CPU attend que le `WBus` se libère
Le CPU envoie l'adresse dans `spif_cmd`
Le CPU attend que le `WBus` se libère
Le CPU envoie la commande dans `spif_cmd`
La carte TGA lit `dat[0]`

Le CPU lit dans la carte TGA
Le CPU attend que le `WBus` se libère
Le CPU envoie l'adresse dans `spif_cmd`
Le CPU attend que le `WBus` se libère
Le CPU envoie la commande dans `spif_cmd`
La carte TGA écrit `dat[1]`
Le CPU attend que la carte ait finit d'écrire
Le CPU lit `dat[1]`

2.2 Schéma de fonctionnement de la dérivation

2.2.1 Interface avec le projet de Denis Oddoux

Le circuit de Denis Oddoux stocke la matrice entière du jeu de la vie dans une RAM statique de la Pamette accessible directement. Au fur et à mesure que des nouveaux pixels sont calculés, la RAM statique est mise à jour.

Les pixels nouvellement calculés sont également émis en direction de la dérivation qui doit les écrire dans la mémoire écran de la carte TGA.

La dérivation tient à jour un compteur lui permettant de savoir à quelle adresse écrire ces pixels.

2.2.2 Écriture dans la mémoire écran

Lors qu'elle est active, la dérivation prend le contrôle du bus `WBus`. Il n'est donc plus possible d'envoyer de mot de commande via `spif_cmd`. Par contre le CPU peut encore remplir `dat[0]` ou lire `dat[1]` ainsi que consulter les mots d'état `monitor[0..4]`.

La dérivation fonctionne comme une machine à états. Chaque état correspond à l'émission d'un certain mot sur le bus de données `WBus<0..31>`. Le passage à l'état suivant est conditionné par les trois bits d'état `WBus<32..34>`.

La suite des états de la machine à états suit les protocoles explicités dans la section 2.1. L'adresse et le mot de commande PCI, normalement envoyés par le CPU via `spif_cmd` sont envoyés directement sur le bus de données de `WBus`. La dérivation attend explicitement que la carte TGA réponde en initiant une requête, puis elle s'occupe de fournir directement sur le `WBus` les mots de donnée.

Voici la suite des états successifs de la dérivation :

État	Mot écrit sur `WBus<0..31>`	Condition attendue sur `WBus<32..35>`
0	—	IDLE
1	adresse	—
2	—	IDLE
3	commande	—
4	—	IDLE
5	—	AUI
6-8	—	—
9-12	données	—
13	—	IDLE

Les symboles — indiquent soit qu'aucun mot de donnée n'est envoyé (s'il apparaît dans la $2^{ème}$ colonne), soit que la dérivation passe à l'état suivant au cycle suivant sans tester l'état du `WBus` (s'il apparaît dans la $3^{ème}$ colonne).

Arrivée a l'état 14, la dérivation recommence un nouveau transfert en revenant à l'état 0.

2.2.3 Contrôle de la dérivation

Quand le CPU a besoin d'écrire ou de lire dans la carte TGA, il faut au préalable qu'il désactive la dérivation pour libérer le bus `WBus`.

Pour cela, j'ai ajouté une nouvelle adresse spéciale en écriture : l'adresse 0xC00000. Comme pour 0x800000 (`spif_cmd`), une écriture à cette adresse n'emprunte pas le chemin classique jusqu'à `dat[0]`. Le mot de donnée écrit est interprété comme un ordre d'activation (bit 0 à 1) ou de désactivation (bit 0 à 0) de la dérivation.

En mode désactivé, la dérivation bloque dans l'état 0 et le contrôle du `WBus` revient à la partie "classique" du Bridge. Il est important de remarquer que la désactivation de la dérivation n'altère en rien le cycle en cours ; le bus n'est libéré qu'au prochain début de cycle, c'est à dire au prochain passage à l'état 0 de la dérivation.

En plus de l'ajout de l'adresse spéciale en écriture 0xC00000, l'adresse spéciale en lecture 0xC00000 correspondant au registre de statu `monitor[4]` a été modifiée. Les bits libres de ce mots de statu reflètent maintenant l'état de la dérivation.

Le CPU doit donc, avant chaque écriture ou lecture dans la carte TGA, désactiver la dérivation, puis attendre que la dérivation revienne dans l'état 0 en lisant `monitor[4]`. A ce moment, on est sûr que le bus est libre et que le Bridge fonctionne exactement comme si la dérivation n'existait pas. Quand le CPU a terminé ses accès à la carte TGA, il lui suffit de réactiver la dérivation.

Une interruption de la dérivation par le CPU se traduit par un certain nombre de pixels émis par le jeu de la vie non mis à jour dans la mémoire écran. Comme le compteur d'adresse de la dérivation continue de tourner même quand la dérivation est désactivée, la dérivation reprend l'écriture des pixels au bon endroit dans la mémoire écran.

2.3 Routage

Une des étapes délicates de la conception d'un circuit FPGA est le routage. En particulier, le Bridge modifié doit absolument tourner à 33 MHz pour être en phase avec le bus PCI.

Sans informations spécifiques de placement, les outils de conceptions de circuit de Xilinx ne permettent d'atteindre qu'un chemin critique de 50 ns, alors que celui-ci doit théoriquement être de 30 ns.

Grâce à des informations de placement adaptées, je suis descendu à 34 ns, ce qui est suffisant dans des conditions classiques d'utilisation.

Il est possible d'introduire des barrettes de registres de façon à couper les routes longues. Malheureusement, cela introduit des *latences* qui sont gênantes lorsqu'il s'agit de répondre à un changement d'état du bus en un nombre de cycles fixé.

Je n'ai donc introduit des registres que dans les circuits de statu où la latence n'est pas gênante.

3 Initialisation de la carte TGA

Cette initialisation est un processus assez compliqué et qui n'est nécessaire qu'une seule fois. C'est pourquoi j'ai décidé de ne pas l'inclure dans le

circuit programmable. Cette opération est donc effectuée par la station de travail à travers le Bridge.

3.1 Fonctionnement de l'initialisation

L'initialisation de la carte consiste à placer des valeurs dans certains registres particuliers :

- registres de balayage horizontal et vertical,
- pixel-clock (vitesse d'émision des pixels vers le moniteur),
- palette des couleurs,
- mode d'opération de la carte (8 bits par pixel),
- mode d'écriture des pixels dans la carte (copie simple, d'autres modes comprenant des opérations logiques sont aussi possibles),
- masque d'écriture des pixels (pas de masque).

Les registres de balayage, ainsi que la pixel-clock ont posé le plus de problèmes. Les valeurs a placer dans ces registres dépendent de la résolution de l'écran et de la fréquence de rafraîchissement de l'image. Seuls quelques lots de valeurs sont acceptés. Ils correspondent à des modes graphiques particuliers, tels 1024x768 pixels en 76 images/seconde.

3.2 Programmation des registres de balayage et de pixel-clock

Bien que la signification de chacun de ces registres soit explicitée dans la documentation du chip de la carte TGA, on n'y trouve aucune information sur les valeurs à y placer pour obtenir un mode d'écran viable. En effet, ces valeurs dépendent du moniteur et non de la carte TGA.

Malheureusement, ce genre d'informations n'est pas non plus dans la documentation des moniteurs car la plus part des modes d'écran supportés par les moniteurs courants sont "standards".

Pour trouver ces valeurs "magiques", j'ai lu les sources du driver de carte TGA de XFree, le célèbre serveur X OpenSource. Le driver tout seul ne fait guère que quelques centaines de lignes et j'ai n'ai pas eu beaucoup de mal à isoler la partie d'initialisation de la carte TGA. Il m'a ensuite suffit de fournir à cette procédure d'initialisation les valeurs correspondant à un mode graphique standard dans le format propre à XFree et qui sont facilement disponibles dans le fichier de configuration XF86Config.

En fait, cela n'a pas été aussi simple à cause d'un bug dans le driver de XFree ainsi qu'un facteur multiplicatif non documenté dans les valeurs de la pixel-clock. J'ai trouvé ce dernier facteur grâce à l'utilisation d'un oscilloscope pour comparer le signal de synchronisation de la carte TGA avec celui d'une carte graphique déjà bien configurée par le système d'exploitation.

4 Conclusion

L'intérêt de ce projet est double : réaliser un circuit logique à l'aide de la bibliothèque PamDC et des outils Xilinix, mais aussi comprendre le fonctionnement de l'interface PCI de Pamette.

Réaliser un circuit à partir de rien n'est pas aisé car le fonctionnement des circuits reprogrammables ainsi que des outils de conception (PamDC et les outils Xilinx) est complexe. Mais il est encore plus difficile de comprendre et modifier un circuit déjà existant lorsqu'on ne sait rien de son fonctionnement initial, ce qui est mon cas avec le programme Bridge. J'ai, de même, passé beaucoup de temps à chercher des informations sur la programmation de la carte TGA.

Il n'est pas envisageable en trois mois de comprendre parfaitement le fonctionnement de PamDC ainsi que toutes les subtilités des circuits Xilinx 4020e utilisés. Heureusement, Mark Shand m'a beaucoup aidé lors de ce projet en m'expliquant certaines particularités de la Pamette exploitées dans la programmation du Bridge ainsi qu'en me guidant dans les problèmes de routage. Je le remercie également pour sa patience lors des essais laborieux d'initialisation de la carte TGA.

Je pense maintenant que ce projet était trop ambitieux pour le temps imparti. Il ne faut pas négliger le temps consacré à la résolution des problèmes techniques, surtout quand on travaille à partir d'un travail existant déjà assez complexe et qu'il faut adapter.

Ce projet a été néanmoins très profitable car il m'a mis en contact avec la réalité de la réalisation de circuits programmables.