ÉCOLE DE TECHNOLOGIE SUPÉRIEURE
UNIVERSITÉ DU QUÉBEC

MÉMOIRE PRÉSENTÉ À
L'ÉCOLE DE TECHNOLOGIE SUPÉRIEURE

COMME EXIGENCE PARTIELLE
À L'OBTENTION DE LA
MAITRISE EN GÉNIE ÉLECTRIQUE
M. Ing.

PAR
PELLETIER, Sébastien

IMPLÉMENTATION EN VHDL/FPGA D'AFFICHEUR VIDÉO NUMÉRIQUE (AVN) POUR DES APPLICATIONS AÉROSPATIALES

MONTRÉAL, LE 16 OCTOBRE 2008

PRÉSENTATION DU JURY

CE MÉMOIRE A ÉTÉ ÉVALUÉ

PAR UN JURY COMPOSÉ DE

M. Vahé Nerguizian, directeur de mémoire
Génie électrique à l'École de technologie supérieure

M. Jean-François Boland, président du jury
Département de Génie Électrique à l'École de technologie supérieure

M. Robert Sabourin, membre du jury
Département de Génie de la production automatisée à l'École de technologie supérieure

M. Vartivar Aklian, examinateur externe
CMC ÉLECTRONIQUE

IL A FAIT L'OBJET D'UNE SOUTENANCE DEVANT JURY

LE 2 OCTOBRE 2008

À L'ÉCOLE DE TECHNOLOGIE SUPÉRIEURE

IMPLÉMENTATION EN VHDL/FPGA D'AFFICHEUR VIDÉO NUMÉRIQUE (AVN) POUR DES APPLICATIONS AÉROSPATIALES

PELLETIER, Sébastien

RÉSUMÉ

L'objectif de ce projet est de développer un contrôleur vidéo en langage VHDL afin de remplacer la composante spécialisée présentement utilisée chez CMC Électronique. Une recherche approfondie des tendances et de ce qui se fait actuellement dans le domaine des contrôleurs vidéo est effectuée afin de définir les spécifications du système. Les techniques d'entreposage et d'affichage des images sont expliquées afin de mener ce projet à terme. Le nouveau contrôleur est développé sur une plateforme électronique possédant un FPGA, un port VGA et de la mémoire pour emmagasiner les données. Il est programmable et prend peu d'espace dans un FPGA, ce qui lui permet de s'insérer dans n'importe quelle nouvelle technologie de masse à faible coût. Il s'adapte rapidement à toutes les résolutions d'affichage puisqu'il est modulaire et configurable. À court terme, ce projet permettra un contrôle amélioré des spécifications et des normes de qualité liées aux contraintes de l'avionique.

Mots clés : VGA, vidéo, VHDL, PROM, SP305, Wishbone

IMPLÉMENTATION EN VHDL/FPGA D'AFFICHEUR VIDÉO NUMÉRIQUE (AVN) POUR DES APPLICATIONS AÉROSPATIALES

Pelletier, Sébastien

ABSTRACT

The Purpose of this project is to develop a digital display controller with the VHDL language to replace the specialized device presently used at CMC Electronics. To define the system specifications, an extensive research is performed on tendencies and on what is nowadays made in the field of the video display controllers. Pictures storage and display techniques are explained to lead this project to completion. The new controller is developped on an electronic platform having FPGA, VGA port and some RAM memory data storage. It is programmable and takes little space in FPGA, what allows it to be inserted in whatever new mass technology at low cost. Because it is modular and configurable, it can be quickly adapted to all possible display resolutions. In the short term, this project will allow a better control on avionic specifications and quality standards constraints.

Keywords: VGA, video, VHDL, PROM, SP305, Wishbone

REMERCIEMENTS

Premièrement, je voudrais souligner le soutient, les encouragements ainsi que la patience dont a fait preuve Sophie Bouchard, ma conjointe, durant toute la durée du projet.

Je tiens à remercier Vartivar Aklian pour toute l'aide et le support qu'il a apporté et aussi pour la patience dont il a fait preuve.

Je voudrais remercier Vahé Nerguizian pour son encadrement et aussi pour la confiance qu'il a témoignée pendant ces années de recherche.

Un merci spécial à Rigoberto Avelar pour avoir pu rapidement faire les modifications nécessaires sur la plateforme de développement afin que le projet puisse continuer sans délais.

Merci aussi à CMC Électronique et à l'École de technologie supérieure qui ont fourni le budget et l'infrastructure sans quoi ce projet n'aurait jamais été possible.

Merci à Frédéric Arteau pour son ouverture et sa patience.

TABLE DES MATIÈRES

Page

LISTE DES TABLEAUX

LISTE DES FIGURES

LISTE DES ABRÉVIATIONS, SIGLES ET ACRONYMES

AHB	Advanced High-Performance Bus
AMBA	Advanced Microcontroller Bus Architecture
APB	Advanced Peripheral Bus
ASB	Advanced System Bus
ASIC	Application-Specific Integrated Circuit
AVN	Afficheur Vidéo Numérique
BSD	Berkeley Software Distribution
CAN	Controller Area Network
Codec	Codeur/décodeur
CPLD	Complex Programmable logic Device
CRT	Cathode Ray Tube
DCM	Digital Clock Manager
DDR	Double Data Rate
DDWG	Digital Diaplay Working Group
DEL	Diode électroluminescente
DVI	Digital Visual Interface

EEPROM	Electrical Erasable Programmable Read Only Memory
FFC/FPC	Flexible Flat Cable
FPGA	Field Programmable Gate Array
JTAG	Joint Test Action Group
LCD	Liquid Crystal Display
LVDS/RSDS	Low Voltage Differential Signal/ Reduced Swing Differential Signal
OPB	On-Chip Peripheral Bus
PCI	Peripheral Component Interconnect
PDA	Personnal digital Assistant
PDF	Portable Document Format
PERL	Practical Extraction and Report Language
PGP	Pretty Good Privacy
PLA	Programmable Logic Array
PLB	Processor Local Bus
PROM	Programmable Read Only Memory
RAM	Random Access Memory
RGB	Red, Green, Blue

SATA	Serial ATA
SDRAM	Synchronous Dynamic Random Access Memory
SERDES	SERializer/DESerializer
SMA	SubMiniature version A
SoC	System On Chip
SRAM	Static Random Access Memory
UART	Universal Asynchronous Receiver-Transmitter
USB	Universal Serial Bus
UXGA	Ultra eXtended Graphics Array (1600 x 1200)
VGA	Video Graphics Array
VHDL	VHSIC (Very High Speed Integrated Circuit) Hardware Description Language
VIODC	Video Input Output Daughter Card
XAPP	Xilinx Application Notes
XGA	eXtended Graphics Array (1024 x 768)
ZBT	Zero Bus Turnaround

INTRODUCTION

Depuis plusieurs années déjà, la place des fonctions vidéo pour interfacer les équipements de vol en remplacement des simples cadrans et voyants lumineux n'est plus à discuter. En effet, un système d'affichage vidéo permet beaucoup plus de flexibilité aux instruments d'avioniques car il est facilement possible d'en modifier l'interface sans en changer le fonctionnement ou vice-versa. Il est aussi possible d'intégrer plusieurs instruments en un seul par l'ajout de menus et autres fonctions qui permettent de passer rapidement d'un instrument à l'autre. Actuellement, la quasi-totalité des produits offerts par CMC Électronique utilise un Afficheur Vidéo Numérique (AVN) pour faire l'interface entre l'usager (le pilote) et l'instrument d'avionique.

Jusqu'à maintenant, les technologies utilisées par la compagnie pour le contrôle de l'affichage proviennent de compagnies extérieures, ce qui diminue beaucoup la flexibilité d'intégration de ces technologies à l'intérieur des systèmes de CMC Électronique. Nous verrons à la section 1.1 les raisons qui poussent la compagnie à développer sa propre expertise en matière d'AVN.

Ce projet a pour objectif la conception en langage VHDL (« Very High Speed Integrated Circuit Hardware Description Language ») et l'implémentation dans une composante FPGA (Field Programmable Gate Array) d'un AVN pour les applications aérospatiales. Il vise à développer une architecture configurable et adaptable aux différents besoins de ces applications afin de remplacer le contrôleur d'écrans qu'utilise présentement CMC Électronique dans leurs systèmes d'avionique. Le projet final est un contrôleur d'écran à plusieurs résolutions et fréquences de rafraîchissement. Il est conçu de manière à répondre aux besoins actuels et futurs de l'entreprise CMC Électronique en matière de contrôle d'écran et il reflète l'avancement technologique des dernières années. Ce module est évolutif car il est possible de l'adapter aux différentes technologies utilisées au sein de l'entreprise.

Ce document est divisé en plusieurs chapitres qui présentent les différents aspects ayant rapport à l'élaboration de ce projet. Voici donc un plan de ce qu'on retrouve dans ce

document et qui vous permet de suivre le cheminement logique suivit lors de la rédaction de ce mémoire.

Le premier chapitre a pour buts de mettre en contexte et de structurer le travail à accomplir. Le deuxième chapitre met en lumière les possibilités et les limitations d'un tel projet pour ensuite définir les spécifications fonctionnelles envisageables lors de sa réalisation. Le chapitre 3 utilise ce qui a été trouvé au chapitre 2 en relations avec les besoins de CMC Électronique pour déterminer la plateforme de développement à utiliser afin de mettre au point le contrôleur vidéo. Suite à ces deux derniers chapitres vient le quatrième qui sert à donner en détail les spécifications retenues qui serviront de guide tout au long du développement du système d'affichage numérique.

Le chapitre 5 explique tous les principes nécessaires à la réalisation du projet. Le fonctionnement de la mémoire PROM (« Programmable Read Only Memory ») et de l'affichage d'une image y sont expliqués. Il en est de même pour l'organisation des fichiers *.mcs et *.bmp dont on élabore sur la manière d'utiliser le langage PERL pour extraire des données d'images et de les insérer dans le fichier de configuration du FPGA.

Le chapitre 6 se veut le plus complexe et volumineux puisque c'est lui qui inclut l'explication et la simulation des différentes parties du système. Il est suivit du chapitre 7 qui permet la mise en route de tout le projet, étape par étape, jusqu'à l'affichage d'une image sur un écran LCD. On termine avec le chapitre 8 qui permet de faire une synthèse de tout le travail accompli et qui permet aussi d'apporter des solutions aux différents problèmes restants à résoudre pour la continuité du projet.

Le code VHDL du contrôleur et du bootloader en entier seront placés à la première annexe, suivit par les bancs d'essais, des scripts de compilations, des scripts PERL et, pour terminer, avec le fichier de contraintes top.ucf. Ce mémoire est écrit de manière à faciliter la consultation ponctuelle tout en permettant d'être lu comme un guide de l'utilisateur.

CHAPITRE 1

PROBLÉMATIQUE ET PRÉPARATION DU PLAN DE DÉVELOPPEMENT

Ce chapitre a pour but de décrire la problématique rencontrée chez CMC Électronique en matière d'AVN et par le fait même, il permet d'expliquer les motivations de la compagnie pour développer un système de contrôle vidéo propriétaire. Aussi, ce chapitre définit un plan de développement qui répond à cette problématique et apporte une solution d'avenir dans ce domaine.

La section qui suit montre en détail chaque aspect de la problématique et la section 1.2 décrit la méthode de travail qui est suivie pour l'élaboration du projet.

1.1 Problématique rencontrée

Depuis plusieurs années, CMC Électronique utilise des contrôleurs vidéo numériques provenant de fabricants extérieurs. Ces composantes sont très performantes, mais sont aussi très peu adaptées et ne rencontrent pas les standards minimums pour les besoins de l'entreprise. Il en résulte donc des dépenses importantes pour ces composantes, des dépenses qui nous amènent à remettre en question leur utilisation et, par conséquent, de les remplacer par des composantes propriétaires à CMC Électronique. Plusieurs aspects seront discutés plus bas pour justifier le développement d'une solution pour un AVN par CMC Électronique.

1.1.1 Aspect technique

L'affichage utilisé en avionique ne requière pas de calculs vidéo sophistiqués comme on peut en voir dans les systèmes d'affichages pour des images 3D par exemple. De plus, l'affichage requis à l'intérieur d'un cockpit demande un débit d'informations relativement lent donc, nul besoin d'avoir une bande passante très élevée comme pour des ordinateurs de maisons, d'autant plus que la résolution d'affichage n'est pas aussi grande que celle qu'on possède à la maison. Les écrans pour l'instrumentation sont plus petits que le plus petit des écrans que

vous pouvez trouver en ce moment sur le marché des écrans résidentiels. Nous sommes donc en présence d'un système simple au niveau technique. La complexité des systèmes utilisés en avionique vient surtout du respect des critères de fiabilité et de stabilité dans des conditions extrêmes de température et de pression atmosphérique dont on parlera plus loin.

Le défi de CMC est jusqu'à maintenant de trouver des composantes qui répondent à ces deux critères de simplicité et de fiabilité tout en maintenant un coût minimal. Les composantes offertes sur le marché sont de plus en plus complexes et coûtent de plus en plus chères, alors que les besoins en avionique n'évoluent pas aussi rapidement et ne sont pas de cette complexité dans la majorité des cas.

Le développement d'une solution propriétaire permet non seulement de développer une expertise technique dans le domaine des AVN, mais aussi une meilleure adaptation aux différents besoins de l'entreprise. Le rythme de l'évolution technique du contrôleur vidéo propriétaire étant plus lent que celui du marché actuel, il est plus facile de développer une technologie dont les standards de fonctionnalité technique sont largement documentés et éprouvés par l'utilisation résidentielle.

1.1.2 Aspect sécurité et fiabilité

Comme mentionné plus haut, le plus grand défi pour développer des produits d'avionique est d'assurer leur fiabilité et leur stabilité de fonctionnement dans des conditions de température et de pression atmosphérique extrêmes. Lorsque CMC Électronique utilise des composantes d'une tierce compagnie, il est nécessaire de certifier la fiabilité et la sécurité de ces composantes, alors qu'elles n'ont pas été nécessairement conçues à cet effet. Ceci demande énormément de temps et de dépenses puisque ces technologies n'ont pas une longue durée de vie sur le marché. Il devient plus facile pour CMC Électronique de répondre à ces hauts standards de qualité en prenant en charge le développement du contrôleur vidéo numérique, puisque ce dernier est développé dans cette optique et avec les critères de fiabilité définis chez CMC Électronique. De plus, le changement de composantes FPGA n'affecte que les aspects physiques du design, ce qui permet de faire des changements technologiques ainsi

que leur certification de fiabilité/stabilité plus rapidement que dans le cas d'un changement de technologie non propriétaire.

1.1.3 Aspect évolutif

En utilisant des composantes de fabrication extérieure, il est difficile de contrôler leur évolution. Pour chaque nouvelle technologie ou nouvelle composante d'affichage, il est souvent requis de recommencer tout le développement de la plateforme du circuit imprimé ainsi que l'adaptation de la composante avec les autres déjà présentes sur la carte. Ceci est très coûteux en temps et en argent.

En développant un contrôleur vidéo chez CMC Électronique dans un langage de spécification physique, comme le VHDL, il est désormais possible de faire évoluer rapidement et sans trop de dépenses additionnelles les applications qui demandent de la vidéo puisque l'AVN fera maintenant partie intégrante de la section numérique du système d'avionique. Il est même possible de garder un design inchangé en le faisant migrer vers une nouvelle technologie de composante programmable (FPGA), ce qui permet une liberté accrue dans le choix de la technologie.

De plus, il est maintenant possible d'ajouter des fonctionnalités qui répondent aux besoins de chaque système sans pour autant nécessiter de modification physique dans le reste du produit.

1.1.4 Aspect énergétique

Avec les composantes vidéo utilisées présentement chez CMC, il n'est pas possible d'intégrer leurs fonctions en une seule composante puisque chacune d'elles est indépendante et a une fonctionnalité fixe. Ainsi, le nombre élevé de composantes sur un circuit augmente la consommation d'énergie du système.

En développant un système d'affichage numérique en un langage comme le VHDL, son intégration avec d'autres fonctions en une seule composante devient possible. Ceci nous

laisse donc envisager une diminution significative de la puissance électrique consommée, surtout s'il est possible de gérer efficacement l'activation et la mise en veilleuse de certaines parties du système lorsqu'elles ne sont pas sollicitées.

1.1.5 Aspect économique

Avec l'intégration de système en une seule composante, la complexité du développement de circuit imprimé diminue. L'unification des systèmes permet aussi de réduire le nombre de composantes sur le circuit, ce qui réduit les coûts de productions.

Aussi, le marché des composantes programmables de types FPGA ont depuis quelques années atteint le marché de masse. Ceci leur permet d'augmenter leur facteur d'échelle, réduisant ainsi les coûts par rapport à la production de composantes à fonctionnalité fixe comme celles utilisées présentement chez CMC Électronique.

Pour ces raisons, il devient plus avantageux d'utiliser un contrôleur vidéo développé en VHDL chez CMC Électronique plutôt qu'un contrôleur video à fonctionnalité fixe de provenance extérieure.

1.1.6 Synthèse de la problématique

En résumé, l'utilisation par CMC Électronique d'une technologie d'affichage vidéo provenant d'un fabriquant extérieur occasionne des inconvénients qui peuvent être énormément atténués ou même éliminés par le développement de cette technologie d'affichage par CMC Électronique. C'est pourquoi ce projet est mis de l'avant. Cependant, pour le mener à bien, il est nécessaire de structurer le travail à accomplir en se donnant un plan de développement. Les sections suivantes de ce chapitre expliquent ce plan.

1.2 Explication du plan de développement utilisé

Voici le plan de développement suivi pour faire avancer le travail. Avant de se lancer dans des explications détaillées, il est préférable de visualiser schématiquement la logistique de chaque tâche sur la Figure 1.1 et la Figure 1.2 qui suivent :

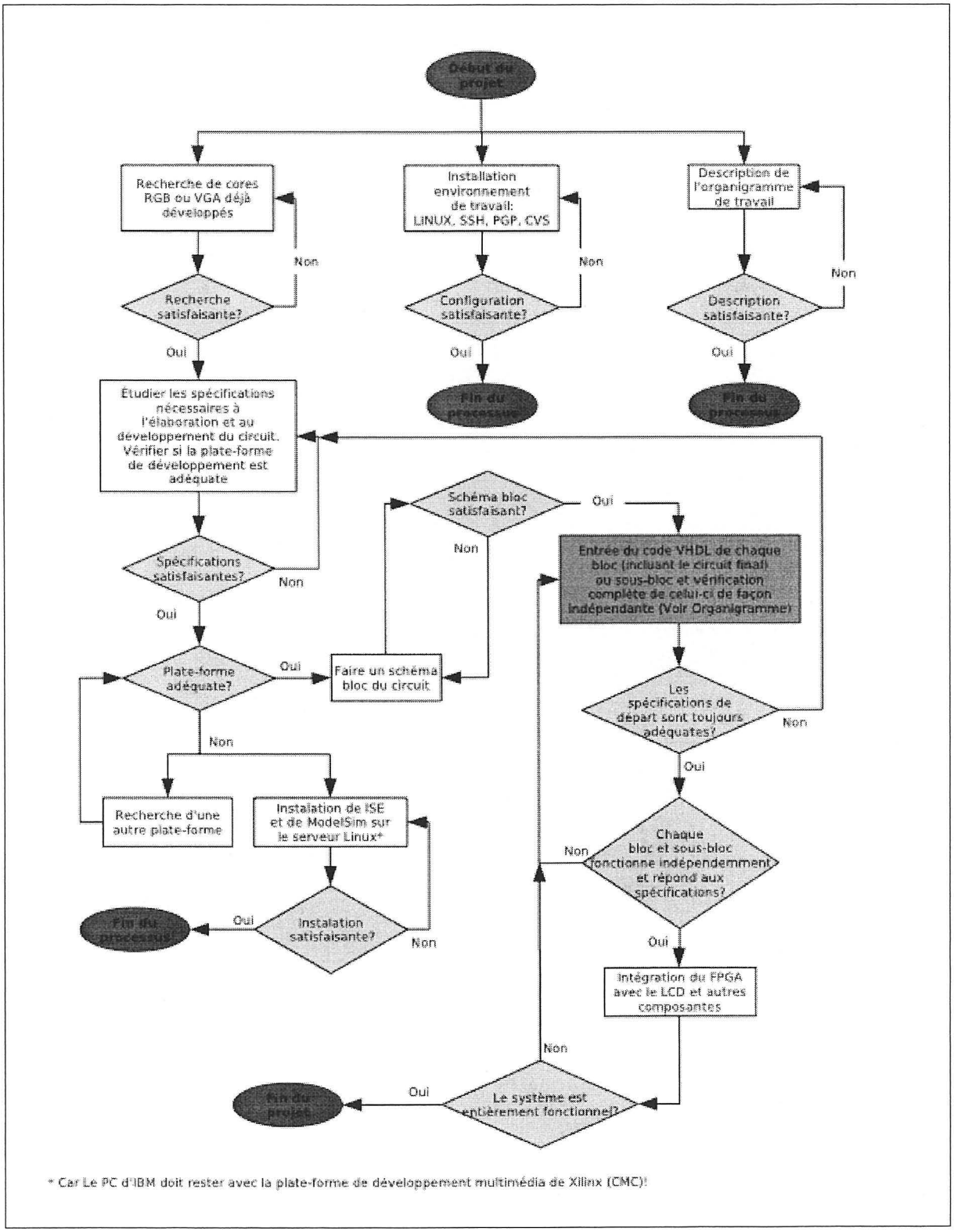

Figure 1.1 Organigramme représentant la méthodologie globale du travail à effectuer.

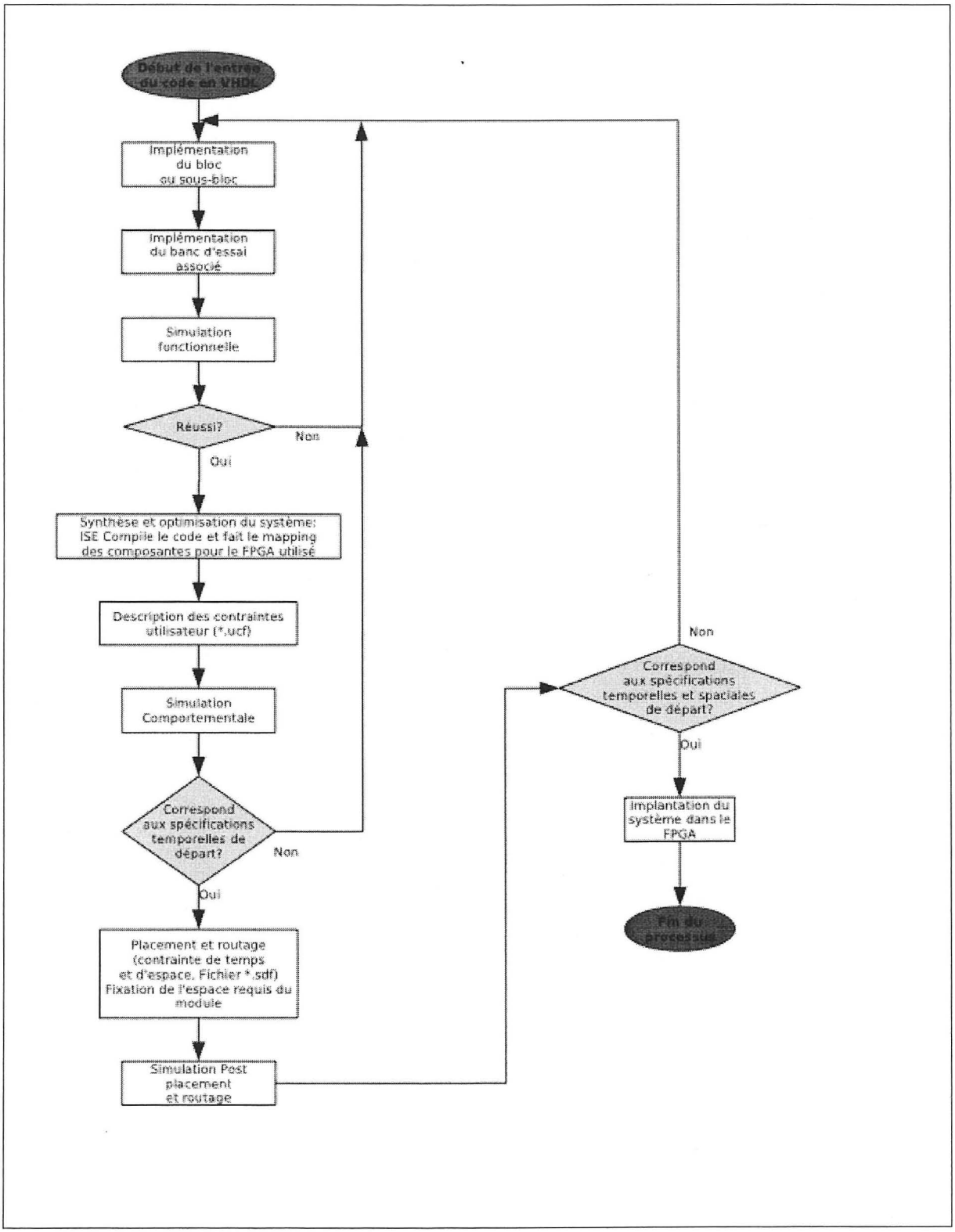

Figure 1.2 Organigramme du processus d'entrée du code VHDL jusqu'à l'implantation.

Les sections suivantes décrivent plus en détail chacune des étapes exposées dans la Figure 1.1 et la Figure 1.2.

1.2.1 Recherche de cores RGB ou VGA déjà développés

Pour faciliter le démarrage du projet, une recherche détaillée des contrôleurs vidéo RGB (Red, Green, Blue) et VGA (Video Graphics Array) existants est effectuée afin de se donner une idée précise de ce qui est atteignable comme spécifications de départ et aussi pour déterminer rapidement une topologie qui répond aux besoins de CMC Électronique. Cette étape est entièrement décrite au CHAPITRE 2.

1.2.2 Installation de l'environnement de travail et liste des logiciels utilisés

Pour des fins de sécurité informatique, il a été considéré d'installer les outils de travail sur un environnement Linux. Cependant, pour des raisons de compatibilité avec certains pilotes du câble JTAG (« Joint Test Action Group ») et aussi par manque de temps à résoudre ces problèmes, un environnement Windows XP et les logiciels suivant sont utilisés :

A. ISE 8.2i de Xilinx;
B. Modelsim 6.2;
C. Chipscope de Xilinx;
D. Microsoft Office 2003;
E. Microsoft Visio 2003;
F. Notepad ++ 4.0.2;
G. Winzip.

Aussi, pour communiquer de manière sécuritaire et confidentielle entre CMC Électronique et l'ÉTS, il est nécessaire de se doter d'un outil d'encryptions de fichiers. Pour cela, deux moyens sont utilisés : le format d'encryption « Pretty Good Privacy » (PGP) ou le format de compression de fichier *.zip avec une double compression nécessitant des mots de passe.

Pour plus de détails sur l'utilisation des outils de développement utilisés, référez-vous à un didacticiel d'ISE 8.2i, de ModelSim et/ou de VHDL.

1.2.3 Étude des spécifications nécessaires à l'élaboration et au développement du circuit et choix de la plateforme de développement

L'étude approfondie des différents contrôleurs vidéo existants (CHAPITRE 2) faisant ressortir les spécifications de départ du projet, il est possible de déterminer les besoins en matière de plateforme de développement tout en minimisant la marge d'erreur par rapport au matériel choisi. Par la suite, un magasinage intensif des plateformes de développement est effectué afin de trouver le meilleur rapport qualité/prix ainsi que le meilleur choix selon l'aspect évolutif et l'aspect documentation. Cette étape se reflète au CHAPITRE 3 qui nous explique le cheminement emprunté pour faire le choix de notre plateforme de développement. Une fois toute l'information rassemblée, un consensus est finalement possible pour déterminer la liste de toutes les spécifications (physiques et logicielles) du projet. Le CHAPITRE 4 en fait l'énumération.

1.2.4 Faire un schéma bloc du circuit

À cette étape, il est nécessaire de connaître en détail les caractéristiques finales voulues pour le projet d'afficheur vidéo numérique. Le schéma bloc évolue tout au long du développement du code VHDL, c'est pour cette raison que seule la version finale est présentée dans ce document. Les différentes parties de ce schéma bloc seront présentées dans le CHAPITRE 6. Il est à noter qu'il est impossible de dessiner le schéma bloc final détaillé en une seule figure. En attendant, voici un schéma bloc préliminaire du système tel qu'il était perçu au début du projet:

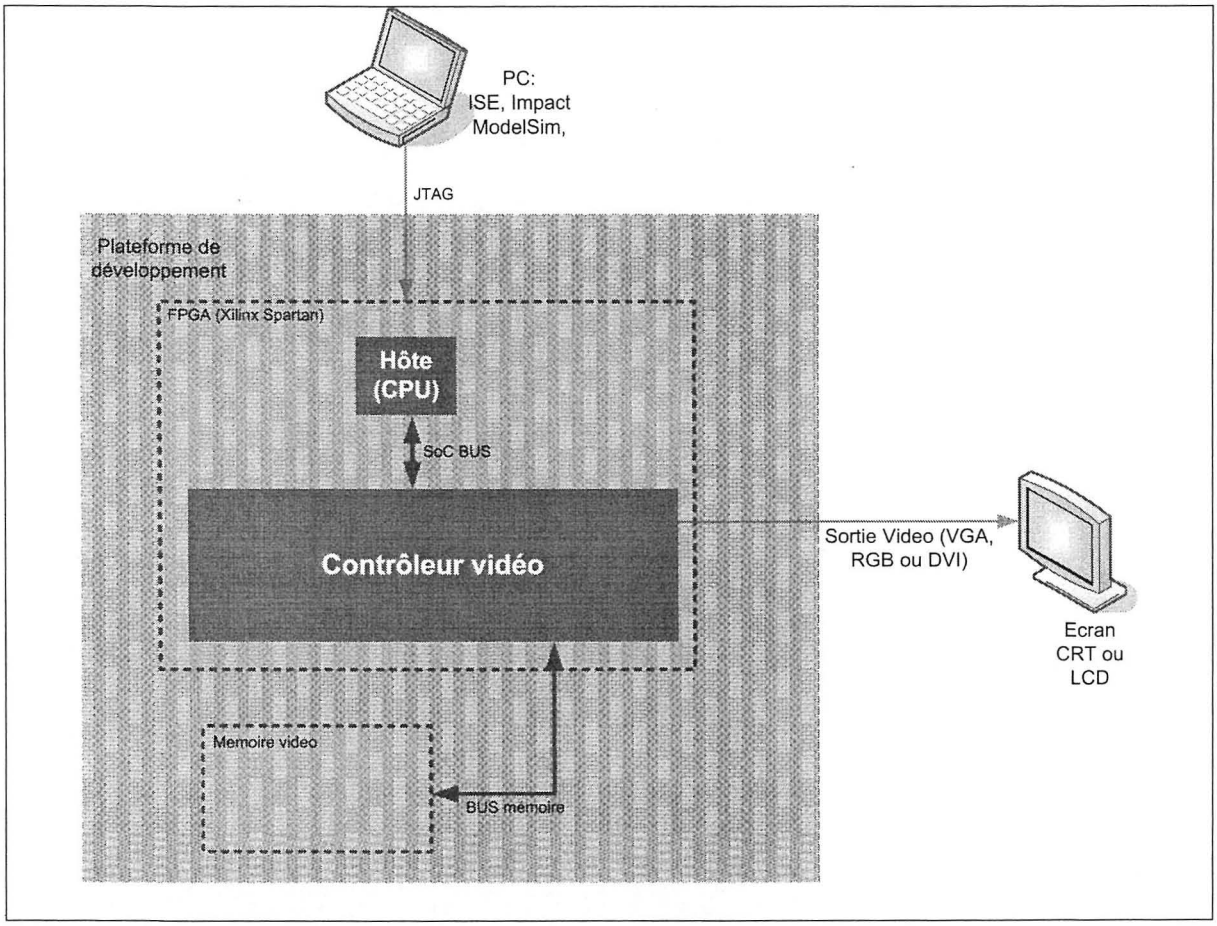

Figure 1.3 Schéma bloc préliminaire du système.

Ce dessin montre bien que le travail est de configurer et de programmer une plateforme de développement par un port JTAG afin de lui permettre d'afficher des images sur un écran. Le travail est concentré sur le FPGA qui doit comprendre un module d'affichage et un système de contrôle hôte qui permet de passer des commandes ou de charger des images dans le module d'affichage.

1.2.5 Entrée du code VHDL en fonction du schéma bloc

Le code VHDL est écrit en respectant la structure du schéma bloc du système (CHAPITRE 6) défini en fonction des spécifications fixées au départ (CHAPITRE 4) afin de permettre une compréhension rapide du code et de faciliter le déverminage du système. Le code possède

aussi le style et la structure qui correspondent aux normes internes de CMC Électronique (section 4.3).

Chaque bloc du code est simulé pour vérifier la fonctionnalité du modèle. Suite à cela, une simulation fonctionnelle globale du système est effectuée (voir section 6.6). Des modifications sont apportées tant que les modèles ne répondent pas aux spécifications de départ.

1.2.6 Synthèse et optimisation du système

Une fois la simulation fonctionnelle réussie, une première synthèse et optimisation qui permet de gérer le type des pièces programmées peut être fait. Un circuit plus performant est obtenu en le réorganisant et en optimisant les équations pour chaque bloc et sous bloc, si cela le permet. Les résultats de synthèse sont présentés au CHAPITRE 8.

1.2.7 Placement et routage

Cette étape est la dernière avant l'implantation dans le FPGA. Elle permet de faire les derniers ajustements en fonction des contraintes de temps et d'espace reliées à la fois à la technologie utilisée et aux spécifications de départ du système. Cette étape permet, en plus, de fixer l'emplacement de chaque bloc et sous bloc afin d'encadrer et de structurer le système à l'intérieur du FPGA. La simulation à partir de cette étape ressemble énormément au fonctionnement réel du circuit physique puisque le modèle tient compte des délais de propagation nominaux et des contraintes de placement et routage. Les contraintes utilisées sont montrées aux sections 7.2 et 7.3.

CHAPITRE 2

RECHERCHE BIBLIOGRAPHIQUE DE CŒUR EXISTANT ET SPÉCIFICATIONS FONCTIONNELLES PRÉLIMINAIRES DU PROJET SUR LES AVN

Ce chapitre présente la recherche effectuée afin de prendre connaissance des principes et des projets existants pouvant être utilisés comme guide de départ au projet de pilote d'écran en VHDL. Ce travail permet une définition plus aisée des spécifications fonctionnelles finales du projet. Pour cette raison, il est nécessaire de parler des avantages et des inconvénients de chaque cœur étudié.

Cette recherche est aussi le point de départ, notre guide de magasinage en quelque sorte, pour le choix de la plateforme de développement acquise afin de mettre au point ce projet. C'est pourquoi la dernière section de ce chapitre donne une synthèse des caractéristiques fonctionnelles retenues pour le projet mais ne constitue pas encore la dernière liste des spécifications désirée pour notre AVN. Des précisions s'ajoutent à ces dernières suite au choix de la plateforme de développement et il est nécessaire de constituer un chapitre indépendant pour toutes les définir. Le CHAPITRE 4 a donc été écrit à cette fin.

2.1 Spécifications de départ minimales recherchées pour le projet

Ce contrôleur doit minimalement répondre aux spécifications utiles tirées de celles de la composante présentement utilisée par la compagnie et aussi de celles définies par la nature des technologies utilisées. Le cœur du système :

a. doit appartenir à la compagnie CMC Électronique;
b. doit être entièrement écrit en langage VHDL;
c. doit permettre un interfaçage rapide et facile avec de la mémoire « Random Access Memory » (RAM) et/ou un processeur;
d. doit permettre une bande passante élevée (difficile de dire ce qui est atteignable avant l'étude de ce qui existe);

e. doit permettre de maintenir la compatibilité avec les systèmes déjà utilisés chez CMC Électronique (sortie RGB);
f. doit permettre d'utiliser les dernières technologies de connectivité avec les nouveaux écrans pour une possibilité accrue d'évolution du système dans le futur;
g. doit permettre une mémoire RAM suffisante pour contenir deux images de résolution 1024 x 768 à encodage de 24 bpp (bits par pixel), ce qui correspond à 4.8 Mo (Mega octets) environ;
h. doit permettre un mode d'économie d'énergie activé de manière logicielle.

2.2 Recherche sur les bus SoC

Un bus « System on Chip » (SoC) permet de connecter facilement et rapidement deux composantes ou plus par un lien de partage d'informations ou de données. Chacun des bus possède son propre protocole.

Pour des raisons pratiques, il est favorable de commencer par définir notre choix en matière de bus interne entre les blocs du système ordiné qui sont en cause. En effet, certains contrôleurs d'écran sur le marché utilisent déjà de ces bus. Ceci peut donc influencer nos choix technologiques.

Ce que nous recherchons en cette matière est un bus facile à adapter et qui permet d'intégrer notre contrôleur d'affichage dans n'importe quel système où un écran est requis. L'autre facette du bus doit être sa bande passante adaptable pour des systèmes très rapides.

2.2.1 Bus Sériel SERDES :

Un bus SERDES (serializer/deserializer) permet de transformer de l'information de nature parallèle en information de nature sérielle, ce qui permet de transférer les données à haut débit avec un nombre réduit de connexions. Il est possible par la suite de recevoir l'information de manière synchrone et de la remettre en parallèle.

Il est convenu de ne pas utiliser de bus SERDES puisque cela entraîne une complexification du « design » matériel afin de répondre aux contraintes de haute vitesse d'horloge de ce genre de bus pour le transfert de données d'images (plus de 1 GHz, ce qui est beaucoup trop élevé). Pour le moment, nous nous contentons de fabriquer un système facilement adaptable et éprouvé. Il sera possible, éventuellement, d'adapter un bus SERDES à partir du bus SoC parallèle choisi plus loin, si cela s'avère nécessaire.

2.2.2 Bus parallèle CoreConnect (IBM)

C'est le bus d'interconnections SoC le plus documenté en apparence ([30] IBM Corporation, 1999, [31] IBM Corporation, 2005 et [32] IBM Corporation, 2008). Plusieurs modèles ont été développés afin de répondre aux divers besoins liés aux systèmes ordinés. Il existe 2 niveaux différents : le « Processor Local Bus » (PLB) pour les systèmes de haute performance comme dans le cas d'une communication entre un processeur et une mémoire, ainsi que le « On-Chip Peripheral Bus » (OPB) pour les systèmes de moindre performance. Le niveau PLB possède une spécification dédiée pour les bits de contrôle organisés en guirlande (« daisy chain ») permettant de libérer de la bande passante pour le reste du PLB, mais rendant la conception de ce type de bus un peu plus complexe. Le bus d'IBM permet une extension jusqu'à 128 bits de données. Il a été développé pour interfacer un processeur avec d'autres périphériques, c'est-à-dire pour fabriquer efficacement un système ordiné complexe et stable dans une seule puce. Rien ne spécifie sur le site si son utilisation est gratuite ou non.

Voici un exemple de schéma bloc :

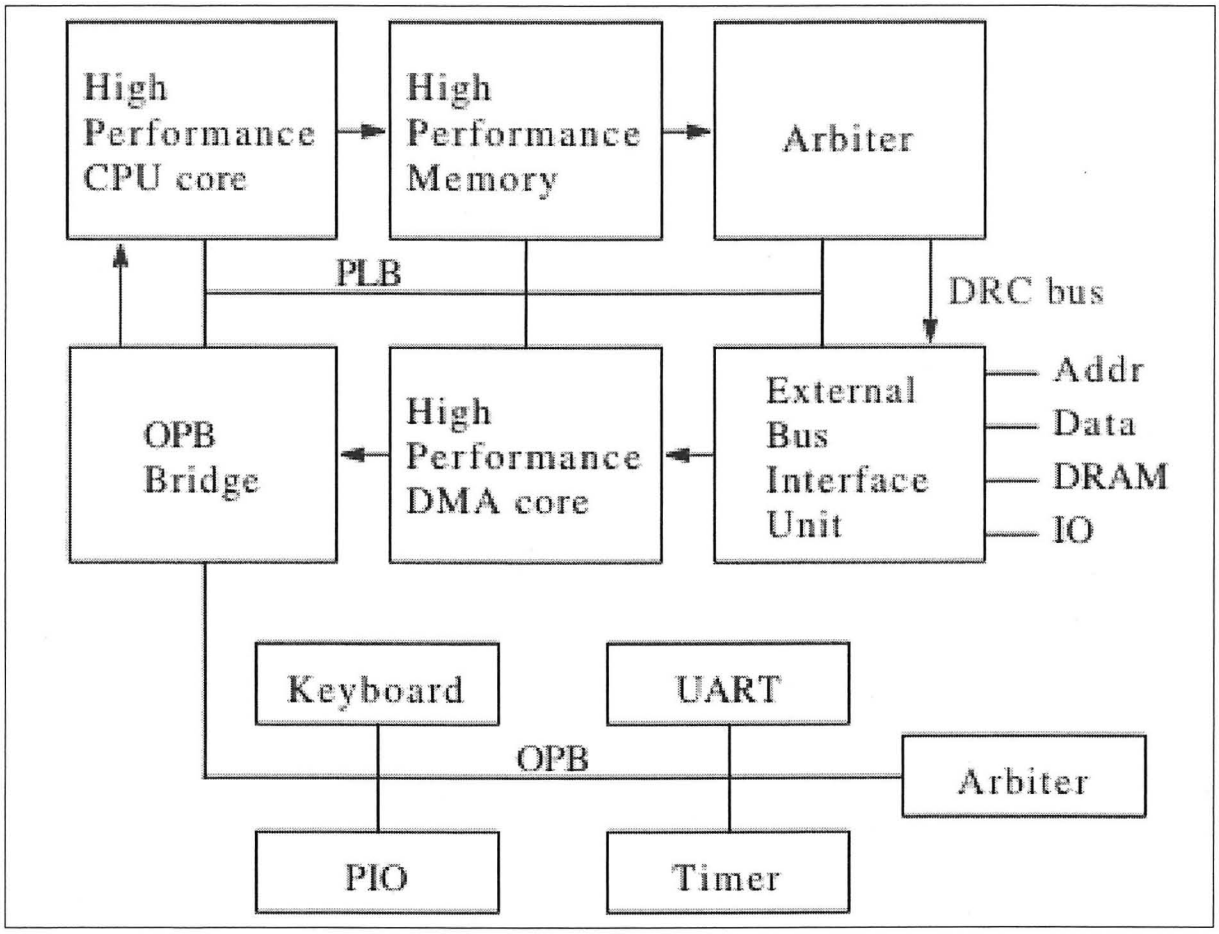

Figure 2.1 Schéma bloc du bus CoreConnect.

(Tiré de [47] Usselmann, 2001)

Source : cette figure est tirée du document de M. Rudolf Usselmann, *OpenCores SoC Bus Review*, p. 3, sur la page OPENCORES.ORG

2.2.3 Bus parallèle AMBA (ARM) :

L'AMBA (« Advanced Microcontroller Bus Architecture ») a été développé avec le même objectif que le CoreConnect d'IBM et lui ressemble un peu, mais ne possède pas de partie dédiée pour les bits de contrôle dans le modèle haute performance, ce qui le simplifie légèrement. Par contre, ce modèle se définit de deux manières différentes selon les besoins, l'AHB (« Advanced High-performance Bus ») étant le plus complet et l'ASB (« Advanced System Bus ») étant une forme simplifiée et légèrement modifiée de l'AHB. L'AMBA possède aussi une troisième spécification pour connecter des périphériques de moindre

performance, il se nomme l'APB (« Advanced Peripheral Bus »). L'utilisation du bus AMBA semble gratuite. Pour les détails, référez-vous à [2] ARM Limited, 1999 ou à [47] Usselmann, 2001.

Voici un exemple de schéma bloc :

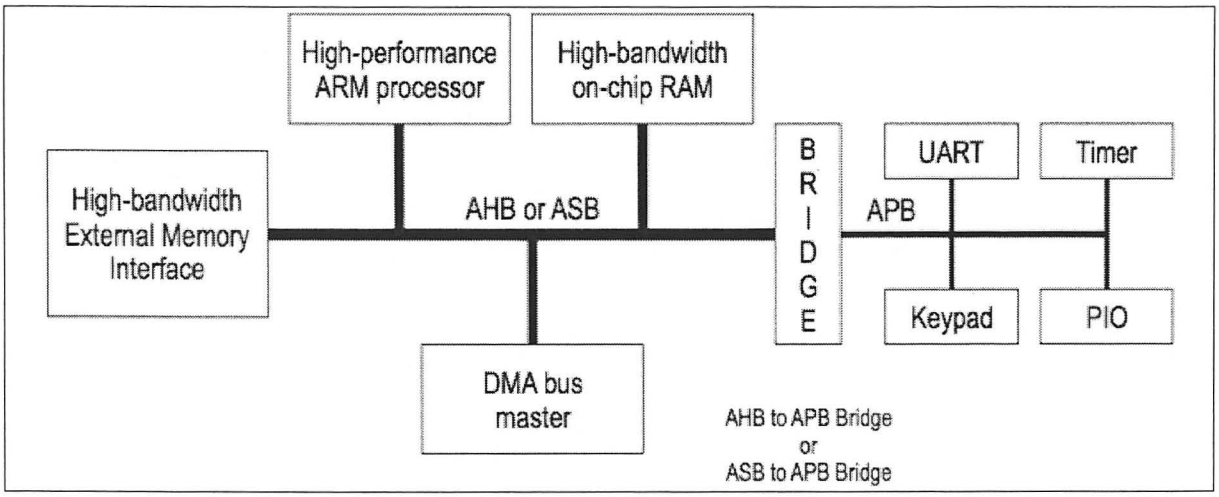

Figure 2.2 Schéma bloc du bus AMBA.

(Tiré d'ARM Limited. 1999)

Source : cette figure est tirée du document d'ARM Limited, 1999, *AMBA™ Specification*, p. 1-4, sur la page web de Gaisler Research section documentation.

2.2.4 Bus parallèle Wishbone :

Le BUS Wishbone a été développé expressément pour interconnecter n'importe quels modules ensemble, que ce soit des composantes d'un système ordiné comprenant un processeur et de la mémoire ou des composantes non standard ou des périphériques sans processeur ni mémoire. Le BUS Wishbone est tout désigné pour une implantation simple, rapide et ne prend que peu de ressources logiques ([47] Usselmann, 2001). Une seule spécification est nécessaire pour toutes les performances nécessaires (pourquoi compliquer les choses quand on peut faire cela simplement), pour ajouter des modules de performance différente dans un même système, il suffit d'organiser deux bus Wishbone de différentes performances et le tour est joué. Le bus permet de modifier, selon les besoins, la fréquence

de transmission ainsi que la largeur du bus de données et du bus d'adresse. Son utilisation est gratuite et toute la documentation est disponible gratuitement sur le site internet de OPENCORES.ORG ([23] Herveille, 2002).

Voici un exemple de schéma bloc :

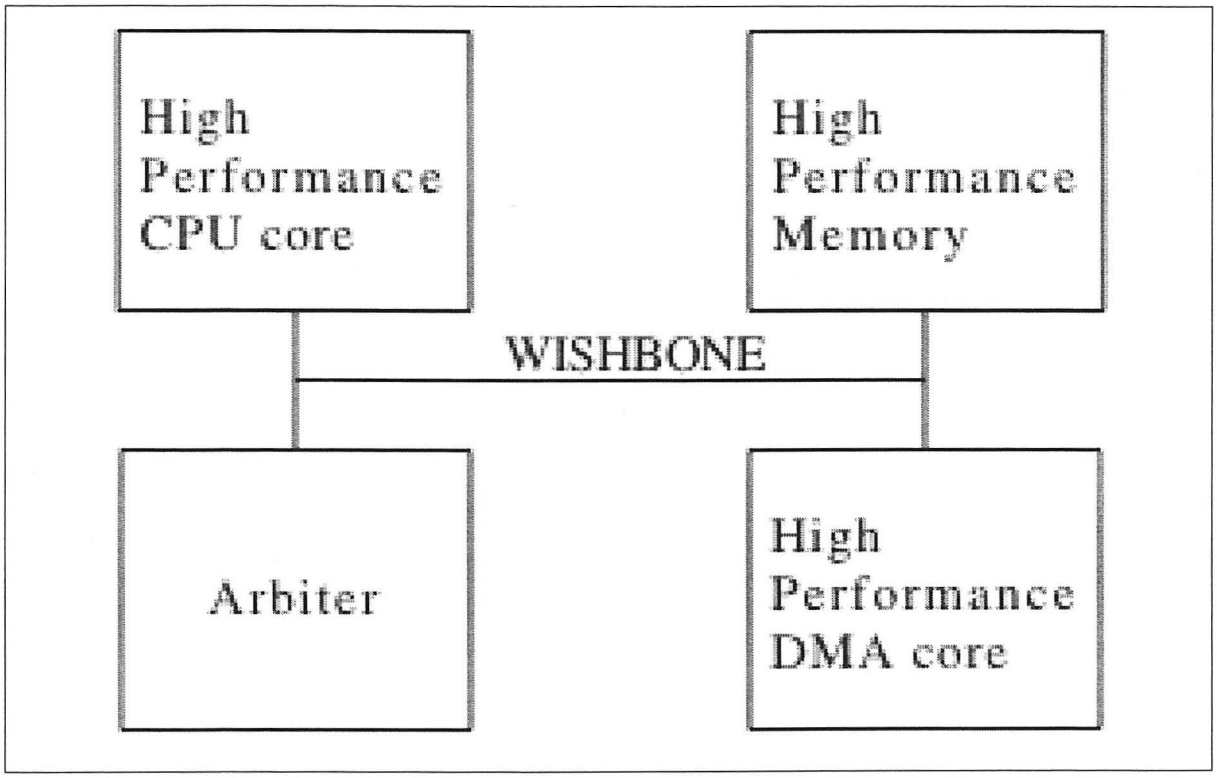

Figure 2.3 Schéma bloc du bus Wishbone.

(Tiré de [47] Usselmann, 2001)

Source : cette figure est tirée du document de M. Rudolf Usselmann, *OpenCores SoC Bus Review*, p. 9, sur la page OPENCORES.ORG

2.2.5 Notre choix de BUS

Au niveau de la performance, il est certain que tous rencontrent ce qui est demandé par notre module d'affichage. Par contre, celui que répond le mieux au critère de simplicité d'implantation et d'utilisation à cause de sa versatilité et de sa gratuité, c'est le bus Wishbone.

Les autres bus ne sont pas avantageux dans notre cas, car ils sont spécifiquement développés pour interfacer des périphériques dans un système ordiné complexe comprenant beaucoup de normes protocolaires. Il devient alors lourd de les utiliser dans des cas simples. En d'autres mots, les utilisateurs de ces bus (AMBA et CoreConnect) doivent s'adapter à leurs spécifications et non l'inverse comme le permettent un peu plus les spécifications du bus Wishbone. Pour ces raisons de simplicité, il est clair que notre choix s'arrête sur le bus Wishbone.

2.3 Recherche sur les protocoles et signaux de sortie vidéo :

Cette section a pour objectif de discuter des possibilités envisagées pour la communication entre le contrôleur vidéo et l'écran qui y est connecté. Nous étudions les différents cas pour s'assurer de prendre les bonnes décisions au départ afin d'adapter facilement le projet aux technologies déjà utilisées chez CMC et aussi pour qu'il puisse facilement évoluer vers des technologies émergentes.

2.3.1 Port RGB (ou VGA) :

Un port RGB (ou VGA) est un port analogique qui ne fait qu'envoyer les signaux nécessaires au contrôle de l'écran jumelé aux couleurs de chaque pixel. Nous devons être en mesure de fournir un port RGB (ou VGA) en sortie du système final pour garder la compatibilité avec ce qui se fait présentement chez CMC Électronique. Cependant, une autre solution se présente à nous: le port DVI (Digital Visual Interface) qui a plusieurs avantages (voir la section 2.3.2). L'utilisation du standard de connexion VGA nous permet d'utiliser facilement la majorité des écrans sur le marché.

2.3.2 Port de DVI (Digital Visual Interface) développé par le DDWG (Digital Display Working Group)

La rédaction des spécifications de la première version de cette interface a été terminée au début avril 1999. Cette interface est développée pour permettre l'échange de données vidéo

numériquement afin de conserver l'intégralité et la qualité du signal vidéo pendant son transport. Cette technologie permet d'augmenter le débit de données, d'améliorer la résolution de l'image et aussi d'accroître le nombre d'images par seconde sur les écrans. Alors que le standard VGA plafonne autour d'une résolution de 1600 x 1200, le standard DVI fait son entrée sur le marché pour permettre de continuer à augmenter la résolution et aussi la vitesse de rafraîchissement d'image. De plus, afin de garder une certaine compatibilité avec les écrans actuels, le standard DVI prévoit une partie de ses spécifications pour des signaux analogiques. Ceci permet de connecter les écrans VGA actuels à l'aide d'adaptateurs au port DVI (RGB compris).

Voici un exemple de schéma bloc du standard DVI :

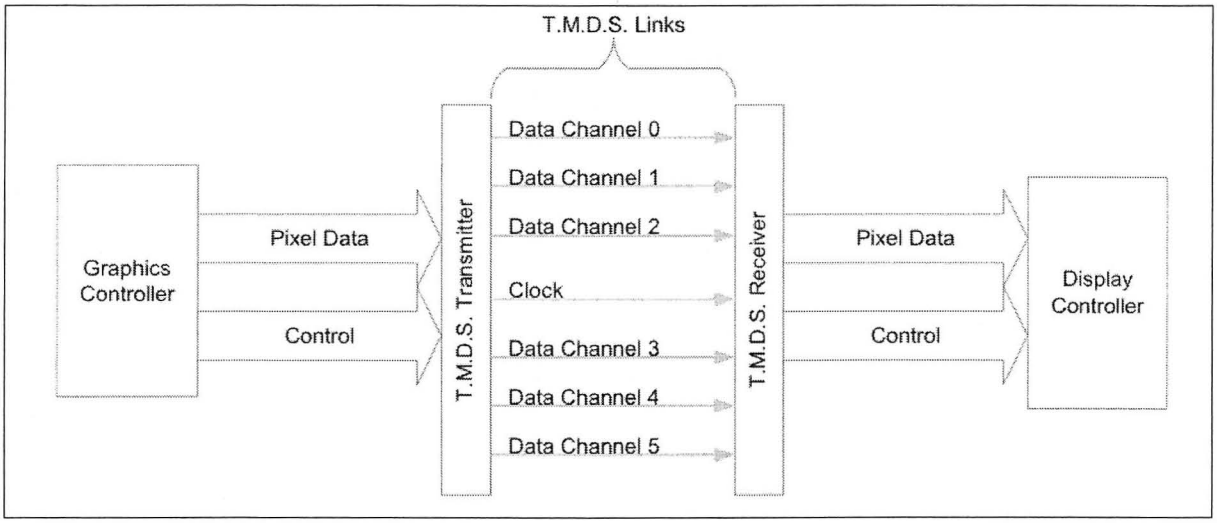

Figure 2.4 Schéma bloc du bus DVI.

(Tiré de Digital Display Working Group. 1999)

Source : cette figure est tirée du document de Digital Display Working Group, 1999, *Digital Visual Interface DVI*, p. 10, url : http://www.ddwg.org/lib/dvi_10.pdf.

Dans notre cas, le standard DVI possède beaucoup trop de bande passante pour nos besoins immédiats. Il sera toujours possible de faire migrer le contrôleur vidéo d'une technologie d'affichage analogique vers une technologie numérique, simplement en supprimant le convertisseur numérique/analogique et en le remplaçant par un module de formatage des données numériques au standard nécessaire. Dans ce sens, il est présentement possible

d'utiliser les signaux analogiques VGA à travers le standard physique du DVI puisque ce dernier a été développé afin de permettre une transition douce du VGA vers le DVI. Ceci nous laisse donc l'opportunité de commencer notre développement par une technologie VGA. De plus, étant donné que l'écran utilisé actuellement communique par un port RGB, nous n'avons pas à mettre d'effort sur le développement additionnel d'une nouvelle connectivité.

2.4 Cores (cœurs) VGA, RGB ou DVI disponibles:

Cette section fait un survol sur les solutions disponibles au moment du démarrage du projet. Nous tenterons d'identifier les caractéristiques et les fonctionnalités utiles pour les besoins de CMC Électronique. Comme vu précédemment lors de l'élaboration du plan de développement (section 1.2.3), les spécifications qui ressortent à la fin de ce chapitre ne sont pas finales, mais servent plutôt comme guide pour le choix de notre plateforme de développement au chapitre suivant. Pour en savoir plus sur les spécifications finales retenues pour le projet, référez-vous au CHAPITRE 4.

2.4.1 GRLIB (Gaisler research):

Après vérification, le groupe de modules GRLIB ne contient aucun cœur de contrôleurs d'écran. Leur solution propose plutôt d'utiliser un contrôleur interfacé par le BUS PCI ou une solution provenant d'une tierce source. Par contre, tous les autres modules pourraient être utilisés pour une amélioration ultérieure du système (processeur SPARC entre autres). Voici quand même les informations disponibles pour ce modèle (source : [19] Gaisler et al. 2005) :

A. Nom du projet: grlib;
B. Langage(s) utilisé(s): VHDL;
C. Niveau de développement : approuvé pour FPGA, ASIC (« Application-Specific Integrated Circuit »);
D. BUS de communication : AMBA;

E. État de développement : Production/Stable;
F. Coûts d'acquisition : Gratuits (Version évaluation uniquement);
G. Coûts d'utilisation commerciale : à voir.

2.4.2 Mistral Software :

Ce système semble intéressant puisqu'il répond au standard « eXtanded Graphics Array » (XGA) et plus, ce qui veut dire que sa résolution maximale possible peut dépasser 1024 x 768 pixels. Il permet une sortie RGB 8:8:8 ou 10:10:10, ce qui permet 32 bpp. Le code est écrit en langage VHDL. Il n'y a pas plus d'information que le petit fichier *.pdf (« Portable Document Format ») de 2 pages ([38] Mistral Software Pvt. Ltd. 2004) et le site internet (url : http://www.mistralsoftware.com/home.htm qui est en construction au moment de la recherche). Un autre point moins intéressant, c'est qu'il ne semble pas posséder de BUS de communication et son schéma bloc est plutôt simpliste. Nous n'avons pas d'information sur les autres fonctions qu'il fournit. La documentation sur ce modèle n'est pas accessible et le code VHDL semble encore en développement. De plus, le code source n'est pas disponible gratuitement. Cette option devra être mise de côté.

Voici le schéma bloc du projet VGA-XGA Controller :

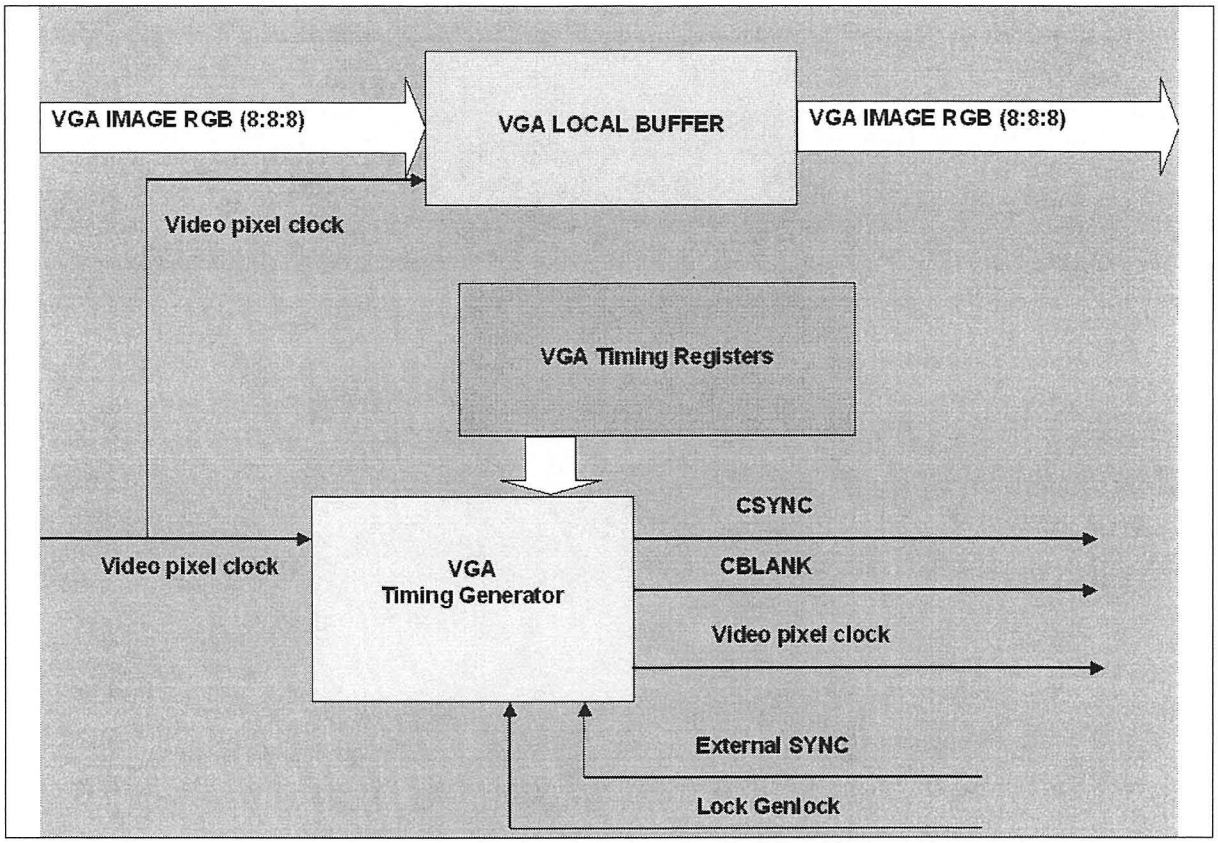

Figure 2.5 Schéma bloc du projet VGA-XGA Controller.

(Tiré de [38] Mistral Software Pvt. Ltd. 2004)

Source : cette figure est tirée du document de Mistral Software Pvt. Ltd. 2004, *VGA-XGA Controlller*, p. 1, url : http://www.mistralsoftware.com/VGA-XGA Controller.pdf.

Voici quand même les informations disponibles pour cette solution :

A. Nom du projet: VGA-XGA Controller;
B. Langage(s) utilisé(s): VHDL;
C. Niveau de développement : approuvé pour FPGA, ASIC;
D. BUS de communication : Aucun Spécifique;
E. État de développement : en développement;
F. Coûts d'acquisition : à voir (non gratuit);
G. Coûts d'utilisation commerciale : à voir.

2.4.3 TVOUT_CTRL (Cast inc.)

Cast inc. propose plusieurs cores intéressants, et celui-ci semble répondre à ce que nous recherchons. Il est écrit en langage VHDL. Le système est fait pour la télé haute définition et pour les solutions vidéo embarquées comme les téléphones portables, les PDA ou autres. Il permet une sortie RGB 8:8:8, ce qui permet une précision de couleurs de 24 bpp, mais ne permet pas de savoir la résolution maximale. Le Bus AMBA sert d'interface avec les autres modules SoC. Il permet un accès DMA unidirectionnel grâce à un contrôleur dédié. Il possède aussi un mode d'économie d'énergie, mais il est impossible de savoir s'il est possible de l'activer de manière logicielle. Les autres fonctions recherchées ne sont pas spécifiées. Le core pourrait servir de référence si nous pouvions y avoir accès gratuitement, mais étant donné que ce n'est pas le cas, cette option devra être laissée de côté.

Voici le schéma bloc du projet TVOUT_CTRL Video Display Controller Core, il peut toujours donner des idées :

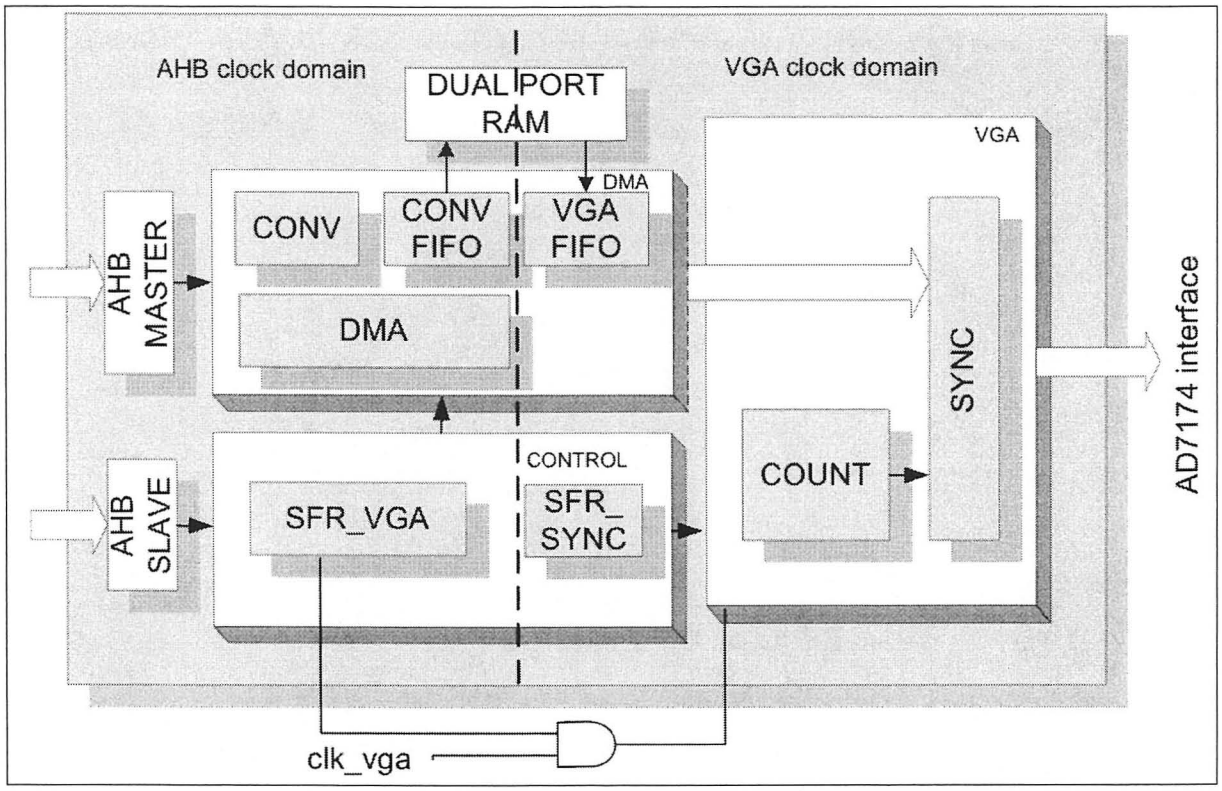

Figure 2.6 Schéma bloc du projet TVOUT_CTRL Video Display Controller Core.

(Tiré de [5] Cast inc. 2005)

Source : cette figure est tirée du document de Cast inc. 2005, *TVOUT_CTRL Video Display Controller Core*, p. 1, url : http://www.cast-inc.com/cores/tvout_ctrl/cast_tvout_ctrl.pdf.

Voici quand même les informations disponibles pour cette solution :

A. Nom du projet: TVOUT_CTRL Video Display Controller Core;
B. Langage(s) utilisé(s): VHDL;
C. Niveau de développement : approuvé pour FPGA, ASIC;
D. BUS de communication : AMBA (AHB);
E. État de développement : Production/Stable;
F. Coûts d'acquisition : à voir (non gratuit);
G. Coûts d'utilisation commerciale : à voir.

2.4.4 LogiCVC Compact Video Controler (Xylon logicbricks)

Ce module est intéressant ([72] Xylon, 2005). Il a l'avantage de supporter plusieurs types d'écrans. Cependant, la résolution maximale supportée est de seulement 1024 x 768 et le nombre de couleurs maximales est de 256, ce qui le limite donc à 8 bits par pixel. Une fonction intéressante y est implémentée : il est possible de générer des couleurs avec un seul bit par signal RGB en alternant les bits « on » et « off » d'un pixel à la manière d'un hacheur dans un délai déterminé. Cette fonction s'appelle XCOLOR™.

Le bus de communication est générique. Ceci ne semble pas être un problème puisque le système semble avoir été fait pour s'interfacer avec un processeur et de la mémoire facilement. La largeur du bus de données est de 16 bits. Il faut donc fabriquer une interface qui assemble deux données sur un bus de 32 bits. Sa largeur de bus pose un problème pour les évolutions futures du système.

Son coût ne semble pas élevé car sur le site de logicbricks, les gens le comparent au prix de pièces physiques achetées sur le marché actuel (coût équivalent à l'utilisation), soit 11,50 USD par module implanté. Cette option devra être laissée de côté malgré tout car elle ne répond pas suffisamment à nos critères de départ.

Voici tout de même le schéma bloc du projet LogiCVC Compact Video Controler :

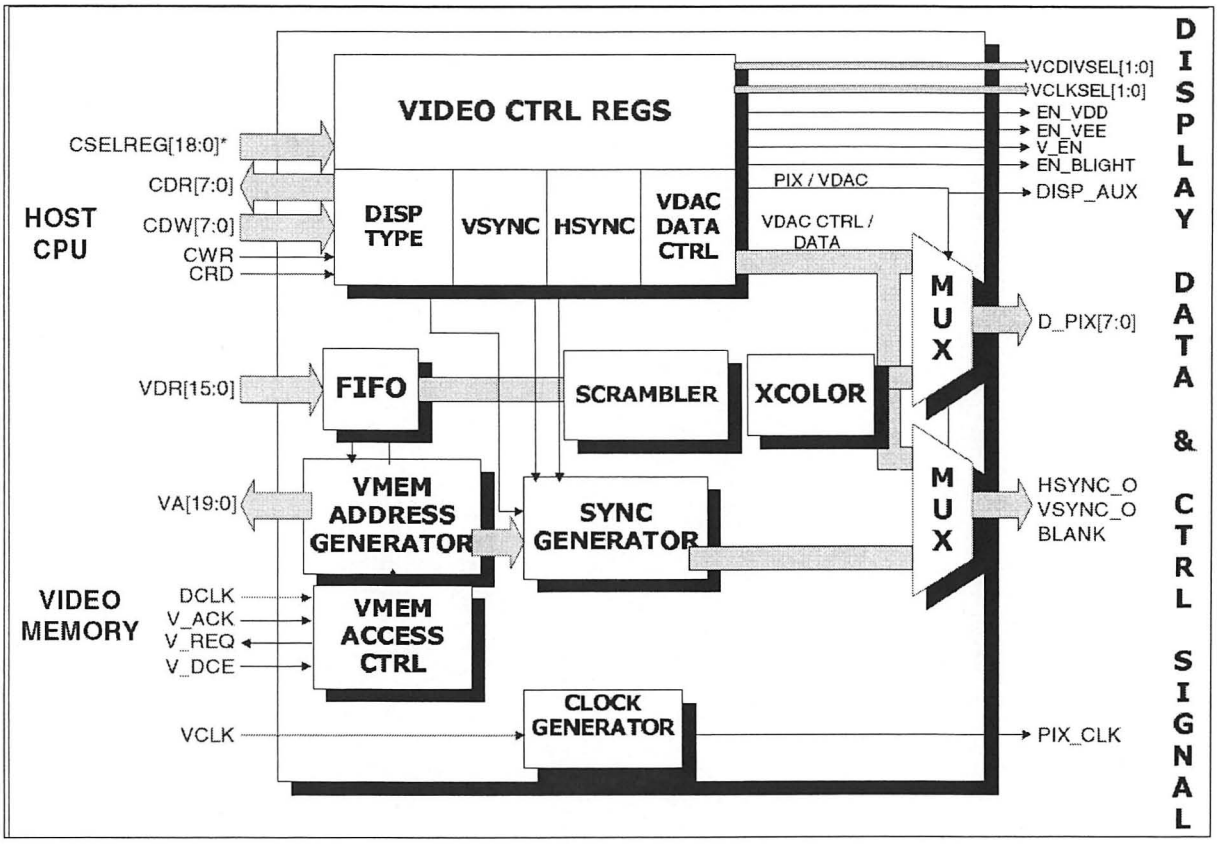

Figure 2.7 Schéma bloc du projet LogiCVC Compact Video Controler.

(Tiré de [50] Xilinx. 2001)

Source : cette figure est tirée du document de Xilinx. 2001, *logiCVC Compact Video Controller*, p. 1, url : http://www.xilinx.com/products/logicore/alliance/xylon/xylon_logicvc.pdf.

Voici quand même les informations disponibles pour cette solution :

A. Nom du projet: LogiCVC Compact Video Controler;
B. Langage(s) utilisé(s): VHDL;
C. Niveau de développement : approuvé pour FPGA, ASIC;
D. BUS de communication : générique;
E. État de développement : Production/Stable;
F. Coûts d'acquisition : environ 11,50$ US (non gratuit);
G. Coûts d'utilisation commerciale : à voir.

2.4.5 INT416-SXGA (Intrinsix)

Ce module ([33] Intrinsix Corp. 2005 et [34] Intrinsix Corp. 2005) est intéressant car il répond aux normes du standard VESA. Il permet donc une résolution maximale de 1280 x 1024 à 85Hz et 16,8M de couleurs. Le bus est générique, mais il est adaptable facilement aux différents bus du marché selon la compagnie (AMBA, PCI, PCI EXPRESS). Si on regarde le schéma bloc, on remarque que sa mémoire vidéo n'est pas partagée avec d'autres modules et qu'il y a un contrôleur/arbitre de mémoire, ce qui permet de penser que les images doivent être chargées avant de démarrer l'affichage. Ceci est une bonne chose dans notre cas, car cela permet de monopoliser la mémoire vidéo pour le contrôleur. Le code est écrit en VERILOG, ce qui nous obligerait à réécrire le tout en VHDL. Étant donné que tous les exemples d'applications données dans la description sont interfacés par un bus PCI, et les outils de test fournis sont des applications Windows, il est logique de penser que ce système est conçu pour la conception de cartes vidéo dans des systèmes ordinés d'ordinateurs personnels et, par le fait même, est plus performant que nos besoins à ce moment-ci du développement. Ce core a aussi l'inconvénient d'être payant. Il devra donc être mis de côté pour toutes ces raisons.

Voici le schéma bloc du projet (quasiment illisible sur le web) INT416-SXGA :

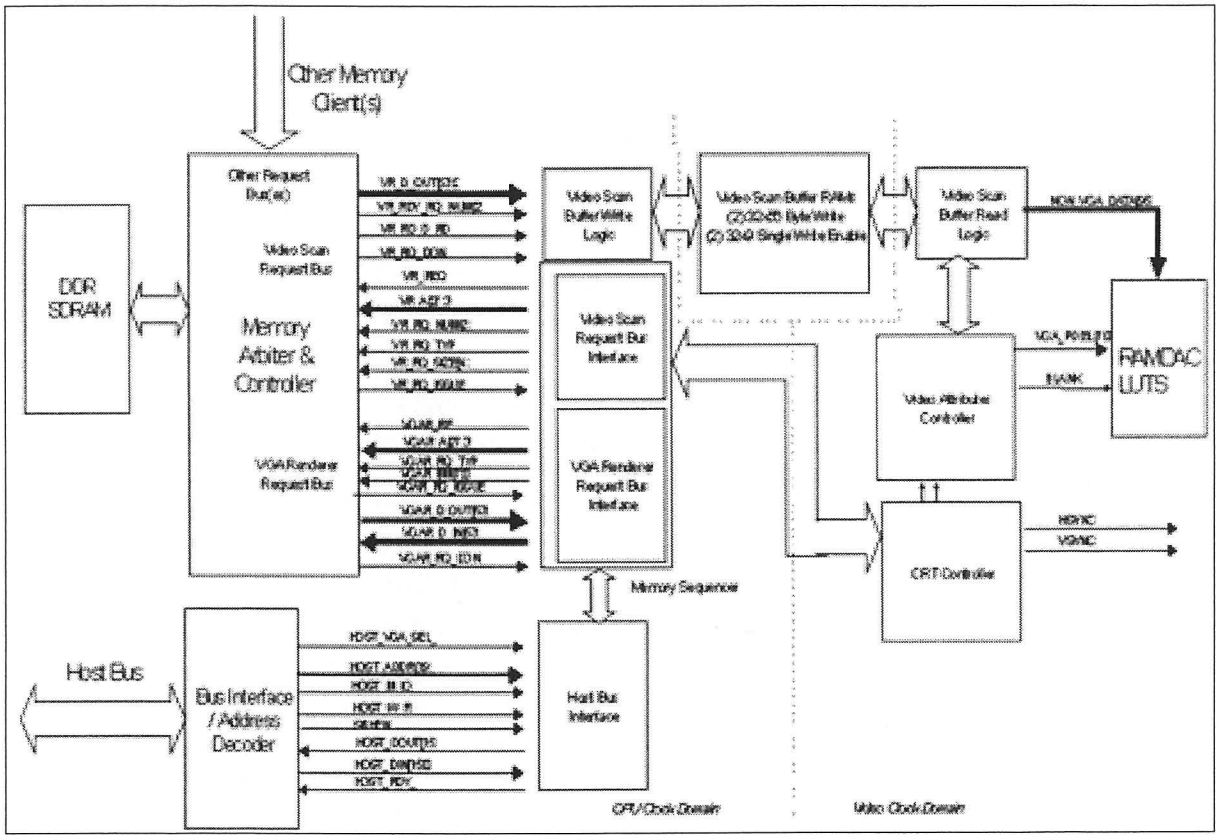

Figure 2.8 Schéma bloc du projet INT416-SXGA.

(Tiré de [33] Intrinsix Corp. 2005)

Source : cette figure est tirée du document de Intrinsix Corp. 2005, *VGA & SXGA Compatible HDL Cores*, p. 1, url : http://www.intrinsix.com/intrinsix-ip/vga/Intrinsix_VGA_and_SXGA_Product_Overview.pdf

Voici quand même les informations disponibles pour ce modèle :

A. Nom du projet: INT416-SXGA;
B. Langage(s) utilisé(s): VERILOG;
C. Niveau de développement : approuvé pour FPGA, ASIC;
D. bus de communication : Générique, adaptable aux différents bus du marché;
E. État de développement : Production/Stable;
F. Coûts d'acquisition : à voir (non gratuit);
G. Coûts d'utilisation commerciale : à voir.

2.4.6 Xilinx

Le site internet de Xilinx nous renvoie à ces entreprises :

A. Voir Cast Inc. (Section 2.4.3);
B. Voir Xylon (Section 2.4.4);
C. Voir Intrinsix (Section 2.4.5).

2.4.7 Altera

Le site internet d'Altera nous renvoie à ces entreprises :

A. Voir Cast Inc. (Section 2.4.3);
B. Voir OpenCores (Section 2.4.12).

2.4.8 Sources du laboratoire pour le cours ELE-748 de l'École de Technologie Supérieure :

Le modèle élaboré pour ce laboratoire est plutôt simple. Il possède une résolution fixe de 800 x 600, mais rien ne spécifie le nombre de couleurs. La plate-forme matérielle utilisée pour le laboratoire est la carte ML401 de Xilinx avec un triple convertisseur N/A de 8 bits/canal : le convertisseur numéro ADV7125KST50 de AnalogDevices achemine les signaux au port VGA de la carte. Ceci me permet d'affirmer qu'il est possible de faire du 24 bpp en théorie, mais il faudrait voir le code plus en détail. La seule description disponible du code source est le protocole de laboratoire du cours ([42] Perrault, 2005). Le code étant fourni, il faudra s'y attarder pour plus de détails, mais je doute que les spécifications soient très élaborées. Il peut être utilisé comme exemple d'explication mais, à mon avis, ce serait une erreur de l'utiliser comme point de départ étant donné sa trop grande simplicité. Il n'y a pas de schéma bloc pour ce module puisque le système d'affichage tient sur quelques lignes de code VHDL uniquement.

Voici les informations disponibles pour ce module :

A. Nom du projet: Laboratoire de ELE-748;
B. Langage(s) utilisé(s): VHDL;
C. Niveau de développement : approuvé pour FPGA;
D. BUS de communication : Aucun Spécifique;
E. État de développement : Stable mais simple;
F. Coûts d'acquisition : gratuit;
G. Coûts d'utilisation commerciale : gratuit.

2.4.9 VGACON

Ce module est gratuit, mais n'est pas très intéressant car il n'est pas assez complet pour nos besoins. En effet, la résolution est de 64 x 60 avec 8 couleurs. Donc, ce n'est pas avec ce système qu'on pourra travailler à la base. Il pourra seulement être utilisé comme exemple pour des fins de compréhension, mais ce sera tout.

Voici son symbole, nous n'avons aucun schéma bloc de ce système :

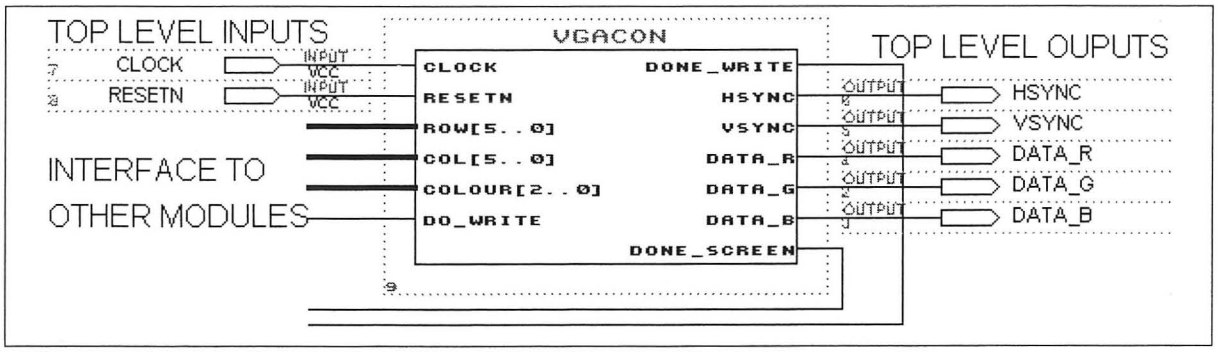

Figure 2.9 Symbole du projet vgacon.

(Tiré de [45] Singh et Egier. 2004)

Source : cette figure est tirée du site internet de Singh, Deshanand et Aaron Egier. 2004, *A Simple VGA Controller*, url : http://www.eecg.toronto.edu/~jayar/ece241_05F/vga_new/index.html

Voici les informations disponibles pour cette solution :

A. Nom du projet: vgacon;
B. Langage(s) utilisé(s): VHDL;
C. Niveau de développement : code de base pour FPGA Altera FLEX10K20;
D. bus de communication : Aucun Spécifique;
E. État de développement : à voir, donne des exemples fonctionnels;
F. Coûts d'acquisition : Gratuit;
G. Coûts d'utilisation commerciale : à voir.

2.4.10 XESS

Comme pour le système VGACON, le code de XESS pourra nous guider pour la compréhension seulement et ne pourra pas nous aider par la suite car il n'est pas assez complet. Effectivement, la résolution n'est pas très grande et ne permet que 64 couleurs. Par contre, ce qui est intéressant, c'est que nous avons accès non seulement au code VHDL, mais aussi à la partie matérielle du port VGA. Il n'y a pas de schéma bloc pour ce module, le code VHDL suffit ([48] XESS Corp., 2004 et [49] XESS Corp., 2005).

Voici les informations disponibles pour cette solution :

A. Nom du projet: VGA;
B. Langage(s) utilisé(s): VHDL;
C. Niveau de développement : exemple de développement pour FPGA;
D. bus de communication : Aucun Spécifique;
E. État de développement : exemple Stable;
F. Coûts d'acquisition : gratuit;
G. Coûts d'utilisation commerciale : gratuit.

2.4.11 Expressive IV (Parallel Systems co.)

Comme pour le système VGACON, le code pourra nous guider au départ seulement pour comprendre les principes d'affichage et ne pourra pas nous aider par la suite car il n'est pas

assez complet. Effectivement, la résolution maximale n'est pas spécifiée ([17] Expressive System, 2000). Par contre, le système peut atteindre 16M de couleurs selon le code qui est d'ailleurs écrit en VHDL et qui est accessible gratuitement (en partie du moins).

Voici le schéma de fonctionnement (qui est quasiment illisible sur internet) du projet Expressive IV :

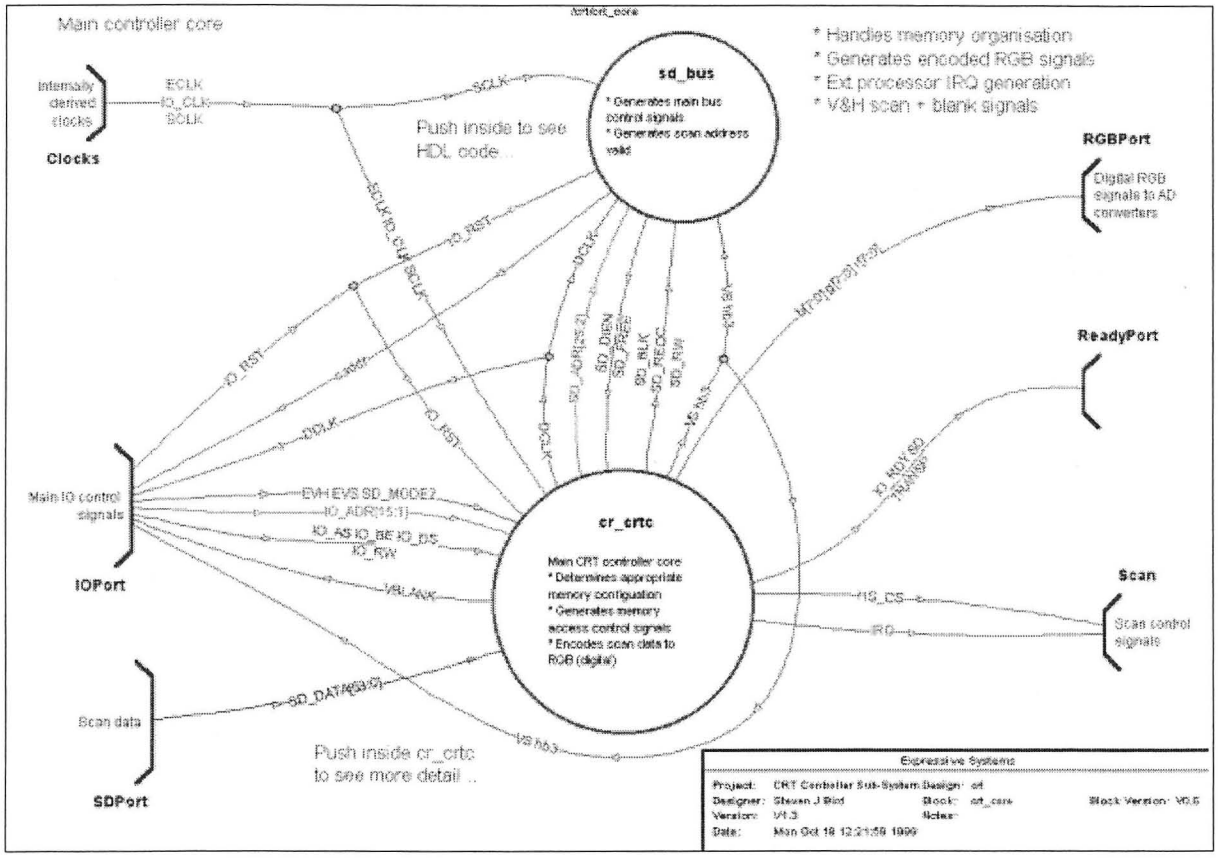

Figure 2.10 Schéma bloc du projet Expressive IV.

(Tiré de [17] Expressive System. 2000)

Source : cette figure est tirée du site internet d'Expressive System. 2000, *VHDL Source for the following design sheet*, url : http://www.parallel-systems.co.uk/expressive-vhdl.htm

Voici les informations disponibles pour cette solution :

A. Nom du projet: Expressive IV;
B. Langage(s) utilisé(s): VHDL;

C. Niveau de développement : exemple de développement pour FPGA;
D. bus de communication : Aucun Spécifique;
E. État de développement : exemple Stable;
F. Coûts d'acquisition : gratuit;
G. Coûts d'utilisation commerciale : gratuit.

2.4.12 VGA/LCD Core v2.0 (Opencores):

Le contrôleur VGA/LCD de Opencores ([24] Herveille, 2003) supporte les écrans CRT et LCD avec des résolutions et des fréquences de rafraîchissement programmables. Il peut supporter 3 écrans à la fois (Noir et blanc). Il supporte les modes de couleurs suivantes : 32bpp, 24bpp (mais seulement 24 bits de sortie RGB disponibles pour le moment), 16bpp (deux pixels par mot de 32 bits), 8bpp en niveaux de gris et 8bpp en pseudo couleur avec deux possibilités de tables de correspondance accessibles de l'extérieur du module. La table des couleurs est située à l'intérieur du core afin de réduire la bande passante nécessaire de la mémoire et permettre un débit de données plus rapide. Les données d'images sont lues automatiquement via l'interface maître WISHBONE. Pour ce qui est de la sortie, elle est standard RGB (VGA).

La résolution maximale donnée à l'appendice A du document vga_core.pdf est de 800 x 600, mais dans les exemples de calculs, il semble être possible de pousser les résolutions plus loin (1024 x 768 ou plus). La mémoire vidéo est extérieure au cœur primaire, ce qui permet une solution plus flexible pour la mémoire. Il est donc possible de partager la mémoire système avec un processeur ou de la rendre dédiée à la vidéo si nécessaire.

Les applications vidéo plus performantes, comme dans les jeux vidéo ou pour la lecture vidéo continue, bénéficient de la fonction d'alternance de deux banques vidéo. De plus, pour réduire le scintillement et l'encombrement des images, on change de page mémoire vidéo et/ou table de couleurs à chaque retour vertical (ce qui est logique).

Deux curseurs matériels permettent une flexibilité additionnelle. Ils peuvent être de deux formats différents et leur image est stockée dans le cœur afin de limiter les accès mémoire.

Le système utilise une interface WISHBONE pour les communications inter modules, ce qui est intéressant. La version du code le plus à jour est écrite en VERILOG malheureusement. Par contre, une version moins récente, avec sensiblement la même topologie, mais qui possède moins de fonctionnalités, est disponible en VHDL. Il est aussi à remarquer que la documentation est bien détaillée et que l'utilisation du module est gratuite. Par contre, étant donné que le code est écrit sous une licence de type BSD, il nous est impossible de le modifier sans le republier gratuitement sur l'internet. Nous ne pouvons donc, en aucun cas, utiliser ce code puisque il est primordial de pouvoir l'adapter à nos besoins tout en restant propriétaire.

Voici le schéma bloc du projet vga_lcd (version VERILOG) :

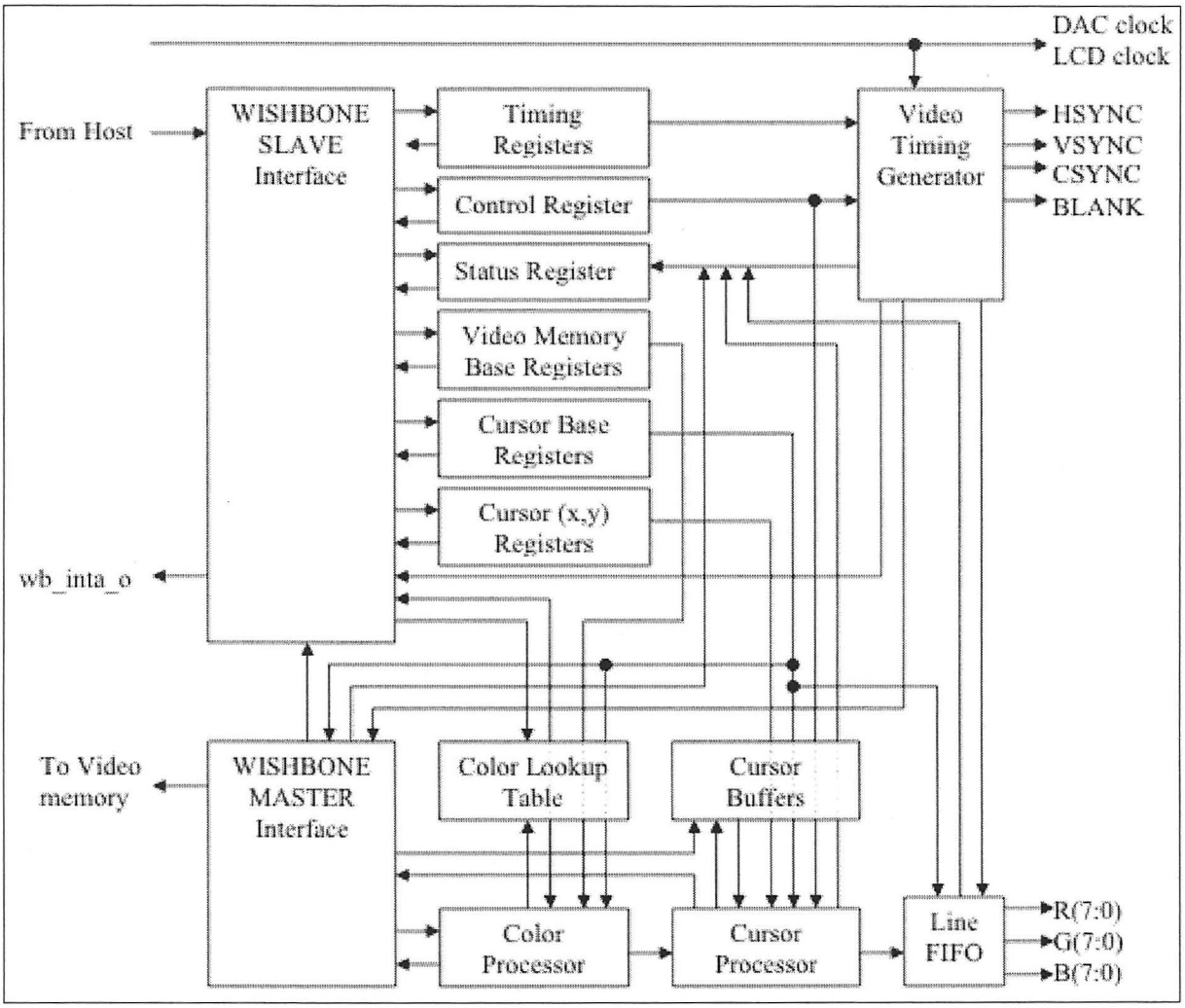

Figure 2.11 Schéma bloc du projet vga_lcd.

(Tiré de [24] Herveille, 2003)

Source : cette figure est tirée du document de M. Richard Herveille, 2003, *VGA/LCD Core v2.0*, url : http://www.opencores.org/cvsweb.shtml/vga_lcd/doc/vga_core.pdf

Voici les informations disponibles pour ce modèle :

A. Nom du projet: vga_lcd;
B. Langage(s) utilisé(s): VERILOG (dernière version) et VHDL (ancienne version);
C. Niveau de développement : approuvé pour FPGA, ASIC;

D. BUS de communication : Wishbone;

E. État de développement : Production/Stable;

F. Coûts d'acquisition : Gratuit (licence de type BSD);

G. Coûts d'utilisation commerciale : Gratuit (licence de type BSD).

2.5 Conclusion sur le choix des spécifications fonctionnelles de départ du projet:

Pour répondre aux spécifications de départ, le code en entier doit être réécrit. Effectivement, aucun cœur ne nous offre suffisamment de performance et de versatilité tout en étant gratuit à utiliser. Voici les tableaux qui résument les technologies étudiées avec leurs principales caractéristiques.

Tableau 2.1

Résumé des recherches sur les bus SoC avec leurs principales force et faiblesses

Bus étudié	Complexité d'utilisation	Bande passante	Licence
SERDES	Élevée	Très élevée	NA
CoreConnect	Élevée	Élevée	Non gratuite
AMBA	Élevée	Élevée	Non gratuite
Wishbone	Basse	Élevée	Gratuite

Tableau 2.2

Résumé des recherches de technologies de connectivité avec les écrans

Technologie	Complexité	signal	connecteurs physiques	Popularité	Compatibilité
RGB	Peu de contraintes	Analogique	Prise coaxiale (encombrant)	Peu répandu	Présentement utilisé chez CMC Électronique
VGA	Peu de contraintes	Analogique	DB15 Haute densité	Très répandu	Compatible RGB
DVI	Contraintes fréquences élevées	Numérique ou Analogique	Connecteur DVI	En émergence	Compatible RGB

Tableau 2.3

Résumé des recherches de coeurs avec leurs principales force et faiblesses

Coeur étudié	Langage	Niveau de développement	BUS SoC	Licence d'utilisation
GRLIB	VHDL	FPGA, ASIC	AMBA	Non gratuit
Mistral Software	VHDL	En développement	Générique	Non gratuit
TVOUT_CTRL	VHDL	FPGA, ASIC	AMBA	Non gratuit
LogiCVC	VHDL	FPGA, ASIC	Générique	Non gratuit
INT416	Verilog	FPGA, ASIC	Générique	Non gratuit
Code ELE-748	VHDL	Exemple base	Générique	Gratuit
VGACON	VHDL	Exemple base	Générique	Inconnue
XESS	VHDL	Exemple base	Générique	Gratuit
Expressive IV	VHDL	Exemple base	Générique	Gratuit
VGA/LCD Core v2.0	Vérilog (VHDL)	FPGA	Wishbone	Gratuit (BSD)

Quelques projets embryonnaires peuvent nous servir d'exemple de démarrage pour la compréhension, mais ils ne permettent pas d'établir notre topologie de départ sur leur base. De plus, étant donné que l'entreprise CMC Électronique veut rester propriétaire du code source, il nous est impossible d'utiliser le code source provenant d'OPENCORES.ORG (qui serait le plus intéressant) puisque cela nous obligerait à garder le code source ouvert (licence de type BSD). Ce code nous permet cependant d'établir les limitations technologiques atteignables pour ce type de projet.

L'étude des contrôleurs vidéo existants nous a permis de définir les spécifications de base qui nous guideront dans l'écriture du code VHDL. Voici un tableau résumé de ce que le code source du projet d'AVN doit remplir comme fonction de base :

Tableau 2.4

Spécifications de départ des fonctions devant être incluse dans le code source

Spécifications	Description
Sortie RGB	Le système doit posséder une sortie RGB (ou VGA) standard pour la compatibilité avec les systèmes existants de CMC Électronique et aussi avec la possibilité d'une mise à jour vers un port DVI.
Resolution minimale d'affichage	Le système doit permettre au minimum les résolutions de 1024 x 768 couleur 8 bpp (256 couleurs) et de 800 x 600 couleur 24 bpp (16 millions de couleurs).
Mémoire video externe	Le système doit utiliser une mémoire vidéo externe au FPGA pour limiter l'utilisation des « blockRAM » de ce dernier.
BUS SoC Wishbone	Le système doit utiliser un bus SoC Wishbone pour permettre un interfaçage rapide, simple, bien documenté et gratuit d'utilisation.
Deux banques d'image accessibles	Le système doit utiliser deux banques d'images en alternance pour permettre des performances accrues et une meilleure qualité des transitions d'image

Code source VHDL	Le code source doit être écrit en VHDL selon les standards et le style de la compagnie CMC Électronique
Mode d'économie d'énergie	Le système doit permettre d'activer de manière logicielle un mode d'économie d'énergie

Voyons maintenant dans le chapitre qui suit, à la lumière de ces caractéristiques fonctionnelles, quelle plateforme de développement répond le mieux à nos besoins.

CHAPITRE 3

ÉTUDE DES POSSIBILITÉS DE PLATEFORMES DE DÉVELOPPEMENT

3.1 Introduction, ce qui guide notre recherche :

Ce chapitre permet d'analyser les différentes possibilités qui s'offrent à nous présentement en matière de plateforme de développement. Les critères utilisés pour arrêter notre choix sont déterminés par les caractéristiques fonctionnelles du système à modéliser et aussi à certaines contraintes extérieures au fonctionnement du système. La carte de développement doit :

a. être à un prix raisonnablement bas;
b. permettre une sortie Vidéo avec une connexion VGA ou DVI possédant un minimum de 8 bits par pixel (8 :8 :8);
c. posséder des connecteurs d'entrées/sorties configurables (user I/O) afin de permettre l'évolution du système à l'aide de connecteurs;
d. posséder un FPGA avec une grande quantité de portes dont on priorise la Série Spartan III de Xilinx pour son bas prix et aussi parce que son environnement de développement est bien connu;
e. posséder un certain nombre de diodes électroluminescentes (DEL) et d'interrupteurs pour faciliter le déverminage;
f. posséder une documentation complète et détaillée facilement accessible.

Nous envisageons plusieurs possibilités puisqu'il n'est pas certain que les solutions offertes correspondent exactement à ce que nous recherchons. Pour cette raison, il est possible de composer avec plusieurs solutions groupées qui pourront éventuellement répondre à nos besoins.

3.2 Les solutions étudiées

Nous faisons ici la liste de chaque possibilité étudiée, intéressante ou non. Pour chaque possibilité, une explication des avantages et inconvénients valide l'éventualité de son utilisation ou non dans le projet.

3.2.1 Solutions complètes avec DVI

Il est question de solutions complètes ici puisque les solutions mentionnées dans cette section nous permettraient de développer un contrôleur vidéo avec un port de sortie vidéo DVI sans aucune modification ou rajout sur la plateforme de développement. Ces solutions sont intéressantes si leur coût n'est pas prohibitif.

3.2.1.1 La carte Sendero développée par Microtronix :

Voici les informations de base de la plateforme développée par la compagnie Microtronix ([37] Microtronix, 2005) :

A. Modèle : 6997-01-01 (Sendero Evaluation Kit);
B. Prix : 895$ US.

Voici une photo de cette carte de développement :

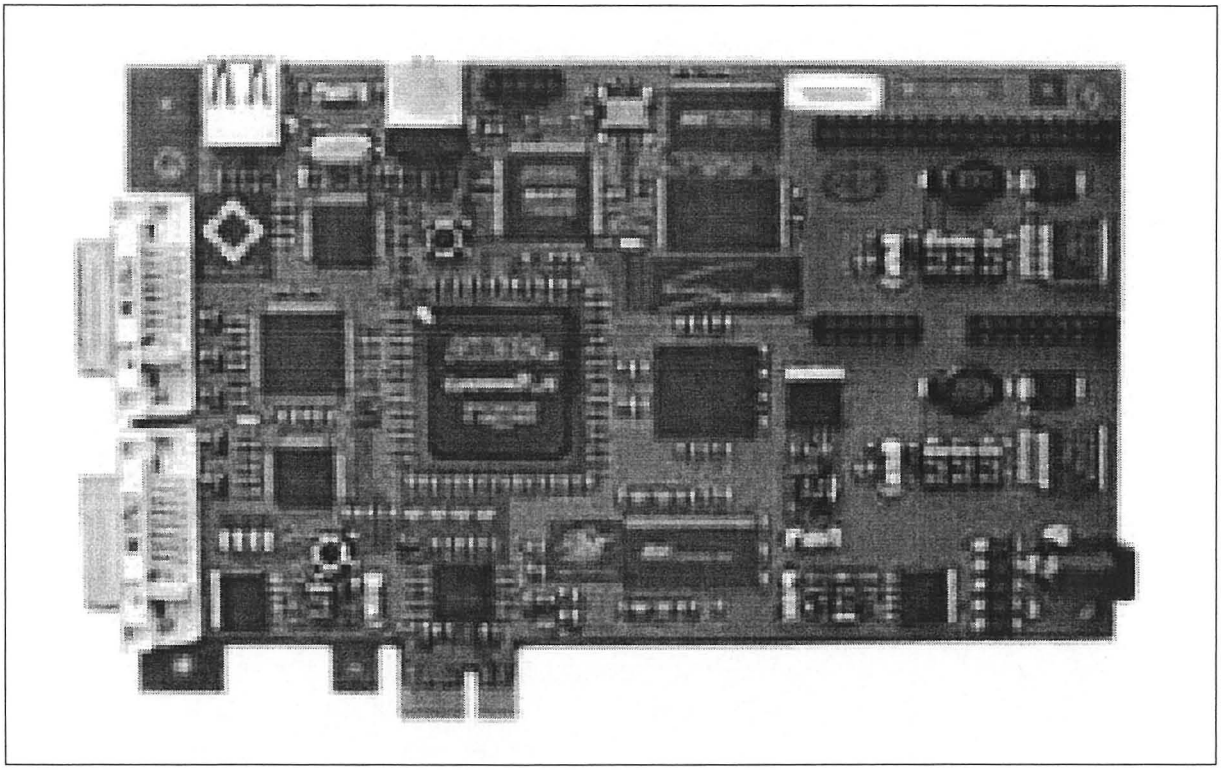

Figure 3.1 Carte modèle : 6997-01-01 (Sendero Evaluation Kit).

(Tiré de [37] Microtronix, 2005)

Source : cette figure est tirée de la page web de Microtronix. 2005, *Sendero Evaluation Kit*, url : http://www.microtronix.com/acatalog/Standard_Development_Kits.html.

3.2.1.1.1 Caractéristiques de la carte d'évaluation « Sendero »:

Voici les caractéristiques importantes pour nos besoins de la carte de développement « Sendero ». Elle possède :

a. une composante Cyclone II d'Altera (bas prix);
b. une mémoire de haute performance pour les applications d'imagerie;
c. un contrôleur mémoire DDR (« Double Data Rate ») et QDRII qui s'interface avec la mémoire QDRII SRAM (« Static Random Access Memory ») (666 Mbps) et la

mémoire DDR SDRAM (« Synchronous Dynamic Random Access Memory ») (333 Mbps);

d. des interfaces DVI en entrée et en sortie qui permettent d'envoyer et de recevoir des flux vidéo ("Streams");
e. deux ports USB (« Universal Serial Bus ») 2.0, un maître et un esclave;
f. une interface PCI Express qui supporte les applications à 1X;
g. une extension sous forme d'une carte fille appelée « Santa Cruz » permettant une amélioration des fonctionnalités;
h. des designs de références, de la documentation et un compilateur « Avalon Bus Core » pour les mémoires, USB et DVI qui sont fournis à l'achat.

Cette carte possède beaucoup d'avantages, mais ne répond pas à nos critères de départ puisqu'elle est trop performante pour nos besoins. En effet, il est inutile pour notre application d'avoir de la mémoire vidéo aussi performante car nous n'utiliserons pas ces capacités. Pour cette raison, le prix à payer pour cette carte nous force à la laisser de côté. L'utilisation d'une composante Altera n'est pas exactement ce qui est recherché, mais l'idée d'une composante à bon marché reste une chose intéressante sur cette solution.

3.2.1.2 La Carte de développement vidéo haute performance VIODC développée par Xilinx :

Voici les informations de base de la plateforme de développement vidéo haute performance développée par la compagnie Xilinx ([70] Xilinx. 2006) :

A. Modèle : HW-V4SX35-VIDEO-SK-US (Virtex-4 Video Starter Kit);
B. Prix : 1495$ US;
C. Composante Xilinx utilisée: XC4VSX35-FF668-10C;
D. Carte mère : ML402: SX XtremeDSP Evaluation Platform;
E. Carte fille : Video Input Output Daughter Card: (VIODC).

Voici des images de cette solution qui comprend plusieurs cartes :

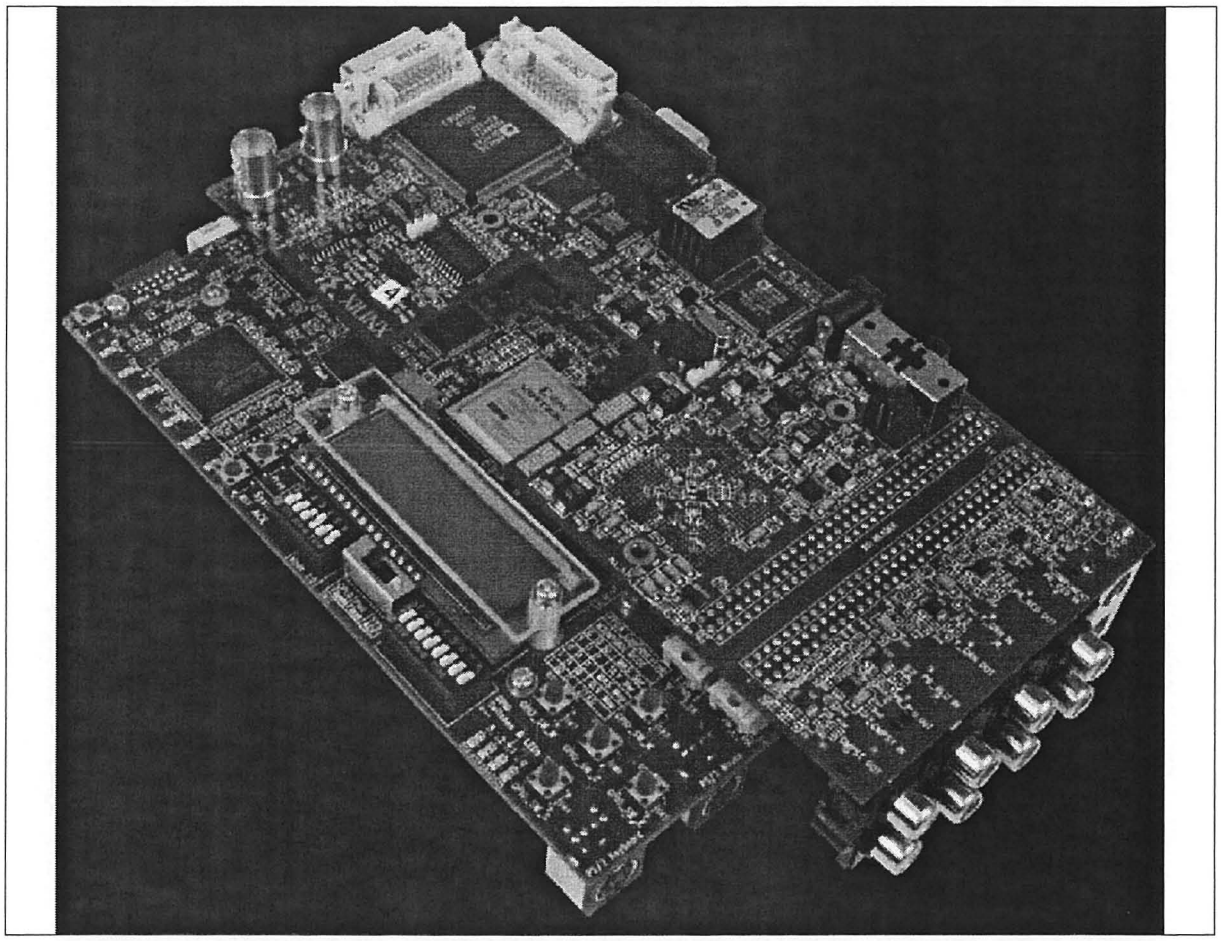

Figure 3.2 Assemblage de l'ensemble de développement XC4VSX35-FF668-10C.

(Tiré de [70] Xilinx, 2006)

Source : cette figure est tirée de la page web de Xilinx. 2006, *Virtex-4 Video Starter Kit*, url : http://www.xilinx.com/xlnx/xebiz/designResources/ip_product_details.jsp?key=HW-V4SX35-VIDEO-SK-US&sGlobalNavPick=PRODUCTS&sSecondaryNavPick=BOARDS.

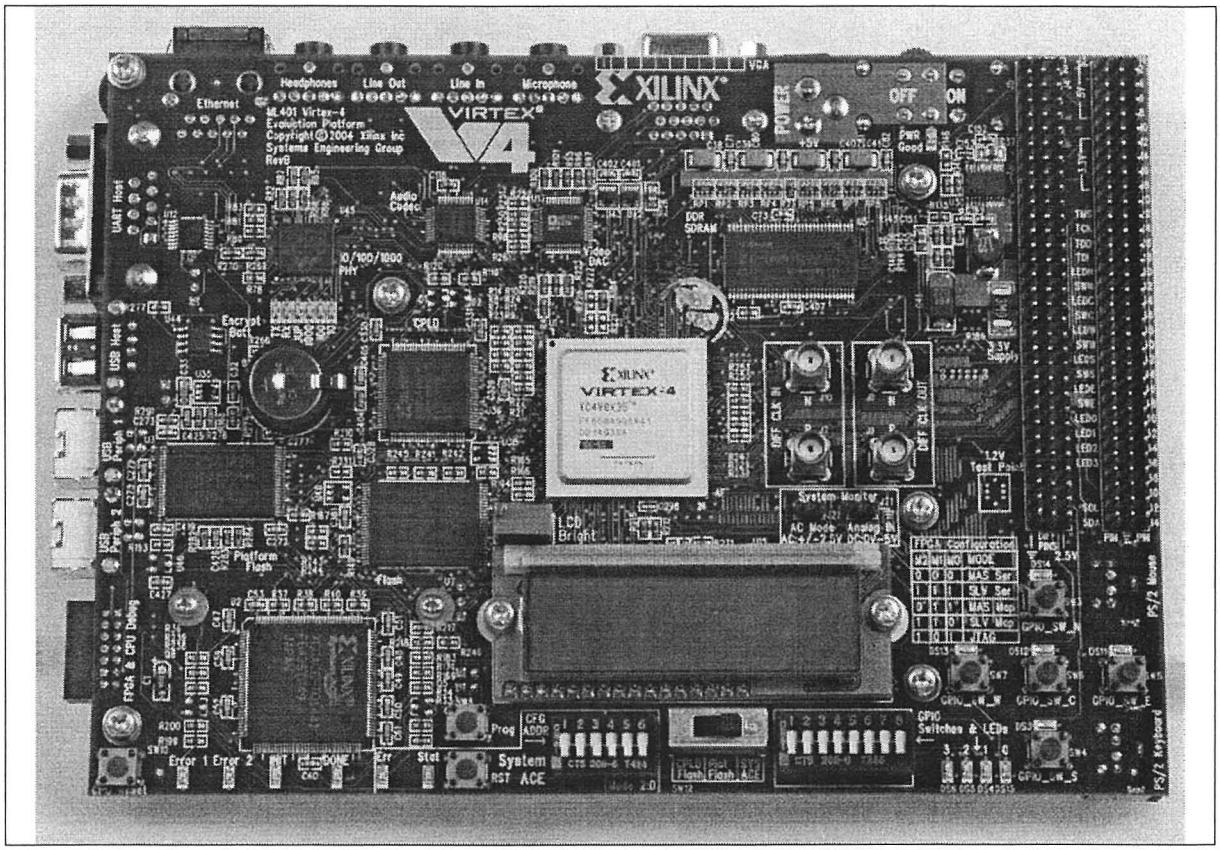

Figure 3.3 Carte mère ML40x uniquement.

(Tiré de [69] Xilinx, 2006)

Source : cette figure est tirée de la page web de Xilinx. 2006, *Virtex-4 ML402 SX XtremeDSP Evaluation Platform*, url : http://www.xilinx.com/xlnx/xebiz/designResources/ip_product_details.jsp?sGlobalNavPick=PRODUCTS&sSecondaryNavPick=BOARDS&iLanguageID=1&key=HW-V4-ML402-USA .

3.2.1.2.1 Caractéristiques de la solution vidéo haute performance de Xilinx:

Cet ensemble de cartes comprend :

a. une plateforme d'évaluation ML402 SX XtremeDSP (HW-ML402-USA);
b. une carte fille d'entrées/sorties vidéo (HW-XGI-VIDEO);
c. une caméra/capteur d'images couleur CMOS 742x480x60Hz RGB à balayage progressif pour l'automobile Micron MT9V022;
d. des entrées et sorties S-Video, composites, HD-SDI, DVI et VGA;

e. deux câbles Ethernet;
f. un adaptateur DVI à VGA;
g. deux Cartes Compact Flash de 32-MB avec socle de lecture SD et HD avec des démonstrations préenregistrées;
h. les logiciels « Xilinx ISE System Generator for DSP », « EDK », et « Ethernet MAC Evaluation » limités à la composante XC4VSX35;
i. le CD « VSK » (Manuels d'usagé, Schémas des circuits, Exemples de design, Gerber, et plus).

Encore une fois, cette solution est beaucoup trop performante pour nos besoins et son coût est beaucoup trop élevé. Cependant, la carte-fille VIODC pourrait devenir une option intéressante si le projet prenait de l'ampleur dans le futur. Nous verrons cela plus loin dans la section 3.3.3.

3.2.1.3 La carte ML402 :

Cette carte constitue la carte maîtresse de l'ensemble VIODC, sa description est donnée dans les sections suivantes, mais voici d'abord ses informations de base ([69] Xilinx, 2006) :

A. Modèle : HW-V4-ML402-USA;
B. Prix de la carte seule : 595$ US;
C. Composante Xilinx utilisée: XC4VSX35-FF668-10C.

3.2.1.3.1 Caractéristiques de la carte ML402 :

La plateforme de développement ML402 comprend :

a. une horloge cadencée à 100 MHz et deux socles d'horloge;
b. une mémoire de 8 Mb ZBT (Zero Bus Turnaround) SRAM;
c. une mémoire de 64 MB DDR SDRAM;
d. une mémoire Flash de 64 Mb;

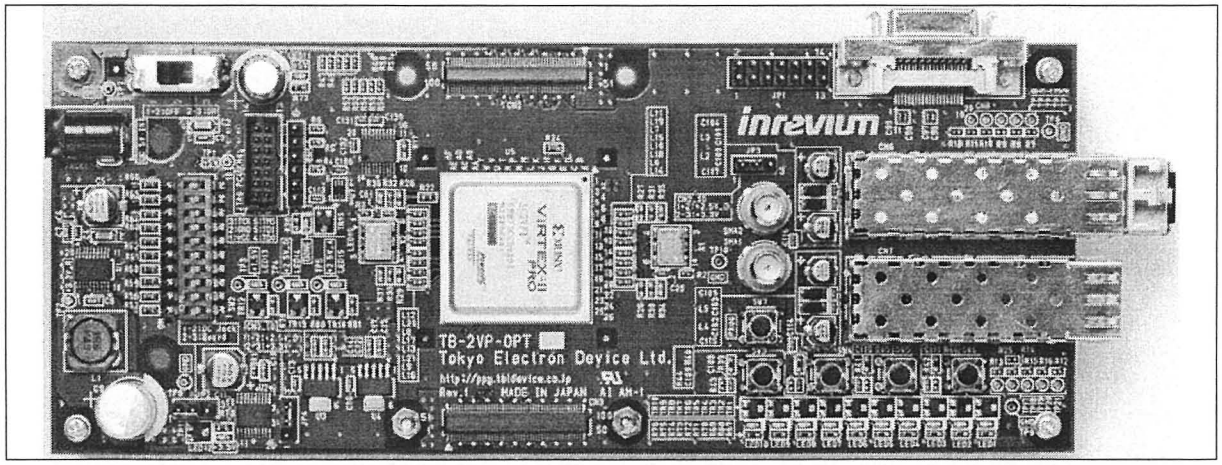

Figure 3.4 Carte modèle TB-V2P-OPT -2G.

(Tiré de [25] HiTech Global. 2005)

Source : cette figure est tirée de la page web de HiTech Global. 2005, *Optical Module with Virtex 2 Pro RocketIO*, url : http://www.hitechglobal.com/TED/OpticalModule.htm .

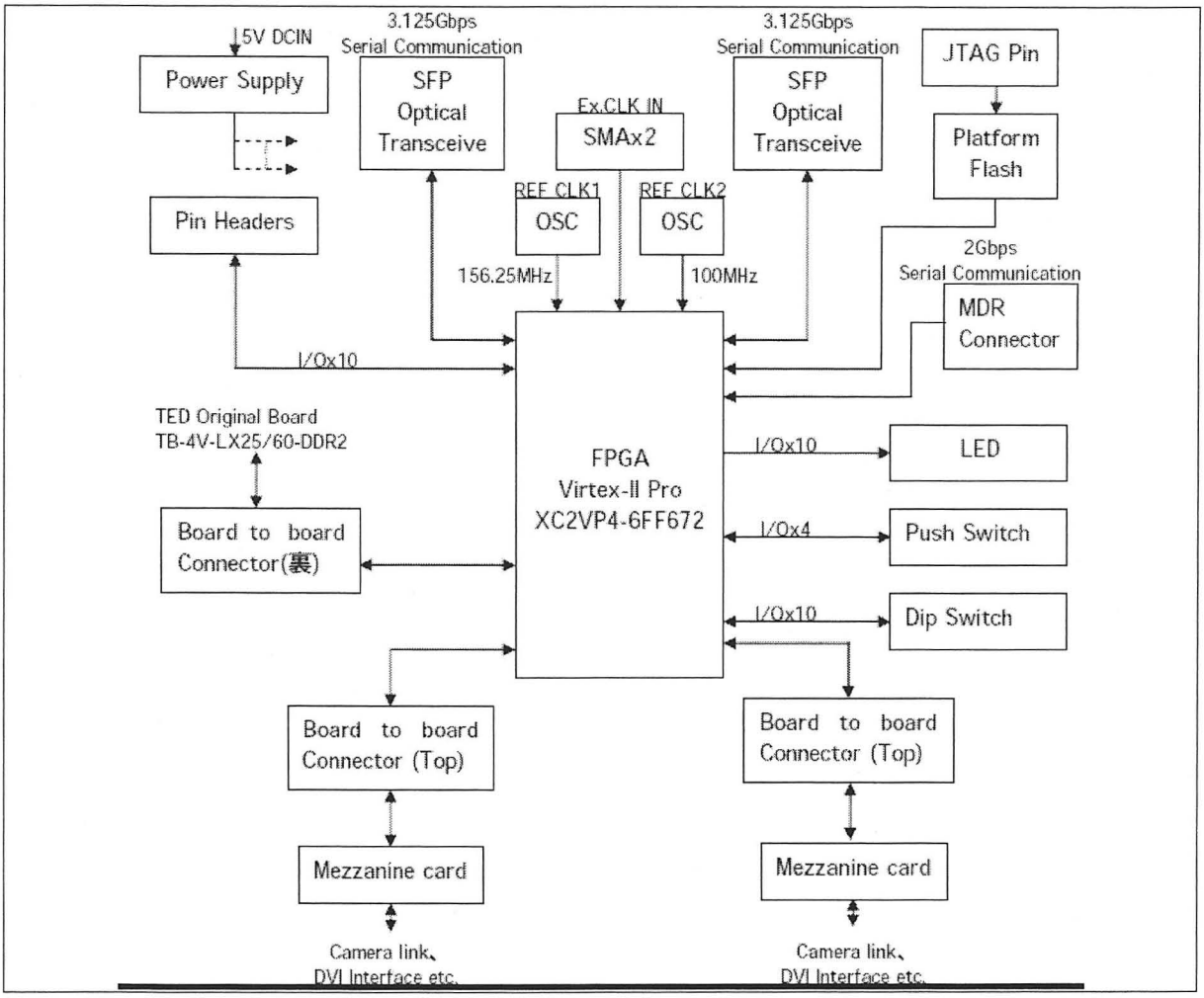

Figure 3.5 Schéma bloc de la carte modèle TB-V2P-OPT -2G.

(Tiré de [25] HiTech Global. 2005)

Source : cette figure est tirée de la page web de HiTech Global. 2005, *Optical Module with Virtex 2 Pro RocketIO*, url : http://www.hitechglobal.com/TED/OpticalModule.htm .

Pour pouvoir utiliser un port DVI, il est nécessaire d'ajouter une carte mezzanine sur les ports d'extension de la carte maîtresse. Voici les photographies des cartes mezzanine DVI Tx et DVI Rx

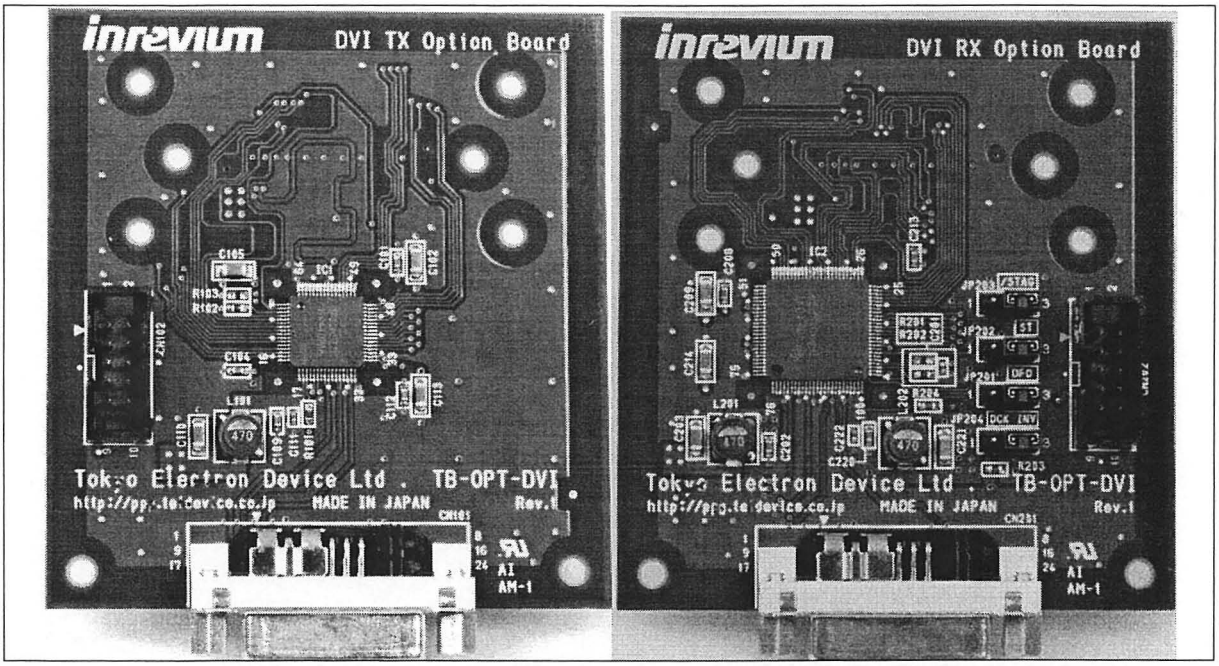

Figure 3.6 Cartes filles RX et TX pour le support DVI.

(Tiré de [27] HiTech Global. 2006)

Source : cette figure est tirée de la page web de HiTech Global. 2006, *Spartan 3E Display Solution Board*, url : http://hitechglobal.com/TED/Spartan3E.htm .

3.2.1.4.1 Les caractéristiques de la solution avec la carte TB-V2P-OPT -2G d'Inrivium:

La plateforme TB-V2P-OPT -2G d'Inrivium comprend :

a. deux canaux RocketI/O avec connexion au Module AFBR-57R5AP d'Agilent (supporte des signaux sériels jusqu'à un débit de 3.125Gbps);
b. deux canaux RocketI/O avec connexion à un port MDR;
c. un connecteur WR-100P-VF-1 pour supporter la connexion de cartes mezzanines pour un lien caméra ou une interface DVI;
d. un connecteur WR-100P-VF-1 pour supporter des opérations avec la carte Virtex-4 DDR2/DVI;
e. un guide de l'utilisateur « Hardware »;
f. des exemples d'applications.

Cette carte possède un énorme désavantage par rapport à nos besoins, elle ne possède pas de mémoire suffisante pour permettre de stocker les images. Elle est de plus très chère puisqu'elle est trop puissante pour nos besoins Il est nécessaire de continuer à chercher.

3.2.1.5 La carte TB-3S-1600E-IMG de la compagnie Inrevium :

Voici les informations de base pour la solution avec la carte TB-3S-1600E-IMG de Inrevium ([27] HiTech Global. 2006) :

A. Modèle : TB-3S-1600E-IMG (Spartan-3E Display Solutions Board);
B. Prix de la carte mère : 1395$ US;
C. Composante Xilinx utilisée: Spartan III XC3S1600E-4FG484C;
D. Modèle de la carte fille : TB-SUB-DVI (Interface DVI Tx/Rx);
E. Prix pour la carte fille : 498$US.

Pour mieux visualiser l'aspect de la carte, voici une photographie de celle-ci :

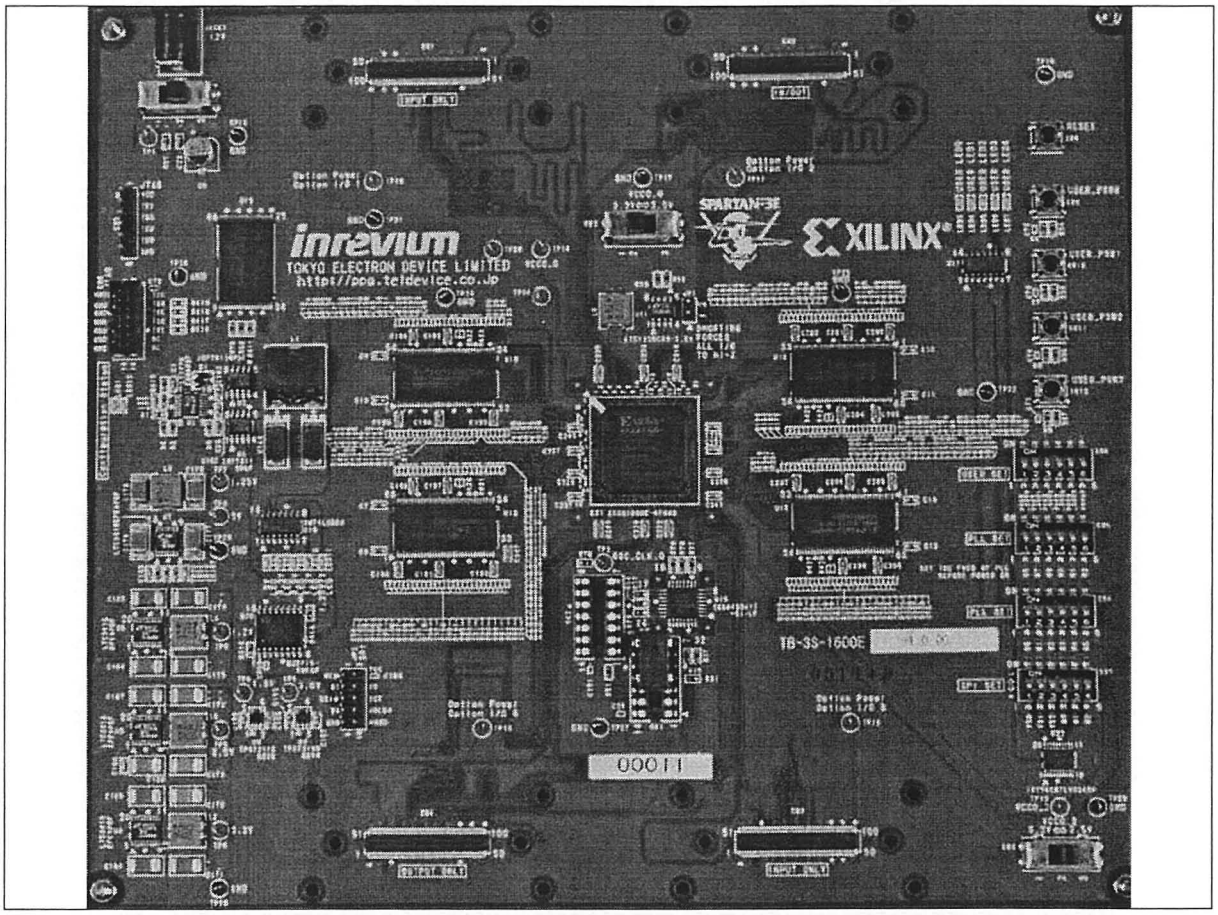

Figure 3.7 Carte modèle TB-3S-1600E-IMG.

(Tiré de [27] HiTech Global. 2006)

Source : cette figure est tirée de la page web de HiTech Global. 2006, *Spartan 3E Display Solution Board*, url : http://hitechglobal.com/TED/Spartan3E.htm .

Pour les cartes filles, ce sont les mêmes qu'à la section précédente, référez-vous à la Figure 3.6.

3.2.1.5.1 Les caractéristiques de la carte TB-3S-1600E-IMG:

La carte TB-3S-1600E-IMG comprend :

a. une mémoire PROM XCF08P-VO48 pour la configuration du FPGA;
b. quatre composantes de mémoire Elpida DDR modèle EDD5116ADTA (512Mbit x16);
c. une mémoire générale SPI de modèle ST Micro M25P16-VMF6P;
d. une horloge externe à PLL programmable et socle OSC;
e. quatre connecteurs d'expansion WR-100S-VF-1;
f. une interface DVI via les cartes filles DVI Tx/Rx Option Modules (jusqu'à Ultra eXtended Graphics Array (UXGA)).

Malgré que plusieurs critères de cette solution répondent à nos besoins, cette carte ne répond pas aux objectifs budgétaires du projet. Nous devons mettre cette option de côté.

3.2.1.6 La carte TB-3S-1500-IMG de Xilinx :

Voici les informations de base de la carte de développement TB-3S-1500-IMG ([26] HiTech Global. 2006) :

A. Modèle : TB-3S-1500-IMG;
B. Prix : 2000$ US;
C. Composante Xilinx utilisée: Spartan III (XC3S1500-4FG676 ou XC3S2000-FG676).

Avant de faire la description de cette solution, voici une image de la carte :

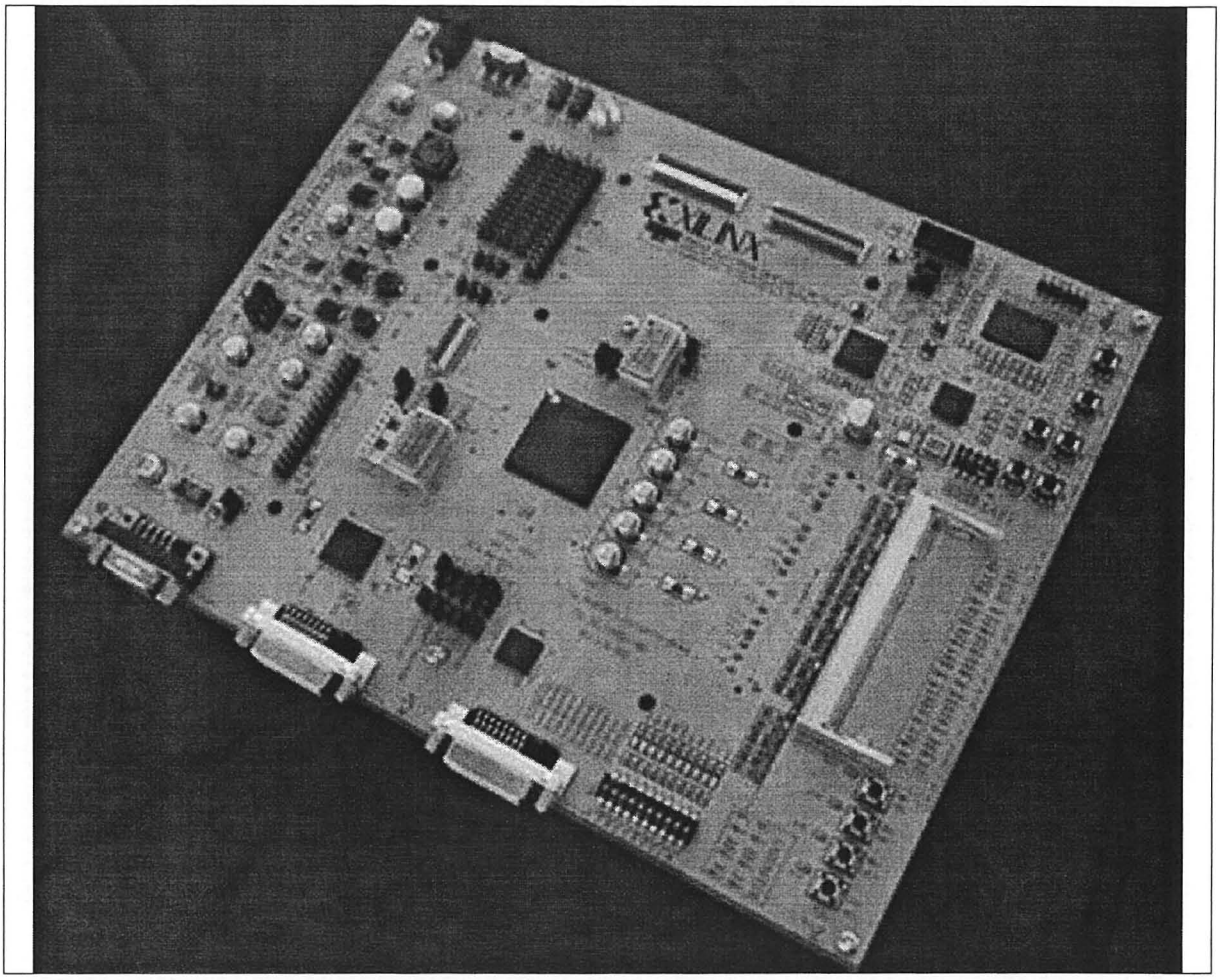

Figure 3.8 Carte modèle TB-3S-1500-IMG.

(Tiré de [26] HiTech Global. 2006)

Source : cette figure est tirée de la page web de HiTech Global. 2006, *Xilinx Spartan 3 III Board with DVI & FFC*, url : http://www.hitechglobal.com/ted/tb3s1500img.htm.

3.2.1.6.1 Les caractéristiques de la plateforme de développement TB-3S-1500-IMG:

Cette section décrit les caractéristiques de la carte TB-3S-1500-IMG et aussi ce que la trousse du produit contient à l'achat. La trousse de cette carte contient :

a. deux fentes mémoire DDR SDRAM SO-DIMM;
b. une mémoire ROM de configuration du FPGA;
c. une mémoire FLASH ROM de 32 Mbits (2M * 16bits)
d. une mémoire SDRAM de 64 Mbits (4M * 16bit)
e. un connecteur FFC/FPC pour LVDS/RSDS qui permettent d'optimiser le développement d'interfaces vidéo;
f. deux ports DVI (Tx/Rx) pour permettre les entrées/sorties d'images sources;
g. des modules d'expansions optionnels;
h. des boutons, des DEL et des interrupteurs;
i. un port parallèle;
j. un port sériel;
k. un connecteur JTAG;
l. un guide de l'utilisateur;
m. un guide de référence pour la mémoire DDR SDRAM 266Mbps.

Cette solution est intéressante et répond à tous les critères sauf un seul, son prix. Nous devons rejeter cette solution pour cette raison.

3.2.2 Les solutions composées de plusieurs pièces provenant de manufacturiers différents

Cette section fait la liste des solutions qui permettent de répondre aux besoins du projet tout en faisant en sorte que le coût d'achat soit le plus bas possible. Il est question en premier lieu des cartes mères et par la suite des cartes filles permettant d'ajouter un port DVI à ces

différentes cartes. Il est certain que ces solutions demandent un peu de développement pour l'adaptation des connecteurs, mais il est nécessaire de faire cette étude.

3.2.2.1 Les cartes mères:

La section suivante décrit les possibilités de cartes maîtresses qui peuvent êtres utilisées en intégrant une connexion DVI d'une tierce compagnie.

3.2.2.1.1 La plateforme XUPV2P de la compagnie Digilent:

Voici les informations de base pour cette plateforme très avantageuse pour son prix éducationnel ([12] Digilent Inc. 2006) :

A. Modèle : XUPV2P;
B. Prix éducationnel : 299$ US;
C. Prix industriel : 1599$ US;
D. Composante Xilinx utilisée: Virtex-2 Pro XC2VP30.

Voici aussi une image de la carte :

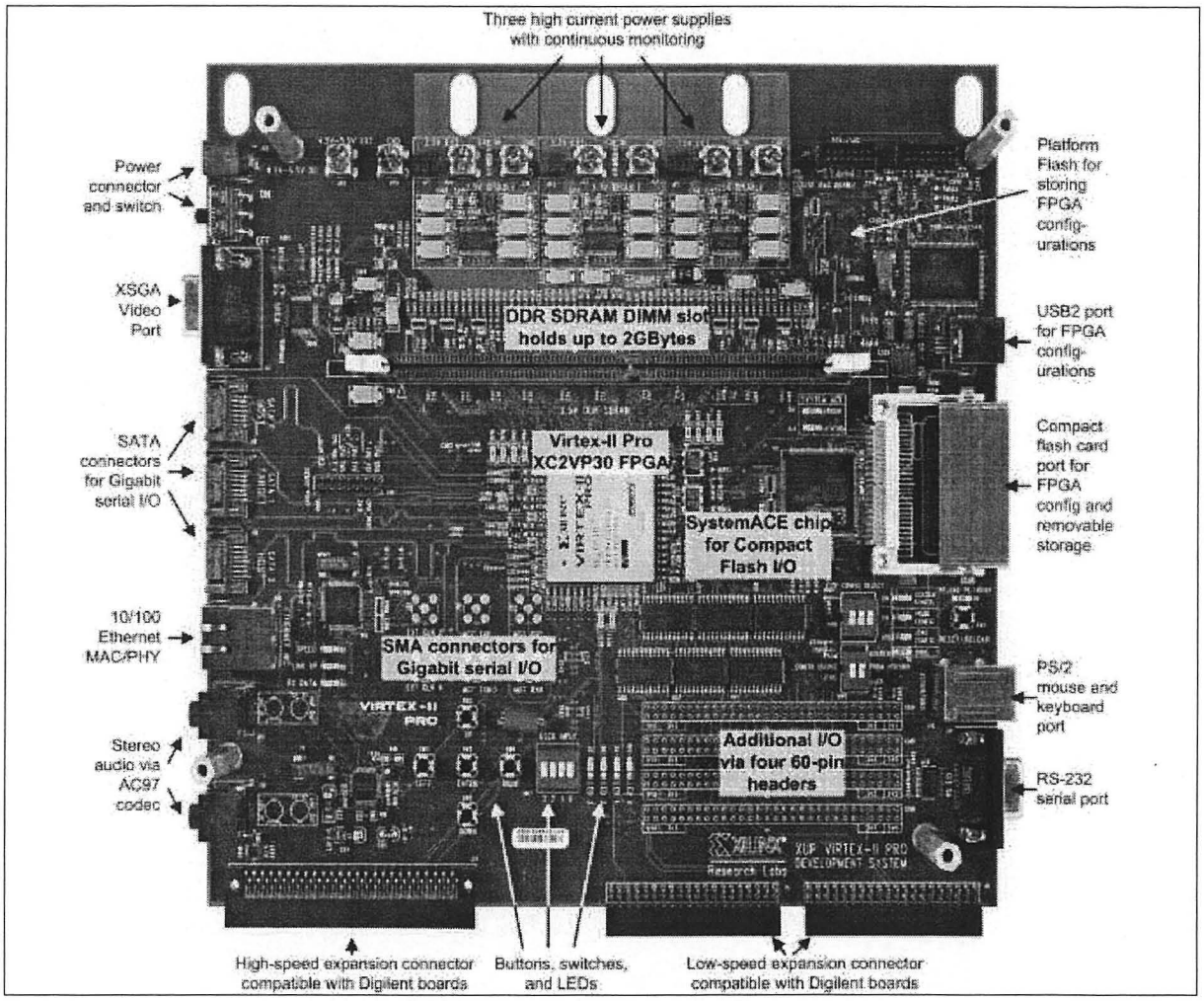

Figure 3.9 Carte modèle XUPV2P.

(Tiré de [12] Digilent Inc. 2006)

Source : cette figure est tirée de la page web de Digilent Inc. 2006, *Virtex-II Pro Development System*, url : http://www.digilentinc.com/Products/Detail.cfm?Prod=XUPV2P&Nav1=Products&Nav2=Programmable.

3.2.2.1.1.1 Les caractéristiques de la carte XUPV2P:

Cette carte comprend :

a. une mémoire DDR SDRAM DIMM de 2GB;
b. un port Ethernet 10/100;
c. un port USB 2.0;

d. une fente pour carte Compact Flash;
e. un port vidéo XSGA;
f. un Codec Audio;
g. trois ports SATA;
h. un port PS/2;
i. un port RS-232;
j. des connecteurs d'expansions.

Son prix est intéressant dans la mesure où il est possible d'avoir un prix éducationnel. Par contre, la carte est plutôt mal documentée sur le web. De plus, elle possède une technologie FPGA en fin de cycle commercial et elle risque d'être insuffisante en nombre de portes pour nos besoins. Pour ces raisons, nous devons mettre cette solution de côté.

3.2.2.1.2 La carte HW-SP305-US de la compagnie Xilinx :

La plateforme de développement HW-SP305-US est celle qui répond le plus à ce qu'on recherche. Voici les informations de base de cette solution ([60] Xilinx, 2005) :

A. Modèle : HW-SP305-US;
B. Prix : 495$ US;
C. Composante Xilinx utilisée: Spartan III XC3S1500-FG676-10.

Pour mieux visualiser la carte, voici une photographie de celle-ci :

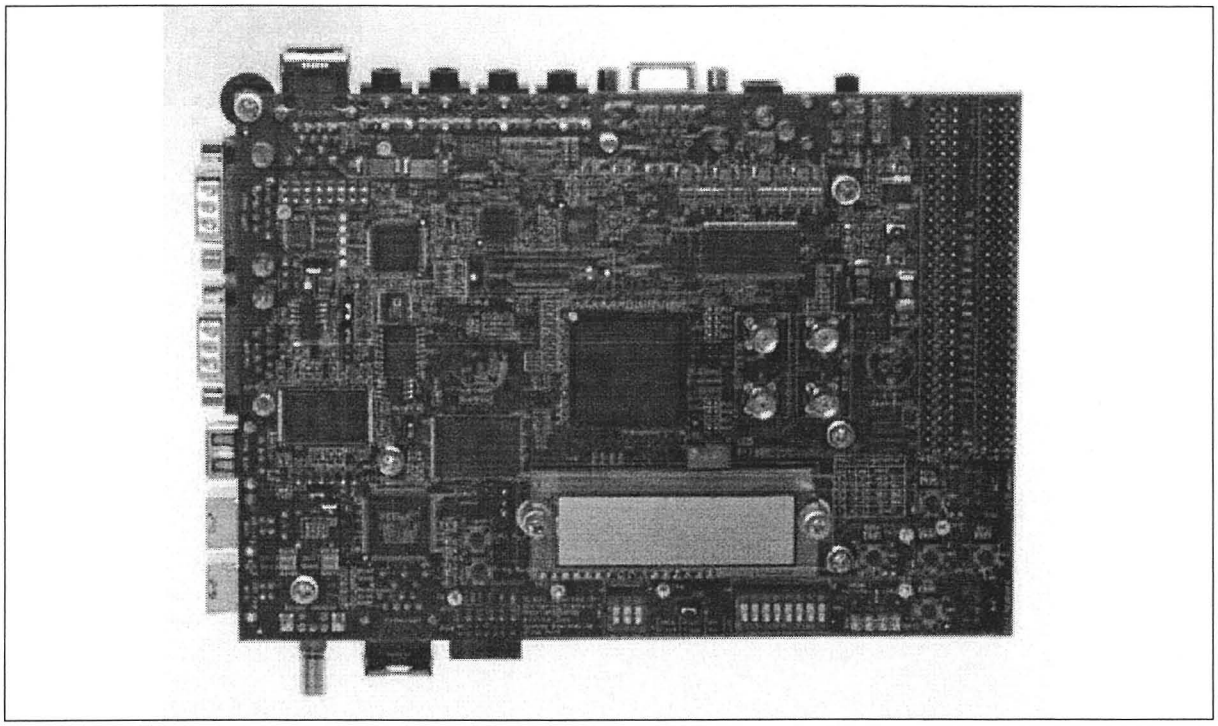

Figure 3.10 Carte modèle HW-SP305-US.

(Tiré de [60] Xilinx. 2005)

Source : cette figure est tirée de la page web de Xilinx. 2006, *Spartan-3 SP305 Development Board*, url : http://www.xilinx.com/xlnx/xebiz/designResources/ip_product_details.jsp?key=HW-SP305-US.

3.2.2.1.2.1 Les caractéristiques de la carte HW-SP305-US:

La carte SP305 comprend :

a. une horloge de 100 MHz externe et deux autres socles d'horloge;
b. une mémoire DDR SDRAM de 64 MB;
c. une mémoire ZBT SRAM de 9Mb;
d. une mémoire Flash linéaire de 8 MB;
e. une mémoire EEPROM de 4 Kb IIC;
f. quatre connecteurs SMA pour des horloges différentielles;
g. deux ports PS/2;

h. deux ports audio stéréo (entrée/sortie);
i. deux UART (« Universal Asynchronous Receiver-Transmitter ») pour les ports RS-232;
j. un port SPI;
k. un port CAN;
l. un port IIC;
m. trois ports USB Ports (un maitre, deux esclaves);
n. un connecteur JTAG PC4;
o. un port Ethernet RJ-45 (connecté sur composante SMSC MAC/PHY);
p. un port Ethernet RJ-45 (connecté sur composante Intel PHY);
q. un encodeur rotatif avec bouton poussoir/interrupteur;
r. plusieurs boutons, DEL et interrupteurs;
s. 32 broches d'expansion (32 simples ou 16 différentielles et 16 d'ordre générale);
t. un port DB 15 VGA (FMS3818krc Fairchild);
u. un écran LCD de 16 x 2 caractères.

Cette solution répond aux critères de prix, de technologie FPGA et de quantité de mémoire disponible. Elle possède un convertisseur vidéo VGA intégré et possède aussi des connecteurs d'expansion. Elle sera considérée pour une étude approfondie à la section 3.3.2 pour assurer la qualité du choix effectué.

3.2.2.1.3 La carte ADS-XLX-SP3-EVL1500 de la compagnie Avnet :

Voici les informations de base de cette carte ([3] Avnet Electronics Marketing, 2006) :

A. Modèle : ADS-XLX-SP3-EVL1500;
B. Prix : 749$ CAD selon courriel du fournisseur (499$ US selon Avnet);
C. Composantes Xilinx utilisées: Spartan III XC3S400-FG456 ou XC3S1000L-FG456 ou XC3S1500-FG456.

Voici une photographie de la carte pour aider à visualiser un peu plus ses possibilités :

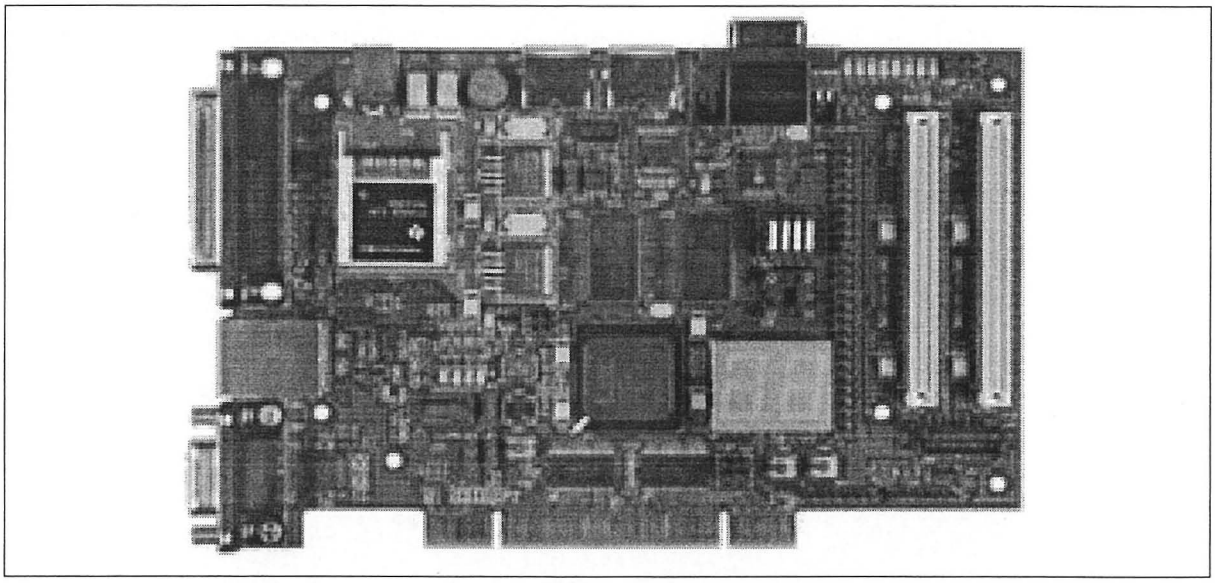

Figure 3.11 Carte modèle ADS-XLX-SP3-EVL1500.

(Tiré de [3] Avnet Electronics Marketing, 2006)

Source : cette figure est tirée de la page web d'Avnet Electronics Marketing, 2006, *Xilinx® Spartan™-3 Evaluation Kit*, url : http://www.em.avnet.com/evk/home/0,1719,RID%253D0%2526CID%253D7816%2526CCD%253DUSA%2526SID%253D4742%2526DID%253DDF2%2526SRT%253D1%2526LID%253D18806%2526PVW%253D%2526BID%253DDF2%2526CTP%253DEVK,00.html.

3.2.2.1.3.1 Les caractéristiques de la carte ADS-XLX-SP3-EVL1500 :

La carte ADS-XLX-SP3-EVL1500 comprend :

a. une mémoire PROM Flash de Xilinx pour la configuration du FPGA;
b. une horloge de 66 MHz et un socle pour un oscillateur supplémentaire;
c. un port JTAG équivalent au câble Parallèle III de Xilinx;
d. deux connecteurs d'expansion AvBus;
e. un connecteur de 50 broches pour accès facile aux entrées/sorties (incluant quatre paires LVDS);
f. un connecteur universel 32 bits PCI;
g. un port Ethernet 10/100;
h. un port DB15 VGA;

i. une console RS-232;
j. deux ports PS2;
k. une mémoire SRAM de 1MB;
l. une mémoire EEPROM série de 256kb;
m. quatre commutateurs;
n. deux boutons poussoirs;
o. huit DEL;
p. un afficheur double de 7-segments.

Cette solution est intéressante, mais comprend les mêmes caractéristiques de base que la carte SP305 décrite à la section 3.2.2.1.2 pour un coût plus élevé. Elle ne possède pas suffisamment de boutons de déverminages. La mémoire vidéo est insuffisante, nous devons laisser tomber cette option.

3.2.2.1.4 La carte ADS-XLX-SP3-DEV1500 de Silica (Avnet) :

Voici les informations de base de la carte ADS-XLX-SP3-DEV1500 ([43] Silica (Avnet), 2006) :

A. Modèle : ADS-XLX-SP3-DEV1500;
B. Prix : 850$ US;
C. Composantes Xilinx utilisées: Spartan III XC3S1500/2000-FG676.

Pour avoir une meilleure idée de la carte, voici une photographie de celle-ci :

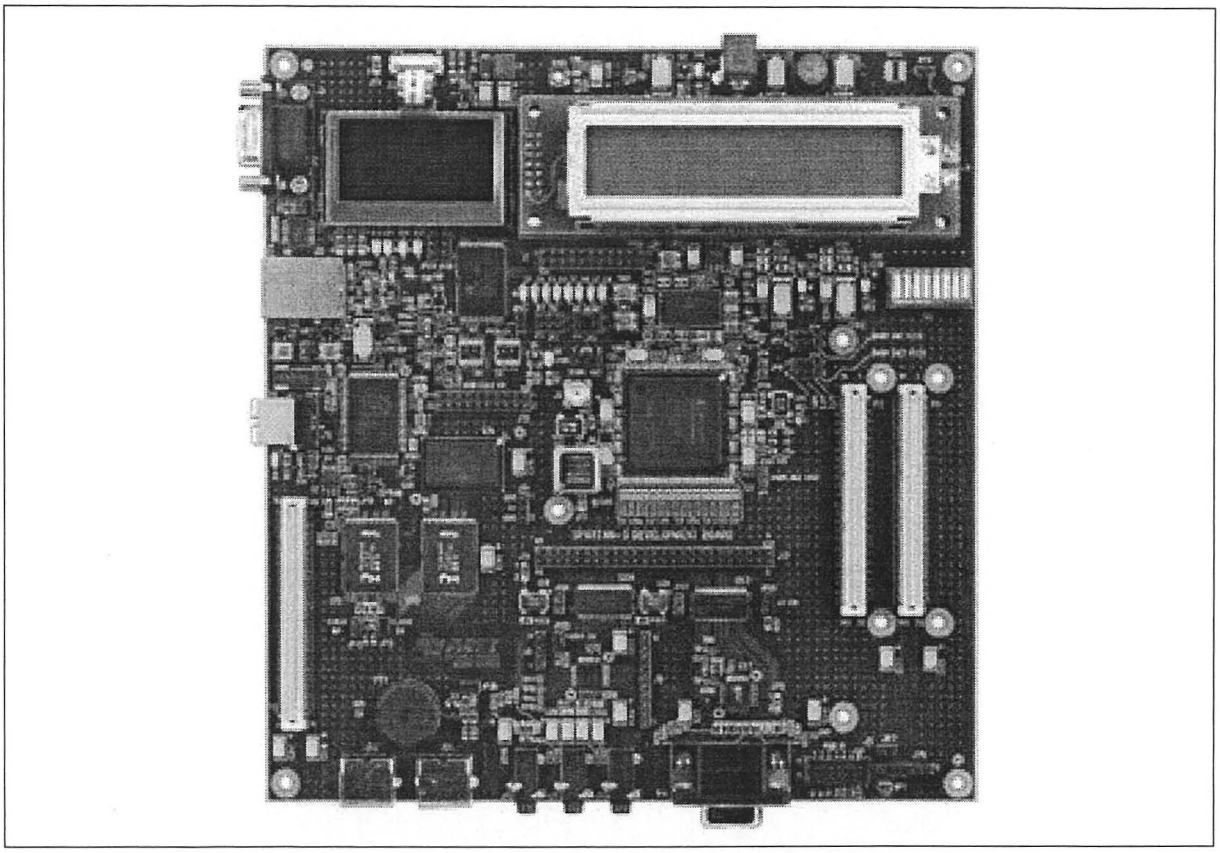

Figure 3.12 Carte modèle ADS-XLX-SP3-DEV1500.

(Tiré de [43] Silica (Avnet), 2006)

Source : cette figure est tirée de la page web de Silica (Avnet), 2006, *SP3 Development Kit*, url : http://www.silica.com/en/products/evaluationkits/SP3%20Development%20kit.html.

3.2.2.1.4.1 Les caractéristiques de la carte ADS-XLX-SP3-DEV1500 :

La carte ADS-XLX-SP3-DEV1500 comprend :

a. un afficheur LCD de 2 x 16 caractères;
b. un écran graphique de 128 x 64 pixels OSRAM OLED;
c. un port DB15 VGA avec le convertisseur videoADV7123 à 330 MHz max;
d. un Codec audio;

e. deux ports PS2;
f. huit commutateurs miniatures;
g. deux boutons poussoirs;
h. huit DEL;
i. une sonnette piézo-électrique;
j. trois ports d'expansion de 140 broches AvBus;
k. jusqu'à 30 paires LVDS;
l. un connecteur de 50 broches pour accès rapide aux entrées/sorties;
m. une mémoire DDR SDRAM de 32 MB;
n. une mémoire FLASH de 16 MB;
o. une mémoire SRAM de 2 MB;
p. un port RS-232;
q. un port Ethernet 10/100;
r. un port USB 2.0;
s. une mémoire PROM Flash de Xilinx pour la configuration du FPGA;
t. un port JTAG supportant le câble Parallèle IV de Xilinx;
u. une horloge de 100 MHz max avec option d'horloge différentielle à 125 MHz;
v. un guide de l'utilisateur et les schémas de câblage;
w. des exemples de code VHDL fonctionnels.

Cette carte est intéressante, mais elle possède des composantes qui sont inutiles pour le travail à effectuer. De plus, son prix est assez élevé. Nous devons donc mettre cette solution de coté.

3.2.2.1.5 La carte GR-XC3S-1500 de Gaisler Research

Voici les informations de base de la carte GR-XC3S-1500 ([28] HiTech Global, 2006 et [18] Gaisler Research, 2006) :

A. Modèle: GR-XC3S-1500 (Xilinx Spartan 3™ Leon Development Board);
B. Prix: 995$ US;

C. Composantes Xilinx utilisées: Spartan III XC3S1500-4.

Pour mieux visualiser la carte, voici une photo de celle-ci :

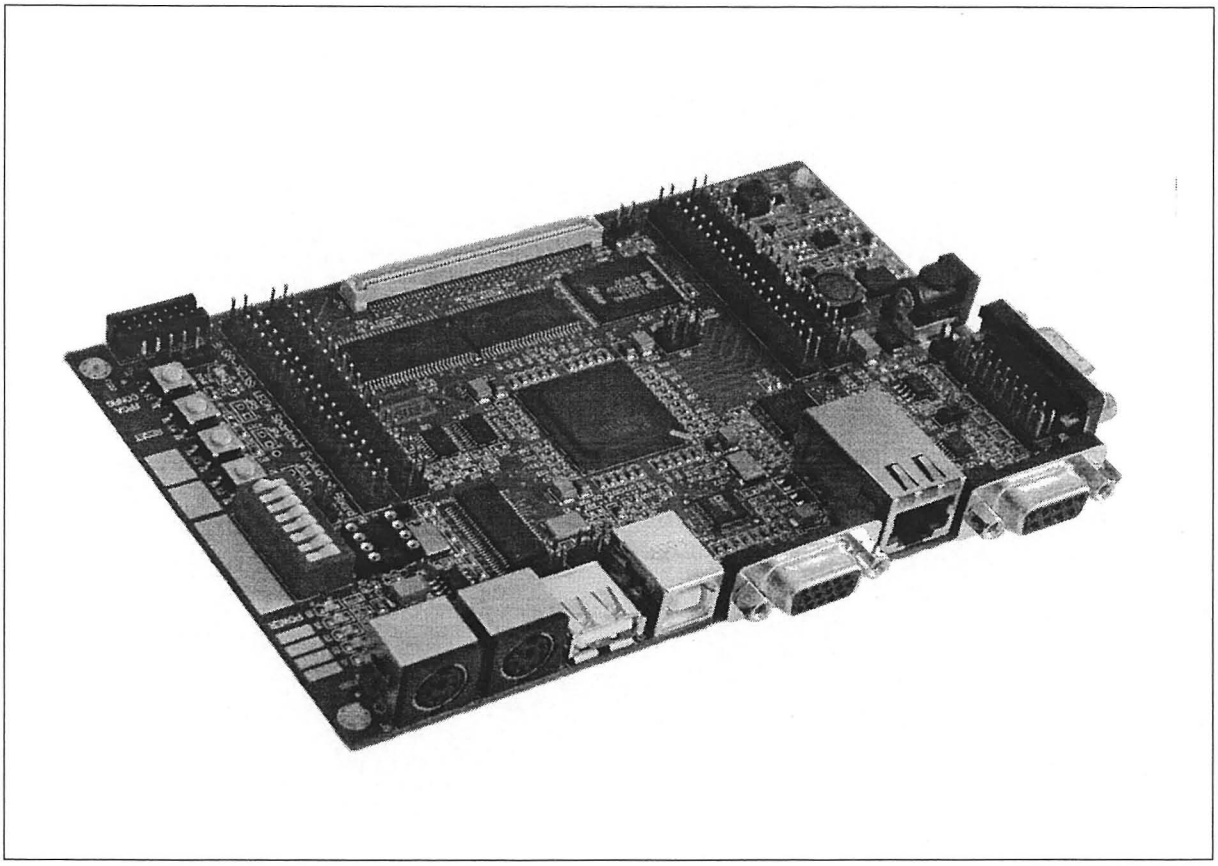

Figure 3.13 Carte modèle GR-XC3S-1500.

(Tiré de [28] HiTech Global. 2006)

Source : cette figure est tirée de la page web de HiTech Global, 2006, *Xilinx Spartan 3™ Development Board*, url : http://hitechglobal.com/Boards/GR-S31500.htm.

3.2.2.1.5.1 Les caractéristiques de la carte GR-XC3S-1500:

La plateforme de développement GR-XC3S-1500 de Gaisler Research contient :

a. une mémoire PROM Flash de Xilinx pour la configuration du FPGA;
b. deux ports série haut débit;
c. une mémoire PROM flash de 8 MB (8M x 8);
d. une mémoire SDRAM PC133 de 64 MB (16M x 32);
e. un port Ethernet 10/100 PHY (LTX971A), incluant le câble;
f. un port VGA avec convertisseur vidéo 24 bits (ADV7125-50, 330 Mhz);
g. deux ports USB-2.0 PHY (CY7C68000), incluant les câbles;
h. deux ports RS232 (1Mbaud), incluant les câbles;
i. deux ports PS/2;
j. un connecteur d'expansion de 120 broches (AMP-177-984-5) pour usage général ou pour interface LVDS;
k. programmation JTAG incluant le câble;
l. une horloge de 25MHz, une autre de 50MHz et un socle pour une autre supplémentaire;
m. un CD-ROM avec la documentation et les schémas de câblage;
n. les fichiers précompilés des processeurs LEON2 et LEON3.

Cette option est intéressante dans la mesure où le projet peut demander un processeur. Mais présentement, les besoins ne sont pas à ce niveau, donc pour des raisons budgétaires, encore une fois, nous devons laisser cette option de coté.

3.2.2.2 Cartes filles :

Voici la section qui identifie les possibilités d'interfacer un port DVI sur une des cartes mères présentées précédemment. Nous avons surtout remarqué la carte SP305 de la section

3.2.2.1.2. Voici donc les options de cartes filles avec port DVI qui sont disponibles au moment d'écrire ces lignes.

3.2.2.2.1 La carte DVI_1x1 de HAPS :

Voici les informations de base de la carte DVI_1 x 1 ([21] Hardi Electronics, 2006) :

A. Modèle : DVI_1 x 1;
B. Prix : non disponible.

Voici la photo de cette carte pour donner une idée :

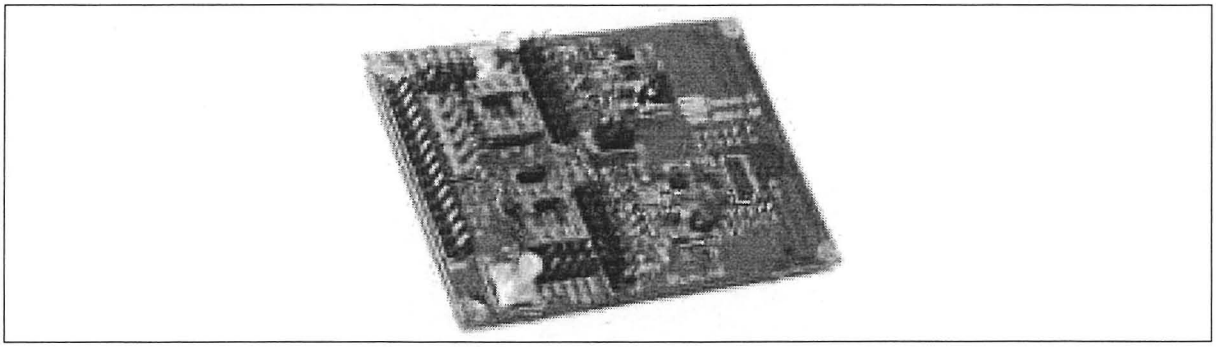

Figure 3.14 Carte modèle : DVI_1x1.
(Tiré de [21] Hardi Electronics. 2006)

Source : cette figure est tirée de la page web de Hardi Electronics, 2006, *DVI_1x1*, url : http://www.hardi.com/haps/dvi_1x1.htm.

3.2.2.2.1.1 Les caractéristiques de la carte DVI_1 x 1:

La carte DVI_1 x 1 contient

a. un système qui régénère l'horloge DVI à partir des signaux d'entrée série et qui retourne ce signal d'horloge vers la carte mère;
b. un sélecteur d'horloge : l'horloge du canal de sortie peut être fournie soit par un oscillateur programmable sur la carte, soit par la régénération à partir du flux des signaux d'entrée ou soit directement en provenance de la carte mère;
c. une horloge et des composantes DVI contrôlables à partir de la carte mère (HAPS);

d. des broches libres sur le connecteur d'expansion HAPS;
e. deux ports RS-232 sur des connecteurs en peigne;
f. deux ports DVI (entrée/sortie) sur des connecteurs en peigne;
g. une fréquence de pixel garantie de 100 Mpixels/s;

Il est à noter que la carte peut être alimentée par une source de 2.5 ou de 3.3 V et que le connecteur à utiliser pour l'interfaçage avec la carte mère est de modèle SAMTEC QTH-060-01-L-D-A.

Aucune information n'a été reçue suite à un courriel pour connaître le coût d'achat de cette carte. Nous devons donc laisser tomber cette solution.

3.2.2.2.2 La carte TB-SUB-DVI d'Inrevium :

Voici les informations de base de la carte TB-SUB-DVI ([27] HiTech Global, 2006) :

A. Modèle : TB-SUB-DVI (Module Tx);
B. Prix : 498$ US (Interface DVI Tx/Rx).

Pour mieux visualiser la carte, référez-vous à la Figure 3.6 de la page 51.

3.2.2.2.2.1 Caractéristiques de la carte TB-SUB-DVI:

La carte TB-SUB-DVI contient un port DVI permettant une résolution compatible UXGA. Le connecteur à utiliser pour l'interfacer avec une carte mère est le modèle: JAE Electronics WR-100S-VF-N1 (100 broches). Cette solution est très chère pour ce qu'elle offre, elle est donc mise de coté elle aussi.

3.3 Analyse approfondie des solutions retenues

Nous retenons quelques possibilités, la carte ML401 couplée avec la carte d'extension VIODC ou la carte SP305 qui ressemble beaucoup à la carte ML401, mais avec un Spartan 3

plutôt qu'un Virtex 4, toujours couplée avec la carte fille VIODC (si possible). Plusieurs problèmes sont répertoriés sur la carte ML401, mais aucun problème connu à ce jour sur la carte SP305. Voyons ce qui en est avant de prendre une décision.

3.3.1 Rapport de problèmes sur la plateforme ML401

Les cartes de développement ML40x possèdent une liste impressionnante de problèmes répertoriés dans un document internet ([65] Xilinx, 2006) et qui pourraient causer des difficultés lors du développement du contrôleur vidéo. Voici donc quelques-uns de ces problèmes ainsi que les implications que ces derniers pourraient engendrer sur le développement de notre AVN.

3.3.1.1 La polarité du socle de la pile pour la clé d'encryptions est inversée

Solution: pour les ML401, 402 et 403, installer une pile CR1025 plus petite et à l'envers. Note: des diodes bloquent le FPGA, il ne voit donc pas la tension inversée de la pile.

Impact sur le développement du contrôleur vidéo : dans le cas de la première phase de développement du contrôleur vidéo, il n'y a pas d'impact majeur à ce problème. Cependant, si la carte est utilisée pour un autre projet qui demande l'utilisation de l'encryption des données de configuration, cela peut causer des problèmes. Il est préférable de faire l'achat d'une carte qui ne possède pas ce genre d'obstacle si c'est possible.

3.3.1.2 Les broches de mise à la terre pour la composante ADV7125

Sur les ML401, 402 et 403, certaines broches de mise à la terre du DAC ADV7125 (U13) sont connectées sur la mise à la terre numérique plutôt que celle analogique, ce qui occasionne une augmentation du bruit vidéo VGA.

Solution: Les broches 1,2, 14, 15, 39 et 40 de U13 (le convertisseur vidéo) doivent être connectées à la mise à la terre analogique (broche 2 de FB6).

Impact sur le développement du contrôleur vidéo : Le risque de rencontrer des problèmes d'affichage augmente considérablement. On ne peut accepter d'ajouter un doute sur les composantes physiques en plus de ceux qui sont engendrés lors du développement d'un cœur en VHDL. Ce problème est directement relié à notre projet et c'est celui qui cause le plus d'embarras.

3.3.1.3 Difficulté de connexion avec les logiciels XMD et ChipScope

Sur le ML401 avec le Virtex 4 LX25, les utilisateurs peuvent avoir certaines difficultés à s'y connecter à l'aide des logiciels XMD et ChipScope à cause d'une erreur silicium.

Solution: La réponse AR #20060 sur le site de Xilinx ([62] Xilinx, 2006) spécifie que la solution dépend de la version du logiciel ISE utilisée, il faut faire la mise à jour de certains fichiers. Le modèle de référence du processeur MicroBlaze de la plateforme ML401 possède déjà la correction à apporter pour que la connexion se fasse correctement.

Impact sur le développement du contrôleur vidéo : dans l'éventualité qu'on utilise les logiciels XMD et ChipScope pour le déverminage, la perte de temps engendrée par ces mises à jour est à prendre en considération pour le choix de notre plateforme, malgré l'indication qui mentionne que les corrections sont déjà apportées au modèle de référence du processeur MicroBlaze.

3.3.1.4 Programmation intermittente par le CPLD au démarrage:

En mode de configuration CPLD + flash linéaire, le design de référence du CPLD ne réussit pas toujours à configurer le FPGA à la mise sous tension. Cependant, le FPGA se programme toujours correctement après avoir appuyé sur le bouton PROG.

Solution: Le design de référence du CPLD a été mis à jour et le fichier *.jed a été mis à jour sur le site de référence de la carte ML40x.

Impact sur le développement du contrôleur vidéo : On doit reprogrammer le CPLD avec le nouveau fichier *.JED, ce qui ne cause pas de problème à long terme.

3.3.1.5 Errata silicium du Virtex 4L X 25:

Plusieurs erreurs de fonctionnalités internes du FPGA sont rapportées, certaines sont réparables facilement, d'autres moins. Voici les problèmes qui sont susceptibles de rendre le travail plus difficile.

3.3.1.5.1 Erreur avec le FIFO16

Le module FIFO16 ne génère pas correctement les signaux fanions ALMOST EMPTY, EMPTY, ALMOST FULL et FULL après les séquences d'opérations suivantes:

A. Une lecture ou une écriture atteint la valeur de décision des fanions ALMOST_EMPTY_OFFSET ou ALMOST_FULL_OFFSET;
B. Une simple lecture ou écriture est exécutée suivi simultanément d'une lecture ou d'une écriture alors que les fronts d'horloges de lecture et d'écriture sont très près les uns des autres.

Une corruption des données peut être provoquée à cause de l'échec des fanions, même si ceux-ci ne sont pas utilisés. Ce problème ne se produit pas si la lecture et l'écriture ne sont pas simultanées.

Solution: les utilisateurs qui doivent utiliser des lectures et écritures simultanées peuvent télécharger des macros, mais les performances décrites dans les spécifications nominales ne sont pas garanties. Voir la réponse de Xilinx AR #22462 pour plus de détails ([64] Xilinx 2006).

Impact sur le développement du contrôleur vidéo : l'utilisation de Fifo dans notre contrôleur peut être compromise ou peut nous causer bien des désagréments. Les registres

Fifo sont très importants dans le genre de système qu'on doit développer et on ne peut se permettre de compromettre l'intégrité des données sur un système avionique.

3.3.1.5.2 L'instruction JTAG INTEST n'est pas supportée sur la machine à états du port JTAG

L'instruction JTAG INTEST mentionnée dans le guide de configuration du Virtex-4 n'est pas supportée.

Solution: pas de solution.

Impact sur le développement du contrôleur vidéo : il nous serait difficile de faire des tests et de la vérification sur le circuit interne du FPGA. Il est plus sage de choisir une autre technologie de FPGA pour des applications en avionique, surtout si on utilise cette composante en production.

3.3.1.5.3 Les problèmes rencontrés avec les modules DCM

L'attribut CLKOUT_PHASE_SHIFT ne peut pas être assigné à la valeur VARIABLE_CENTER car cette valeur n'est pas supportée.

Solution: aucune solution.

Lorsque les sorties CLKFX et/ou CLKFX180 sont utilisées comme unique sortie du DCM, il peut y avoir un problème avec la génération du signal LOCKED lorsque la fréquence d'entrée sur le signal CLKIN est en dehors de l'intervalle de temps spécifié par CLKIN_FREQ_DLL_(HF or LF)_(MS or MR)_MIN/MAX.

Solution: utiliser les réponses #20529 et #23624 pour générer adéquatement le signal LOCKED ([66] Xilinx, 2006 et [67] Xilinx, 2006).

Pour générer des horloges identiques de haute performance avec le moins de déphasage ou de biais (« skew ») sur l'ensemble du FPGA, il est préférable d'utiliser les caractéristiques ChipSync™.

Solution: suivre les indications de la réponse #20529 de Xilinx pour atteindre une spécification de ±300 ps sur l'attribut CLKIN_CLKFB_PHASE.

Impact sur le développement du contrôleur vidéo : beaucoup de perte de temps et d'incertitudes sur le bon fonctionnement du FPGA.

De plus, certaines erreurs sont plutôt dans les caractéristiques nominales du Virtex de Xilinx qui sont erronées. Voir le fichier d'errata pour plus de détails ([68] Xilinx, 2006).

3.3.1.6 Résumé de ce qui peut affecter notre travail:

Beaucoup des problèmes de la carte ML401 peuvent affecter les performances et aussi l'avancement du travail car ils se trouvent directement reliés à l'application qu'on veut développer. Le cas des mises à la terre sur le convertisseur vidéo peut nous affecter directement puisque nous ne pourrons jamais être certains de ce qui sort sur le port VGA. Nous pouvons contourner le problème en utilisant la carte fille qui utilise un port VGA indépendant, mais ce n'est sûrement pas une solution envisageable. Les problèmes de Fifos peuvent directement affecter les performances de nos propres Fifos dans le module VGA qu'on cherche à implémenter. Les problèmes de JTAG pourraient affecter le déverminage et aussi les vérifications éventuelles.

Les erratas dans le silicium du Virtex dérangent un peu plus. Cela risque de nous obliger à modifier légèrement le design de base pour l'adapter au Virtex, mais quand viendra le temps de remplacer le Virtex par une autre composante, nous serons obligé de modifier le code en conséquence, ce qui n'est pas souhaitable.

3.3.2 L'analyse de la solution SP305

Selon Jim Burnham, représentant pour les plates-formes de développement chez Xilinx (contacté par courriel), la carte SP305 est fabriquée avec les mêmes spécifications au niveau de ses connecteurs d'extension que celles de la carte ML401. Toutes les tensions et les entrées/sorties sont respectées afin de garder les spécifications identiques entre les plates-formes de développement. J'ai d'ailleurs vérifié sur les PCB de la carte SP305 et de la carte d'extension VIDEO VIODC et tout concorde effectivement. Les connecteurs J5, J4, J3 et J6 du SP305 concordent respectivement avec les connecteurs J8, J5, J4 et J9 sur la carte d'extension VIDEO ([55] 2005 et [70] Xilinx, 2006).

La carte de développement SP305 ne semble pas avoir d'errata sur le site de Xilinx ni sur le web en général. Par contre, le Spartan 3 en possède certains. Ils ont été pour la plupart corrigés avec les dernières versions du silicium, mais en cas de silicium moins récents, il est important d'y jeter un coup d'œil. Voici donc ce qui a été découvert sur le SPARTAN 3.

3.3.2.1 Les problèmes avec le Spartan 3

La plupart des problèmes surviennent soit à la mise sous tension (power-up) du FPGA ou suite à sa configuration. Dans tout les cas, il y a possibilité de contourner ou régler le problème. Ces anomalies ont été corrigées dans les versions plus récentes identifiées par les codes AGQ et EGQ (ces codes font partie du numéro de modèle). La Carte SP305 possède un EGQ525.

Les versions antérieures au code EGQ 0532 (incluant AGQ) ne permettent pas de faire une relecture du code du FPGA. Toutes les nouvelles versions ont été corrigées. Ce serait le seul problème connu à ce jour pour la carte SP305.

3.3.2.2 Ce qui peut affecter notre travail:

Il n'y a pas de problèmes majeurs qui semblent affecter le bon déroulement de notre travail. Le Spartan 3 a été lancé initialement en 2003 (plutôt que 2004 pour le Virtex 4), ce qui permet de penser que la plupart des erreurs ont été corrigées depuis. Le dernier document des Gerber de la carte SP305 date de 2005 comparativement à 2004 pour la ML401.

Note: le circuit imprimé et le schématique montrent bien une mise à la terre séparée du reste du circuit pour la partie du port VGA et du ADV7125 sur la carte SP305 ([55] Xilinx, 2005), ce qui n'est pas le cas pour la LM401 ([53] Xilinx, 2004).

3.3.3 L'analyse de la solution avec la carte-fille VIODC:

La carte d'extension vidéo ne semble pas posséder d'erreur majeure connue. Le XC2VP7 ne possède qu'un seul problème et ne nous concerne pas car ce problème affecte seulement son alimentation ([51] Xilinx 2003), ce qui a sûrement été compensé sur la carte lors de sa fabrication.

La carte VIODC accède au port DVI par l'intermédiaire du XC2VP7. Il ne semble pas évident à la première vue en regardant le schéma que ce soit bien un XC2VP7, mais en regardant bien les symboles, le modèle du FPGA de la carte est bien XC2VP7. Il aurait été préférable de n'avoir aucun intermédiaire entre le port DVI et la carte mère, mais le Virtex pro permettant plus de bande passante, on peut surement atteindre des vitesses de pointe plus élevées que prévu.

Il pourrait être intéressant d'utiliser la carte VIODC seule sans aucune autre carte et avec le PPC intégré au Virtex pro, mais avec 11 088 cellules logiques (1LC= 1LUT(4input) + 1FF + Carry logic), il n'est pas certain qu'on puisse faire tout ce qu'on prévoit, surtout que cette carte n'inclus pas de mémoire externe au FPGA XC2VP7.

3.4 Conclusion sur le choix de la plateforme de développement

Pour bien comprendre le choix qui a été fait, voici un tableau récapitulatif des solutions regardées :

Tableau 3.1

Résumé des recherches de plateformes de prototypage avec leurs principales force et faiblesses

Plateforme étudée	Prix (US)	Supplément carte fille	Connecteurs d'expansion	FPGA	Documentation
Sendero (DVI)	895$	aucun	PCI Express + Carte fille	Cyclone II	Fournie à l'achat
ML40x + VIODC (DVI +VGA)	1495$	Carte VIODC	Non disponnible	XC4VSX35 + Virtex II Pro	Disponible WEB
ML40x (VGA)	595$	Aucun	32 broches d'expansion	XC4VSX35	Disponible WEB
TB-V2P-OPT -2G (VGA + DVI)	1100$ (+495$)	Carte TB-SUB-DVI (+495$)	Non disponnible	Virtex-II Pro XC2VP4	Non mentionné
TB-3S-1600E-IMG (DVI)	1395$ (+495$)	Carte TB-SUB-DVI (+495$)	Non disponnible	Spartan III XC3S1600E	Non mentionné
TB-3S-1500-IMG (2 x DVI)	2000$	Aucun	Connecteurs génériques	Spartan III (XC3S1500-ou XC3S2000	Fournit à l'achat
XUPV2P (VGA)	299$	Aucun	4 x 60 broches	Virtex-2 Pro XC2VP30.	Mal documenté
HW-SP305-US (VGA)	495$	Aucun	32 broches d'expansion	Spartan III XC3S1500	Disponnible WEB
ADS-XLX-SP3-EVL1500 (VGA)	499$	Aucun	2 connecteurs AvBus (50 broches)	Spartan III XC3S400 ou XC3S1000L ou XC3S1500	Non mentionné
ADS-XLX-SP3-DEV1500 (VGA)	850$	Aucun	3 connecteurs AvBus (140 broches)	Spartan III XC3S1500/2000	Fournit à l'achat
GR-XC3S-1500 (VGA)	995$	Aucun	2 x 120 broches	Spartan III XC3S1500	Fournit à l'achat

La plateforme qui retient le plus notre attention est la carte de modèle : HW-SP305-US fabriquée par Xilinx et dont le prix est de 495$US. Nous choisissons cette carte car elle répond aux critères de départ. En effet, elle possède :

a. le prix raisonnable de 495 $US, ce qui en fait une des moins chères;

b. une sortie Vidéo avec une connexion VGA possédant un minimum de 8 bits par pixel (8 :8 :8) ainsi qu'une possibilité de connexion avec la carte VIODC pour une connectivité DVI;
c. plusieurs connecteurs d'entrées/sorties configurables (user I/O) qui permettent une évolution du système en cas de besoins;
d. un FPGA avec une grande quantité de portes de la Série Spartan III de Xilinx (pour son bas prix et aussi parce que son environnement de développement est bien connu);
e. un certain nombre de DEL et d'interrupteurs pour faciliter le déverminage;
f. une documentation complète et détaillée facilement accessible.

De plus, la compatibilité avec la carte fille VIODC (Video Input Output Daughter Card) de Xilinx a été vérifiée, ce qui permet d'ajouter des fonctionnalités DVI sur cette carte si les besoins se présentent (modèle HW-XGI-VIDEO-US au prix de 995$US). Aucun problème majeur ne nous empêche de croire que la carte SP305 est la meilleure solution pour nous.

Ce qui nous intéresse sur la carte SP305, c'est aussi sa mémoire ZBT qui permet un accès mémoire rapide et aussi la capacité de la mémoire PROM flash pour la programmation du FPGA. En effet, la mémoire PROM xcf32p fabriquée par Xilinx nous permet non seulement de configurer le FPGA, mais aussi d'emmagasiner nos données d'images lorsque la carte n'est pas alimentée.

Cette carte possède aussi un affichage LCD, plusieurs DEL et plusieurs interrupteurs qui permettrons un interfaçage intéressant avec les différentes fonctions du projet.

CHAPITRE 4

SPÉCIFICATIONS RETENUES POUR NOTRE AFFICHEUR VIDÉO NUMÉRIQUE

Ce chapitre décrit en détail les spécifications retenues qui sont utilisées afin d'implémenter un afficheur vidéo numérique répondant aux besoins de base de CMC Électronique. Il sera question uniquement des points retenus puisque les raisons qui nous ont poussés à faire ces choix ont été discutées dans les chapitres précédents.

4.1 Spécifications fonctionnelles

Suite à l'analyse faite sur les différents cœurs existants au CHAPITRE 2, les spécifications fonctionnelles finales du circuit implémenté en VHDL dans le FPGA ont été déterminées. Cette section les décrit en détail. Le contrôleur vidéo numérique comprend :

a. une sortie VGA (RGB, utilisées présentement chez CMC) qui permet aussi de s'adapter dans l'éventualité d'une migration vers un port DVI;
b. une résolution d'affichage allant jusqu'à 1024 x 768 et un taux de rafraichissement de 60 Hz en mode 24 bits par couleur;
c. une mémoire vidéo externe au FPGA pour l'entreposage des images, cette mémoire possède une bande passante élevé et fonctionne avec Pipeline (mémoire ZBT);
d. une possibilité d'utiliser les modes d'affichage 8 bits avec table de couleurs ou 24 bits;
e. deux bus Wishbone : un pour le processeur host (ou bootloader) et un pour la mémoire vidéo, ce qui permet un interfaçage rapide et une optimisation de la bande passante de la mémoire vidéo;
f. deux banques d'image, ce qui permet de faire un changement rapide de l'image affichée par un changement de la banque d'image sélectionnée sans provoquer de déformation d'image;
g. un mode d'économie d'énergie activé de manière logiciel;

h. un module de chargement (bootloader) permettant au minimum de charger les images en mémoire vidéo et de configurer le module d'AVN au démarrage.

4.2 Spécifications Physiques

Cette section décrit en détail les caractéristiques physiques du projet final. Ces spécifications sont tirées de l'analyse des différentes plateformes de développement disponibles sur le marché décrite au CHAPITRE 3. Les caractéristiques physiques mentionnées ici sont celles utilisées pour le projet uniquement et ne comprennent pas toutes les caractéristiques de la carte de développement SP305. Pour plus de détail sur cette carte, veuillez vous référer à ([58] Xilinx, 2005). Voici la liste de ces caractéristiques physiques :

A. La plateforme de développement utilisée est la carte de modèle HW-SP305-US fabriquée par la compagnie Xilinx;
B. La composante programmable utilisée est le Spartan III de modèle XC3S1500-FG676-10C fabriqué par la compagnie Xilinx;
C. L'horloge physique sur la carte est cadencée à 100 MHz
D. La configuration du FPGA s'effectue au moyen d'une mémoire PROM flash de modèle XCF32P en mode « FPGA Master Serial »
E. L'utilisation de la mémoire PROM flash permet aussi d'emmagasiner les données d'images dans l'espace restant avant de les charger en mémoire vidéo.
F. Une mémoire vidéo de 8 Mo de type ZBT (Zero Bus Turnaround) à la base. Les broches de cette mémoire permettent une expansion de la mémoire jusqu'à 32 Mo théoriquement.
G. Les composantes physiques sur la plateforme sont alimentées à 3.3 VDC
H. La programmation de la PROM flash se fait par une communication JTAG à l'aide du logiciel Impact de la suite ISE 8.2 de la compagnie Xilinx et du câble de programmation de modèle : Parallel Cable IV (DLC7).
I. Le convertisseur VGA de modèle ADV7125KST50 possède 24 bits d'entrées et permet une fréquence d'horloge pixel maximale de 50 MHz.

J. Des connecteurs d'expansion sont disponibles dans l'éventualité d'ajout de fonctionnalités plus complexes. Par exemple, on peut ajouter un port DVI.

4.3 Le guide à utiliser pour un code VHDL bien écrit

Le code écrit en VHDL doit être régit par certaines contraintes de mise en page, de syntaxe et de contenu afin de faciliter la relecture du code, sa compréhension et sa maintenance. Ceci n'est pas un guide didactique du langage, mais bien un guide qui permet de bien mettre en forme le texte pour ainsi mieux comprendre ce qu'il décrit. Cette section se divise en trois sous-section afin facilité la lecture.

4.3.1 Le guide des déclarations et de la syntaxe du code

Plusieurs règles de syntaxe permettent de rendre la lecture et la compréhension du code plus aisées. Les voici :

A. Le code doit respecter la syntaxe standard de 1993 (Compatible VHDL-93);
B. Chaque fichier possède le même nom que l'« entity » qu'il contient.
C. Les noms des signaux, des variables, des fonctions, des types, des entités etc. doivent décrire la fonction et non le type, ils doivent avoir un sens complet. Les noms comme : « signal1 » ou « toto2 » sont à proscrire;
D. Un suffixe descriptif doit être ajouté à la fin du nom pour identifier le genre de l'objet :

- _s pour les genres « signal »;
- _i pour les genres « in »;
- _o pour les genres « out »;
- _io pour les genres « inout »;
- _v pour les genres « variable »;
- _f pour les genres « file »;
- _c pour les genres « constant »;
- _a pour les genres « alias »;

- _t pour les genres « type »;
- _p pour les genres « pakage »;
- _r pour les signaux dans un « record ».
- _lib pour les noms de librairie.
- _a0, _a1, _a2 etc. pour identifier une architecture (il peut y en avoir plusieurs pour une seule entité).

E. Tous les signaux actifs bas (polarité négative) commencent par le préfix « n_ ». Par exemple, le nom « n_reset_s » est très bien pour un signal de réinitialisation actif bas;
F. Tous les bus ou groupes de signaux sont notés en petit boutiste (« little endian ») et écris avec le bit le plus significatif à gauche et le bit le moins significatif à droite;
G. Toutes les instanciations d'un « component » doivent posséder le même nom que ce « component » suivit par un index pour identifier chaque instance. Par exemple : nom0 et nom1 sont des noms d'instance acceptés;
H. Les instanciations des component se font par emplacement et non explicitement;
I. Lorsque c'est possible, il est préférable de grouper les signaux en définissant un type.

4.3.2 Le guide de la bonne mise en forme du code

La manière de mettre en page le texte contribue aussi à la lisibilité du code. Voici les règles à suivre :

A. Tous les fichiers possèdent une seule « entity » suivit de son « architecture »;
B. L'indentation se fait par des tabulations de 2 caractères d'espace afin de limiter la perte d'espace inutile;
C. Favoriser la réutilisation des fonctions et des blocs de logique lorsque c'est possible. Un code modulaire est toujours apprécié.
D. Essayer de garder un plan de l'arborescence des fichiers;
E. Utiliser une maximum de deux ou trois « process » par architecture afin de minimiser les fonctions en « spaghetti »;
F. La déclaration des signaux d'une entité doit être faite dans l'ordre : les « in » pour débuter, les « inout » ensuite et les « out » pour terminer. Cependant, il est souvent

plus facile de comprendre les relations entre les signaux lorsqu'ils sont groupés selon leur rôle et leur fonction. Il est même recommandé de placer un commentaire entre chaque groupe de déclaration pour exprimer l'appartenance de chaque groupe;

G. Il est impératif d'avoir une seule déclaration de « signal », de « type », de « variable » etc. par ligne afin de permettre l'ajout de commentaire pour chacune d'elles (voir aussi le point E de la section 4.3.3);
H. Les commentaires doivent être bien indentés et bien alignés pour facilité la lecture;
I. Utiliser des recettes éprouvées de syntaxe qui permettent une synthèse prédéterminée du code afin d'éviter la perte de temps et aussi pour s'assurer d'une fonctionnalité optimale du circuit inféré. Voici quelques exemples :
    - La forme d'un processus (« process ») de réinitialisation qui s'active de manière asynchrone et que se relâche de manière synchrone;
    - La forme d'un processus synchrone;
    - La forme d'un processus synchrone avec réinitialisation asynchrone;
    - La forme d'un processus synchrone avec réinitialisation synchrone;
    - La forme d'un processus qui infère un blocRAM.

4.3.3 Le guide pour bien commenter le code

Pour s'assurer de la bonne documentation du code VHDL, il est primordial de se doter de règles pour bien le commenter. Voici ces règles :

A. Le but d'un commentaire est de faire comprendre le système, décrit dans le code VHDL, par une personne qui n'est pas directement impliquée dans le développement de ce code.
B. Les commentaires doivent suivre de près la partie fonctionnelle qu'ils décrivent. Ils ne peuvent donc pas se trouver uniquement dans un bloc de texte en entête avec aucun commentaire dans la partie exécutable.
C. Les commentaires doivent être descriptifs. Il n'est pas suffisant de traduire bêtement ce qu'on lit dans le code VHDL.
D. Chaque fichier commence par un entête qui donne les informations suivantes :

- Le nom du fichier;
- Le nom du projet;
- La composante cible, c'est-à-dire celle qui inclura la présente composante;
- Les dépendances, c'est-à-dire les composantes qui permettent de fabriquer la présente composante;
- Le type de fichier, il peut être de trois catégories: « déclaration et définition d'une entity et de son architecture », « définitions et déclarations de types, de fonctions et de constantes » ou « déclaration et définition d'un banc d'essai »;
- Le ou les noms complet de ou des auteurs;
- L'établissement ou l'entreprise d'attache du fichier;
- Les versions avec les dates de chaque mise à jour et les changements qui ont été fais entre chaque version avec son auteur;
- L'entête peut contenir aussi les limitations du code, les erreurs connues ou investigations de résolution en cours.

E. Chaque déclaration de « signal », de « variable », de « type », de « in », de « out », de « inout » et de « alias » doit être suivie d'un commentaire explicatif du rôle de cet objet dans le code. Il est donc impératif d'avoir une seule déclaration par ligne (voir le point G de la section 4.3.2). Des explications en bloc de commentaire peut s'avérer utile, mais il n'est pas conseillé de procéder uniquement ainsi car dans l'éventualité de plusieurs modifications au code, le commentaire peut finir par s'avérer inconsistant;

F. Chaque déclaration de « process », de « block », de « function » etc. doit être impérativement précédée d'un commentaire descriptif de son rôle et/ou de son fonctionnement et/ou de ses limites. Dans certains cas, les paramètres et/ou les résultats attendus peuvent y être décrits pour clarifier le fonctionnement de l'objet.

G. Lorsque la fonctionnalité est représentée par des données (peu importe leur représentation : fichier texte ASCII ou constantes câblées) comme dans le cas de microcode ou de matrice de connexion d'un PLA (Programmable Logic Array), un texte explicatif complet de cette fonctionnalité doit précéder le code;

H. Si possible, utiliser des outils de documentation automatique comme par exemple : VHDLDOC;

Il peut s'avérer difficile de suivre toutes ces règles au départ, mais le travail pour les assimiler en vaut la peine. Lorsque le projet prend de l'ampleur, il devient plus facile de s'y retrouver et de faire des relations entre les modules.

Ceci termine le CHAPITRE 4. Ce dernier permet de mettre sur papier les différents aspects du projet d'AVN. La première section de ce chapitre décrit les spécifications fonctionnelles du système. La deuxième décrit les spécifications physiques. La dernière détermine les règles d'écriture du code source du système. Ces trois sections forment les barèmes qui guide la conception du contrôleur video.

CHAPITRE 5

EXPLICATION DE LA MÉCANIQUE D'AFFICHAGE D'UNE IMAGE ET DES PRINCIPES CONNEXES À LA RÉALISATION DU PROJET

Ce chapitre est le dernier avant de passer à l'explication du code VHDL et de sa synthèse. Il est erroné de croire que son importance est moindre puisque tous les concepts qui y sont présentés sont en relation directe avec le code décrit au prochain chapitre. C'est d'ailleurs pour cette raison qu'il est placé en amont de ce dernier. Il est donc important de bien lire chaque section pour en saisir tous les détails avant de passer au chapitre suivant. Si certaines parties vous semblent moins claires, référez-vous aux différents documents sources donnés le long du texte.

5.1 Mécanique d'affichage d'une image

La présente section relate des concepts mis en œuvre pour permettre l'affichage d'une image numérique sur un écran. Il est primordial de bien comprendre chacun de ces concepts afin de pouvoir continuer la lecture de ce document. Les sources d'information de cette section sont [14] Engdahl, 2005 et [15] Engdahl, 2005.

5.1.1 Les signaux de contrôle VGA

Un système d'affichage VGA possède généralement sept signaux physiques de contrôle afin de permettre un affichage adéquat. Trois de ces signaux sont de nature analogique et correspondent aux trois couleurs primaires que sont le rouge, le vert et le bleu. On les nomme aussi très souvent les signaux RGB (pour « Red », « Green », « Blue »). Leur valeur de sortie en tension se tient dans la plage de 0 à 0.7 Volt. L'intensité des couleurs est obtenue en faisant varier la tension. Habituellement, la variation de tension est générée en faisant une conversion d'une valeur numérique binaire en une tension grâce à un convertisseur numérique vers analogique (c'est d'ailleurs ce qui se passe sur notre plateforme de développement). Les quatre signaux restant sont les signaux de synchronisation

horizontale, **verticale**, **composite** et **blank**. Ceux-ci sont des signaux numériques qui contrôlent les temps d'affichage de chaque pixel et les délais du balayage de l'image. L'explication détaillée de ces signaux sera donnée à la section 5.1.3. Voyons maintenant comment générer les couleurs.

5.1.2 Méthode de génération des couleurs

Comme expliqué précédemment, les couleurs sont générées à partir des trois couleurs primaires RGB en convertissant les valeurs binaires en un niveau de tension. Pour le projet, nous utilisons un convertisseur permettant des valeurs binaires de 8 bits pour chaque couleur, ce qui donne en tout 24 bits pour chaque pixel à l'entrée du convertisseur.

5.1.2.1 Profondeur de huit bits avec table des couleurs

Il n'est pas toujours nécessaire d'accéder à une grande variété de couleurs comme dans le cas de couleur encodée sur 24 bits, d'autant plus que l'espace mémoire requis pour emmagasiner une image de 24 bits par couleur est assez grand. Pour diminuer la plage des couleurs utilisées, on utilise une table de 256 couleurs. Chaque couleur de la table possède un encodage de 24 bits et on accède à chacune d'elles en indiquant leur index à l'entrée de cette table. Cet index est de 8 bits pour pouvoir accéder aux 256 couleurs de la table. Donc, chaque couleur de pixel est emmagasiné en mémoire dans son encodage de 8 bits. On dit alors que la profondeur des couleurs de l'image est de 8 bits. L'image prend alors le tiers de l'espace d'une image dont la profondeur est de 24 bits.

5.1.2.2 Profondeur de huit bits par troncature

Il est aussi possible de tronquer les couleurs. Pour faire cela, il suffit de sauvegarder uniquement les bits les plus significatifs de chaque couleur primaire (RGB). Par exemple, pour une profondeur de 8 bits, on sauvegarde les 3 bits les plus significatifs du rouge et du vert et les deux bits les plus significatifs du bleu. Ce qui nous donne un total de 8 bits pour

chaque pixel. Pour régénérer les pixels de 24 bits nécessaires au convertisseur, nous pouvons facilement recopier le dernier bit sauvegardé de chacune des couleurs, ce qui nous permet de générer le blanc et le noir qui sont respectivement 0xFFFFFF et 0x000000. Par exemple, si le pixel sauvegardé est la valeur 0x3A (« 0011 1010 »). On a les trois bits « 001 » pour le rouge, « 110 » pour le vert et « 10 » pour le bleu. En recopiant le dernier bit de chaque couleur, on a la valeur de 24 bits 0x3FC080. Cette manière de faire permet de sauvegarder une image avec une profondeur de 8 bits sans nécessiter de table de couleurs. Par contre, la répartition du spectre des couleurs est moins homogène.

5.1.2.3 Profondeur de 16 bits par troncature

Pour augmenter la répartition du spectre des couleurs, on peut utiliser le même principe, mais en encodant les pixels sur 16 bits. Ceci nous permet donc de coder le rouge sur 6 bits et de coder le vert et le bleu sur 5 bits. On utilise le même remplissage des bits manquant que précédemment, ce qui permet toujours de donner le noir et le blanc. On a par exemple un pixel de 16 bits de valeur 0xAB3C (« 1010 1011 0011 1100 »). Ceci donne les 6 bits du rouge « 101010 », les 5 bits pour le vert « 11001 » et les 5 bits pour le bleu « 11100 ». En étirant les bits les moins significatifs de chaque couleur, on a la valeur 0xA8CFE0 à l'entrée du convertisseur.

5.1.2.4 Profondeur de 24 bits

La profondeur de 24 bits ne nécessite aucune manipulation des couleurs puisque chaque pixel affiché est sauvegardé tel quel en mémoire. Il suffit donc de lire le pixel de 24 bits et de l'envoyer directement vers le convertisseur VGA. Il est important de bien respecter la fenêtre temporelle d'affichage pour s'assurer de la qualité de l'image. Voici justement l'explication du fonctionnement des signaux permettant de bien synchroniser l'affichage.

5.1.3 L'affichage VGA et la génération des signaux de synchronisation de l'image

Cette section a pour but d'expliquer la théorie de l'affichage VGA d'une image. Avant de débuter, voici une figure qui permet de bien comprendre la mécanique avant d'aller plus en détail.

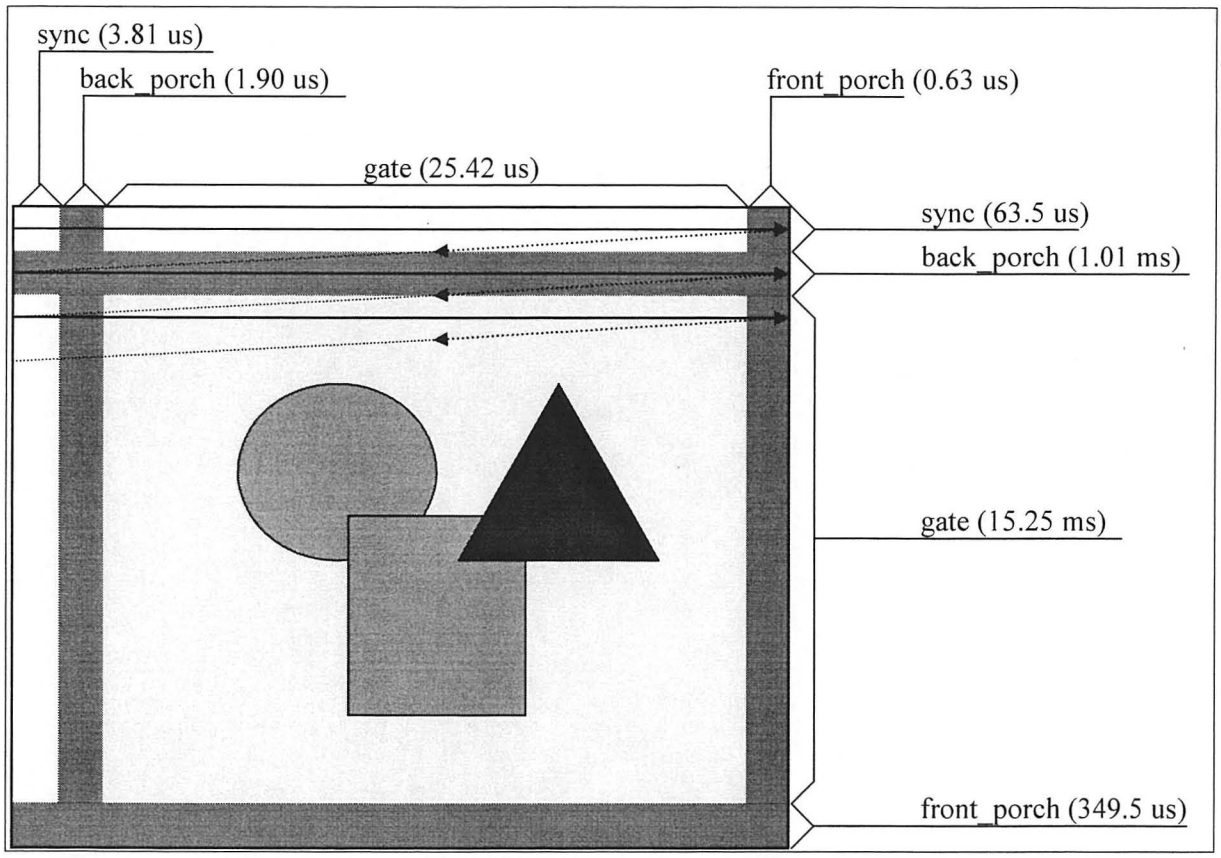

Figure 5.1 Les champs temporels de l'affichage VGA.

On remarque sur cette figure quatre champs de temporisation pour chaque axe. Ces champs nommés sync, back_porch, gate et front_porch sont les noms donnés à chaque portion du temps utile pour générer les signaux numériques de synchronisation. Vous pouvez remarquer en vous référant à la Figure 5.1 que les champs temporels ne font pas tous partie de la section visible de l'image. Seul le champ gate y correspond, c'est la « porte » de l'affichage des pixels. D'ailleurs, c'est par cette partie qu'on définie la résolution d'affichage : par exemple 640 x 480 signifie 640 pixels affichés horizontalement et 480 pixels affichés verticalement. Les autres champs, en dehors de la région affichée, sont nécessaires pour bien positionner

l'image sur l'écran. Le champ sync permet de déterminer le début du balayage. Le back_porch est le délai qui sépare les champs sync et gate. Finalement, le champ front_porch est celui qui sépare la fin de l'affichage dans un balayage et le début du champ sync du prochain balayage.

La temporisation de chaque champ se décrit en nombres de pixels. Chaque résolution d'image possède une quantité déterminée de pixels par ligne et par colonne. L'affichage commence toujours en haut à gauche et se fait par un balayage de chaque ligne pour se terminer en bas à droite. La Figure 5.1 démontre le sens de l'affichage par les flèches. Le retour à la ligne suivante est représenté par une ligne pointillée.

Les signaux numériques de synchronisation sont souvent générés à partir de compteurs. Chaque champ étant le nombre de cycles (ou pixels) que les compteurs doivent faire, on initialise toujours les compteurs à ces valeurs de champs au début de l'affichage afin d'effectuer la temporisation. Le

Tableau 5.1 montre les valeurs de chaque champ pour les résolutions les plus communes. Il va sans dire que la fréquence à laquelle ces compteurs fonctionnent est très importante et doit être précise afin de créer les bons délais sur les signaux de synchronisations.

Tableau 5.1

Valeurs typiques des compteurs pour les résolutions d'images les plus utilisées

		Horizontal (en pixel)				Vertical (en pixel)			
Résolution	Fréquence d'horloge	sync	front_porch	gate	back_porch	sync	front_porch	gate	back_porch
640x480 60 Hz	25.17 MHz	96	16	640	48	2	11	480	31
640x480 75 Hz	31.5 MHz	96	16	640	48	2	11	480	32
640x480 85 Hz	36 MHz	48	32	640	112	3	1	480	25
800x600 60 Hz	40 MHz	128	40	800	88	4	1	600	23
800x600 75 Hz	49.5 MHz	80	16	800	160	2	1	600	21
800x600 85 Hz	56.25 MHz	64	32	800	152	3	1	600	27
1024x768 60 Hz	65 MHz	136	24	1024	160	6	3	768	29
1024x768 75 Hz	78.75 MHz	96	16	1024	176	3	1	768	28
1024x768 85 Hz	94.5 MHz	96	48	1024	208	3	1	768	36

La Figure 5.2 permet de visualiser la génération des signaux de synchronisation (déjà mentionnés dans la section 5.1.1) avec les délais demandés (aussi donné en nombre de pixels) pour un affichage dont la résolution de l'image est de 640 x 480 pixels avec 60 rafraîchissements d'image par seconde et une fréquence pixel de 25,175 MHz.

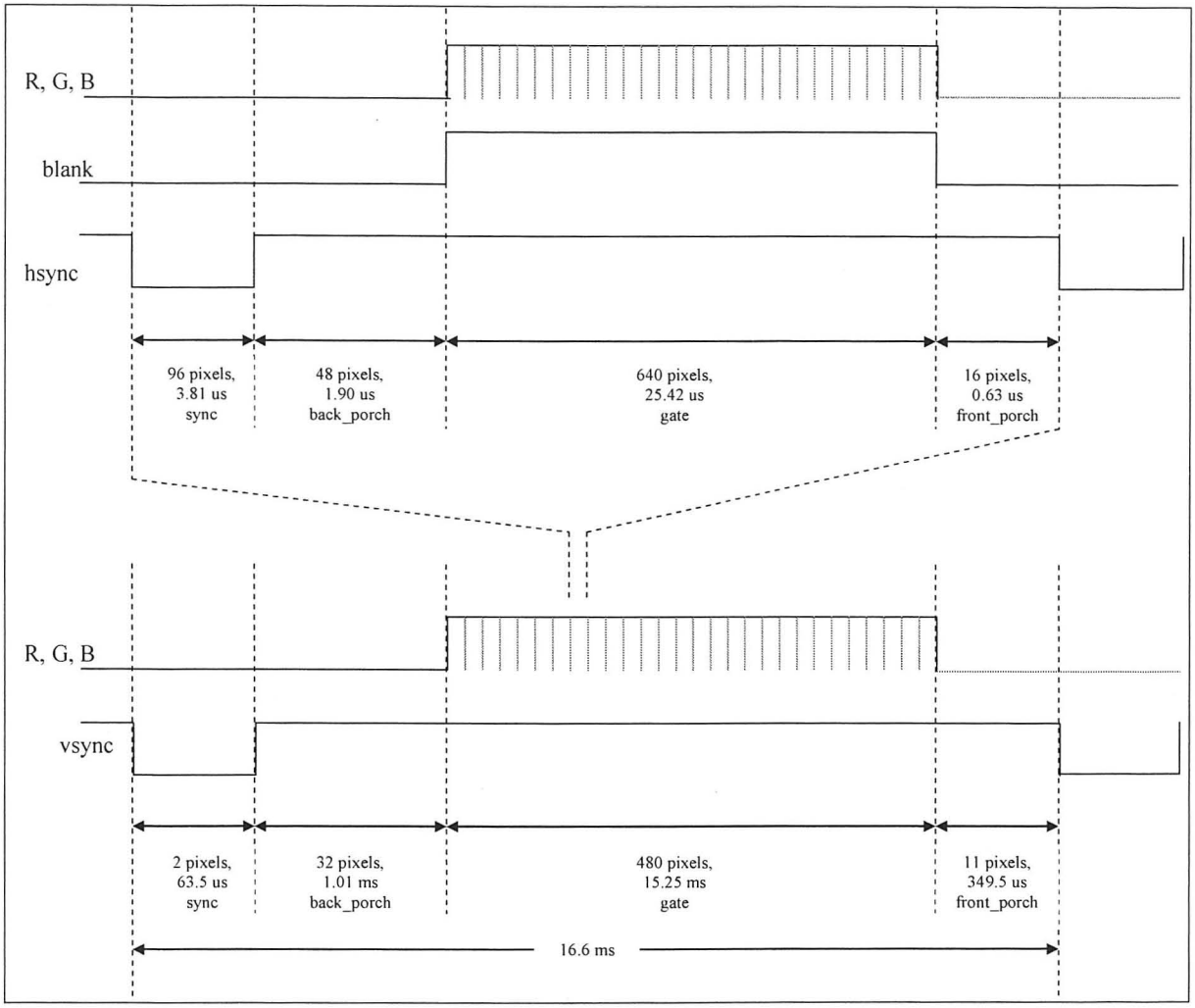

Figure 5.2 Les signaux de synchronisation de l'affichage VGA.

Sur cette figure, chaque axe est contrôlé par un signal numérique de synchronisation. Le signal hsync contrôle la temporisation des champs « sync » horizontaux et le signal vsync, celle des champs « sync » verticaux. Le compteur horizontal fonctionne à chaque cycle de l'horloge alors que le compteur vertical fonctionne une fois à chaque fin de balayage horizontal, c'est-à-dire lorsqu'un nouveau signal hsync est activé. C'est donc dire que la durée d'un « pixel vertical » dure le temps complet d'une ligne totalisant 800 pixels (dans le cas de cette résolution d'image).

Le signal csync mentionné à la section 5.1.1 n'est pas montré sur la Figure 5.2 puisque ce signal est habituellement généré en faisant passer les signaux hsync et vsync dans une porte logique « and » lorsque ceux-ci sont actif bas.

Le signal blank est le signal qui identifie la « porte » d'affichage des pixels. Il est d'ailleurs activé tout le long du champ temporel gate. En d'autres mots, le signal blank permet d'identifier le moment de placer les pixels à l'entrée du convertisseur numérique à analogique. Il est souvent généré en faisant passer un signal blank vertical et un signal blank horizontal dans une porte logique « and », ce qui place le signal de sortie exactement au bon moment dans les deux axes à la fois.

5.2 Structure des fichiers *.bmp

Le format *.bmp est le format d'image utilisé à la base dans les premières versions du système d'exploitation Windows de Microsoft. Étant donné sa simplicité et sa popularité encore très répandue, ce format est très documenté et il est facile d'en extraire les données d'images (voir les sources : [11] Daub, 1998, [8] CommentCaMarche.net, 2007 et [4] Bourke, 1998). Voici donc ce qui est le plus important à savoir au sujet des fichiers bitmap.

Le fichier possède de 3 à 4 blocs d'information disposés de la manière suivante :

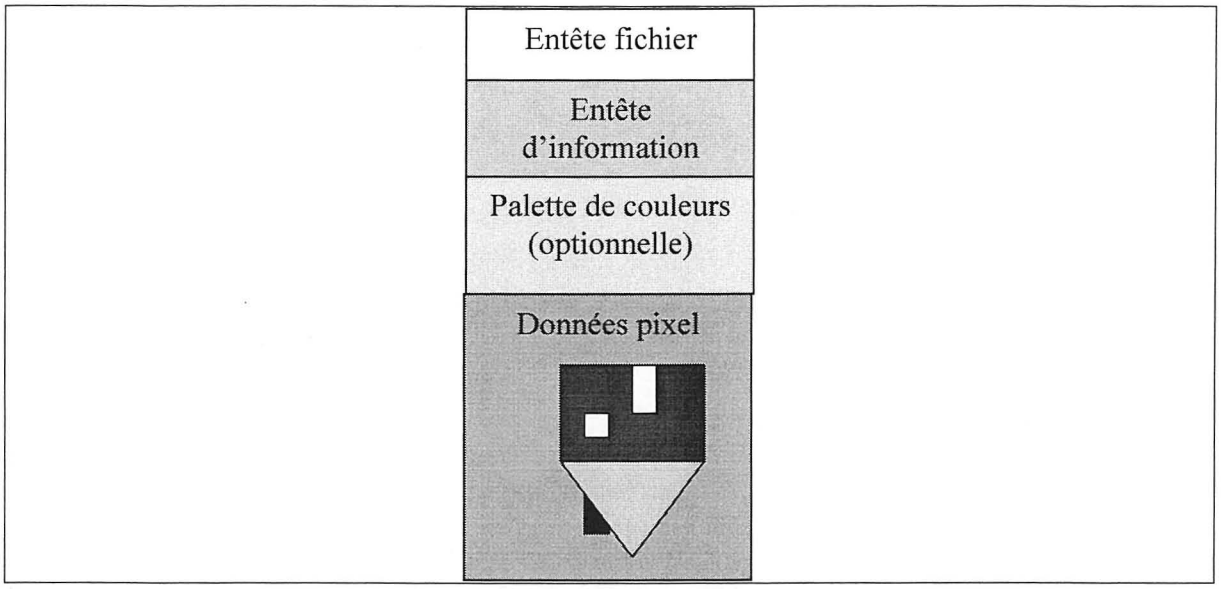

Figure 5.3 Structure d'un fichier *.bmp.

Une note importante à savoir est que les données de pixels sont placées en commençant au bas à gauche de l'image vers la droite et vers le haut par la suite. C'est pour cette raison que sur la Figure 5.3, le dessin de la maison est à l'envers. Ce détail est important car pour afficher correctement une image, on doit commencer par la ligne du haut et descendre ligne par ligne vers le bas. Ceci nous oblige donc à inverser chaque ligne de l'image bitmap pour un affichage adéquat. Pour faciliter les choses, le script bmp2txt.pl procède à cette inversion, nous verrons plus bas comment.

5.2.1 L'entête du fichier

L'entête du fichier fournit les informations sur le type de fichier (bitmap), sa taille et indique où commencent les informations concernant l'image. Voici les champs qu'on y retrouve :

A. La signature (2 octets) : pour le fichier bitmap de Windows, les caractères « BM » (ou 0x424D) forme ce champ;
B. La taille totale du fichier en octet (4 octets);
C. Un champ réservé (4 octets);

D. Le décalage (« offset ») de l'image (4 octets) : donne le nombre d'octets qui séparent le début du fichier et les données pixel.

5.2.2 L'entête d'information

Cet entête est important, mais contient souvent des informations inutilisées. Voici donc ce qu'il contient :

A. La taille de l'entête d'information en octets (sur 4 octets) : La valeur typique est 0x28;
B. La largeur de l'image en pixel (sur 4 octets) : c'est-à-dire le nombre de pixels horizontalement;
C. La hauteur de l'image en pixel (sur 4 octets) : c'est-à-dire le nombre de pixels verticalement;
D. Le nombre de plan (sur 2 octets) : cette valeur vaut toujours 1 par défaut;
E. La profondeur de codage des couleurs (sur 2 octets) : c'est-à-dire le nombre de bits utilisés pour coder la couleur (1, 4, 8, 16, 24 ou 32);
F. La méthode de compression (sur 4 octets) : Cette valeur vaut 0 lorsque l'image n'est pas compressée;
G. La taille totale de l'image en octets (sur 4 octets);
H. La résolution horizontale (sur 4 octets) c'est-à-dire le nombre de pixel/m horizontalement;
I. La résolution verticale (sur 4 octets) c'est-à-dire le nombre de pixel/m verticalement;
J. Le nombre de couleurs de la palette (sur 4 octets);
K. Le nombre de couleurs importantes dans la palette (sur 4 octets). Ce champ peut prendre la valeur 0 si chaque couleur est utilisée.

Les champs A, D, F, H, I, J et K sont toujours à leur valeur par défaut ou bien à zéro dans notre projet puisque nous ne les utilisons pas.

5.2.3 La palette de couleurs (optionnelle)

La palette de couleur est nécessaire uniquement lorsque la profondeur des couleurs est de 8 bits. En effet la palette contient 256 valeurs de couleurs codées sur 32 bits. Chaque couleur contient les trois composantes primaires RGB (24 bits) dans l'ordre suivant : le bleu (sur 1 octet), le vert (sur 1 octet) et le rouge (sur 1 octet). L'octet restant de chaque couleur est réservé pour l'évolution éventuelle du format de fichier. Il n'est pas utilisé dans notre cas.

5.2.4 Les données pixel

Cette section permet d'écrire successivement les pixels de l'image ligne par ligne en commençant par le pixel ***en bas à gauche***. Chaque ligne de l'image doit obligatoirement comporter un nombre d'octets qui est multiple de 4. Si non, la ligne doit être remplie de zéros pour respecter ce critère.

Le nombre d'octet utilisé pour chaque pixel dépend de la profondeur des couleurs de l'image. En mode 256 couleurs, chaque octet contient un pixel (mode 8 bits). En mode 16 millions de couleurs, chaque pixel nécessite 3 octets (mode 24 bits).

5.2.5 Fonctionnement du script bmp2txt.pl

Le script bmp2txt.pl est un script écrit en langage PERL qui permet d'extraire une image d'un fichier en format *bmp et de la convertir en format texte pour une utilisation ultérieure lors de la formation du fichier de programmation *.mcs. Le langage PERL est utilisé pour deux raisons principales. La première est que la suite ISE de Xilinx possède un moteur PERL qui permet de l'exécuter sans devoir installer de logiciels additionnels. La deuxième est que le langage est tout indiqué pour manipuler des fichiers et des champs à l'intérieur de ceux-ci. Cependant, l'expertise de programmation en PERL étant très limitée, il est certain qu'une optimisation de ce script serait souhaitable. D'ailleurs, vous trouverez à la section 8.3.4 une explication de ce qui peut être amélioré par rapports à ce script. Pour des

renseignements sur le langage PERL, référez vous à [36] Marshall 2005 et [41] Perl.org, 2006.

Avant de lancer le script, il suffit d'écrire le nom du fichier *.bmp à convertir avec tout le chemin d'accès dans la variable $bmp_file du script. Voici un exemple de cette ligne de code :

```
my $bmp_file = "Images/i800x600_inc.bmp";
```

Ensuite vous devez inscrire le fichier de destination avec tout son chemin d'accès dans la variable $txt_file. Voici un exemple de cette ligne de code :

```
my $txt_file = "i800x600_inc.txt";
```

Enfin, il ne reste qu'à exécuter le script en faisant dans un terminal (sous Windows, utiliser un terminal DOS) la commande suivante :

```
> xilperl bmp2txt.pl
```

Vous devez exécuter ce script pour chaque image à insérer. Chaque fichier destination doit posséder un nom unique afin de ne pas écraser des données utiles.

Le script se divise en plusieurs sections afin de faciliter son déverminage et aussi pour permettre une meilleure compréhension de son fonctionnement. Les sections qui suivent permettent de décrire chacune des parties du script.

5.2.5.1 Déclarations de départ

Cette section du script permet de déclarer et de définir les variables qui seront nécessaires pour la durée du script. C'est dans cette section, entre autres, que vous inscrivez les noms des fichiers source et destination.

5.2.5.2 Lecture de l'entête

C'est dans cette section qu'on ouvre le fichier *.bmp en mode lecture binaire. Ceci veut dire que le fichier *.bmp n'est pas un fichier texte standard avec des caractères d'imprimerie, mais que les données sont organisées en binaire pour donner des informations sur l'image. La première étape consiste donc à ouvrir le fichier *.bmp en mode binaire et ensuite de charger les informations de l'entête dans une variable nommée « $buffer ».

De cette variable sont ensuite tirées toutes les informations utiles pour l'affichage de l'image par le contrôleur vidéo. La première information est le type de fichier lu, si les deux premiers octets correspondent aux deux lettres « BM », c'est que c'est bien un fichier *.bmp. Ensuite, on extrait la taille du fichier en octets ($file_size), puis on extrait le décalage de l'image ($offset) afin de connaitre exactement l'emplacement des données d'image depuis le début du fichier. Enfin, nous somme capable de connaître le nombre d'octets de données d'une image (incluant sa table de couleurs) en faisant le petit calcul suivant : $nb_byte_data = $file_size-$offset+(256*4).

5.2.5.3 Lecture de l'entête d'information (info header)

Les informations utiles qui proviennent de l'entête d'information sont les dimensions de l'image et la profondeur des couleurs (8, 16 ou 24 bits).

On lit donc la largeur de l'image (en pixels) dans la variable $image_windth. Ensuite on lit la hauteur de l'image dans la variable $image_height. Enfin on lit la profondeur des couleurs (qui identifie le nombre de bits par pixel) dans la variable $nb_bits_pixel.

5.2.5.4 Lecture de la table des couleurs

Dans cette section, on ouvre le fichier d'image uniquement pour remplir la variable tampon $buffer avec les valeurs de la table des couleurs. Ensuite on formate une table de ces données dans la variable matricielle @tmpLUT. On instancie aussi deux autres variables

matricielles qui permettrons de travailler sur la première : @tmpLUT2 et @LUT. Les données seront retravaillées plus loin lors du formatage.

5.2.5.5 Lecture des données de pixels

De la même manière que la table des couleurs, la table des valeurs de pixels est lue et placée dans une variable $buffer. Ensuite, on sépare les données en morceau pour remplir la variable matricielle @tmpimage_data. Par la suite, les variables matricielles @tmpimage_data2 et @image_data sont instanciées pour une utilisation ultérieure dans le script.

5.2.5.6 Formatage de la table des couleurs

La table des couleurs du fichier *.bmp possède des données de quatre octets alors que chaque couleur n'a besoin que de trois octets. Il y a donc un octet de trop pour chaque valeur de couleur provenant du fichier *.bmp. La boucle « for » passe chaque octet et le réécrit dans la variable matricielle @LUT s'il fait partie des trois premiers octets de la couleur (d'où l'utilisation de l'opérateur modulo ou « % » qui permet d'identifier le dernier octet de chaque couleur). Les deux autres lignes de la boucle servent uniquement à l'affichage, ce qui est fort utile pour déverminer.

5.2.5.7 Formatage des données pour remettre l'image à l'endroit

La boucle for de cette section permet de remettre les lignes dans le bon ordre d'affichage puisque celles-ci sont inversées dans le fichier *.bmp. La boucle copie ligne par ligne chaque pixel dans une variable vectorielle (une seule dimension), ce qui facilite la section suivante puisqu'il suffira de balayer simplement la variable ainsi crée pour écrire les données dans le fichier *.txt.

5.2.5.8 Écriture des données dans le fichier user_data.txt

Cette partie du script permet de générer un fichier *.txt qui contient toutes les données nécessaire à l'affichage d'une image. La première chose qu'on rencontre dans cette section de code est un commentaire d'entête qui identifie le fichier.

Ensuite, une ligne d'entête de 16 octets est générée. Elle contient les 32 bits du patron de bits permettant de retrouver le début des données d'image une fois qu'elles sont programmées dans la mémoire PROM. Cet entête contient ensuite une trame de 32 bits pour identifier le nombre d'octets présent dans le fichier (ce qui inclut uniquement les octets de la table des couleurs et ceux de l'image proprement dite). Les 32 bits suivant sont ceux qui permettent une configuration de base du contrôleur vidéo. Présentement, ils configurent le contrôleur en mode 8 bits par pixel. Les derniers 32 bits de cet entête ne sont pas utilisés, mais pourront être utilisés éventuellement.

La ligne suivante du fichier de script débute une boucle « for » qui permet de mettre en forme la table des couleurs de l'image. On lit 32 bits à la fois afin de recréer ce qui sera effectué dans le système de contrôleur vidéo lors du chargement de la table à l'intérieur du système. Ceci permet donc de placer les octets de la table des couleurs dans le bon ordre dans la mémoire PROM.

Enfin, la dernière boucle « for » exécute sensiblement le même travail que la précédente, excepté qu'elle le fait sur les données des pixels de l'image. On découpe ici l'image en tranches de 4 octets pour la placer dans le fichier *.txt. Le fichier *.txt généré par le script doit contenir obligatoirement 16 octets hexadécimal par ligne. Ceci à pour but de faciliter la génération du fichier *.mcs en rendant l'utilisation du script « pc.pl » possible (Pour l'utilisation du script pc.pl, reportez-vous à la section suivante du présent document). La figure suivante montre l'organisation des octets dans le fichier *.txt.

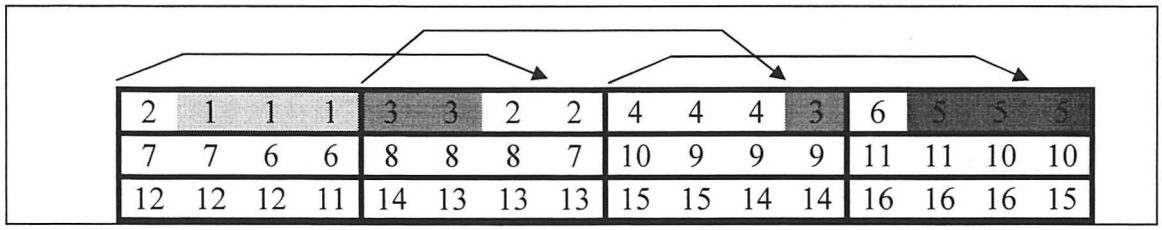

Figure 5.4 Organisation des données dans le fichier utilisateur.

Dans cet exemple chaque chiffre représente un octet et chaque valeur représente un pixel de 24 bits. Donc, le pixel numéro un se place à droite du premier morceau de 32 bits, il reste donc un octet de libre qui est comblé par le premier octet du prochain pixel. La deuxième tranche de 32 bit comble donc les deux octets de droite par ceux restant du pixel numéro deux et le deux de gauche sont comblés par la première partie du pixel trois. On procède ainsi pour tous les pixels de l'image. Cette manière de faire peut sembler bizarre, mais elle permet d'optimiser la bande passante du bus mémoire lors de la lecture de l'image.

5.2.6 Utilisation du script pc.pl fournit avec la note d'application XAPP694 de Xilinx

Le script pc.pl permet d'insérer des données utilisateur à la suite de la configuration du FPGA d'un fichier *.mcs. Ceci permet donc d'utiliser la mémoire PROM de la plateforme de développement comme support pour nos données d'images. Aucune explication du fonctionnement de ce script ne sera faite ici puisque nous n'avons pas conçu ce dernier et son fonctionnement interne exact n'est pas connu avec précision. Voici par contre une petite explication de son utilisation (voir aussi la section 7.5).

Avant d'exécuter le script, il faut premièrement s'assurer de générer un fichier *.mcs à partir du fichier *.bit du projet ISE à l'aide du logiciel Impact (voir section 7.4) et deuxièmement produire un fichier *.txt qui contient toutes les données utilisateurs nécessaires au bon fonctionnement du système (vous pouvez vous référer aux sections 7.5 et 5.2.5 pour ce faire). Pour lancer le script, faite la commande suivante dans un terminal :

```
xilperl pc.pl –ps 32 –pf initial.mcs –pf user_data.txt
```

L'option –ps 32 spécifie la taille de la mémoire PROM. L'option –pf initial.mcs spécifie à partir de quel fichier *.mcs générer le nouveau fichier jumelé. L'option –pf user_data.txt spécifie le fichier *.txt qui contient les données à annexer au fichier de configuration du FPGA. Le nouveau fichier *.mcs généré par ce script se nomme new_initial.mcs. En effet, le script utilise le nom du fichier *.mcs de départ et ajoute le préfixe « new_ » à ce nom pour créer le nom du fichier générer. Cela a l'avantage de ne pas écraser le fichier d'origine et permet alors de générer plusieurs fichiers de configuration identiques mais possédant des données différentes (dans notre cas, des images différentes).

5.3 Fonctionnement de la mémoire PROM

La présente section permet de comprendre la manière dont la mémoire PROM est utilisée dans le projet. Les détails sont tirés de [54] Xilinx, 2004, [56] Xilinx, 2005 et [58] Xilinx, 2005.

La mémoire PROM de la carte SP305 est le modèle xcf32p de Xilinx. Elle permet de programmer le FPGA en mode « Serial Master », « Serial Slave », « Master selectMAP » et « Slave selectMAP ». Afin de permettre la lecture de données d'images, nous avons opté pour le mode « serial slave », le FPGA étant le maître, il génère l'horloge qui synchronise la sortie des données. La raison de ce choix est simplement parce que nous pouvions rapidement fabriquer un prototype de chargeur de démarrage (« bootloader ») à partir du modèle développé par Xilinx qui est expliqué dans la note d'application XAPP694 (voir [54] Xilinx, 2004). Il serait cependant possible de fabriquer un chargeur de démarrage plus rapide en utilisant un mode parallèle (selectMAP) plutôt que sériel. La carte SP305 permet beaucoup de versatilité dans ce domaine, cependant, nous nous attarderons uniquement sur ce qui a été utilisé lors de l'élaboration du projet. Voici le schéma de connexion qui est utilisé sur la carte SP305 afin d'utiliser le mode « serial slave ».

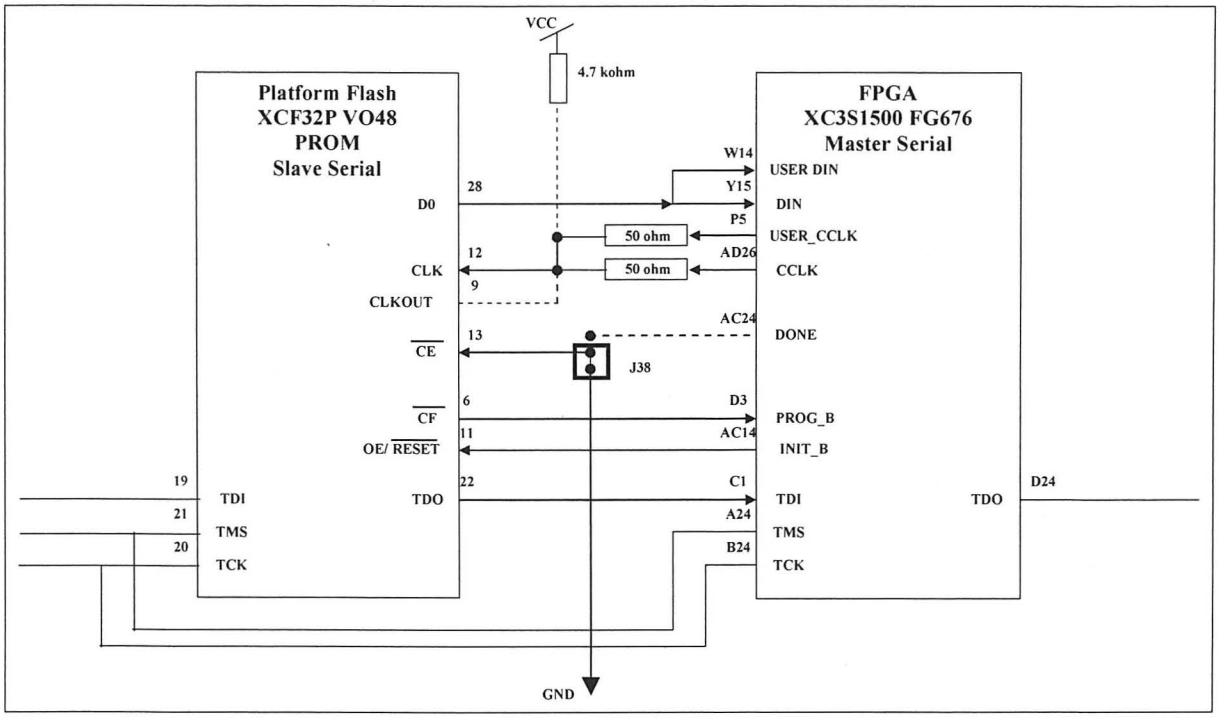

Figure 5.5 Schéma bloc des connexions entre la mémoire PROM et le FPGA sur la plateforme de développement SP305.

Prenez note que la position du cavalier sur les broches J38 doit être **inversée** par rapport à celle spécifiée dans les guides de la plateforme SP305. En effet le cavalier doit faire en sorte que la PROM est toujours active même suite à la configuration du FPGA afin de permettre la lecture des données d'images. Pour la garder active, il suffit de mettre la broche CE à la terre comme montré sur la Figure 5.5, ce qui correspond à placer le cavalier sur la position « DONE RESET » sur la plateforme SP305 comme il est montré sur la Figure 7.25 dans le cercle nommé J38 (plutôt qu'en position « Always Enable »). C'est le seul petit problème rencontré avec la carte SP305 jusqu'à ce jour.

Lorsque la mémoire PROM est programmée, elle lance un signal « PROG » au FPGA afin de le forcer à se reconfigurer. Le FPGA étant en mode « Master Serial », il génère l'horloge de configuration cclk qui alimente la mémoire PROM. Une fois le FPGA configuré, l'horloge de configuration par défaut s'arrête et le circuit interne du FPGA s'active. C'est à ce moment que le module bootloader entre en jeu et génère l'horloge auxiliaire cclk afin de permettre la continuité de la lecture de la mémoire PROM. Sur la Figure 5.5, on observe les deux

résistances de 50 Ohm qui permettent ce subterfuge. La plateforme de développement SP305 a été créée de cette manière spécifiquement pour cette raison. C'est d'ailleurs pour cela que le signal DIN est aussi branché à deux endroits sur le FPGA (voir la Figure 5.5). Le premier endroit est le signal DIN pour la configuration du FPGA et le second est une entrée/sortie configurable par l'usagé afin d'utiliser la mémoire PROM comme support de données.

Il serait possible de modifier le module bootloader afin de rendre la configuration du FPGA parallèle sur huit bits et ainsi, lire la mémoire PROM sur huit bits pour les données d'images. La plateforme permet cette configuration et multiplierait par huit la vitesse de chargement des données dans la mémoire vidéo.

5.3.1 Ordonnancement des données

Les données qui sortent de la mémoire PROM sont ordonnancés par blocs de huit bits en format gros boutiste (big endian), alors que la mémoire vidéo, tout le système d'affichage ainsi que les fichiers d'images *.bmp utilisés sont en format petit boutiste (little endian). Nous devons donc réordonnancer chaque octet qui sort de la mémoire PROM. La figure suivante explique le processus de réordonnancement des bits lors de l'utilisation de la mémoire PROM.

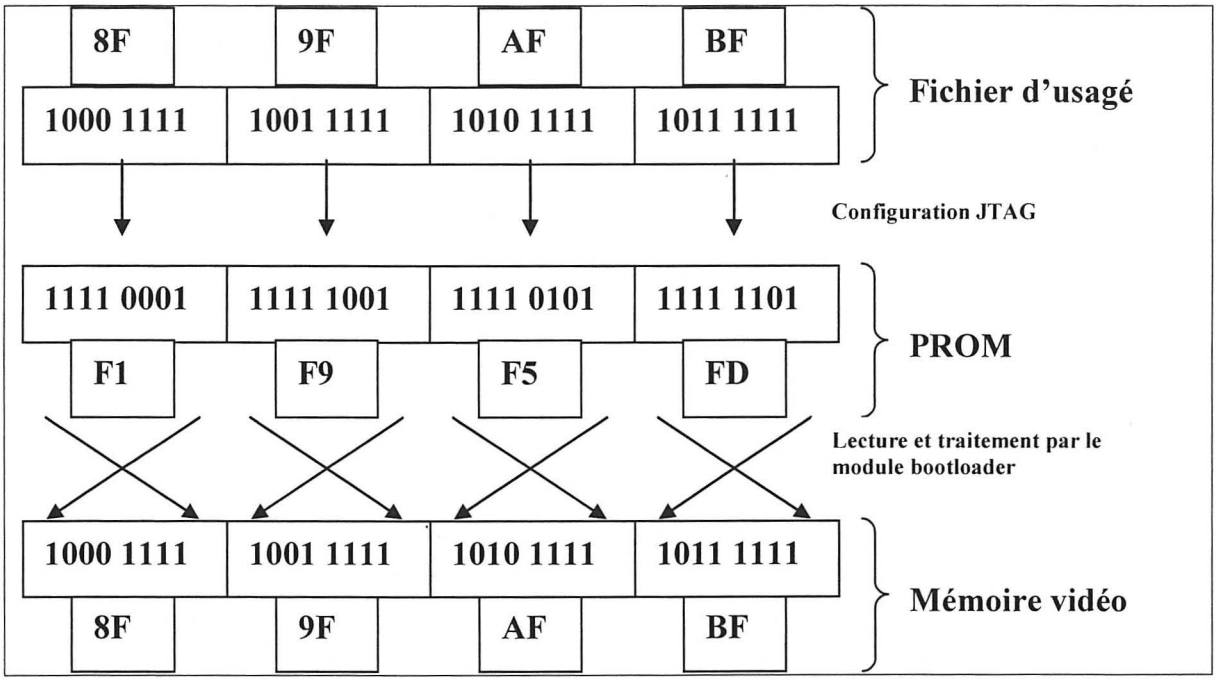

Figure 5.6 Schéma d'explication des processus d'ordonnancement des bits de données pour l'utilisation de la mémoire PROM.

Lors de la lecture de la mémoire PROM, les données de sortie sérielle sont ordonnancées en gros boutiste, même si les données d'origines envoyées sur le port JTAG sont ordonnancées en petit boutiste. La raison en est fort simple : c'est que la lecture de la PROM se fait toujours de l'adresse la moins significative vers la plus significative, ce qui réordonnance les bits dans chaque octets. Il est donc nécessaire de réorganiser les bits lus afin de retrouver les mots binaires originaux. La Figure 5.6 montre que les octets gardent le même ordre, mais que les bits dans chaque octet sont inversés. La boucle « for » du module pattern_reco_loader permet de faire ce réordonnancement simultanément avec la lecture des données de la PROM. Pour plus de détail sur le sujet, référez-vous à [54] Xilinx, 2004.

5.4 Structure des fichiers *.mcs

Le fichier *.mcs est un fichier texte qui permet de remplir la mémoire PROM et qu'on nomme aussi « Intel Object File ». On s'en sert habituellement pour emmagasiner les

fichiers de configuration des FPGA sur une plateforme de développement. La structure des lignes de données d'un fichier *.mcs est montrée dans les trois tableaux suivants.

Tableau 5.2

Entrée de données type 00

:	BC	AAAA	00	HHHH....H	CC
Char. De départ	Nombre d'octets	Adresse hexa	Type	HH = 1 Byte de donnée	Checksum
	2 Char	4 Char	2 Char	2 à 32 Char	2 Char

Tableau 5.3

Entrée de données type 01

:	00	0000	01	FF
Char. De départ	Nombre d'octets	Adresse hexa	Type	Checksum
	2 Char	4 Char	2 Char	2 Char

Tableau 5.4

Entrée de données type 04

:	02	0000	04	HHHH	CC
Char. De départ	Nombre d'octets	Adresse hexa	Type	2 Bytes offset	Checksum
	2 Char	4 Char	2 Char	4 Char	2 Char

Ces trois types de configurations doivent être rigoureusement suivit afin de créer le fichier *.mcs contenant des données utilisateur.

Le type 00 permet de placer des données utilisateur, soit jusqu'à 16 octets par ligne du fichier *.mcs, ce qui totalise 32 caractères hexadécimal dans le champ des données.

Le type 01 sert uniquement à identifier la fin du fichier.

Le type 04 permet d'étendre le champ d'adresse des données. En effet, la plage d'adresse directement accessible est de 4 caractères ou 16 bits, donc $2^{16} = 65536$ données. Ce type de ligne permet alors une extension de la plage d'adresse en ajoutant 16 autres bits d'adresse, ce qui définit les bits 31 à 16 de l'adresse linéaire de 32 bits de base. Cette valeur de deux octets est ajoutée à toutes les données qui suivent cette ligne pour constituer l'adressage absolu des données de la PROM.

Le champ « Checksum » permet de vérifier la validité de la transmission des données. Il est calculé en faisant la sommation en format complément 2 de chaque octet de la ligne en cours.

Le script Perl permettant d'ajouter les données utilisateur à la suite de la section de configuration du FPGA doit être en mesure de calculer tous ces champs afin que le chargement de la PROM s'exécute sans problème. Heureusement, ce script nommé « pc.pl » est fournit avec la note d'application XAPP694 ([54] Xilinx, 2004). Pour l'utiliser, il suffit de mettre les données de départ dans un fichier texte et de les organiser en groupes de 32 caractères hexadécimaux (ou 16 octets) par ligne. Veuillez lire la note d'application XAPP694 pour bien comprendre le processus.

Ceci met fin au CHAPITRE 5. Ce chapitre explique les principes utilisés pour compléter l'élaboration du système d'afficheur vidéo numérique. On y retrouve entre autre : la description de la mécanique d'affichage d'une image sur un écran VGA, une description de la structure d'un fichier *.bmp avec l'explication du fonctionnement du script PERL permettant d'extraire les données d'image, une explication du fonctionnement de la mémoire RAM du système d'afficheur vidéo ainsi que la description de la structure d'un fichier *.mcs servant à contenir toutes les données utiles pour la configuration du FPGA.

CHAPITRE 6

DÉVELOPPEMENT DE LA TOPOLOGIE ET PARTITIONNEMENT DES DIFFÉRENTES FONCTIONS DU SYSTÈME VIDÉO NUMÉRIQUE

Ce chapitre explique en détail la fonctionnalité des différents modules qui composent le système d'Afficheur Vidéo Numérique. Pour chaque module, un symbole et une explication des entrées et sorties sont donnés. Par la suite, une explication du fonctionnement de ce module est appuyée par un schéma bloc et par quelques chronogrammes de simulation. Nous procédons par une approche de haut en bas, ce qui signifie que nous débutons par le module de plus haut niveau et nous terminons par un module qui se trouve à la base de l'arborescence. La section 6.1 décrit en détail la hiérarchie des modules, ce qui facilite le suivi des différentes sections. Par la suite une explication de base vous est fournie sur la manière dont la mémoire PROM est utilisée. Puis, on détaille le contenu du fichier de définition des types qui est utilisé à tous les niveaux du système. Ce fichier se nomme types_p.vhd. Enfin, nous commencons avec le module « top » qui comprend tous les autres modules.

6.1 Hiérarchie des différents modules avec le nom du fichier qui les contient

Cette section permet de situer rapidement chacun des modules dans le hiérarchie du système en partant du module de plus haut niveau qui est le module top et en descendant vers le bas où se trouvent les modules les plus simples. Vous trouverez les fichiers VHDL qui constituent le circuit à l'ANNEXE I dans l'ordre d'apparition des modules de cette section. Les sections qui suivent seront aussi dans cet ordre et permettrons de décrire en détail chacun des modules et des sous-modules Voici donc cette arborescence des modules :

- top0 (top.vhd)
    - types_p (types_p.vhd)
    - wb2zbt_wraper0 (wb2zbt_wraper.vhd)
    - reset_gen0 (reset_gen.vhd)
    - clk_gen0 (clk_gen.vhd)

        - pclk_gen0 (pclk_gen.vhd)
        - mclk_gen0 (mclk_gen.vhd)
    - bootloader0 (bootloader.vhd)
        - pattern_reco_loader0 (perttern_reco_loder.vhd)
        - cclk_gen0 (cclk_gen.vhd)
        - clut_dat_gen0 (clut_dat_gen.vhd)
        - load_fsm0 (load_fsm.vhd)
        - dload_adr_gen0 (dload_adr_gen.vhd)
            - adr_counter0 (adr_counter.vhd)
            - fin_dload_gen0 (fin_dload_gen.vhd)
    - vga0 (vga.vhd)
        - ctrl_register0 (ctrl_register.vhd)
        - sync_gen0 (sync_gen.vhd)
            - timing_gen0 (timing_gen.vhd)
            - timing_gen1 (timing_gen.vhd)
        - pixel_gen0 (pixel_gen.vhd)
            - wb_ctrl0 (wb_ctrl.vhd)
            - fifo0 (fifo.vhd)
            - pixel_processor0 (pixel_processor.vhd)
                - clut0 (clut.vhd)
            - dclk_fifo0 (dclk_fifo.vhd)
            - pixel_counter0 (pixel_counter.vhd)

6.2 Utilisation de la PROM pour la configuration de FPGA et pour la sauvegarde des données d'images.

Normalement, suite à la réussite de sa programmation, le FPGA génère un signal DONE qui est branché par défaut sur la remise à zéro de la PROM. Par contre, afin de permettre la prise de contrôle de la mémoire PROM suite à la configuration du FPGA, nous devons déconnecter le signal DONE de la PROM en plaçant le cavalier J38 de la carte SP305 sur les broches 1 et 2 (ce qui est contraire à la documentation de la carte). En effet, la documentation mentionne de placer le cavalier sur les broches 2 et 3, mais après plusieurs vérifications, nous avons constaté l'erreur : les broches 1 et 3 sont inversées).

La PROM se programme par le port JTAG à l'aide d'un fichier *.mcs qui contient le programme du FPGA et aussi toutes les données nécessaires pour faire fonctionner notre système VGA. Il est à noter que les bits des données dans la PROM sont ordonnancés en

gros boutiste (Big Endian), ce qui fait que chaque octet doit être réordonnancé en petit boutiste (Little Endian) avant d'être utilisé.

Pour plus de détails sur l'utilisation de la mémoire PROM en général et pour l'utiliser comme support pour des données utilisateur, référez-vous à la section 5.3 et aux documents [54] Xilinx, 2004 et [56] Xilinx, 2005.

6.3 Description du fichier des définitions des fonctions et des types communs à tous les fichiers.

Le fichier types_p.vhd contient un ensemble de types, de fonctions et de constantes qui facilite l'homogénéité, la compréhention et l'écriture du code VHDL. Les paragraphes qui suivent expliquent chaque partie de ce fichier.

Le type timing_t est un groupe de signaux qui englobent toutes les valeurs des compteurs dans un axe d'affichage (horizontal ou vertical). Il comprend donc les signaux suivants :

Tableau 6.1

Description des signaux groupés dans le type timing_t

Nom	Description
sync_r	Largeur de l'impulsion du signal sync.
back_porch_r	Intervalle entre le signal sync et le début des pixels visibles (gate_r).
gate_r	Nombre de pixels visibles sur cet axe (horizontal ou vertical).
front_porch_r	Intervalle entre la fin de l'affichage des pixels et le signal sync.

Le type timing_h_v_t est un groupe de signaux qui englobent toutes les valeurs des compteurs des deux axes d'affichage et aussi les constantes qui permettent de générer la fréquence exacte de l'horloge pixel pclk par le DCM du module pclk_gen (section 6.7.3). Il comprend donc les signaux suivants :

Tableau 6.2

Description des signaux groupés dans le type timing_t

Nom	Description
horizontal_r	Groupe des valeurs des compteurs de la génération des signaux de synchronisme pour l'axe horizontal.

vertical_r	Groupe des valeurs des compteurs de la génération des signaux de synchronisme pour l'axe vertical.
pclk_divide_r	Valeur du diviseur d'horloge qui peut prendre les valeurs entières de 1 à 32.
pclk_multiply_r	Valeur du diviseur d'horloge qui peut prendre les valeurs entières de 2 à 32.

Les constantes de ce type permettent de configurer rapidement la résolution et le taux de rafraichissement de l'image. Il sera utile de définir d'autres constantes dans l'avenir afin d'automatiser le changement de résolution. Voici quelques constantes définies :

- res_640_480_60hz_c : affichage de résolution 640x480 avec un taux de rafraîchissement de 60 Hz.
- res_800_600_60hz_c : affichage de résolution 800x600 avec un taux de rafraîchissement de 60 Hz.
- res_1024_768_60hz_c: : affichage de résolution 1024x768 avec un taux de rafraîchissement de 60 Hz.

Le type timing_levels_t est un groupe de signaux qui englobent toutes les valeurs des niveaux actifs pour les signaux VGA. Il est donc possible de configurer les sorties VGA comme actives hautes ou actives basses. Le type comprend les signaux suivants :

Tableau 6.3

Description des signaux groupés dans le type timing_levels_t

Nom	Description
blank_r	Niveau actif du signal blank par défaut à 1.
csync_r	Niveau actif du signal sync composite (csync) par défaut à 0.
vsync_r	Niveau actif du signal sync vertical (vsync) par défaut à 0.
hsync_r	Niveau actif du signal sync horizontal (hsync) par défaut à 0.

Le type clut_load_t est un groupe de signaux qui englobent tous les signaux qui permettent de contrôler le chargement de la table des couleurs du module clut (voir section 6.11.5.5.3). Le type comprend donc les signaux suivants :

Tableau 6.4

Description des signaux groupés dans le type clut_load_t

Nom	Description
load_r	Ce signal permet d'activer le chargement de la table des couleurs.
dat_r	Ce signal est la donnée de pixel qui correspond à l'index de 8 bits (l'adresse de la

	couleur dans la table).
wadr_r	Ce signal est l'index ou l'adresse de chargement de la donnée dans la table des couleurs.

Le type wbm_ctrl_out_t est un groupe de signaux qui englobent tous les signaux qui permettent de contrôler un bus Wishbone du côté maître. Le type comprend donc les signaux suivants :

Tableau 6.5

Description des signaux groupés dans le type wbm_ctrl_out_t

Nom	Description
cyc_r	Signal qui indique un cycle valide du bus en activité.
stb_r	Signal de sélection du transfert de données.
we_r	Signal de « write enable ».
sel_r[4]	Signal de sélection d'octet (granularité).

Le type wbm_ctrl_in_t est un groupe de signaux qui englobent tous les signaux qui permettent de recevoir une réponse d'un périphérique esclave par le maître. Le type comprend donc les signaux suivants :

Tableau 6.6

Description des signaux groupés dans le type wbm_ctrl_in_t

Nom	Description
rty_r	Signal de « retry » qui indique au maitre de recommencer sa requête.
err_r	Signal d'erreur qui indique au maître qu'il y a un problème sur le bus. (pas utilisé pour le moment).
ack_r	Signal d'acquittement du périphérique esclave qui indique au maître que la transaction est réussie.

6.4 Description des entrées et des sorties du module top

La figure suivante représente le symbole du module top qui est le module de plus haut niveau hiérarchique du système :

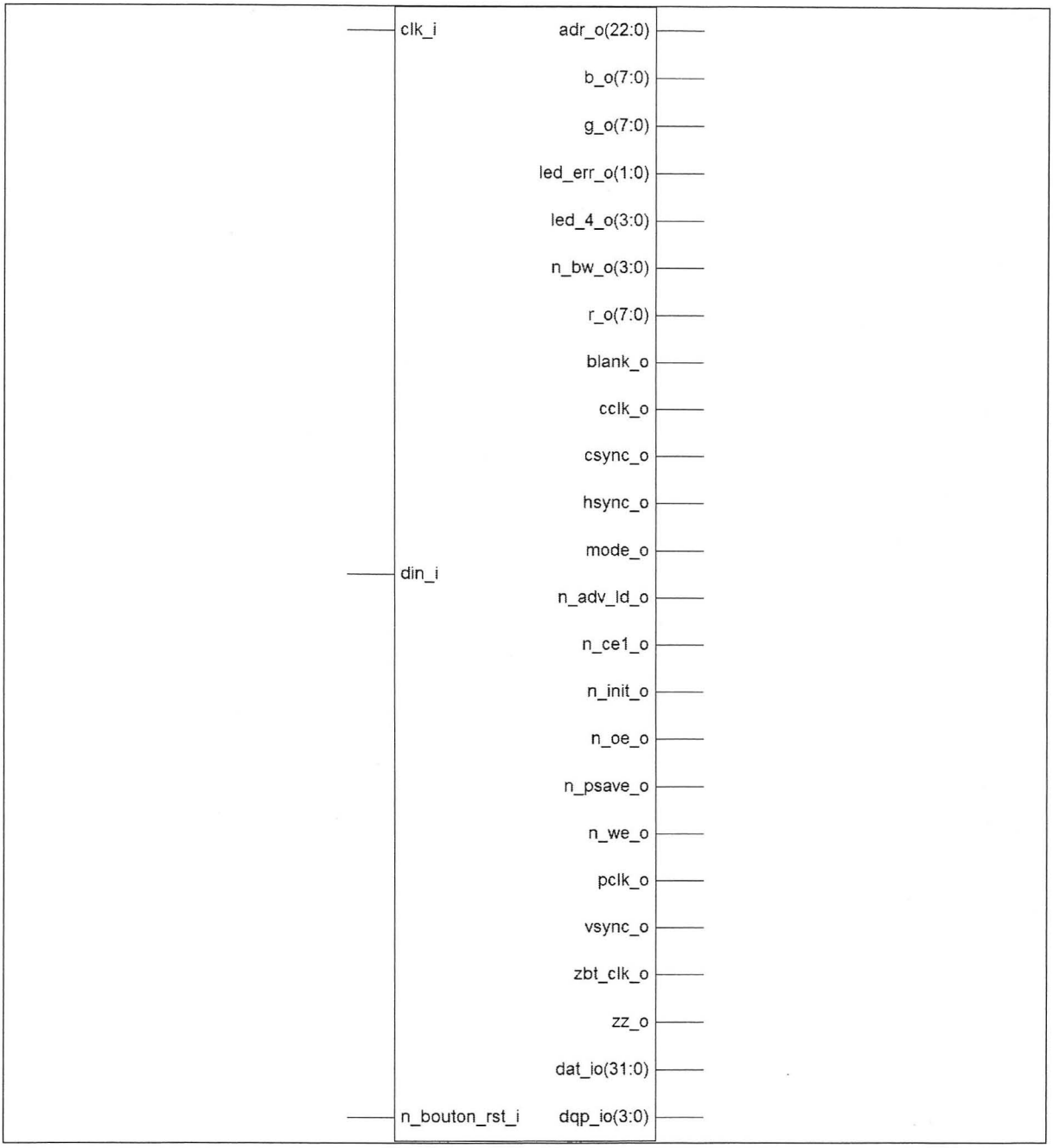

Figure 6.1 Symbole du module top.

Avant de passer à la description du fonctionnement du module top, voici les détails de ses entrées et de ses sorties :

Tableau 6.7

Description des entrées et des sorties du système global (module top)

Nom	Entrée/ Sortie	Description
din	E	Signal sériel qui sort de la mémoire PROM et qui se rend au module top0/bootloader0/pattern_reco_loader0. Il constitue les données sérielles à charger en mémoire.
cclk	S	Signal de pseudo horloge généré par le module /top0/bootloader0/cclk_gen0 en divisant l'horloge mclk à l'aide de bascule. Une division par cinq est effectuée par défaut, mais il est possible de l'optimiser pour qu'elle soit générée au maximum de la vitesse supportée par la mémoire PROM qui est de 40 MHz. Les paramètres de génération de cette pseudo horloge seront discutés à la section 6.10.4.
n_init	S	Signal de réinitialisation de la mémoire PROM. Lorsqu'il est actif (bas), le pointeur interne de la mémoire PROM retourne au tout premier bit de la mémoire. Elle est donc prête pour une relecture complète. Lorsqu'il est inactivé (haut), un front sur l'entrée cclk permet d'incrémenter le pointeur interne d'un bit. Ce bit se présente alors à la sortie din. Lorsque le pointeur est en fin de course, il y restera jusqu'à ce que le signal n_init soit activé de nouveau. Ce signal est directement relié au signal n_clear dans le module bootloader0. Pour plus de détails, référez-vous à la section 5.3.
n_bouton_rst	E	Signal actif bas qui se branche directement sur le bouton reset de la carte de développement
clk	E	Ce signal est directement branché sur la broche de l'horloge globale de la carte SP305. Cette horloge oscille à 100 MHz.
led_err(0)	S	Ce signal est branché sur la DEL nommée Err1 sur la carte. C'est le signal pat_ok provenant du bootloader qui lui est raccordé et qui indique que les données de la PROM ont été situées.
led_err(1)	S	Ce signal est branché sur la DEL nommée Err2 sur la carte. C'est le signal fin_dload provenant du bootloader qui lui est raccordé et qui indique que les données de la PROM ont été entièrement chargées en mémoire vidéo.
led_4[4]	S	Ces quatre signaux sont utiles uniquement lors du déverminage. En fonctionnement normal, ils ne sont pas utilisés et sont branchés directement à la terre.
adr[23]	S	Signal d'adresse de la mémoire vidéo ZBT.
dat[32]	E/S	Signal de données de la mémoire vidéo ZBT.
dqp[4]	E/S	Signal de données supplémentaire de la mémoire vidéo ZBT qui peuvent être utilisés pour identifier la parité de chaque octet de données.
n_bw[4]	S	Signal actif bas qui permet d'identifier les octets valides à transférer avec la mémoire ZBT. On l'appel souvent la granularité des données.
n_we	S	Signal actif bas de « write enable » de la mémoire vidéo ZBT.
n_oe	S	Signal actif bas de « output enable » de la mémoire vidéo ZBT. Ce signal est important uniquement en lecture et permet d'activer les sorties de la mémoire. Autrement, la mémoire est en haute impédance.
n_adv_ld	S	Signal qui permet à la mémoire de charger une adresse explicitement si il est à zéro (0) ou de faire une incrémentation automatique de l'adresse s'il est à un (1). Ce mode est utile dans le cas ou le contrôleur de la mémoire est incapable d'incrémenter l'adresse mémoire assez rapidement pour avoir une nouvelle adresse à chaque cycle, ce qui n'est pas notre cas.
mode	S	Signal de mode de la mémoire ZBT. 0 = linéaire, 1 = interlace.
zz	S	Signal de mise en veilleuse de la mémoire. 0 = mémoire en fonction, 1 = mémoire en veilleuse. N'étant pas utilisé en ce moment dans le système, on le garde à 0 en tout temps.

n_ce1	S	Signal actif bas qui permet de sélectionner la mémoire vidéo ZBT. Ce signal est toujours actif.
zbt_clk	S	Signal d'horloge de la mémoire vidéo ZBT. Ce signal est branché en permanence sur l'horloge mclk qui est générée à l'interne du circuit par le module mclk_gen (section 6.7.4).
r[8]	S	Signal de donnée de la couleur rouge vers le convertisseur vidéo.
g[8]	S	Signal de donnée de la couleur vert vers le convertisseur vidéo.
b[8]	S	Signal de donnée de la couleur bleu vers le convertisseur vidéo.
n_psave	S	Signal actif bas de mise en veilleuse du convertisseur vidéo.
hsync	S	Signal de synchronisation vidéo horizontal vers le convertisseur vidéo.
vsync	S	Signal de synchronisation vidéo vertical vers le convertisseur vidéo.
csync	S	Signal de synchronisation vidéo composite vers le convertisseur vidéo.
blank	S	Signal de synchronisation vidéo qui identifie les sections visibles de l'image. Ce signal se dirige vers le convertisseur vidéo.
pclk	S	Signal d'horloge des pixels. Il est généré par le module pclk_gen (section 6.7.3).

6.5 Description du fonctionnement du module top

Pour débuter, voici un schéma du circuit global dans le FPGA qui nous permettra de mieux situer les différents blocs de fonctionnalité. Ce dessin nous permettra de faciliter l'explication des relations entre les différents signaux qui constituent le système.

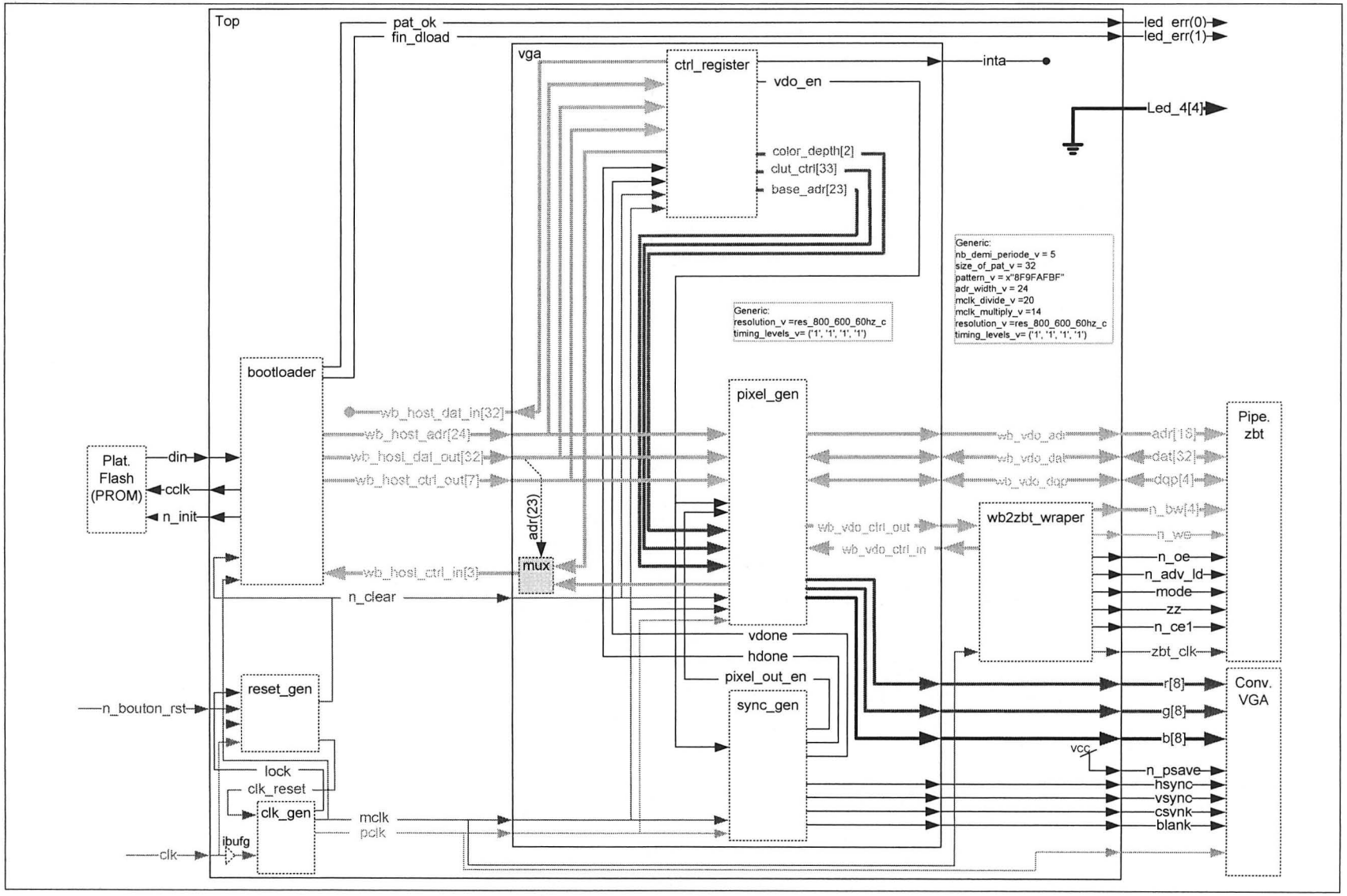

Figure 6.2 Schéma bloc du module top.

En faisant l'abstraction de l'intérieur des boîtes, nous pouvons tout de même visualiser le fonctionnement du système. Pour facilité la compréhension globale, voici trois diagrammes d'états qui servent de référence dans les explications qui suivent.

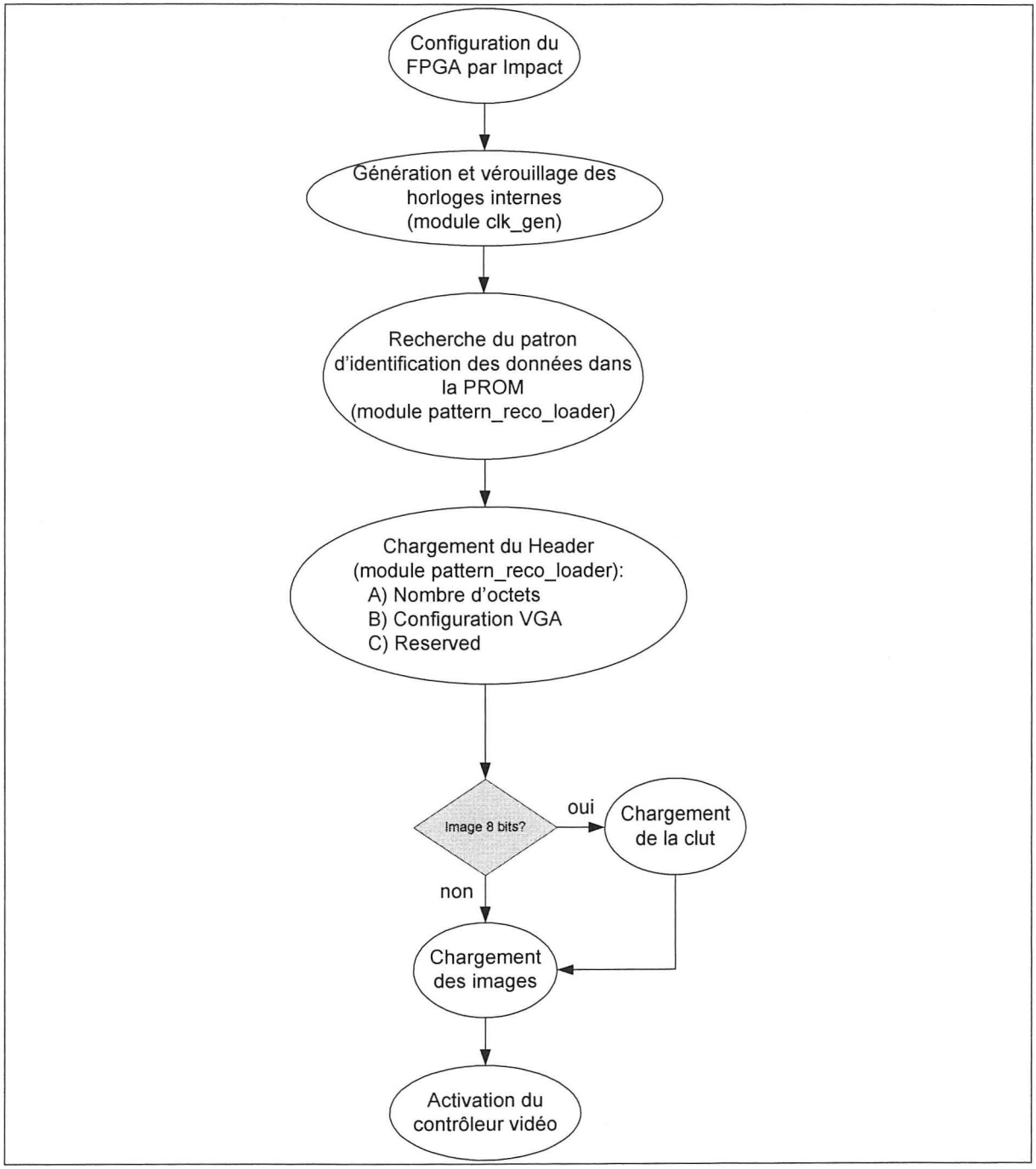

Figure 6.3 Diagramme d'état du chargement des données dans la mémoire vidéo avant l'activation du contrôleur vidéo.

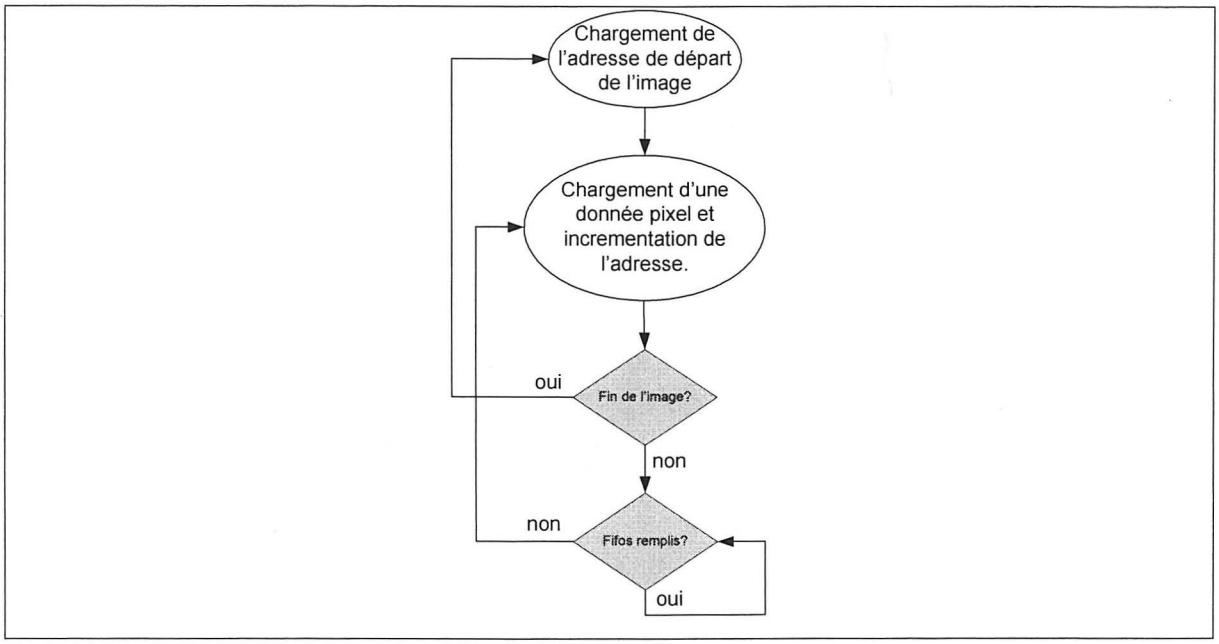

Figure 6.4 Diagramme d'états simplifié du chargement des Fifo du module pixel_gen.

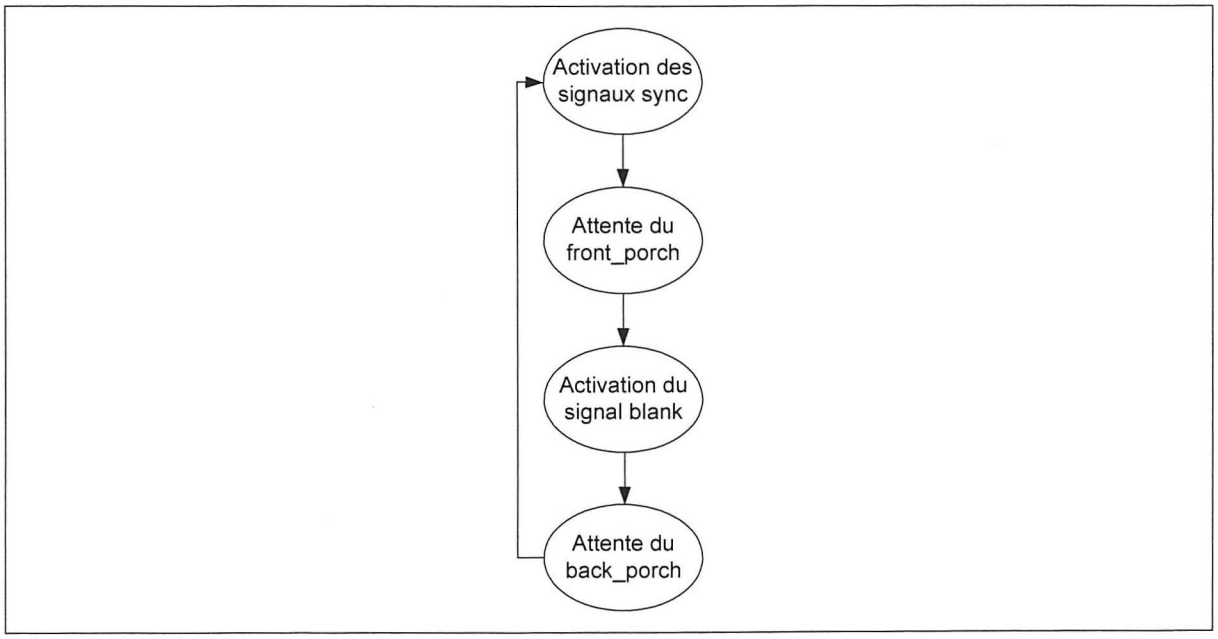

Figure 6.5 Diagramme d'états de la génération des signaux de synchronisation vidéo.

À la mise sous tension de la carte SP305, le FPGA se programme à partir de la PROM en mode serial master, ce qui signifie que le FPGA génère lui-même son horloge de configuration. Cette étape correspond au premier état de la Figure 6.3. Une fois la

configuration terminée, le système s'initialise, le module clk_gen se verrouille sur l'horloge principale externe pour générer les deux horloges internes, mclk et pclk, nécessaires au bon fonctionnement du système, ce qui correspond au deuxième état du diagramme de la Figure 6.3.

Par la suite, le module bootloader prend le contrôle de la mémoire PROM afin de charger les paramètres de configurations du contrôleur VGA et ensuite charger les images dans la mémoire ZBT avec pipeline (par l'intermédiaire du module pixel_gen). Nous verrons plus loin le fonctionnement détaillé du module bootloader, cependant, il est bon de savoir que ce module génère une horloge pour la lecture de la mémoire PROM et lit bit par bit les données reçues de cette dernière. Un registre interne au bootloader permet d'assembler les bits et d'identifier un patron qui détermine l'emplacement du début des données à charger, ce qui correspond à l'état trois de la Figure 6.3. Le signal pat_ok s'active lorsque le patron est détecté (DEL Err1 de la carte SP305).

Les premières données à charger par le bootloader sont des paramètres qui sont chargés dans l'ordre : Le nombre d'octet à charger, la configuration du module VGA et un champ réservé. Ces trois données constituent l'entête qu'on nomme aussi Header (État quatre du diagramme d'état de la Figure 6.3).

On identifie le nombre de bits par pixel dans la donnée de configuration VGA, ce qui nous permet de faire le chargement de la table des couleurs lorsque les images sont de 8 bits. Autrement, on passe directement au chargement des données d'image. À la fin du chargement des données de la mémoire vidéo, un signal fin_dload s'active. Le bootloader active alors les fonctions d'affichage du module VGA, ce qui termine le diagramme d'état de la Figure 6.3.

Le module VGA permet l'affichage de deux images préchargées et sélectionnées en mémoire ZBT. C'est le bootloader (ou tout autre host ou processeur) qui envoie sa sélection d'image au module ctrl_register dans un registre interne. Cette sélection permet donc de commencer la lecture en mémoire ZBT à l'adresse du premier pixel de cette image, ce qui correspond au premier état du diagramme de la Figure 6.4. Chaque pixel lu en mémoire ZBT est par la

suite envoyé dans le module pixel_gen pour y être correctement formaté avant de sortir sur le port RGB. Les pixels passent un à un dans les fifos du module pixel_gen. Les pixels sont chargés tant que les fifos ne sont pas pleins et on fait sortir un pixel à la fois vers le convertisseur vidéo lorsque le signal pixel_out_en est actif (ce qui correspond grossomodo au signal blank). L'image est terminée lorsque les signaux hdone et vdone, provenant du module sync_gen, s'activent. Le module sync_gen génère aussi les signaux de synchronisation horizontale, verticale et composite permettant le contrôle de l'affichage sur l'écran VGA. Les paramètres de temps de ces signaux sont donnés par l'intermédiaire du groupes de variables génériques dont le type timing_h_v_t est définit dans le fichier types_p.vhd. La génération des signaux de synchronisation se fait dans l'ordre décrit par le diagramme d'état de la Figure 6.5. Pour plus de détails sur la génération des signaux VGA, référez-vous aux sections 5.1, 6.11.4 et 6.11.5.

Mentionnons aussi que le module VGA communique avec les modules voisins par l'intermédiaire de deux bus Wishbone. Un pour l'interface avec le module host (bootloader) et un autre pour l'interface avec la mémoire ZBT. Ce qui nous amène au module wb2zbt_wraper qui sert uniquement à adapter le bus wishbone aux ports de la mémoire vidéo ZBT. L'explication détaillée de chacun des modules est donnée dans les sections qui suivent.

6.6 Résultats des simulations du module top

Voici la simulation fonctionnelle du module top :

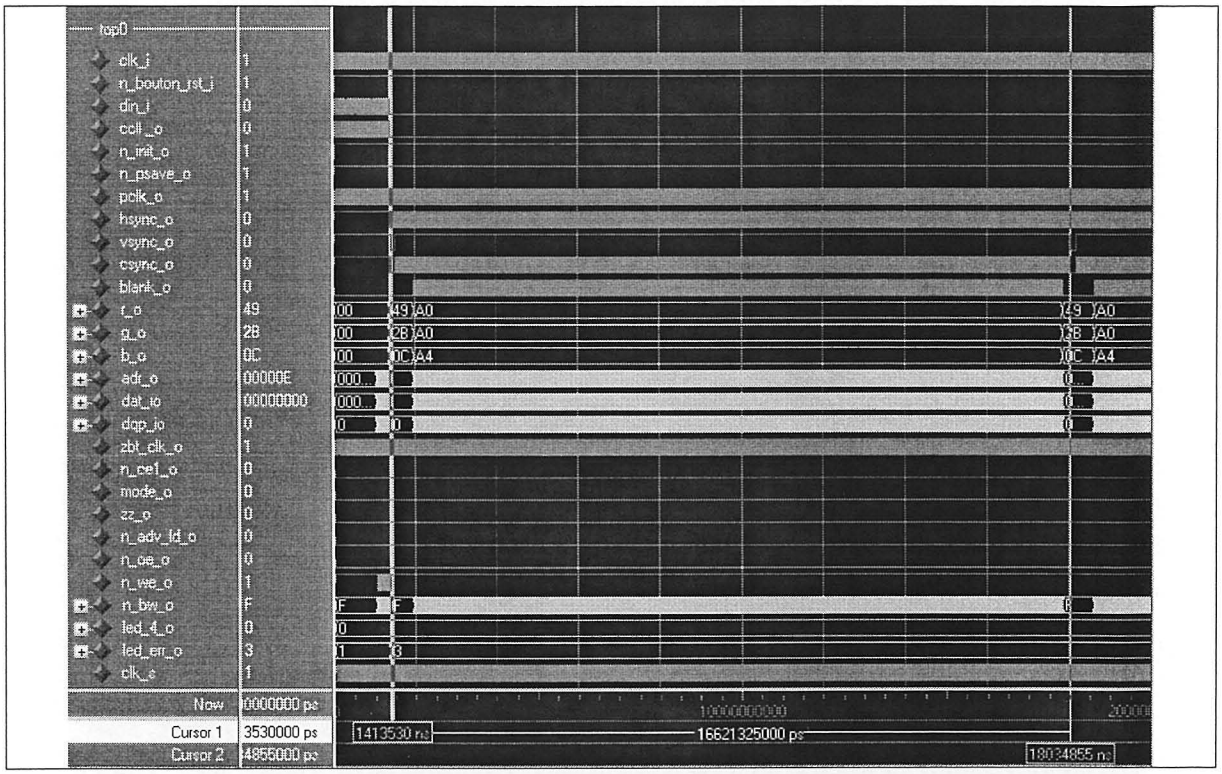

Figure 6.6 Vue générale de la simulation du module top.

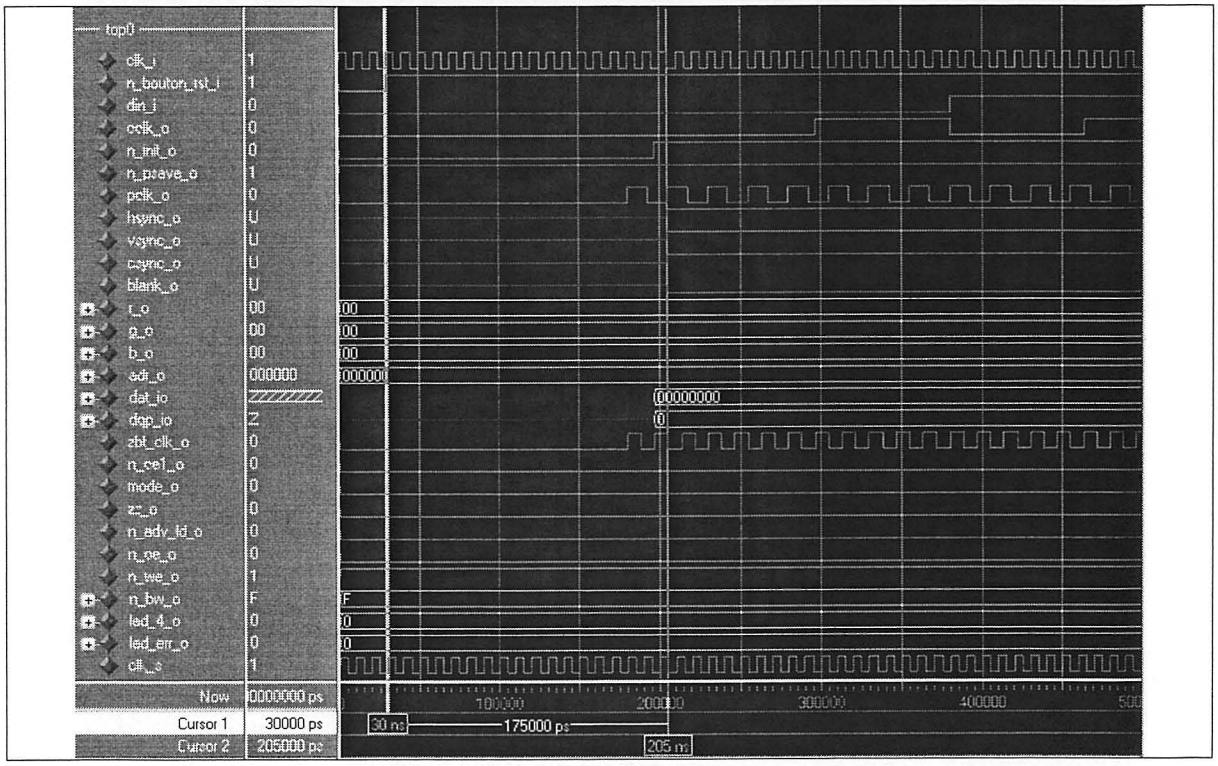

Figure 6.7 Simulation de l'initialisation de départ du module top.

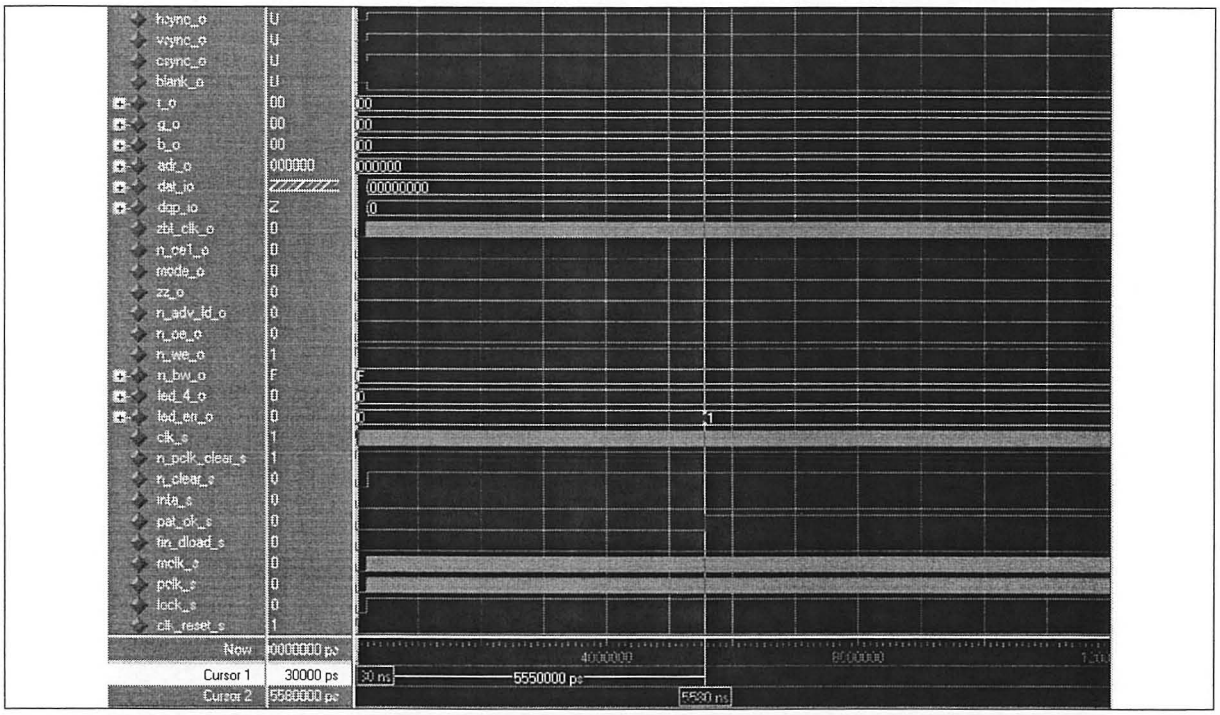

Figure 6.8 Simulation de la détection du patron de bits dans la PROM par module top.

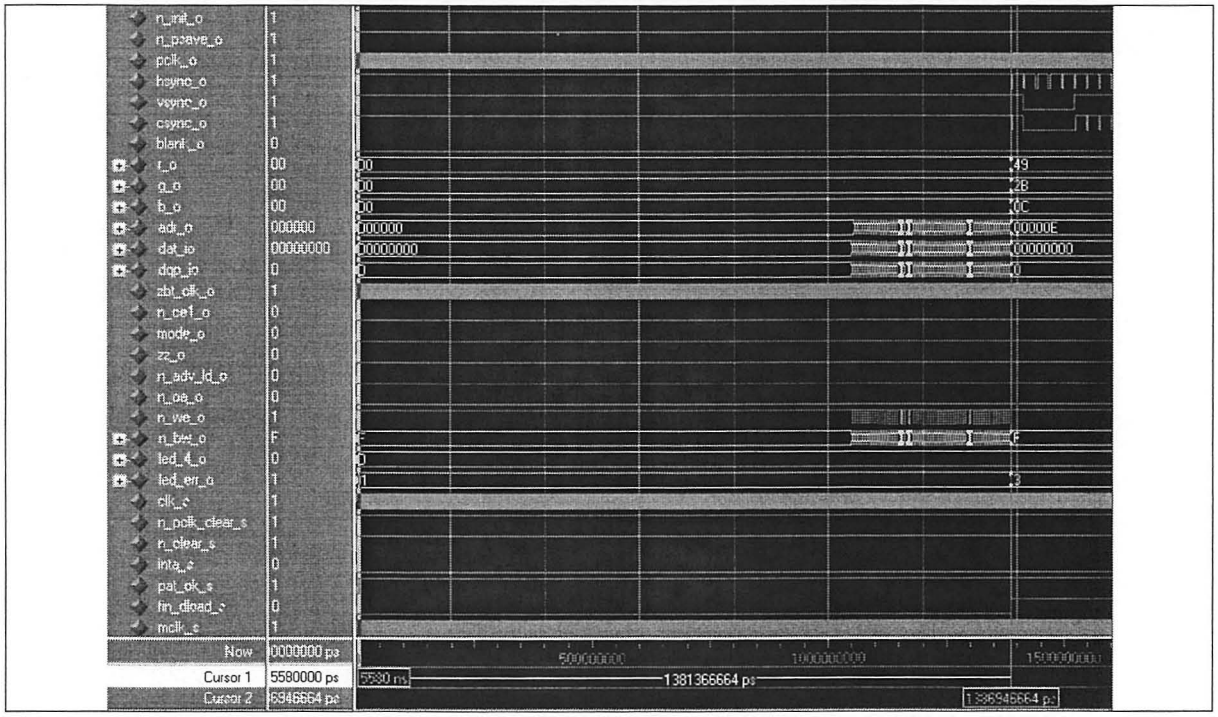

Figure 6.9 Simulation du chargement de la mémoire et activation du contrôleur vidéo dans le module top.

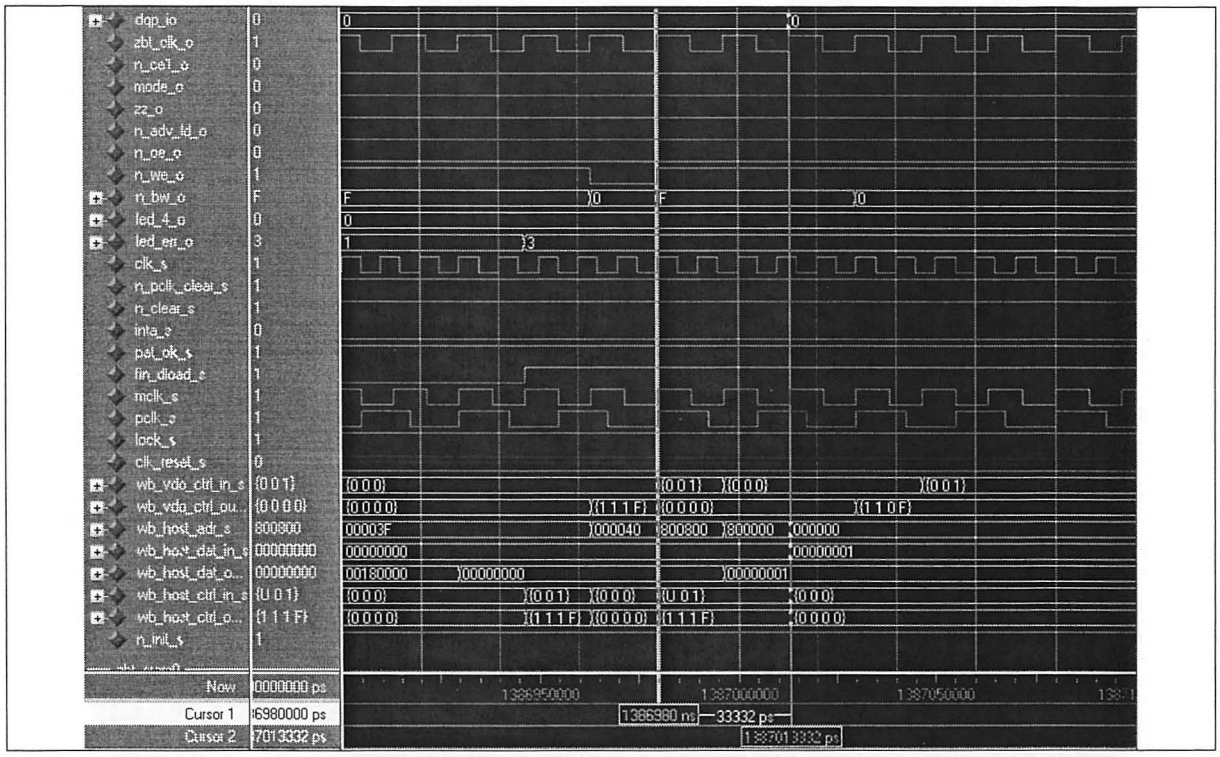

Figure 6.10 Vue détaillée de l'activation du contrôleur vidéo dans le module top.

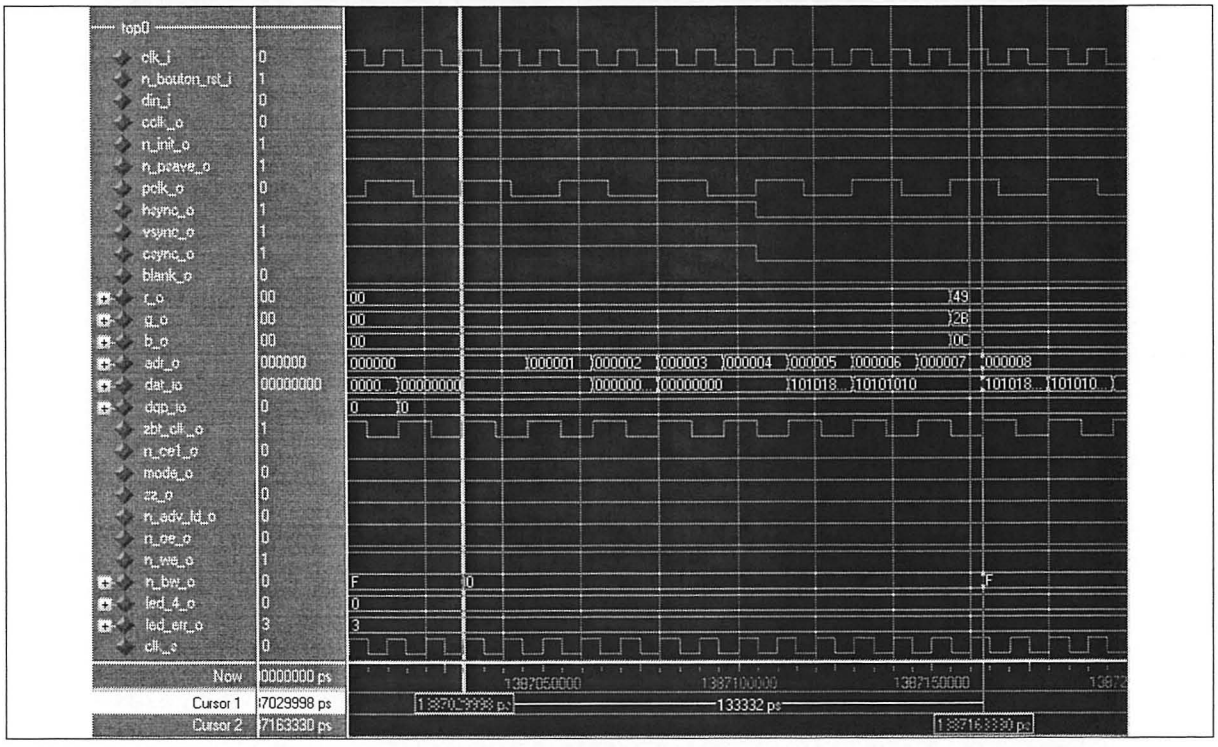

Figure 6.11 Vue d'ensemble du chargement des Fifos par les données mémoire dans le module top.

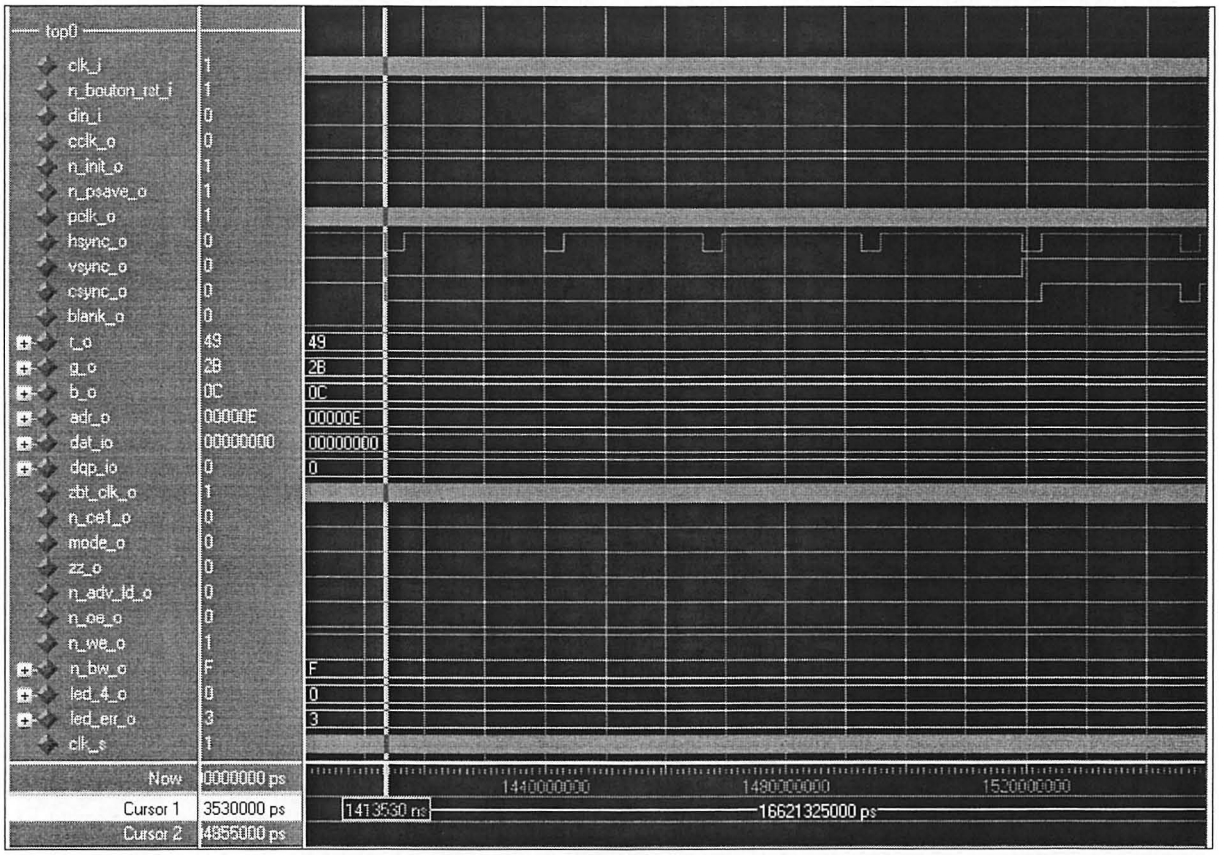

Figure 6.12 Simulation de la génération des signaux sync vertical et horizontal dans le module top.

La Figure 6.6 est une vue générale de la simulation du module top. On peut y remarquer que la première étape, suite à une mise sous tension du système, est le chargement de la mémoire par le bootloader. Nous ne sommes pas en mesure de voir la préparation des horloges sur cette image. En gros, le curseur numéro un correspond à l'activation du contrôleur vidéo qui est la dernière étape du diagramme de la Figure 6.3. Sur la simulation de la Figure 6.6, on peut y voir que l'image recommence toujours avec le même pixel 49-2B-0C qui correspond au premier pixel de l'image affiché dans la simulation. S'il en était autrement, ce serait que le contrôleur ne fonctionne pas correctement. On peut aussi observer que les deux curseurs se trouvent au début d'une impulsion du signal vsync, donc l'intervalle de temps entre les curseurs correspond à la durée de l'affichage d'une image, soit de 16,6 ms pour une image rafraîchie 60 fois par seconde (60 Hz).

La Figure 6.7 correspond à l'initialisation de départ du module top, ce qui correspond à la deuxième étape du diagramme de la Figure 6.3. On peut voir sur cette simulation au curseur numéro un que le signal n_bouton_rst actif bas engendre une remise à zéro de tout le système. Après quelques cycles d'horloge (environ 10), les modules DCM du système reprennent la génération des deux horloges nécessaires au bon fonctionnement de l'affichage et le signal n_init, qui permet de remettre la mémoire PROM à zéro, se désactive (à la valeur 1). À cet instant précis, nous sommes au curseur numéro deux et le système poursuit avec la détection du patron de bit qui permet de trouver l'emplacement des données dans la mémoire PROM.

La Figure 6.8 montre d'ailleurs très bien, au curseur numéro deux, le moment où le patron de bits est détecté. En effet, à cet instant le signal pat_ok s'active et la DEL branchée sur le signal led_err(0) s'allume. Le système charge alors l'entête de 12 octets et toutes les autres données par la suite.

La Figure 6.9 montre le chargement des données jusqu'à l'activation du contrôleur vidéo au temps du curseur numéro deux. Le délai entre l'activation du signal pat_ok au curseur numéro un et le début du chargement de la mémoire ZBT correspond en fait au chargement de la table des couleurs dans le module clut. En effet, cette figure ne permet pas de visualiser cette étape, mais elle sera discutée plus loin dans les sections relatives à la table des couleurs (module pixel_processor section 6.11.5.5 et module clut section 6.11.5.5.3). Enfin, entre 1 ms et le curseur numéro deux, on distingue très bien le chargement de la mémoire sur les signaux de contrôle de celle-ci.

En faisant un agrandissement de la Figure 6.9 vis-à-vis le curseur numéro deux, on obtient la Figure 6.10. Cette dernière figure nous montre le chargement de la dernière donnée en mémoire au curseur numéro un, suivi par la commande pour mettre l'adresse de départ de l'image dans le ctrl_register (wb_host_adr = 0x800800 et wb_host_dat_out = 0x00000000) puis de l'adresse d'activation du contrôleur vidéo ((wb_host_adr = 0x800000 et wb_host_dat_out = 0x00000001). A partir de ce moment, le contrôleur vidéo entre en fonction et devient autonome pour l'affichage des images. Il remplit ses fifos et active ses

signaux de synchronisation vidéo. On verra en détail ces étapes à partir du module VGA (Section 6.11) et dans ses sous composantes.

6.7 Description du module clk_gen (clk_gen.vhd)

Voici le symbole du module clk_gen :

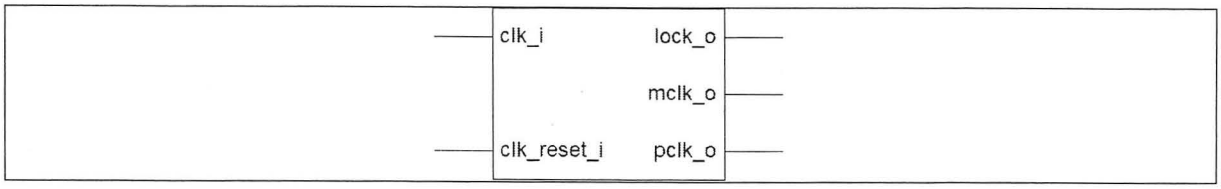

Figure 6.13 Symbole du module clk_gen.

Avant de décrire le fonctionnement du module clk_gen, voici un tableau de ses entrées et de ses sorties :

Tableau 6.8

Description des entrées et des sorties du module clk_gen

Nom	Entrée/ Sortie	Description
clk_reset	E	Signal de remise à zéro des générateurs d'horloges. Ce signal est généré par le module reset_gen (voir section 6.9) à partir de l'horloge externe clk pour s'assurer du bon fonctionnement des DCM.
clk	E	Signal d'horloge externe au FPGA. Cette horloge est générée par la carte SP305 à la fréquence de 100 MHz.
pclk	S	Signal d'horloge pixel qui permet la synchronisation des signaux destinés aux sorties VGA. Elle se doit d'être adaptée avec précision pour donner la résolution et le taux de rafraîchissement désiré de l'image.
mclk	S	Signal d'horloge maîtresse distribuée dans tout le FPGA. Elle se doit d'être plus rapide que l'horloge pclk afin d'assurer une bande passante intéressante pour la lecture en mémoire ZBT et le remplissage adéquat des Fifos.
lock	S	Signal généré par les deux générateurs d'horloge afin d'avertir que la génération des horloges est stabilisée et que les systèmes qui dépendent des ces horloges peuvent commencer leurs opérations sans problème.

6.7.1 Description du fonctionnement du module clk_gen

Le module clk_gen génère les deux signaux d'horloges nécessaires au bon fonctionnement du système en entier. Le signal d'horloge mclk est l'horloge rapide pour tous les modules de

traitement, de contrôle et de communication en général. Le signal d'horloge pclk est l'horloge adaptée à la résolution de l'image et au taux de rafraîchissement désiré de l'écran sur lequel s'effectue l'affichage de l'image. Cette horloge permet donc la synchronisation des pixels avec les signaux VGA. Un changement de domaine d'horloge est nécessaire dans certains modules afin de s'assurer de l'intégrité des signaux en tout temps. Les détails seront expliqués en temps et lieux. Voici le schéma détaillé du module clk_gen :

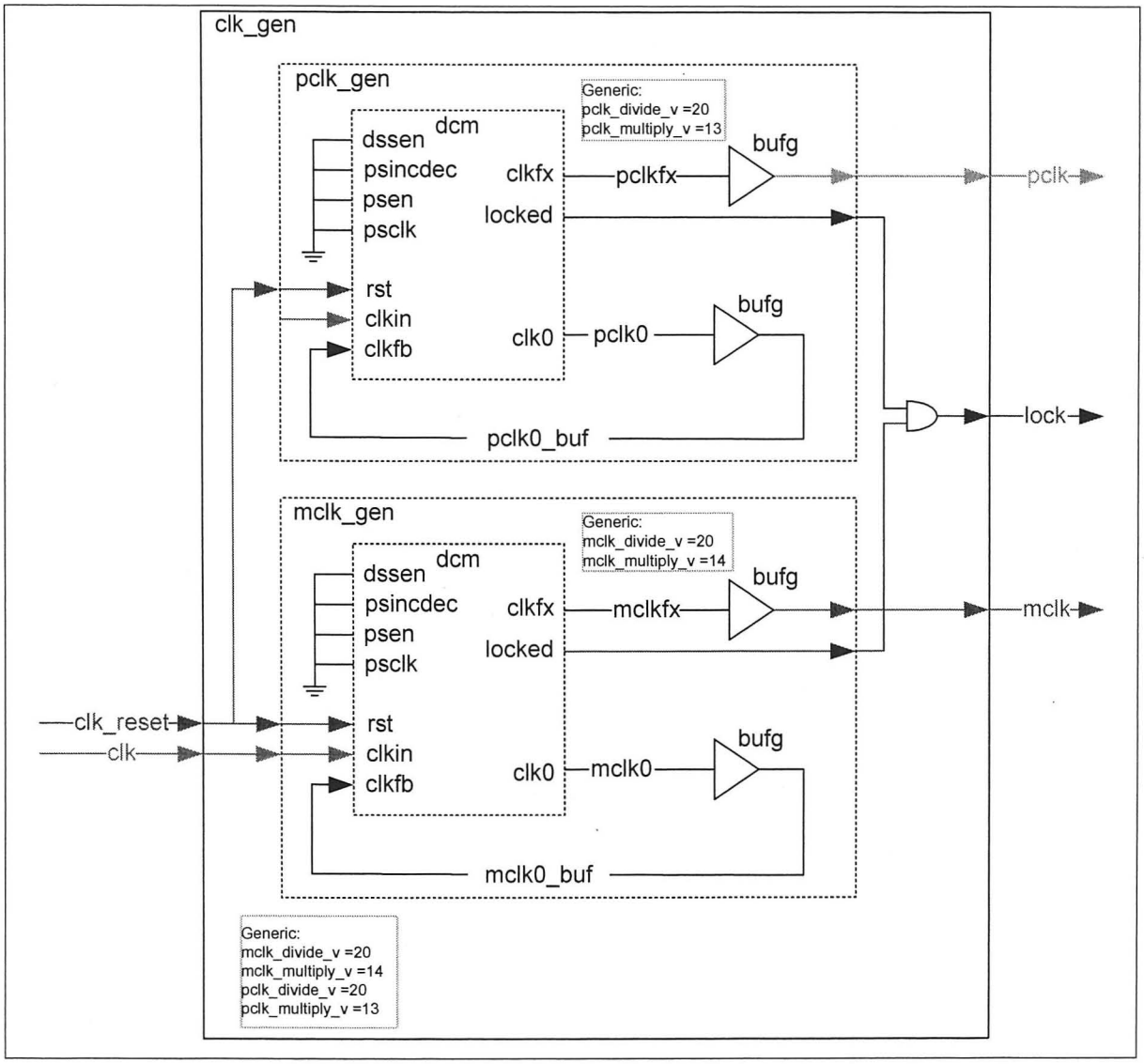

Figure 6.14 Schéma du module clk_gen et de ses sous-modules mclk_gen et pclk_gen.

6.7.2 Résultats des simulations du module clk_gen

Voici la simulation fonctionnelle du module clk_gen :

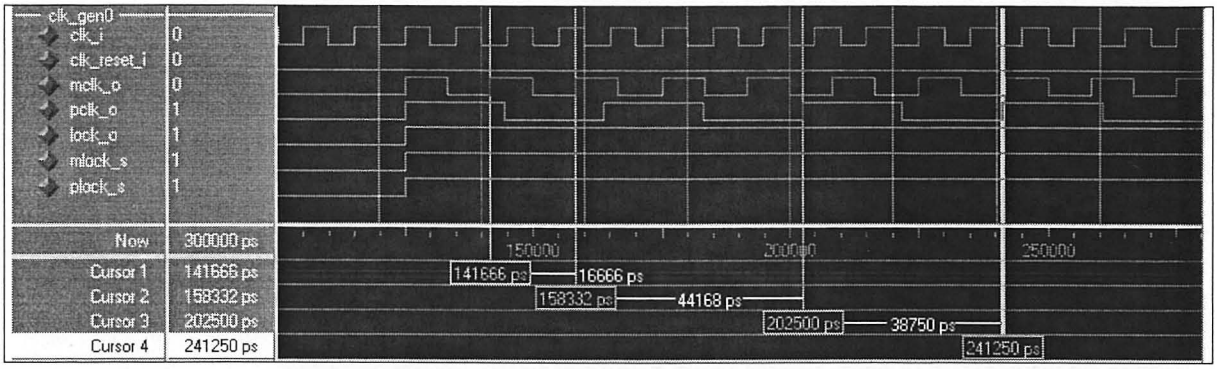

Figure 6.15 Simulation de la génération des horloges par le module clk_gen.

La Figure 6.15 montre bien la génération des deux horloges par le module clk_gen. En effet la fréquence de mclk recherchée étant de 60 MHz, le résultat de la génération de l'horloge par le DCM donne une période de 16 666 ps donc une fréquence de 60,002 MHz. Pour ce qui est de pclk, on recherche une fréquence de 25,17 MHz, on obtient une période de 38 750 ps donc une fréquence de 25.8 MHz. On a donc une variation de 2,5% pour pclk et de 0.003% pour mclk, ce qui est acceptable. Pour de meilleur résultat il suffit de changer les paramètres génériques des DCM et de jouer avec la fréquence d'entrée.

6.7.3 Description du module pclk_gen (pclk_gen.vhd)

Voici le symbole du module pclk_gen

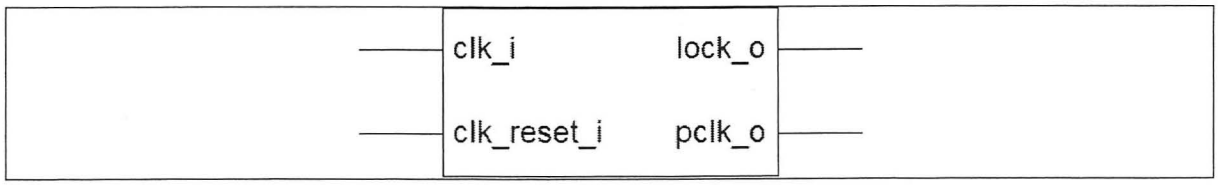

Figure 6.16 Symbole du module pclk_gen.

Avant de décrire le fonctionnement du module pclk_gen, voici un tableau de ses entrées et de ses sorties :

Tableau 6.9

Description des signaux importants du module pclk_gen

Nom	Entrée/ Sortie	Description
clk_reset	E	Signal de remise à zéro des générateurs d'horloges. Ce signal est généré par le module reset_gen (voir section 0 ci-dessous) à partir de l'horloge externe clk pour s'assurer du bon fonctionnement des DCM.
clk	E	Signal d'horloge externe au FPGA. Cette horloge est générée par la carte SP305 à la fréquence de 100 MHz.
pclk	S	Horloge pixel générée à la fréquence voulue pour donner la bonne résolution et le bon taux de rafraîchissement de l'écran.
lock	S	Signal qui s'active lorsque le signal d'horloge généré par le module DCM est stable.
pclkfx	Interne	Signal de sortie du DCM qui possède la fréquence voulue de l'horloge, mais qui n'est pas encore placé sur le bus général de distribution d'horloge. La fréquence d'horloge générée est donnée par la formule suivante : freq_pclk = freq_clk * pclk_multiply / pclk_divide où pclk_multiply et pclk_divide sont des variables entières génériques données en paramètre au module pclk_gen et appliquées comme attributs dans le module DCM afin de générer la bonne fréquence de sortie.
pclk0	Interne	Signal de sortie du module DCM nécessaire pour donner une rétroaction du signal d'horloge généré. Ce signal doit passer dans un suiveur général (bufg) afin de générer le même délai de propagation que la sortie d'horloge pclk.
pclk0_buf	Interne	Signal identique au signal pclk0, mais décalé d'un délai provoqué par un suiveur général (bufg).

6.7.3.1 Description du fonctionnement du module pclk_gen

Le module pclk_gen permet de générer l'horloge pixel à partir d'un module DCM interne du FPGA. Pour une utilisation adéquate des modules DCM, veuillez vous référer aux documents [59] Xilinx, 2005 et [66] Xilinx, 2006.

6.7.3.2 Résultats des simulations du module pclk_gen

Voici la simulation fonctionnelle du module pclk_gen :

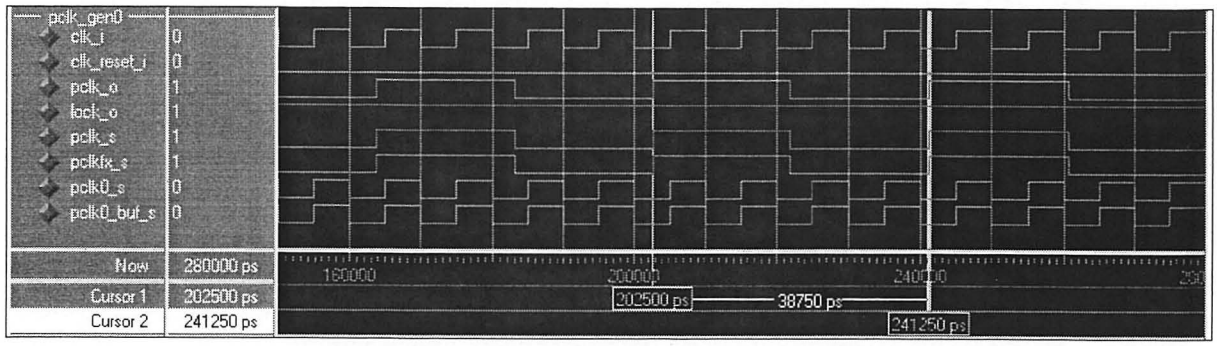

Figure 6.17 Simulation du module pclk_gen.

La Figure 6.17 montre bien la génération de l'horloge pclk. En effet la période de pclk obtenue est de 38 750 ps, ce qui nous donne une fréquence de 25,8 MHz comparativement à la fréquence recherchée de 25,17 MHz. On a donc une variation de 2,5% pour pclk, ce qui est acceptable. Pour de meilleur résultat il suffit de changer les paramètres génériques des DCM et de jouer avec la fréquence d'entrée.

6.7.4 Description du module mclk_gen (mclk_gen.vhd)

Voici le symbole du module mclk_gen :

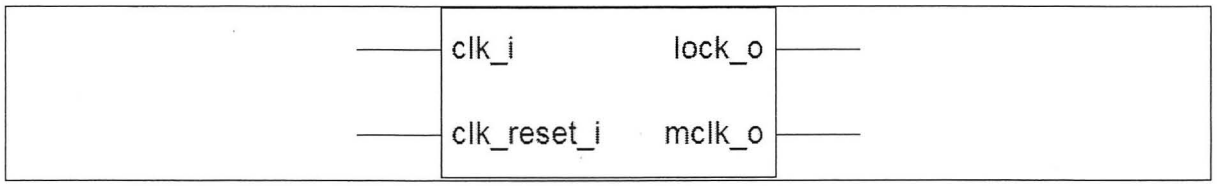

Figure 6.18 Symbole du module mclk_gen.

Avant de décrire le fonctionnement du module mclk_gen, voici un tableau de ses entrées et de ses sorties :

Tableau 6.10

Description des entrées et des sorties du module mclk_gen

Nom	Entrée/ Sortie	Description
clk_reset	E	Signal de remise à zéro des générateurs d'horloges. Ce signal est généré par le module reset_gen (voir section 6.9) à partir de l'horloge externe clk pour s'assurer du bon fonctionnement des DCM.
clk	E	Signal d'horloge externe au FPGA. Cette horloge est générée par la carte SP305 à la fréquence de 100 MHz.
mclk	S	Sortie d'horloge maîtresse générée à la fréquence voulue pour utiliser la bande passante maximale pour la communication avec le host ou la mémoire ZBT. La limite de cette fréquence est influencée par deux facteurs principaux : la mémoire ZBT utilisée et le délai critique généré par la synthèse du fichier *.bit du FPGA. La composante mémoire ZBT permet une fréquence maximale de fonctionnement de 200 MHz ce qui ne risque pas de nous mettre de bâtons dans les roues. Par contre, il sera nécessaire de contraindre convenablement le placement des différentes composantes internes du système afin d'optimiser les délais de propagation.
lock	S	Signal qui s'active lorsque le signal d'horloge généré par le module DCM est stable.
mclkfx	Interne	Signal de sortie du DCM qui possède la fréquence voulue de l'horloge, mais qui n'est pas encore placé sur le bus général de distribution d'horloge. La fréquence d'horloge générée est donnée par la formule suivante : freq_mclk = freq_clk *mclk_multiply / mclk_divide où mclk_multiply et mclk_divide sont des variables entières génériques données en paramètre au module mclk_gen et appliquées comme attributs dans le module DCM afin de générer la bonne fréquence de sortie.
mclk0	Interne	Signal de sortie du module DCM nécessaire pour donner une rétroaction du signal d'horloge généré. Ce signal doit passer dans un suiveur général afin de générer le même délai de propagation que la sortie d'horloge mclk.
mclk0_buf	Interne	Signal identique au signal mclk0, mais décalé d'un délai provoqué par un bufg (suiveur général).

6.7.4.1 Description du fonctionnement du module mclk_gen

Le module mclk_gen permet de générer l'horloge maîtresse à partir d'un module DCM interne du FPGA. Pour une utilisation adéquate des modules DCM, veuillez vous référer aux documents [59] Xilinx, 2005 et [66] Xilinx, 2006.

6.7.4.2 Résultats des simulations du module mclk_gen :

Voici la simulation fonctionnelle du module mclk_gen :

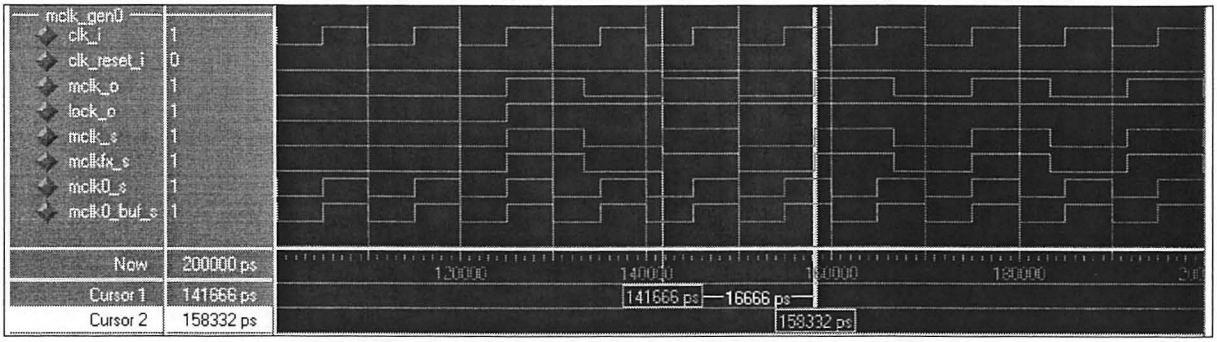

Figure 6.19 Simulation du module mclk_gen.

La Figure 6.19 montre bien la génération de l'horloge mclk. En effet la fréquence de mclk recherchée étant de 60 MHz, le résultat de la génération de l'horloge par le DCM donne une période de 16 666 ps donc une fréquence de 60,002 MHz. On a donc une variation de 0.003%, ce qui est excellent. Pour modifier les fréquences générées, il suffit de changer les paramètres génériques des DCM et de jouer avec la fréquence d'entrée.

6.8 Description du module wb2zbt_wraper (wb2zbt_wraper.vhd)

Voici le symbole du module wb2zbt_wraper :

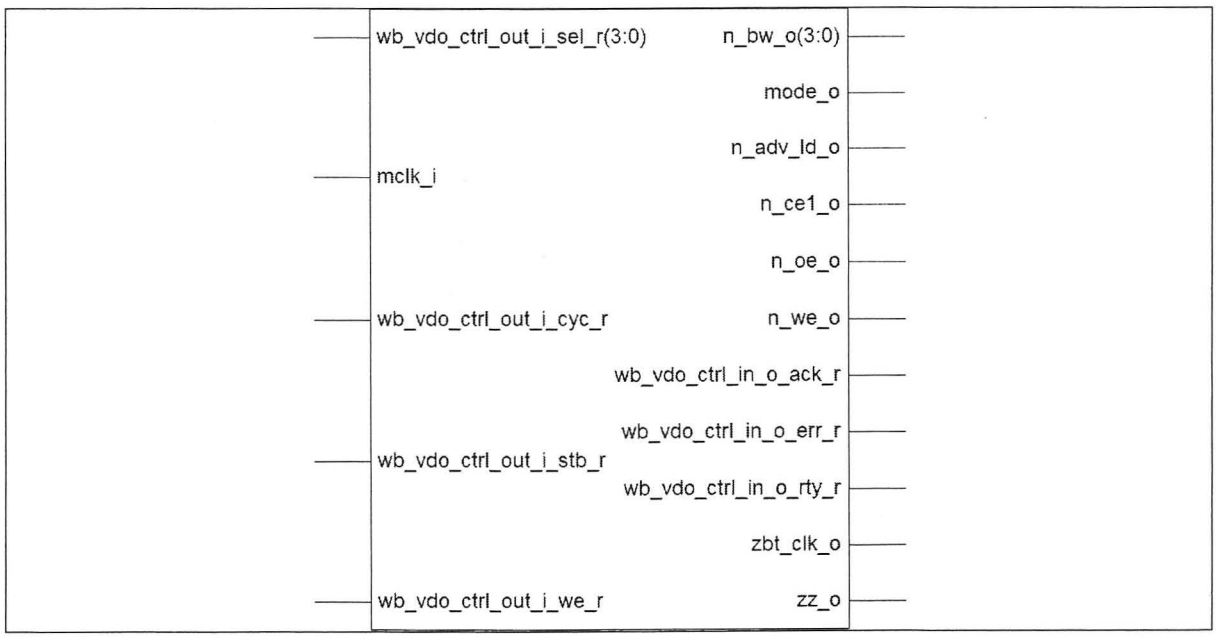

Figure 6.20 Symbole du module wb2zbt_wraper.

Avant de décrire le fonctionnement du module wb2zbt_wraper, voici un tableau de ses entrées et de ses sorties :

Tableau 6.11

Description des entrées et des sorties du module wb2zbt_wraper

Nom	Entrée/ Sortie	Description
wb_vdo_ctrl_out	E	Groupe de signaux contenant tous les signaux de contrôle du bus wishbone fourni par le maître (qui est dans ce cas-ci le module VGA). On y trouve donc les signaux suivants : • sel : ce signal sélectionne les octets actifs pour un transfert entre le maître et l'esclave. • we : signal de « write enable ». il permet l'écriture par le maître lorsqu'il est actif • cyc : signal de validation nécessaire pour toutes les transactions sur le bus Wishbone • stb : signal de sélection de l'esclave pour lequel la requête est destinée
wb_vdo_ctrl_in[3]	S	Groupe de signaux contenant tous les signaux de contrôle du bus wishbone reçus par le maître (qui est dans ce cas-ci le module VGA). On y trouve donc les signaux suivants : • ack : ce signal averti le maître que la requête a été prise en charge par l'esclave • err : signal d'erreur de transmission sur le bus • rty : signal qui averti le maître de réessayer sa requête plus tard (utilisé dans le cas où l'esclave serait incapable de répondre immédiatement
mclk	E	Horloge maîtresse, voir les sections 6.7 et 6.7.4 pour des détails sur l'horloge maîtresse
zbt_clk	S	Signal d'horloge fournie à la mémoire ZBT. C'est une copie conforme du signal mclk, l'horloge maîtresse.
n_bw	S	Signal inverse du signal sel provenant du bus Wishbone. Il a la même utilité et entre dans la mémoire ZBT. Présentement, les lectures et les écritures en mémoire se font toujours sur 32 bits pour rendre les accès mémoires efficaces au maximum. Donc en théorie, on pourrait placer ce signal en permanence à la valeur « 0000 ». Cependant, pour des raisons de possibilité d'extension du système, nous avons gardé la fonctionnalité dynamique de ce signal.
n_we	S	Signal we inversé, il a la même fonctionnalité et entre dans la mémoire ZBT.
n_oe	S	Signal « output enable » qui permet d'activer les sorties de la mémoire ZBT. Ce signal est ignoré par la mémoire lors d'une écriture (n_we = 0). Nous pouvons donc le garder actif en permanence, ce qui réduit les problèmes de timing liés à la latence de réactivation des sorties.
n_adv_ld	S	Signal permettant de charger une adresse si actif (bas) ou de forcer une incrémentation automatique si inactif (haut). Le mode d'incrémentation automatique pourrait être utilisé en mode rafale (burst mode), mais ce mode est inutile dans notre cas car notre module VGA est en mesure de générer une adresse à chaque cycle d'horloge.
mode	S	Permet de mettre l'incrémentation automatique en mode linéaire ou en mode interlacée (inutilisé dans notre cas).
zz	S	Permet de mettre la mémoire en mode sommeil. Nous n'utilisons pas pour le moment cette fonctionnalité, mais il serait possible de l'ajouter

		éventuellement.
n_ce1	S	Signal de sélection de la mémoire. Étant donné que la mémoire ZBT est la seule composante sur le bus qui communiquer avec le module VGA, nous gardons ce signal toujours activé (bas).

6.8.1 Description du fonctionnement du module wb2zbt_wraper

Le module wb2zbt_wraper est uniquement un module d'adaptation entre le bus Wishbone et les broches de connexions avec la mémoire ZBT. Le projet a été pensé de manière à ce qu'uniquement ce module soit modifié dans le cas ou un autre type de mémoire serait utilisée (en autant qu'elle possède aussi un pipeline de deux cycles). Pour plus de détails sur la fonctionnalité de la mémoire ZBT, référez-vous aux documents [9] Cypress, 2004 et [10] Cypress, 2004.

Le processus interne permet de générer le signal ack de manière synchrone lorsque les signaux cyc et stb s'activent, ce qui donne déjà un cycle d'horloge sur le ack. Un deuxième cycle de latence sera nécessaire pour adapter la mémoire ZBT avec le circuit puisque notre mémoire vidéo possède un pipeline de 2 cycles d'horloge. Ce délai supplémentaire nécessaire est implicite dans le module VGA.

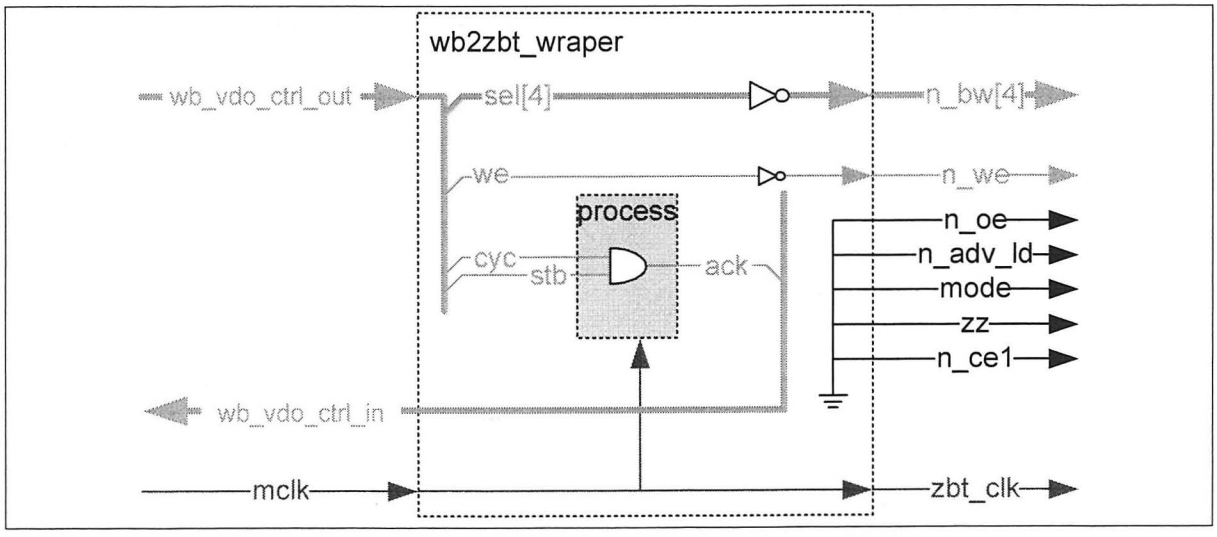

Figure 6.21 Schéma du module wb2zbt_wraper.

6.8.2 Résultats des simulations du module wb2zbt_wraper

Voici la simulation fonctionnelle du module wb2zbt_wraper :

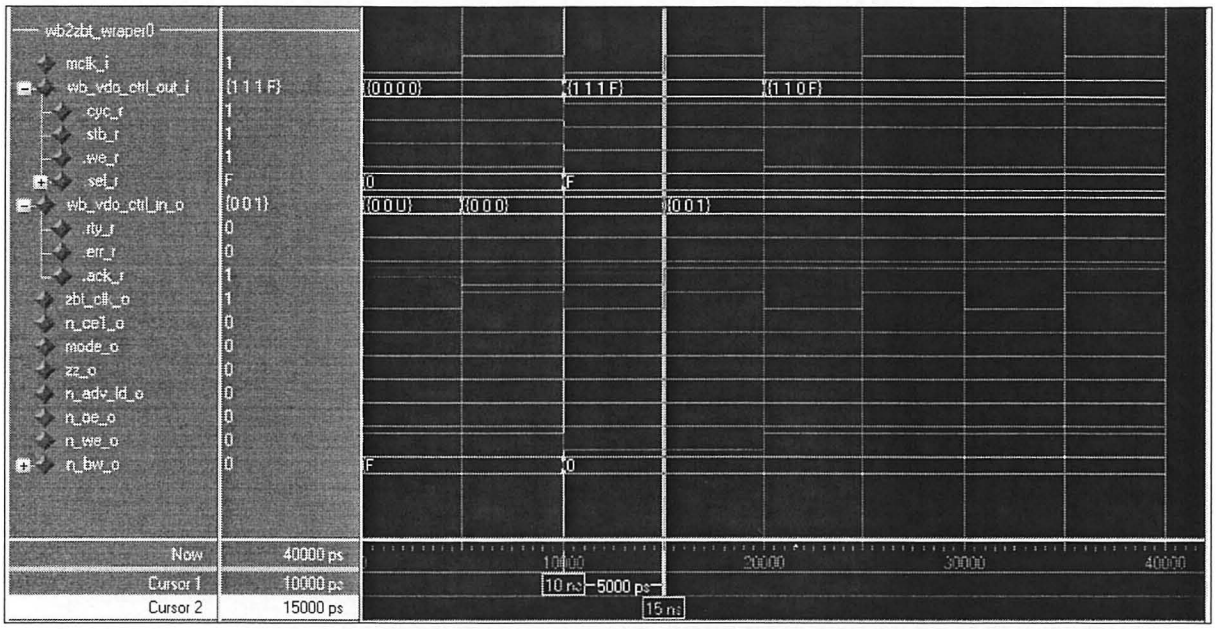

Figure 6.22 Simulation du module wb2zbt_wraper.

Le module wb2zbt_wraper est uniquement utiliser pour fixer les valeurs de base pour les signaux de la mémoire ZBT qui sont inutilisé. Il permet aussi de générer le signal ack sur le bus wishbone de manière synchrone pour assurer la lecture adéquate de la mémoire. Sur la Figure 6.22, on retrouve le signal ack qui s'active de manière synchrone lors d'une écriture. L'écriture est entamée au curseur numéro un et le ack s'active sur le curseur numéro deux.

6.9 Description du module reset_gen (reset_gen.vhd)

Voici le symbole du module reset_gen

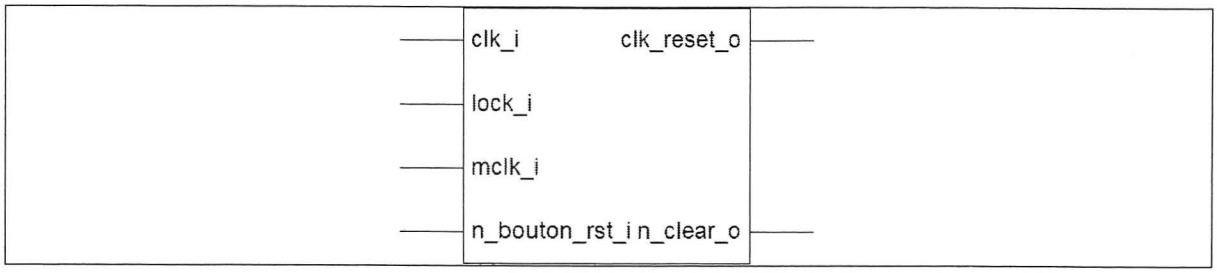

Figure 6.23 Symbole du module reset_gen.

Avant de décrire le fonctionnement du module reset_gen, voici un tableau de ses entrées et de ses sorties :

Tableau 6.12

Description des entrées et des sorties du module reset_gen

Nom	Entrée/ Sortie	Description
mclk	E	Signal d'horloge maîtresse, voir les sections 6.7 et 6.7.4 pour plus de détails.
clk	E	Signal d'horloge fixe, générée à l'extérieur du FPGA sur la carte SP305. Sa fréquence est de 100 MHz.
n_bouton_rst	E	Signal d'entrée provenant du bouton « reset » de la carte SP305. Il est actif bas. Ce signal déclenche le mécanisme de remise à zéro de tout le système. Même les images sont rechargées dans la mémoire ZBT à partir de la PROM lors d'une remise à zéro par ce bouton.
n_clear	S	Signal de remise à zéro pour tous les modules autres que les générateurs d'horloges. Il s'active de manière asynchrone, mais se désactive de manière synchrone avec au moins deux cycles d'horloges entre l'activation et la désactivation pour s'assurer que tous les délais sont respectés, surtout le délai du pipeline de l'horloge ZBT.
clk_reset	S	Signal de remise à zéro des générateurs d'horloges. Il est synchrone avec l'entrée d'horloge clk afin de permettre aux horloges mclk et pclk de se remettre à zéro même lorsqu'elles ne sont pas générées (ou verrouillées).

6.9.1 Description du fonctionnement du module reset_gen

Ce module a pour objectif de générer deux signaux de remise à zéro à partir d'un bouton actif bas relié au signal n_bouton_rst. Le signal clk_reset est synchrone sur l'entrée d'horloge fixe clk de la carte pour permettre une réinitialisation des générateurs d'horloges. Le signal n_clear est un signal de remise à zéro pour tous les autres modules. Il s'active de manière asynchrone, mais se désactive de manière synchrone avec au moins deux cycles d'horloges

entre l'activation et la désactivation pour s'assurer que tous les délais sont respectés, surtout le délai du pipeline de l'horloge ZBT. Voici un schéma bloc du module reset_gen :

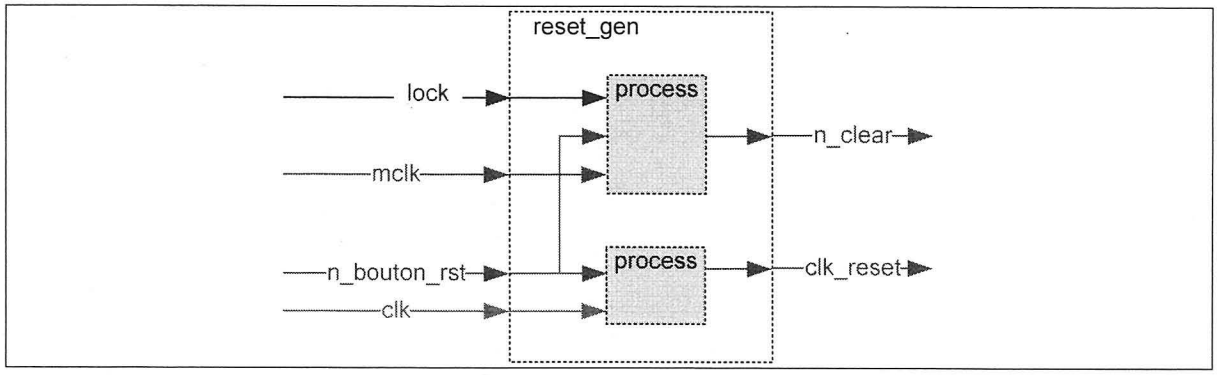

Figure 6.24 Schéma du module reset_gen.

6.9.2 Résultats des simulations du module reset_gen

Voici la simulation fonctionnelle du module reset_gen :

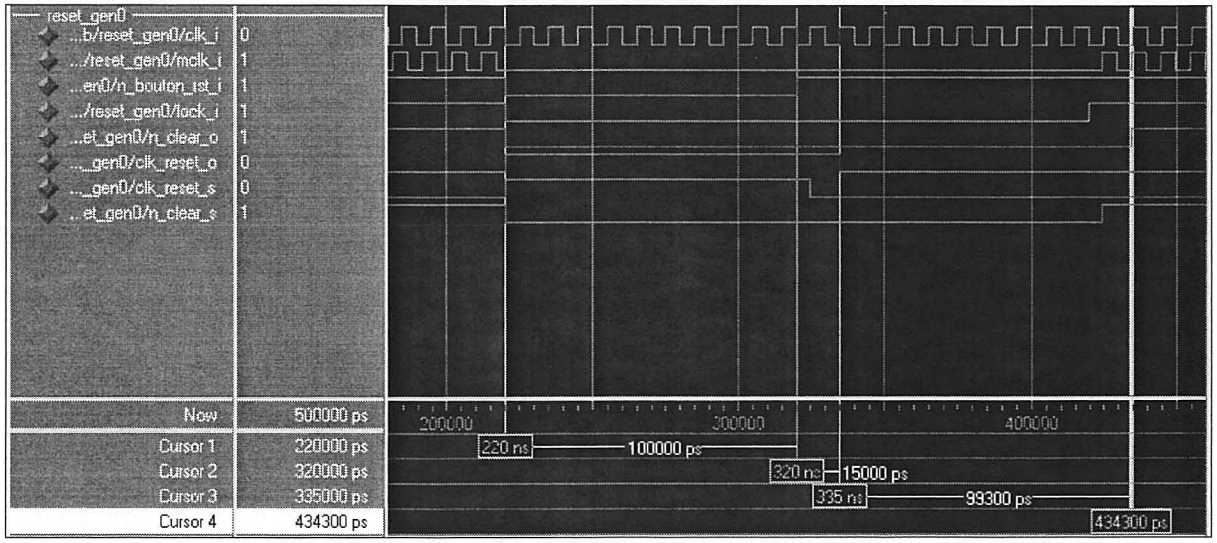

Figure 6.25 Simulation du module reset_gen.

La Figure 6.25 montre au curseur numéro un que le signal n_bouton_rst s'active à zéro, ce qui maintient le module en initialisation jusqu'au curseur numéro deux, soit tant que le signal n_bouton_rst est actif bas. Lorsque le signal n_bouton_rst devient inactif à 1, le signal clk_reset s'inactive de manière synchrone (curseur numéro trois), ce qui permet aux DCM de

générer les horloges à nouveau. Lorsque les DCM sont près, le signal lock s'active, ce qui permet de remettre le reste du système en fonction en rendant le signal n_clear inactif à « 1 » suite à deux fronts montant de mclk.

6.10 Description du module bootloader (bootloader.vhd)

Voici le symbole du module bootloader :

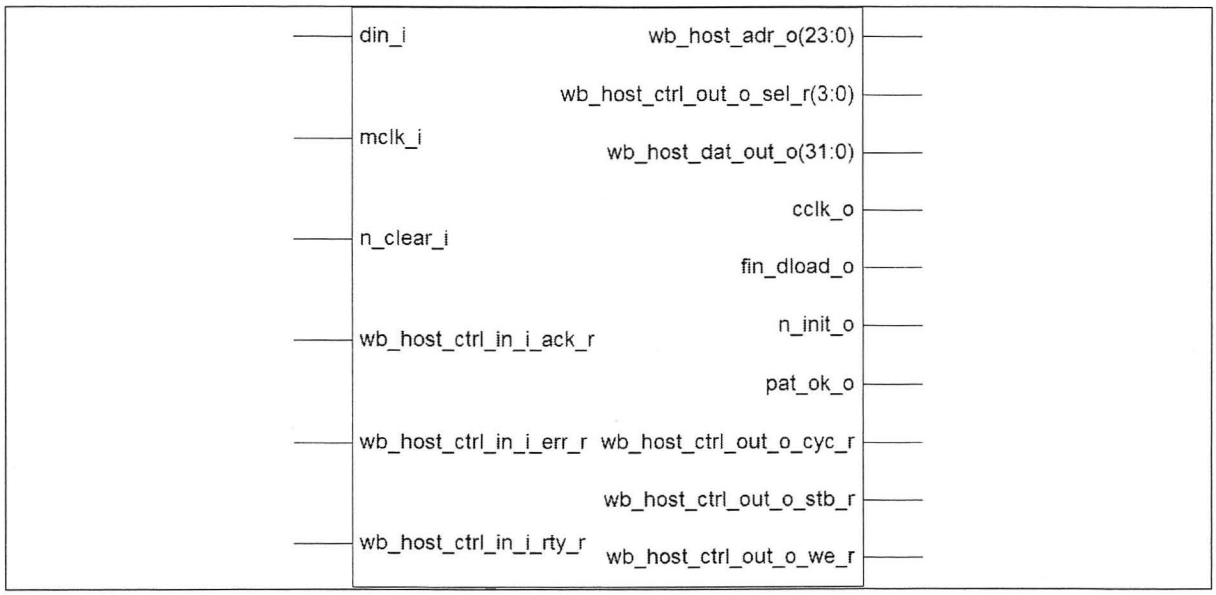

Figure 6.26 Symbole du module bootloader.

Avant de décrire le fonctionnement du module bootloader, voici un tableau de ses entrées et de ses sorties :

Tableau 6.13

Description des entrées et des sorties du module bootloader

Nom	Entrée/ Sortie	Description
mclk	E	Signal d'horloge maîtresse, voir les sections 6.7 et 6.7.4 pour plus de détails.
n_clear	E	Signal de réinitialisation générale du système. Voir la section 6.9 pour plus de détails.
din	E	Signal sériel qui sort de la mémoire PROM et qui se rend au module top0/bootloader0/pattern_reco_loader0. Il constitue les données sérielles à charger en mémoire.
cclk	S	Signal de pseudo horloge généré par le module /top0/bootloader0/cclk_gen0 par division de l'horloge mclk. Une division par cinq est effectuée par défaut, mais il est possible de l'optimiser pour qu'elle soit générée au maximum de la vitesse supportée par la mémoire PROM qui est de 40 MHz. Les paramètres de génération de cette pseudo horloge sont discutés à la section 6.10.4.
n_init	S	Signal de réinitialisation de la mémoire PROM. Lorsqu'il est actif (bas), le pointeur interne de la mémoire retourne au tout premier bit de la mémoire. Elle est donc prête pour une relecture complète. Lorsqu'il est inactivé (haut), un front sur l'entrée cclk permet d'incrémenter le pointeur interne d'un bit. Ce bit se présente alors à la sortie din. Lorsque le pointeur est en fin de course, il y restera jusqu'à ce que le signal n_init soit activé de nouveau. Ce signal est directement relié au signal n_clear.
pat_ok	S	Signal qui averti que le patron de bit recherché dans la PROM a été trouvé. Ce signal déclenche le processus de chargement des paramètres et des données de la PROM vers le module VGA. Il est de plus branché physiquement sur la DEL err0 de la carte SP305.
fin_dload	S	Signal qui indique que toutes les données d'images ainsi que tous les paramètres provenant de la PROM ont été chargées dans la mémoire vidéo (le contrôleur vidéo peut donc être activé). Ce signal met fin au chargement par le module bootloader. Ce signal est généré par le module bootloader0/dload_adr_gen0.
wb_host_adr[24]	S	Signal d'adresse du bus Wishbone du Host. Le maître est le module bootloader et l'esclave est le module VGA.
wb_host_dat_out[32]	S	Signal de donnée du bus Wishbone du Host. Le maître est le module bootloader et l'esclave est le module VGA.
wb_host_ctrl_out[7]	S	Groupe de signaux de contrôle du bus wishbone host. Ce groupe de signaux inclut entre autres les signaux cyc et stb qui permettent d'entamer une transaction sur le bus. Le signal sel permet de savoir la taille de la transaction. Pour plus de détails sur le fonctionnement du bus Wishbone, se référer au document [23] Herveille, 2002.
wb_host_ctrl_in[3]	E	Entrée de contrôle du bus Wishbone pour le module host (bootloader). C'est un groupe de signaux qui permet au module esclave de répondre au maître. Ce groupe de signaux comprend les signaux ack, err et rty. Le signal ack est le seul utilisé présentement et permet de dire au maitre que la transaction est acceptée.

6.10.1 Description du fonctionnement du module bootloader

Le module bootloader permet de charger la mémoire vidéo en faisant une lecture bit par bit dans la mémoire PROM. Il se doit de réordonnancer les bits de chaque donnée afin qu'elle soit valide (voir le module pattern_reco_loader à la section 6.10.3). Il permet aussi de contrôler d'une manière simple le module VGA (voir le module load_fsm à la section 6.10.6). C'est lui, entre autres, qui active l'affichage et qui configure les registres de contrôle du module VGA (dans le module ctrl_register). Voici un schéma bloc du module bootloader :

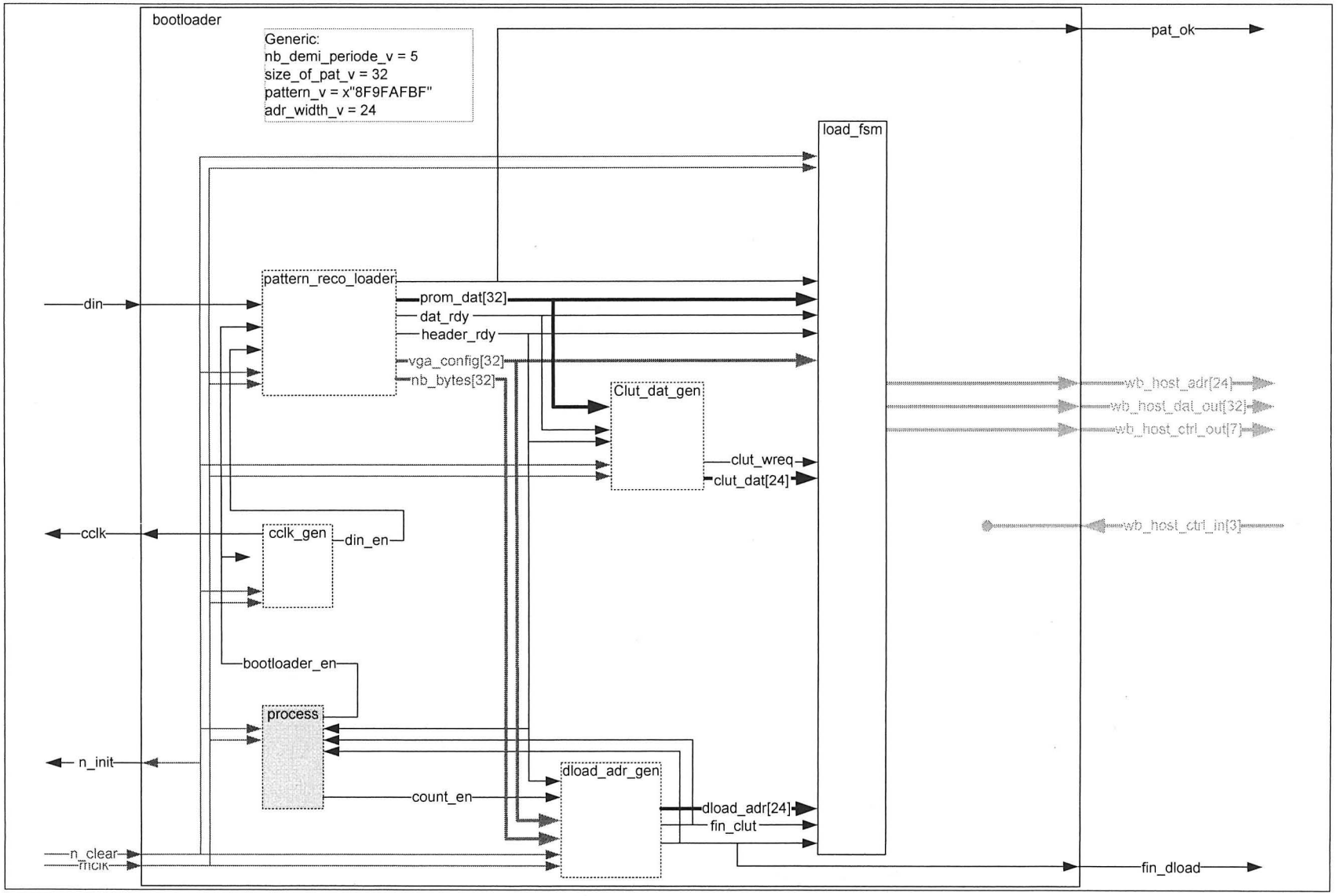

Figure 6.27 Schéma du module bootloader.

Le bootloader à comme première fonction de lire la mémoire PROM bit par bit et de retrouver les données d'images afin de les charger dans la mémoire vidéo. Pour ce faire, un patron de bit est inséré dans la PROM afin de retrouver facilement le début des données d'image. Suite au patron de bits, on a ajouté un entête qui se compose de 12 octets. Les quatre premiers octets donnent le nombre d'octets à lire, les quatre octets suivants donnent la configuration à envoyer au module VGA et identifient entre autres le nombre de bits d'encodage pour chaque pixel. Les quatre derniers octets de l'entête ne sont pas encore utilisés et sont gardés pour une utilisation future. Suite à l'entête, il y a deux possibilités : en mode 8 bits, l'entête est suivi de la table des 256 couleurs et par la suite des données de pixel, autrement, c'est-à-dire en mode 24 bits, ce sont les données de pixel à charger en mémoire vidéo qui suivent l'entête. Pour plus de détails sur le fonctionnement du bootloader, une explication des sous-modules est donnée dans les sections qui suivent :

6.10.2 Résultats des simulations du module bootloader

Voici la simulation fonctionnelle du module bootloader :

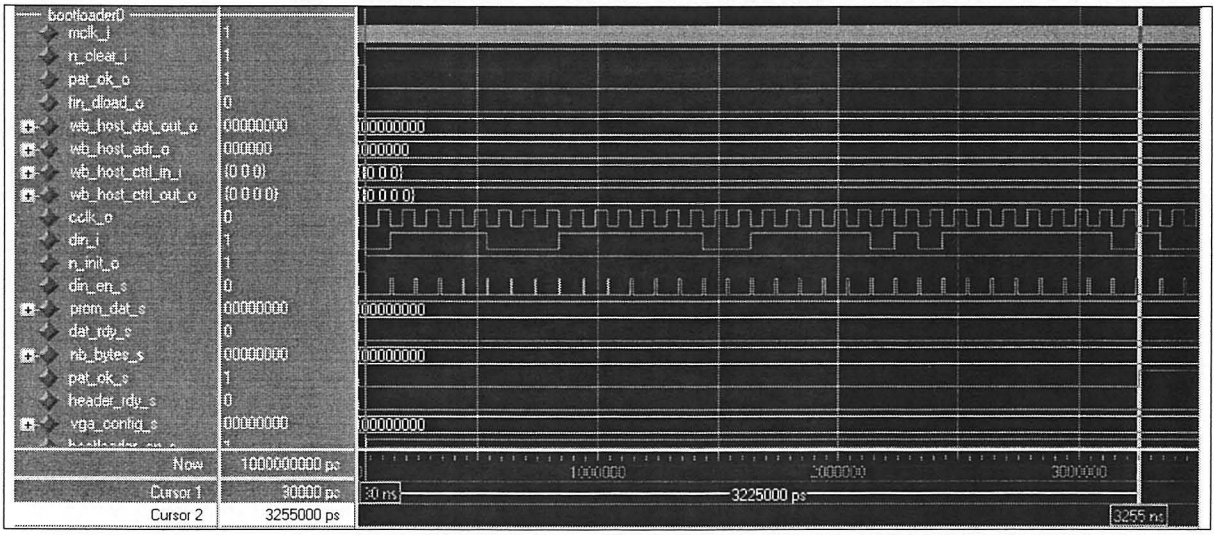

Figure 6.28 Simulation de la détection du patron de bits du module bootloader.

La figure précédente montre le train de bits du patron de 32 bits 0x8F9FAFBF qui est le patron à détecter et qui détermine le début des données utilisateur dans la mémoire PROM.

Le curseur numéro deux marque l'activation du signal pat_ok qui indique au restant du système que le début du chargement des données peut commencer.

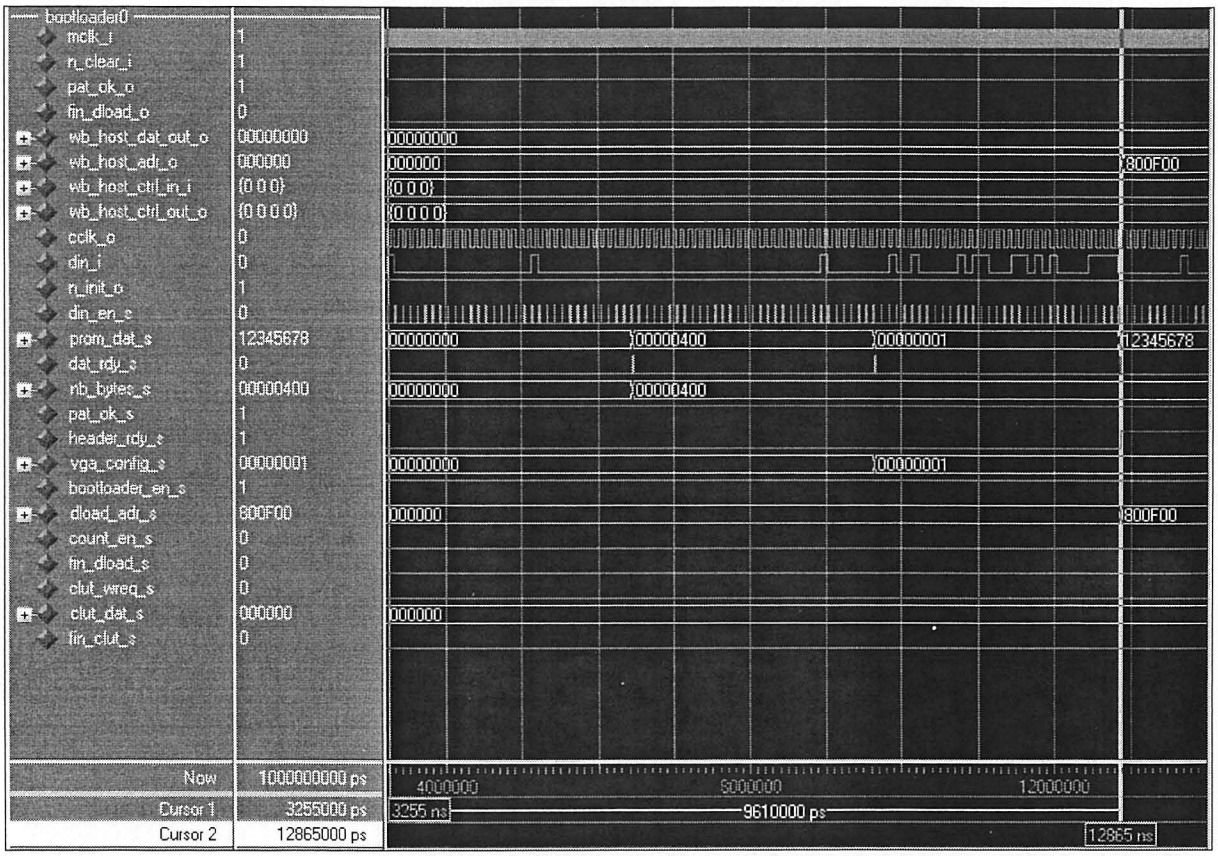

Figure 6.29 Simulation du chargement de l'entête par le module bootloader.

La figure qui précède montre la lecture de l'entête des données dans la PROM. En effet, si on observe le signal prom_dat, on remarque les données suivant : 0x00000400, 0x00000001 et 0x12345678 qui sont respectivement le nombre d'octets à charger, la trame de configuration de base du contrôleur vidéo et une trame de 32 bits réservé pour une utilisation future. On remarque aussi sur cette image que la valeur 0x00000400 est attribuée au signal nb_bytes et que la valeur 0x00000001 est attribuée au signal vga_config. Ce sont des registres du module bootloader qui permettent d'emmagasiner les valeurs de l'entête avant de passer au chargement des données pixel.

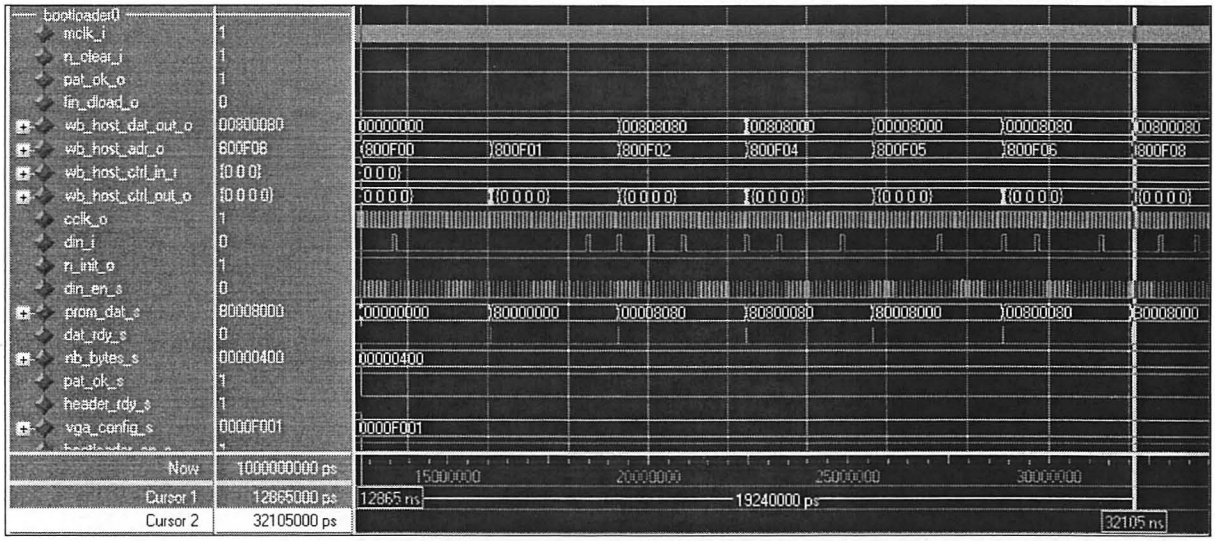

Figure 6.30 Simulation du chargement de la table des couleurs par le module bootloader.

La Figure 6.30 montre le début du chargement de la table des couleurs. Le curseur numéro un pointe où le bus d'adresse wb_host_adr possède la valeur 0x800F00, ce qui correspond à l'adresse de la première valeur de la table des couleurs (module clut). Les 8 bits les moins significatifs de cette adresse sont en fait l'emplacement des 256 couleurs de la table des couleurs. Sur cette figure, le bus d'adresse est incrémenté de un à chaque 24 bits de données lues (de la PROM) puisque la table des couleurs possède des données de 24 bits (8 bits pour le rouge, 8 bits pour le vert et 8 bits pour le bleu).

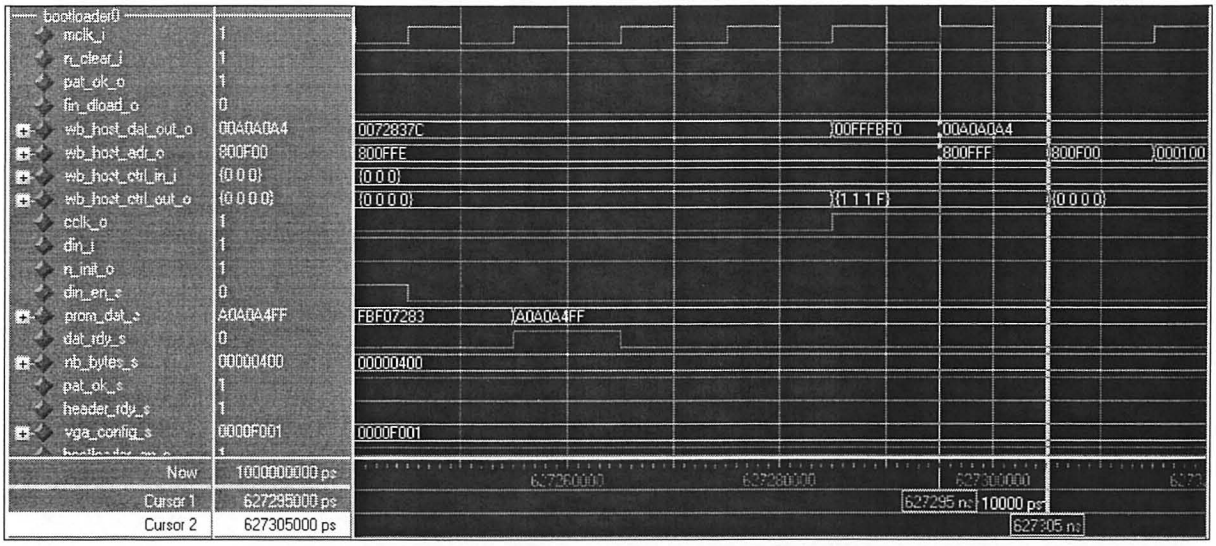

Figure 6.31 Simulation de la fin du chargement de la table des couleurs par le module bootloader.

La Figure 6.31 montre la fin du chargement de la table des couleurs. Le curseur numéro un pointe la dernière adresse de la table, soit l'adresse 0x800FFF. Suite à cela, le module bootloader se met à charger les pixels dans la mémoire vidéo à chaque 32 bits de données de la PROM, ce qu'on ne voit pas ici dans cette image. Vous pouvez voir le chargement de la première donnée de pixel en mémoire ZBT à la figure suivante où pointe le curseur numéro deux.

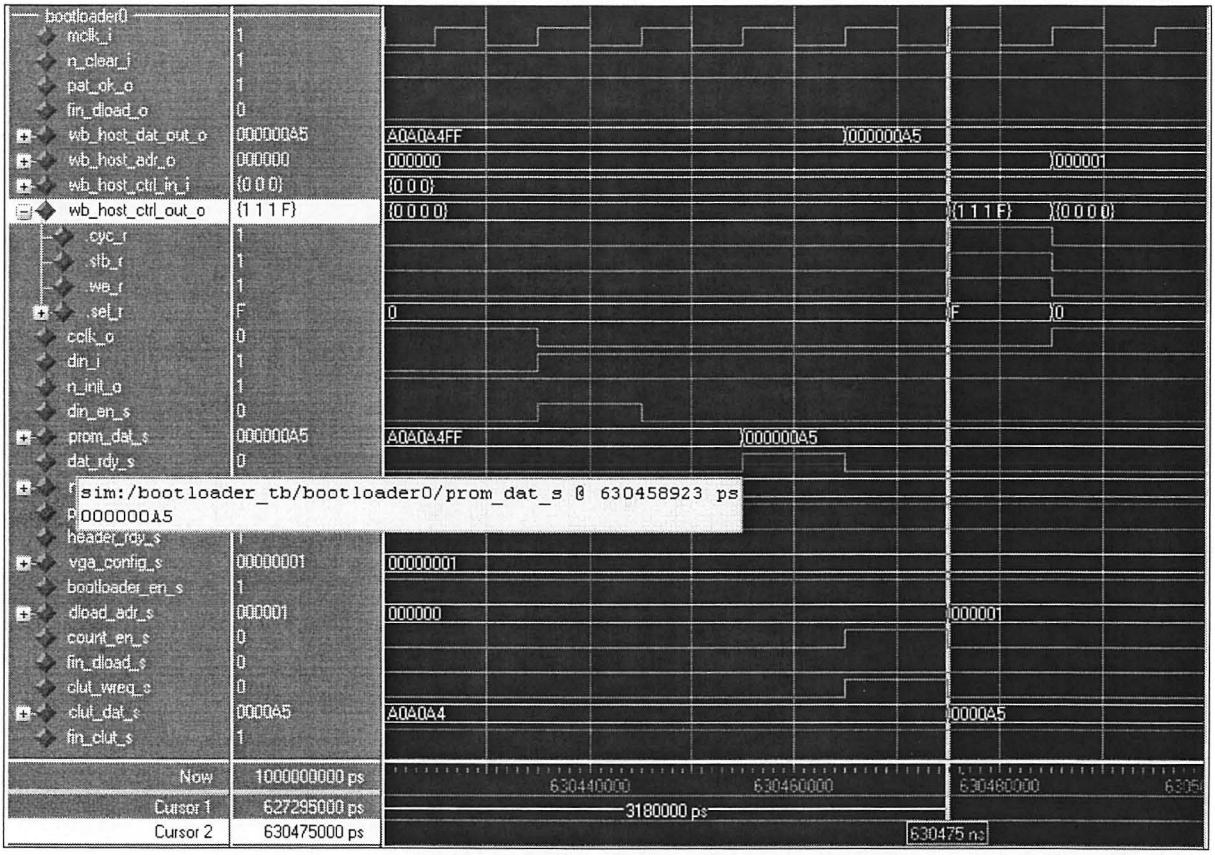

Figure 6.32 Simulation du début de chargement de la mémoire vidéo par le module bootloader.

Sur la Figure 6.32, vous pouvez remarquer que le bit le plus significatif de l'adresse est à zéro, ce qui identifie la destination de la donnée comme étant la mémoire vidéo. Le bootloader envoie la donnée à écrire en mémoire dès qu'il a accumulé les 32 bits sériels provenant de la mémoire PROM. Il active alors les bits cyc, stb et we du bus wishbone pour indiquer qu'une donnée est valide sur le bus. Il n'y a pas d'attente de vérification, ce qui signifie que la donnée doit être impérativement lue sur le champ par le module VGA lorsque le module bootloader l'indique. En d'autres mots, le module VGA n'a pas à activer de signal ack. Dans une version ultérieure, il sera nécessaire d'ajouter une gestion des transactions plus sophistiquée afin de réduire les possibilités d'erreurs de transmission.

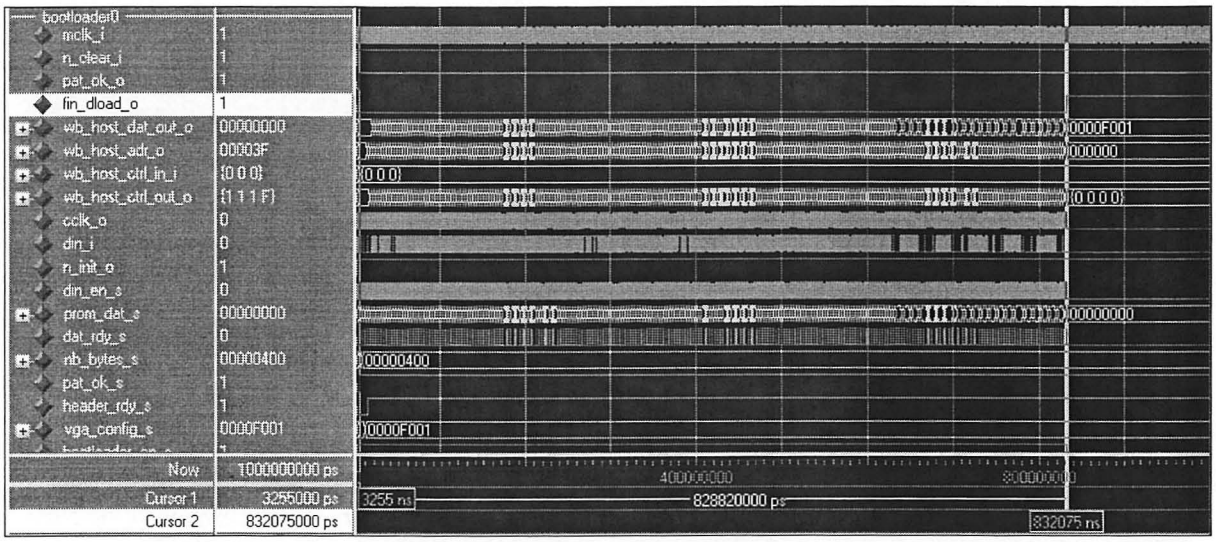

Figure 6.33 Vue générale de la simulation du module bootloader.

La figure précédente a pour unique but de vous montrer que le chargement des données provenant de la PROM se fait en une séquence sans interruption. Les bits sont lus de la PROM à une fréquence de 20 MHz sur le signal cclk (maximum possible de 40 MHz selon les spécifications de la PROM). Étant donné que la mémoire vidéo peut opérer à une fréquence de quatre à cinq fois plus élevée, le temps de traitement entre chaque octet est une éternité. Ici, pour la simulation, nous ne chargeons que les données de la table de couleurs et quelques données de pixels et nous atteignons quasiment 1 ms de temps de chargement. Le chargement de plusieurs images en mémoire peut donc prendre plusieurs millisecondes. Il serait donc intéressant de changer le mode de chargement sériel en un mode de chargement parallèle. Le mode selectMAP serait donc approprié et permettrait d'atteindre des vitesses de configuration d'au moins huit fois supérieure.

6.10.3 Description du module pattern_reco_loader (pattern_reco_loader.vhd)

Voici le symbole du module pattern_reco_loader :

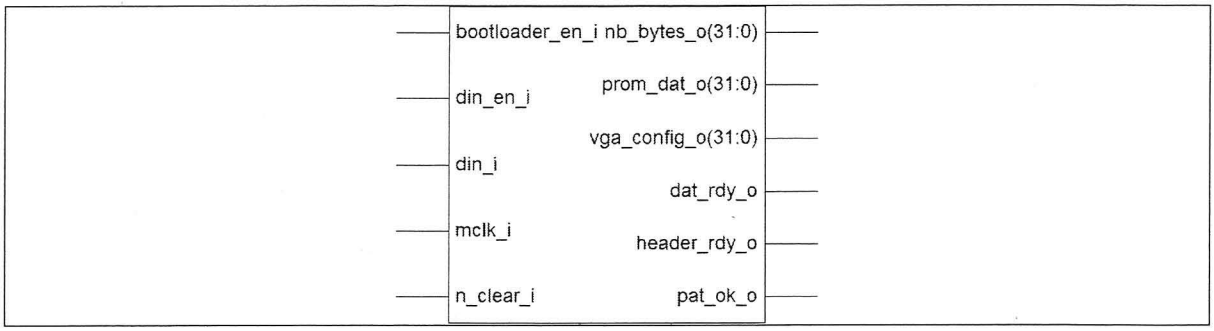

Figure 6.34 Symbole du module pattern_reco_loader.

Avant de décrire le fonctionnement du module pattern_reco_loader, voici un tableau de ses entrées et de ses sorties :

Tableau 6.14

Description des entrées et des sorties du module pattern_reco_loader

Nom	Entrée/ Sortie	Description
mclk	E	Signal d'horloge maîtresse, voir les sections 6.7 et 6.7.4 pour plus de détails.
n_clear	E	Signal de réinitialisation générale du système. Voir la section 6.9 pour plus de détails.
din	E	Entrée sérielle des données provenant de la PROM. Chaque bit de donnée est chargé par l'activation du signal din_en qui provient du module cclk_gen (section 6.10.4).
din_en	E	Signal qui provient du module cclk_gen (section 6.10.4) et qui permet de savoir quand l'entrée sérielle din est opérationnelle. ce signal est actif un seul cycle d'horloge mclk et s'active durant le premier cycle de mclk lorsque l'horloge de la PROM est basse.
bootloader_en	interne	Signal qui signifie aux différents modules internes du module bootloader que le travail du bootloader doit s'effectuer. Ce signal est généré dans le module bootloader (section 6.10). Il s'active tant que le chargement de toutes les données dans la PROM n'est pas terminé (signal fin_dload provenant du module dload_adr_gen, section 6.10.7).
pat_ok	S	Signal qui signifie que le patron de début des données a été détecté. Le chargement des différentes données peut donc s'effectuer.
vga_config[32]	S	Signal qui contient plusieurs paramètres de configuration du contrôleur VGA. Ce signal est envoyé au registre de contrôle ctrl_reg qui est à l'adresse 0h800000 du bus Wishbone et qui se situe dans le module top0/vga0/ctrl_register0. Pour des détails sur le signal ctrl_reg du module ctrl_register, voir la section 6.11.3.
nb_bytes[32]	S	Signal qui détermine combien d'octets le bootloader devra charger. Ce signal est

		propagé vers le module top0/bootloader0/dload_adr_gen0 qui permet de faire un décomptage des octets chargés (voir section 6.10.7).
prom_dat[32]	S	Signal des données de 32 bits qui proviennent directement de la PROM après leur réordonnancement.
dat_rdy	S	Signal qui signifie que les données du vecteur prom_dat sont valides.
header_rdy	S	Signal qui indique lorsque l'entête des données de la PROM est entièrement lu.

6.10.3.1 Description du fonctionnement du module pattern_reco_loader

Ce module permet de lire les données provenant de la PROM de manière sérielle, de réordonnancer chaque paquet de 32 bits reçus, de détecter les différents paramètres de chargement et de transmettre les données d'images au module load_fsm (section 6.10.6) qui est la machine à états finis de chargement du module bootloader (section 6.10). La figure suivante montre un schéma de connexion des différents processus internes du module pattern_reco_loader.

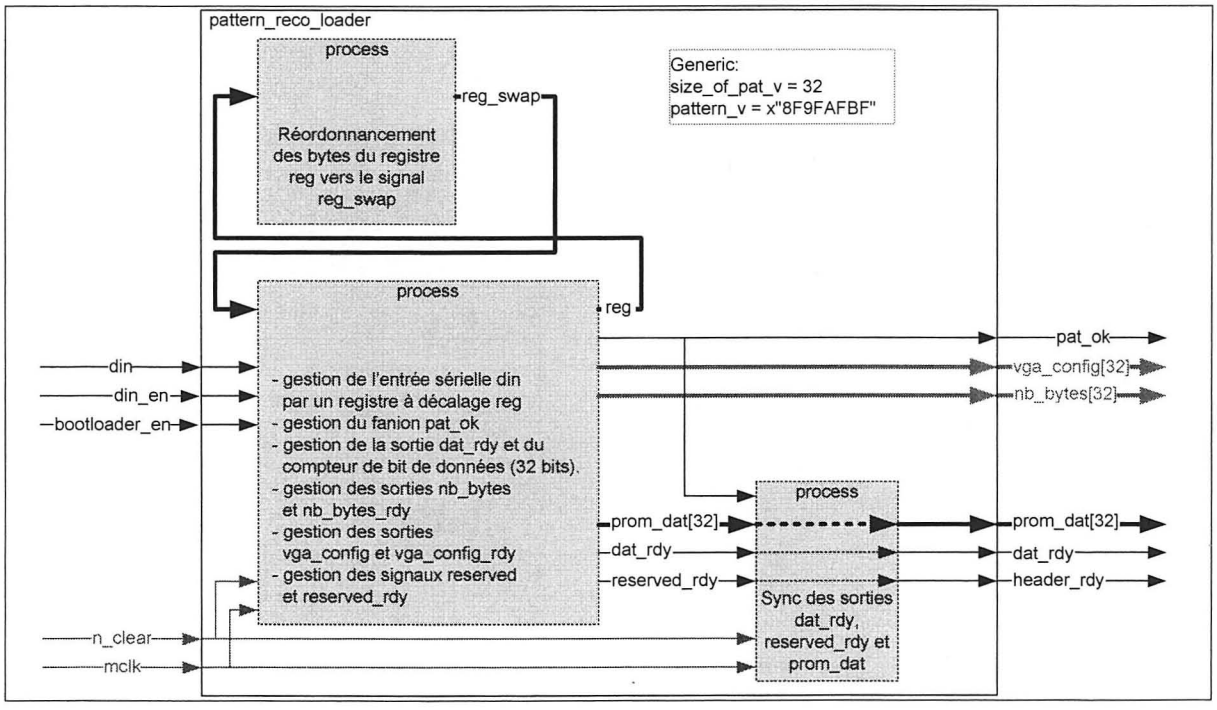

Figure 6.35 Schéma du module pattern_reco_loader.

Le premier processus est celui qui permet le réordonnancement des données reçues dans le registre de réception reg. En effet, la mémoire PROM emmagasine ses données de 8 bits selon un ordonnancement gros-boutiste (big-endian) alors que nous utilisons partout ailleurs

un ordonnancement petit-boutiste (little-endian). Pour plus de détail sur l'ordonnancement des données de la PROM et sur le principe de chargement de données utilisateur à partir de celle-ci, référez-vous aux documents [54] Xilinx, 2004 et [56] Xilinx, 2005.

Le processus principal, celui qui se trouve en bas à gauche sur la figure précédente, agit un peu comme une machine à états finis. Il permet de faire séquentiellement les étapes suivantes :

a. recherche du patron de bits qui identifie le début des données à charger et activation du signal pat_ok qui s'active lorsque le patron est détecté;
b. chargement de l'entête, ce qui inclut les champs de 32 bits suivants :
    - nombre d'octets d'image à charger (signal nb_bytes), ce qui n'inclut pas l'entête. Un signal interne nb_bytes_rdy s'active lorsque le nombre d'octets à charger est connu.
    - les paramètres configurables du contrôleur d'affichage (signal vga_config). Un signal interne vga_config_rdy s'active lorsque le vecteur de configuration est connu.
    - un champ inutilisé pour le moment et gardé pour une utilisation éventuelle (signal reserved). Un signal interne reserved_rdy s'active lorsque le champ est détecté. Ce signal reserved_rdy marque aussi la fin de l'entête dans la PROM.
c. chargement de la table des couleurs dans le cas d'une configuration d'affichage 8 bits;
d. chargement des données de pixel d'images.

Ce processus sert aussi au contrôle de synchronisation de l'entrée sérielle din des données provenant de la PROM.

Le dernier processus, celui situé en bas à droite de la figure précédente permet de synchroniser adéquatement les sorties prom_dat, dat_rdy et header_rdy.

6.10.3.2 Résultats des simulations du module pattern_reco_loader

Voici la simulation fonctionnelle du module pattern_reco_loader :

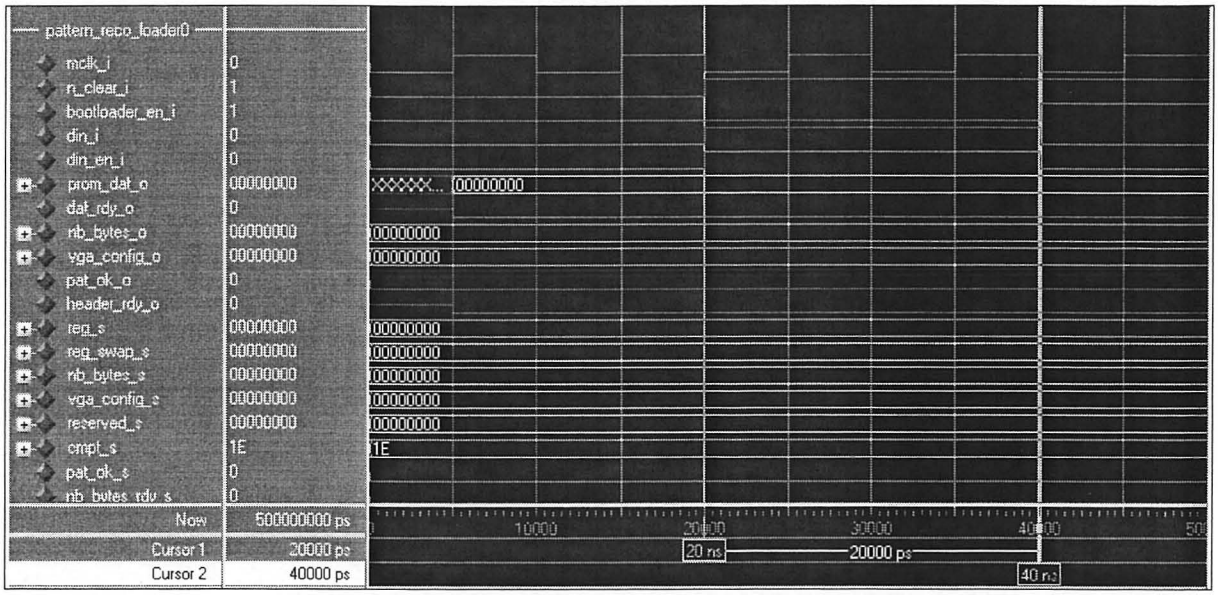

Figure 6.36 Simulation de l'initialisation du module pattern_reco_loader.

La figure précédente montre l'initialisation du module pattern_reco_loader. Ce module se met automatiquement en recherche du patron de bits qui délimite le début des données dans la PROM. C'est pourquoi le signal pat_ok est désactivé après l'initialisation.

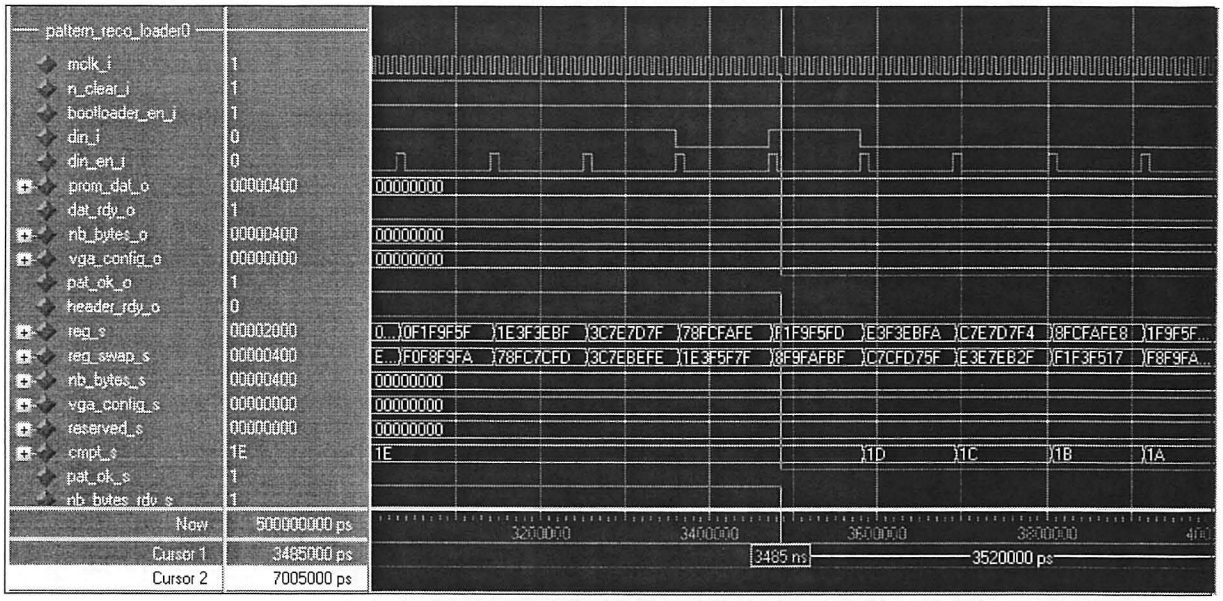

Figure 6.37 Simulation de la détection du patron de bits par le module pattern_reco_loader.

La Figure 6.37 montre bien la détection du patron de bits qui détermine le début des données utilisateur dans la PROM. Le patron à détecter est la valeur de 32 bits 0x8F9FAFBF. Aussitôt que cette trame apparaît sur le signal reg_swap, le signal pat_ok s'active et le chargement des données débute selon l'ordre de présentation du texte qui suit (vous pouvez aussi vous référer à la Figure 6.3 qui exprime la séquence de chargement utilisée par le module bootloader en général).

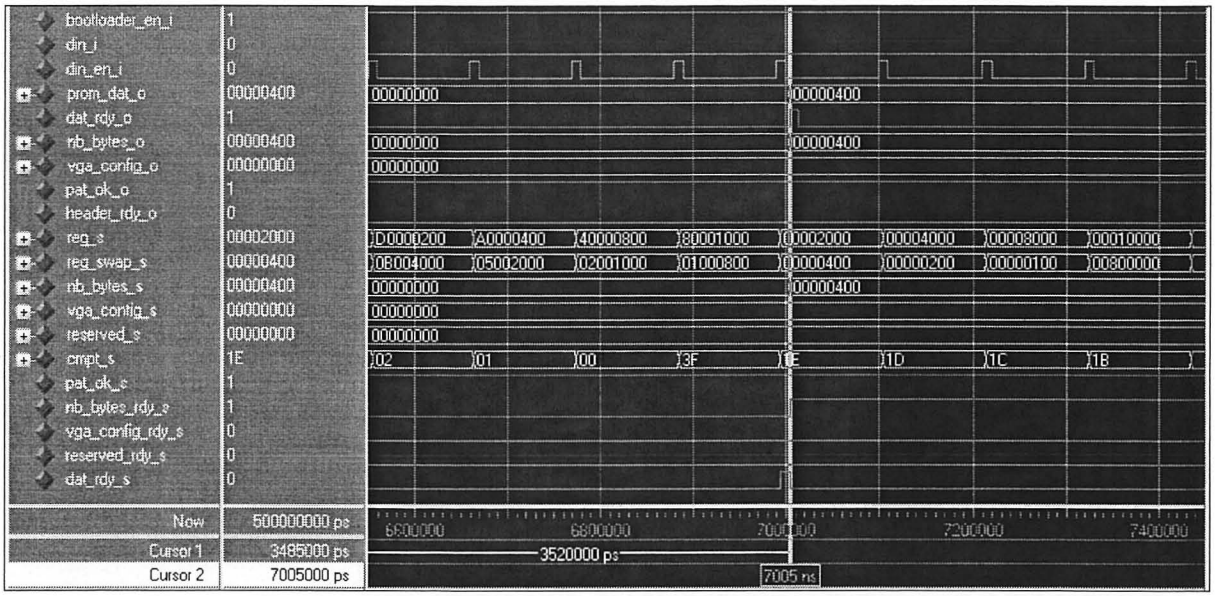

Figure 6.38 Simulation de l'enregistrement du nombre d'octets à charger par le module pattern_reco_loader.

La figure précédente illustre l'acquisition des quatre octets qui suivent la détection du patron de bits. C'est quatre octets font partie de l'entête des données et constituent le nombre d'octet que le système doit charge vers le contrôleur vidéo. Aussitôt que le signal dat_rdy s'active (pour la première fois depuis la détection du patron de bits), la valeur de 32 bits est sauvegardée dans le signal nb_bytes et le signal nb_bytes_rdy s'active. Ceci est identifié par le curseur numéro deux sur l'image précédente.

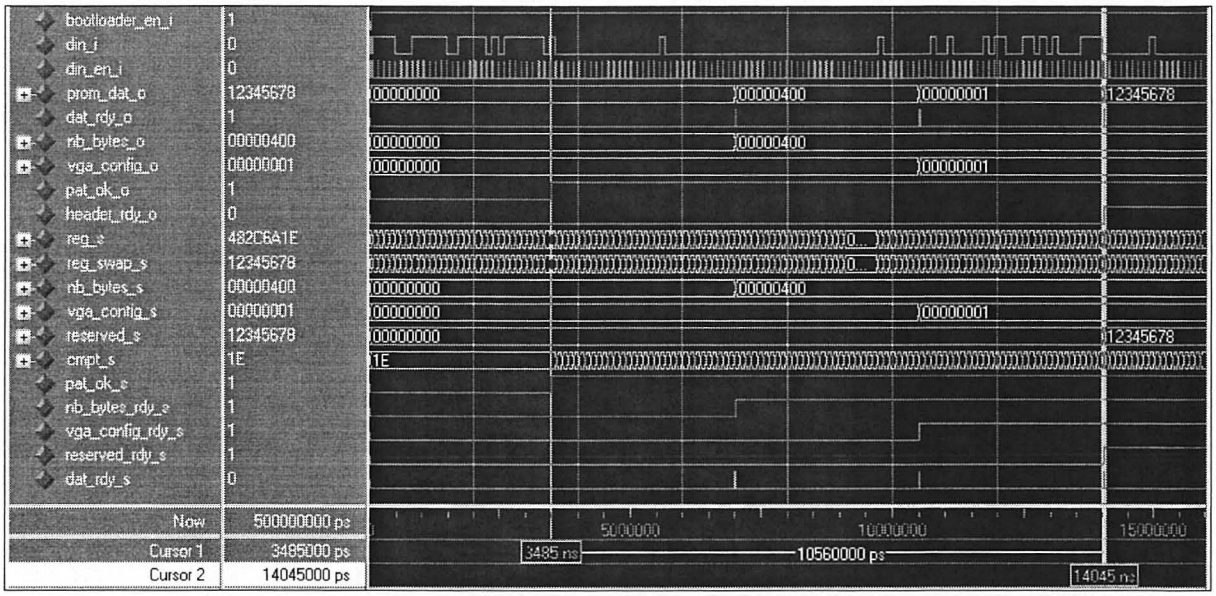

Figure 6.39 Simulation de l'acquisition de l'entête entière par le module pattern_reco_loader.

La Figure 6.39 montre la totalité du chargement de l'entête qui est constituée de trois fois 32 bits. Comme vu précédemment, le premier 32 bits constitue le nombre d'octets à charger et est placé dans le signal nb_bytes. Sur cette image, nb_bytes est égal à 0x00000400. Le signal nb_bytes_rdy s'active simultanément au chargement du signal nb_bytes.

Ensuite, une valeur de 32 bits contenant une trame de configuration du contrôleur VGA est chargée. Cette valeur est mémorisée dans le signal vga_config et le signal vga_config_rdy s'active simultanément. La valeur chargée est la valeur qui sera envoyée et mémorisée dans le registre de contrôle ctrl_reg du module ctrl_register (voir Section 6.11.3).

L'entête se termine par une valeur de 32 bits qui n'est pas encore utilisée, mais qui pourra éventuellement servir pour une configuration plus détaillée. On sauvegarde tout de même la valeur dans un signal nommée reserved et au même moment, on active le signal reserved_rdy qui devient par le fait même le signal d'identification de la fin du chargement de l'entête nommé header_rdy. Le curseur numéro deux pointe justement sur l'instant précis de la fin du chargement de l'entête.

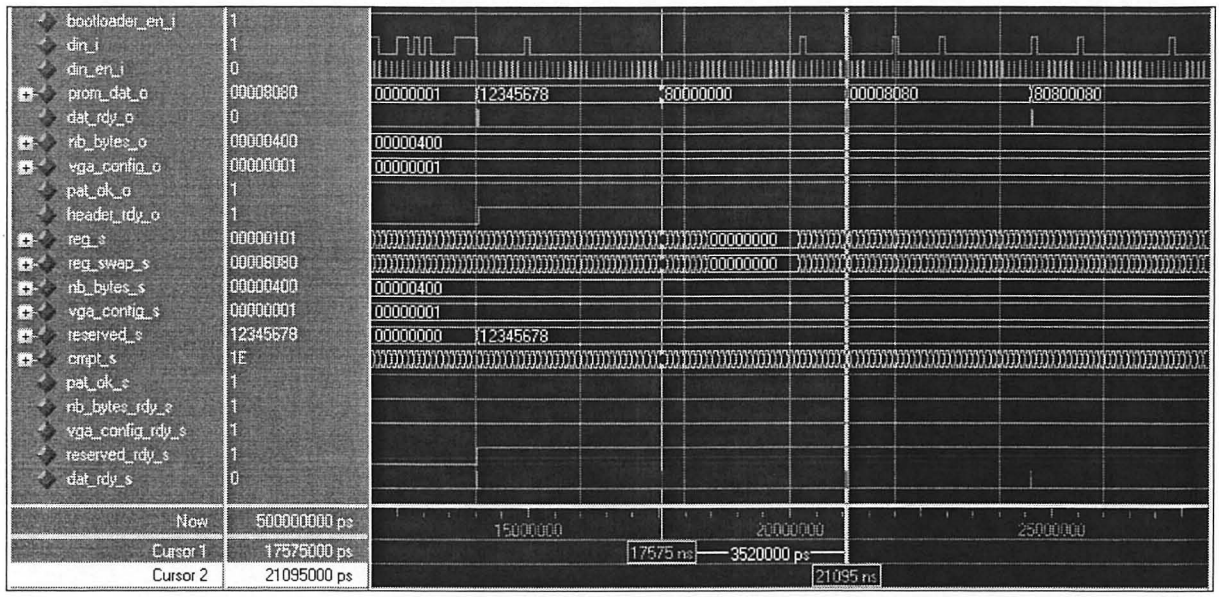

Figure 6.40 Simulation du chargement des données de 32 bits par le module pattern_reco_loader.

La Figure 6.40 montre finalement que le module pattern_reco_loader permet de faire le chargement de données sérielles sur le signal din par une conversion en données de 32 bits. À partir du moment où l'entête est chargé, le module ne fait plus de traitement et devient uniquement un convertisseur série à parallèle dont les valeurs de sortie sont de 32 bits. Un signal nommé dat_rdy permet d'identifier le moment où la valeur de sortie est valide. Les curseurs numéro un et deux de la figure précédente montrent les deux premières valeurs de sortie du module à la suite du chargement de l'entête.

6.10.4 Description du module cclk_gen (cclk_gen.vhd)

Voici le symbole du module cclk_gen :

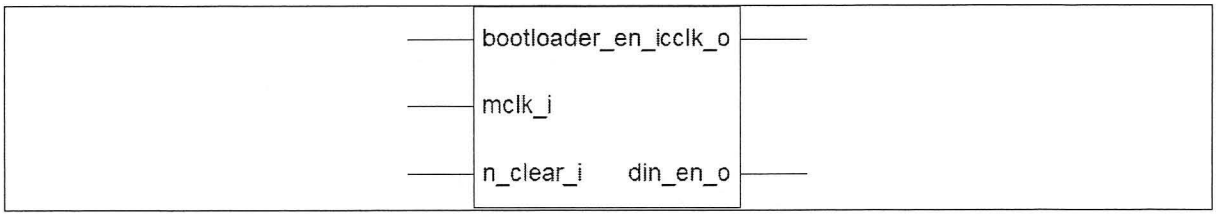

Figure 6.41 Symbole du module cclk_gen.

Avant de décrire le fonctionnement du module cclk_gen, voici un tableau de ses entrées et de ses sorties :

Tableau 6.15

Description des entrées et des sorties du module cclk_gen

Nom	Entrée/ Sortie	Description
mclk	E	Signal d'horloge maîtresse, voir les sections 6.7 et 6.7.4 pour plus de détails.
n_clear	E	Signal de réinitialisation générale du système. Voir la section 6.9 pour plus de détails.
bootloader_en	E	Signal qui signifie aux différents modules internes du module bootloader que le travail du bootloader doit s'effectuer. Ce signal est généré dans le module bootloader (section 6.10). Il s'active tant que le chargement de toutes les données dans la PROM n'est pas terminé (signal fin_dload provenant du module dload_adr_gen, section 6.10.7).
cclk	S	Signal d'horloge de lecture dans la PROM générée par un diviseur d'horloge, à partir de l'horloge maîtresse mclk. Le paramètre de division d'horloge est donné par une variable générique nommée nb_demi_periode_v. Ce paramètre est le nombre de cycles de mclk -2 pour une demi-période de cclk. Par défaut, ce paramètre est à la valeur « 011 » donc 3, ce qui signifie que chaque niveau de cclk correspond à 5 cycles de mclk (3 + 2 = 5).
din_en	S	Signal qui valide la donnée provenant de la PROM. Ce signal s'active durant un cycle de mclk suite à un front descendant de l'horloge cclk. Ce qui permet de s'assurer que les bits provenant de la PROM sont valides puisque ces derniers se présentent sur le front montant de cclk.

6.10.4.1 Description du fonctionnement du module cclk_gen

Ce module permet de générer un signal d'horloge pour permettre l'utilisation de la PROM après la configuration du FPGA. Il permet aussi de générer un signal din_en qui permet de savoir exactement quand lire le bit de donnée de la PROM.

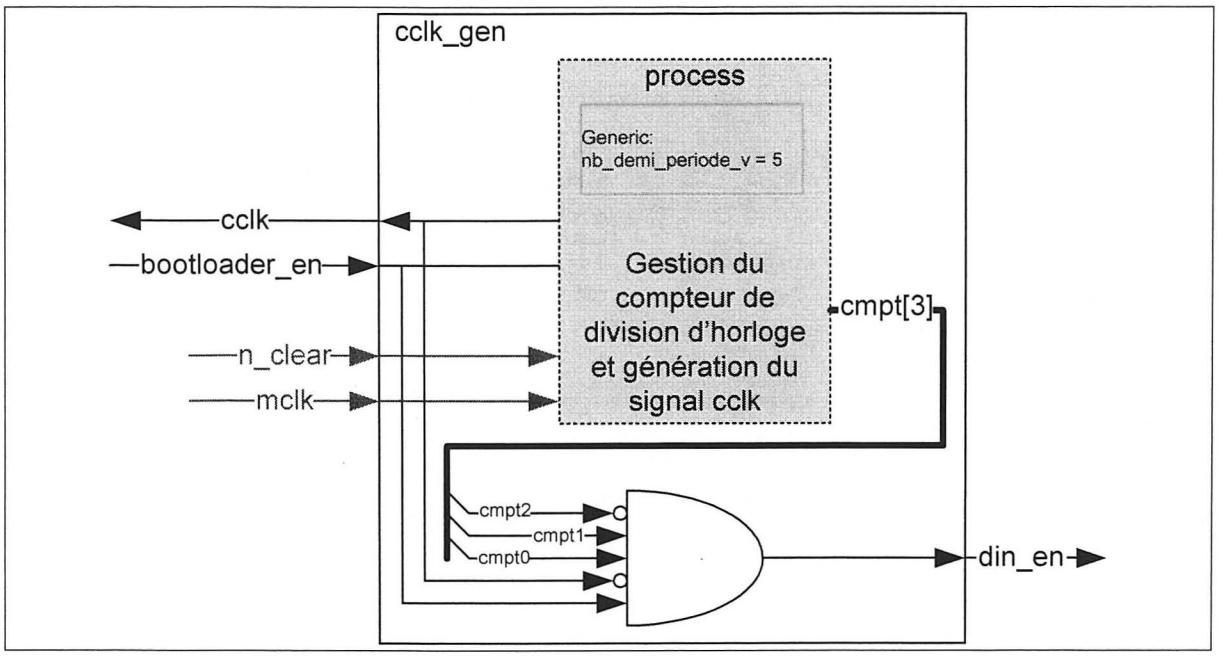

Figure 6.42 Schéma bloc du module cclk_gen.

Un seul processus est utilisé ici. Ce processus permet la division de l'horloge maîtresse par un compteur. Pour chaque compte complet, l'horloge cclk change d'état. C'est pour cette raison que la valeur de départ du compteur correspond à une demi-période. Le compteur est décroissant, ce qui permet une optimisation car le signal de fin de compte est branché sur le bit le plus significatif ce qui élimine la logique de décodage et donc diminue les délais de propagation. Le compteur par défaut démarre à la valeur « 011 » et suit la séquence « 010 », « 001 », « 000 » et « 111 », ce qui donne cinq valeurs et donc cinq cycles d'horloge maîtresse par demi-période.

6.10.4.2 Résultats des simulations du module cclk_gen

Voici la simulation fonctionnelle du module cclk_gen :

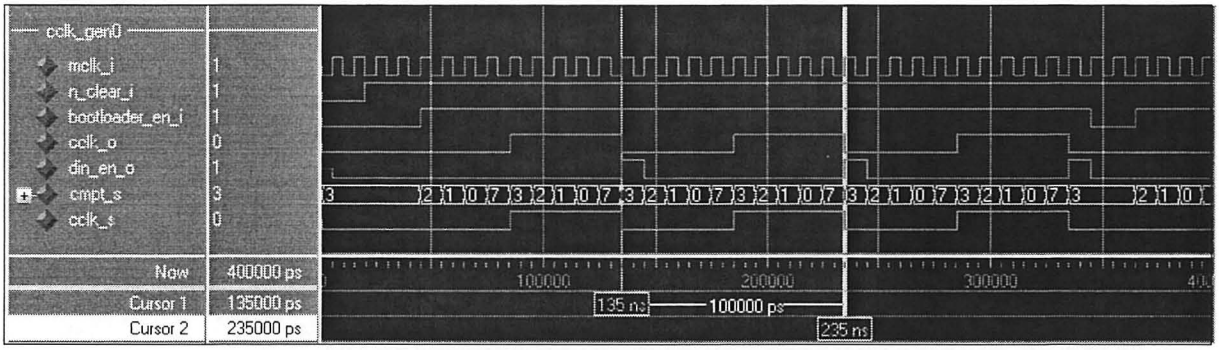

Figure 6.43 Simulation complète du module cclk_gen.

La figure précédente illustre bien le fonctionnement du module cclk_gen. Entre les deux curseurs, on mesure une période complète qui dure exactement 10 cycles d'horloge de mclk. Cette durée est configurée par la valeur générique du compteur de demi-période. Pour cette simulation, la valeur générique est de 0x011, ce qui veut dire que cinq cycles d'horloge sont comptés pour chaque demi-période. On comprend ici pourquoi la valeur 3 en binaire devient un compte de cinq : c'est qu'en fait le compteur passe par zéro et par sa valeur maximale. Ceci nous permet d'utiliser qu'un seul bit pour détecter la fin du compte, soit le bit le plus significatif qui est compt(2). Le signal din_en s'active toujours au niveau bas de cclk et lorsque le compteur commence à compter. Il en est ainsi pour deux raisons : la première est pour s'assurer que la valeur du bit din qui provient de la PROM est stable lors de la génération du signal din_en et la deuxième est parce que c'est un circuit simple.

6.10.5 Description du module clut_dat_gen (clut_dat_gen.vhd)

Voici le symbole du module clut_dat_gen :

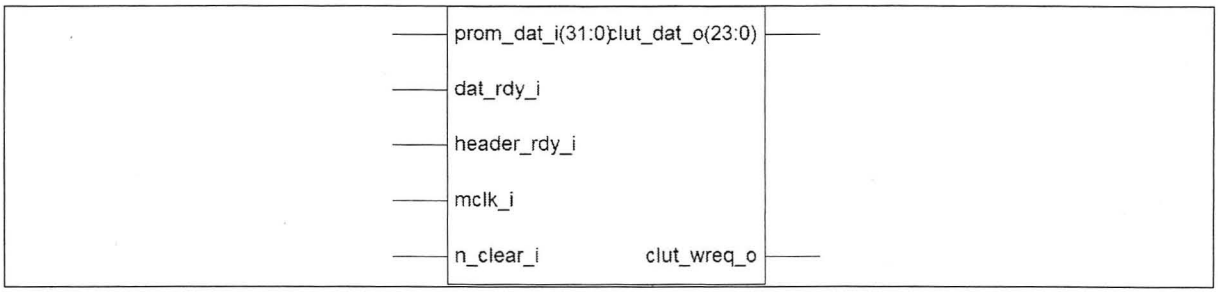

Figure 6.44 Symbole du module clut_dat_gen.

Avant de décrire le fonctionnement du module clut_dat_gen, voici un tableau de ses entrées et de ses sorties :

Tableau 6.16

Description des entrées et des sorties du module clut_dat_gen

Nom	Entrée/ Sortie	Description
mclk	E	Signal d'horloge maîtresse, voir les sections 6.7 et 6.7.4 pour plus de détails.
n_clear	E	Signal de réinitialisation générale du système. Voir la section 6.9 pour plus de détails.
prom_dat[32]	E	Signal des données qui proviennent du module pattern_reco_loader (section 6.10.3). Ce sont les données d'image à envoyer dans la mémoire vidéo.
dat_rdy	E	Signal de validation des données. Il indique que les données du signal prom_dat sont valides.
header_rdy	E	Signal qui indique lorsque l'entête des données de la PROM est entièrement lu. Ce signal est généré par le module pattern_reco_loader.
clut_dat[24]	S	Signal de données à sauvegarder dans la table des couleurs utilisé uniquement lors d'un affichage 8 bits. La table des couleurs est dans le module clut (section 6.11.5.5.3).
clut_wreq	S	Signal qui indique qu'une donnée de couleur peut être écrite dans le module clut.

6.10.5.1 Description du fonctionnement du module clut_dat_gen

Ce module permet de réordonnancer les données en paquets de 24 bits lors du chargement de la table des couleurs en mode d'affichage 8 bits. Voici son schéma bloc :

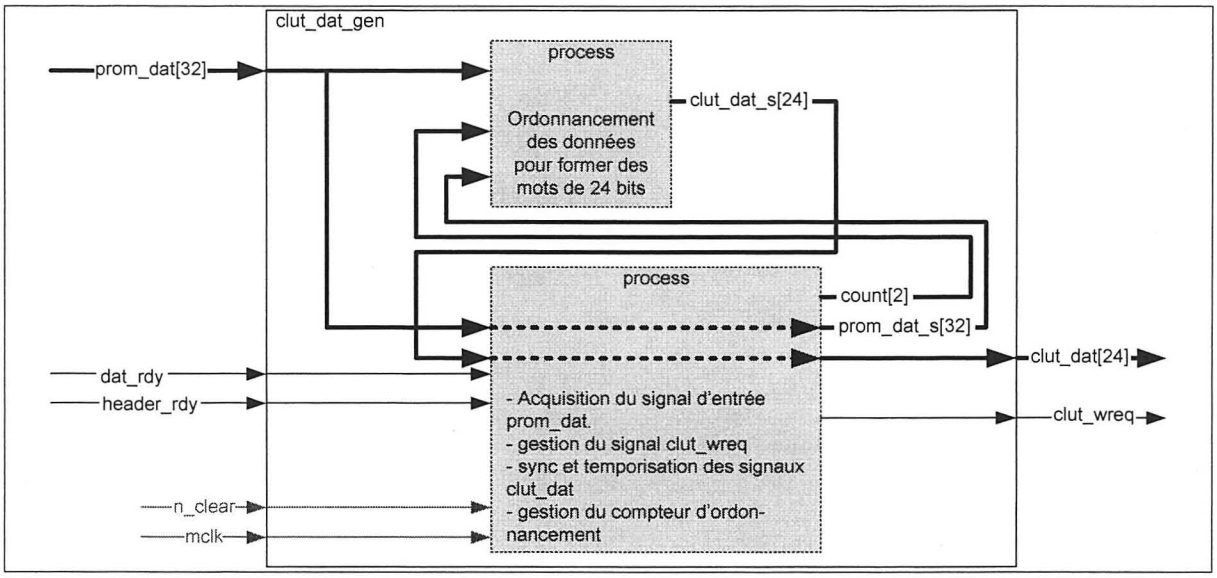

Figure 6.45 Schéma du module clut_dat_gen.

Note concernant les données contenues dans la PROM : Si vous ouvrez un fichier contenant les données de la table, vous remarquerez que les données sont organisées par lignes de 16 octets. Afin de s'assurer d'une efficacité accrue du chargement de la mémoire et de la table des couleurs, les données sont lues de la PROM par paquets de 4 octets.

Si on identifie par un numéro unique chaque valeur de la table des couleurs, le fichier de données utilisateur (servant à former le fichier *.mcs de la PROM) sera structuré de cette manière :

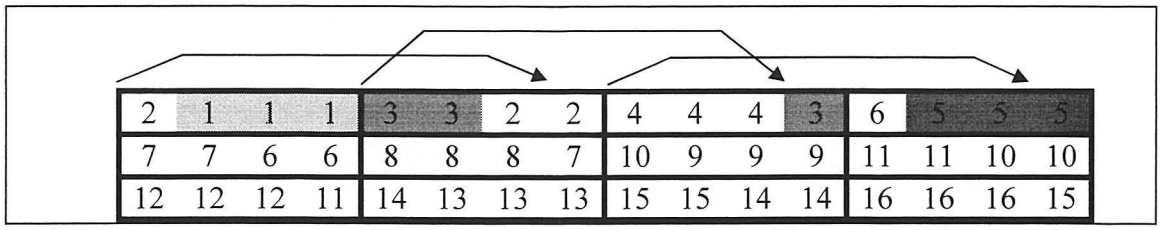

Figure 6.46 Organisation des données dans le fichier utilisateur.

Il faut donc une manière de faire pour lire 32 bits à la fois, mais en traiter uniquement 24. Il reste donc immanquablement toujours un octet inutilisé qui doit être mémorisé et utilisé lors du prochain 32 bits de données.

Le processus du haut de la Figure 6.45 est asynchrone et permet de prendre des données de quatre octets et de générer des données de trois octets qui constituent les couleurs de la table des couleurs dans le cas où le contrôleur est configuré pour un affichage de 256 couleurs (8 bits). À chaque lecture de donnée de quatre octets, le processus garde donc un octet et le met de côté pour les prochains quatre octets. Lorsque trois octets sont accumulés, cela constitue une autre donnée qui est aussi acheminée vers la sortie et le procédé se répète.

Le processus du bas de la Figure 6.45 est synchrone et permet de générer un compteur de contrôle qui permet d'identifier l'octet supplémentaire à isoler pour le réordonnancement des couleurs. Il permet aussi de rendre plusieurs signaux synchrones dont le signal de sortie clut_dat.

6.10.5.2 Résultats des simulations du module clut_dat_gen

Voici la simulation fonctionnelle du module clut_dat_gen :

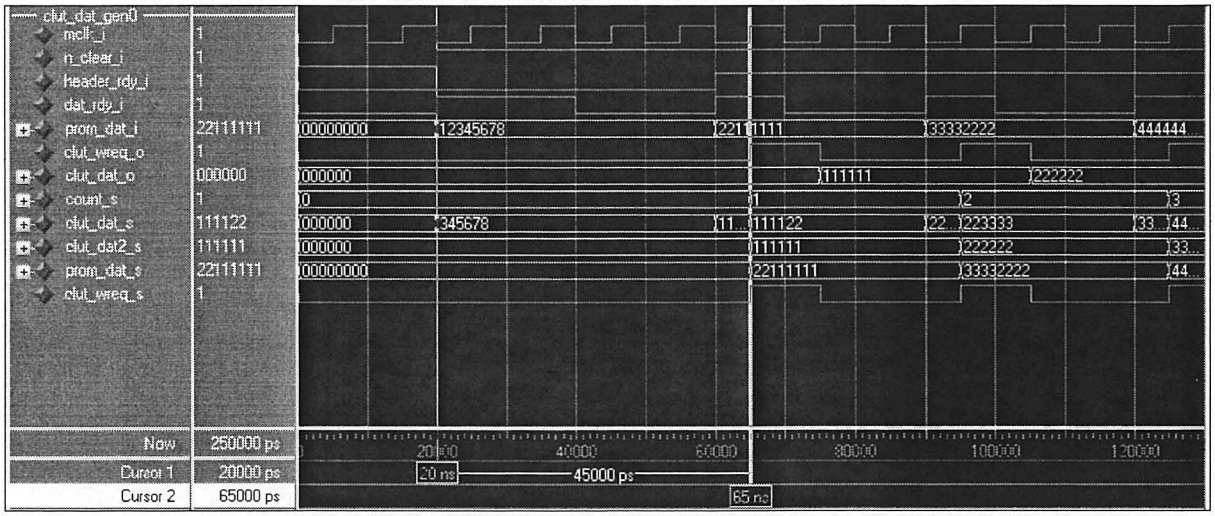

Figure 6.47 Simulation de l'initialisation et du blocage du module clut_dat_gen.

Sur la Figure 6.47, le premier curseur montre l'initialisation du module clut_dat_gen. Dans ce cas, les valeurs sont toutes initialisées à zéro. Par la suite, on teste le blocage du module tout en attribuant une valeur à l'entrée prom_dat. On observe bien que rien ne se passe en sortie puisque le signal header_rdy n'est pas activé. C'est certain puisque le signal header_rdy identifie la fin du chargement de l'entête des données. Alors il est logique que le chargement de la table des couleurs débute uniquement lorsque l'entête est entièrement chargé.

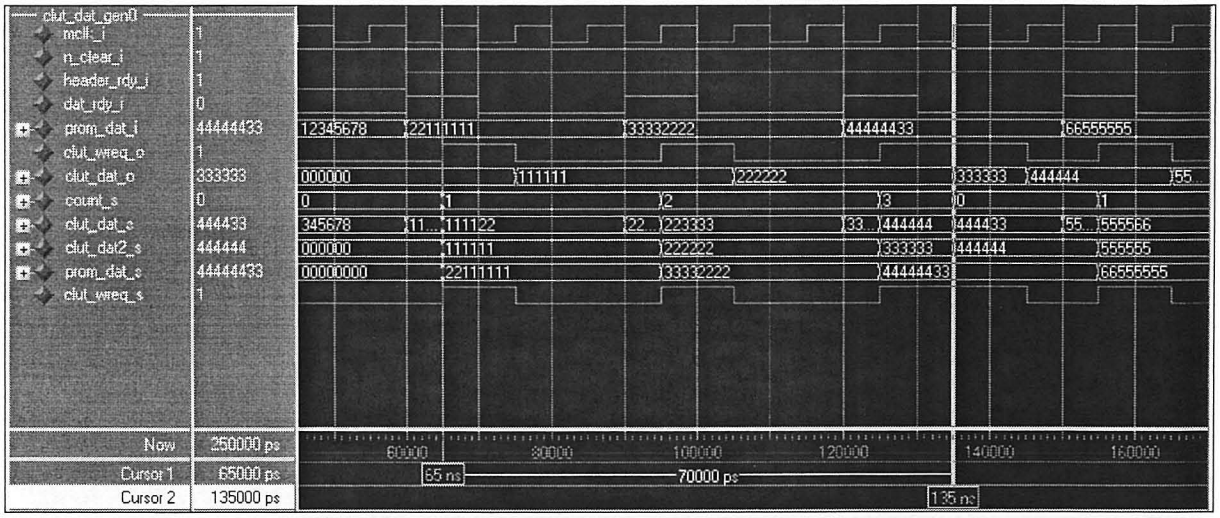

Figure 6.48 Simulation du chargement des quatre premières données de la table des couleurs par le module clut_dat_gen.

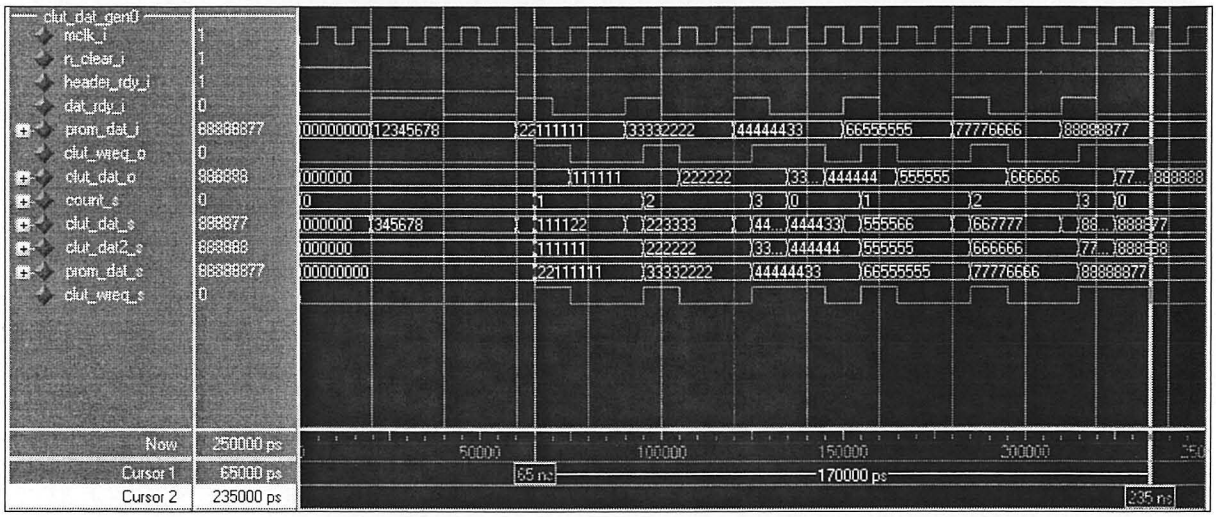

Figure 6.49 Simulation du chargement des huit premières données de la table des couleurs par le module clut_dat_gen.

Sur les deux figures précédentes, on peut voir le traitement effectué sur les données de 32 bits pour en sous tirer des valeurs de 24 bits. Vis-à-vis le curseur numéro un, nous avons la donnée 0x22111111 provenant de la PROM, donc les 24 bits les moins significatifs correspondent à la première donnée de 24 bits de la table des couleurs alors que les huit bits les plus significatifs font partie de la deuxième donnée de la table des couleurs. Donc la valeur 0x22 doit être gardée en mémoire afin d'être concaténée avec la prochaine donnée.

Cette mise en mémoire est faite par le signal prom_dat_s. En continuant ainsi, la prochaine donnée de la PROM contient donc 16 bits de la deuxième valeur de table des couleurs et aussi les deux premiers octets de la troisième valeur. On voit bien que le point de coupure entre les deux valeurs change d'une donnée à l'autre. C'est pour cette raison que le compteur nommé count a été créé. Lorsque le compteur est à zéro, le point de coupure est entre le bit 23 et le bit 24 du signal prom_dat. Lorsqu'il a la valeur un, le point de coupure est entre les bits 15 et 16, à la valeur deux, entre les bits 7 et 8 et à la valeur trois, il n'y a pas de point de coupure car la valeur de 24 bits se trouve entièrement dans la donnée de 32 bits. On peut voir cette dernière constatation vis-à-vis le curseur numéro deux de la Figure 6.48. Suite à cela, le compteur se réinitialise et le module recommence le même principe. On peut voir cela sur la Figure 6.49.

6.10.6 Description du module load_fsm (load_fsm.vhd)

Voici le symbole du module load_fsm :

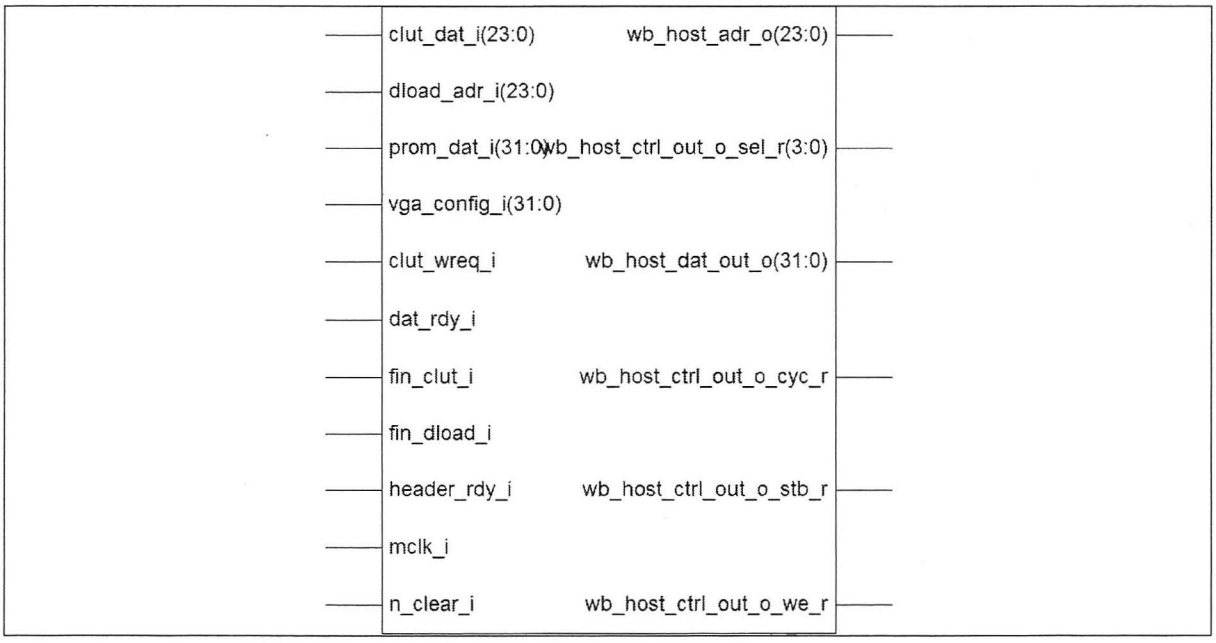

Figure 6.50 Symbole du module load_fsm.

Avant de décrire le fonctionnement du module load_fsm, voici un tableau de ses entrées et de ses sorties :

Tableau 6.17

Description des entrées et des sorties du module load_fsm

Nom	Entrée/ Sortie	Description
mclk	E	Signal d'horloge maîtresse, voir les sections 6.7 et 6.7.4 pour plus de détails.
n_clear	E	Signal de réinitialisation générale du système. Voir la section 6.9 pour plus de détails.
dat_rdy	E	Signal qui signifie que les données du vecteur prom_dat sont valides.
prom_dat[32]	E	Signal des données de 32 bits qui proviennent directement de la PROM après leur réordonnancement.
nb_bytes[32]	E	Signal qui détermine combien d'octets le bootloader devra charger. Ce signal est propagé vers le module top0/bootloader0/dload_adr_gen0 qui permet de faire un décomptage des octets chargés (voir section 6.10.7).
header_rdy	E	Signal qui indique lorsque l'entête des données de la PROM est entièrement lu. Ce signal est généré par le module pattern_reco_loader
clut_wreq	E	Signal qui indique qu'une donnée de couleur peut être écrite dans le module clut.
fin_clut	E	Signal qui indique que le chargement de la table des couleurs (module clut) est terminé.
fin_dload	E	Signal qui indique que toutes les données d'images ainsi que tous les paramètres provenant de la PROM ont été chargés dans le système vidéo (le contrôleur vidéo peut donc être activé). Ce signal met fin au chargement par le module bootloader. Ce signal est généré par le module bootloader0/dload_adr_gen0.
vga_config[32]	E	Signal qui contient plusieurs paramètres de configuration du contrôleur VGA. Ce signal est envoyé au registre de contrôle ctrl_reg qui est à l'adresse 0x800000 du bus Wishbone et qui se situe dans le module top0/vga0/ctrl_register0. Pour des détails sur le signal ctrl_reg du module ctrl_register, voir la section 6.11.3.
clut_dat[24]	E	Signal des données à sauvegarder dans la table des couleurs utilisée uniquement lors d'un affichage 8 bits. La table des couleurs est dans le module clut (section 6.11.5.5.3).
dload_adr[24]	E	Signal d'adresse généré par le module dload_adr_gen (section 6.10.7) et qui sert à identifier l'emplacement où doit se placer la donnée dans la mémoire vidéo.
wb_host_adr[24]	S	Signal d'adresse du bus Wishbone du Host. Le maître est le module bootloader et l'esclave est le module VGA.
wb_host_dat_out[32]	S	Signal de donnée du bus Wishbone du Host. Le maître est le module bootloader et l'esclave est le module VGA.
wb_host_ctrl_out[7]	S	Groupe de signaux de contrôle du bus wishbone host. Ce groupe de signaux inclut entre autres les signaux cyc et stb qui permettent d'entamer une transaction sur le bus. Le signal sel permet de savoir la taille de la transaction. Pour plus de détails sur le fonctionnement du bus Wishbone, référez-vous au document [23] Herveille 2002.

6.10.6.1 Description du fonctionnement du module load_fsm

Ce module est la machine à états finis qui contrôle toutes les étapes de chargement et de configuration du module VGA. Voici un schéma bloc de ce module :

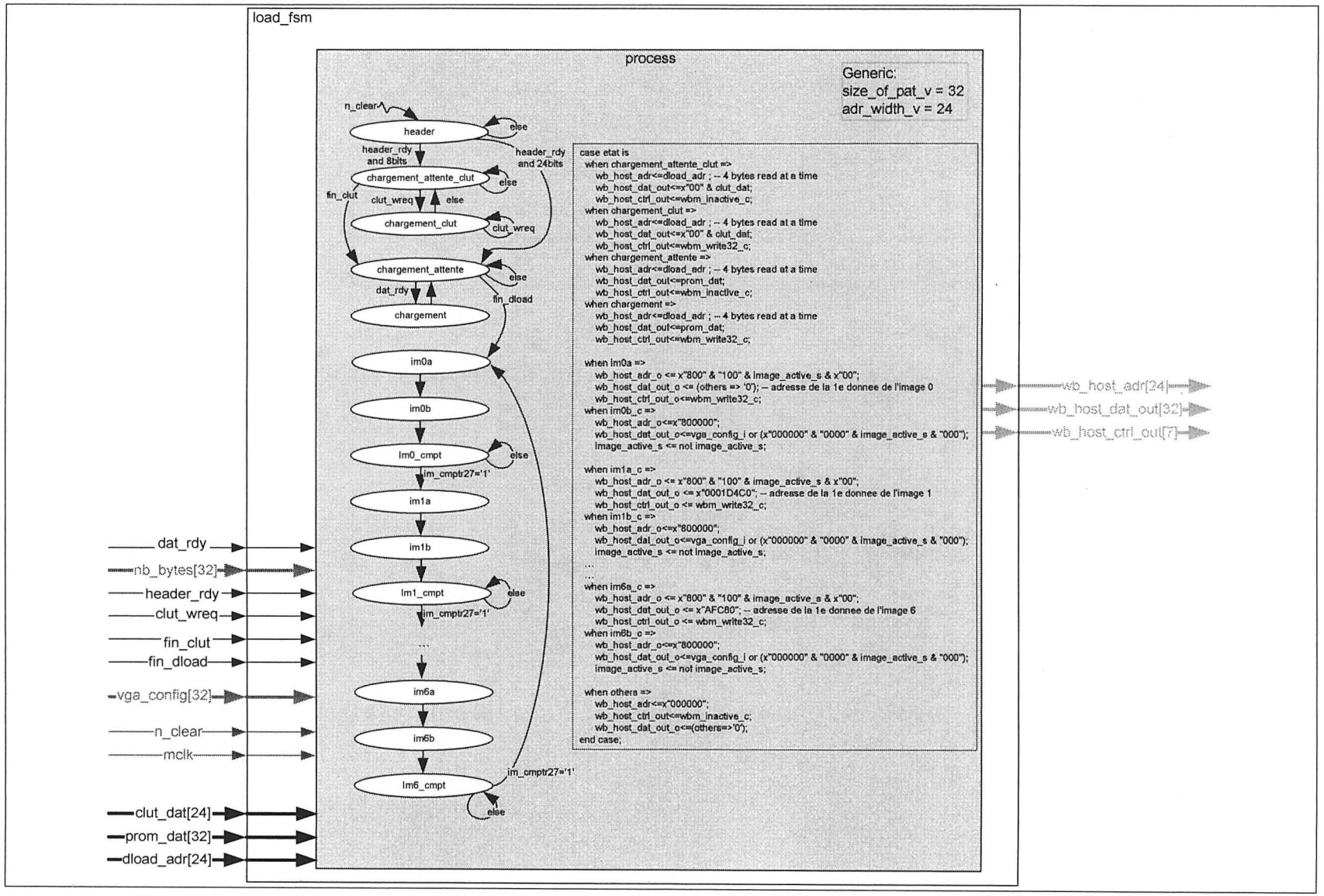

Figure 6.51 Schéma du module load_fsm.

Le système en entier consiste en un seul processus qui constitue une machine à états finis, une machine de Moore pour que toutes les entrées et les sorties soient synchrones.

La machine permet de coordonner les différentes étapes de chargement des données de la PROM et permet aussi de contrôler le changement des images affichées par le contrôleur vidéo. En fait, la machine à état contrôle les signaux placés sur le bus Wishbone. La machine entre en fonction aussitôt que le chargement de l'entête des données est terminé. Pour mieux comprendre le fonctionnement, voici le diagramme d'état descriptif de la machine, il correspond à celui dessiné sur le schéma de la Figure 6.51, mais avec des détails explicatifs :

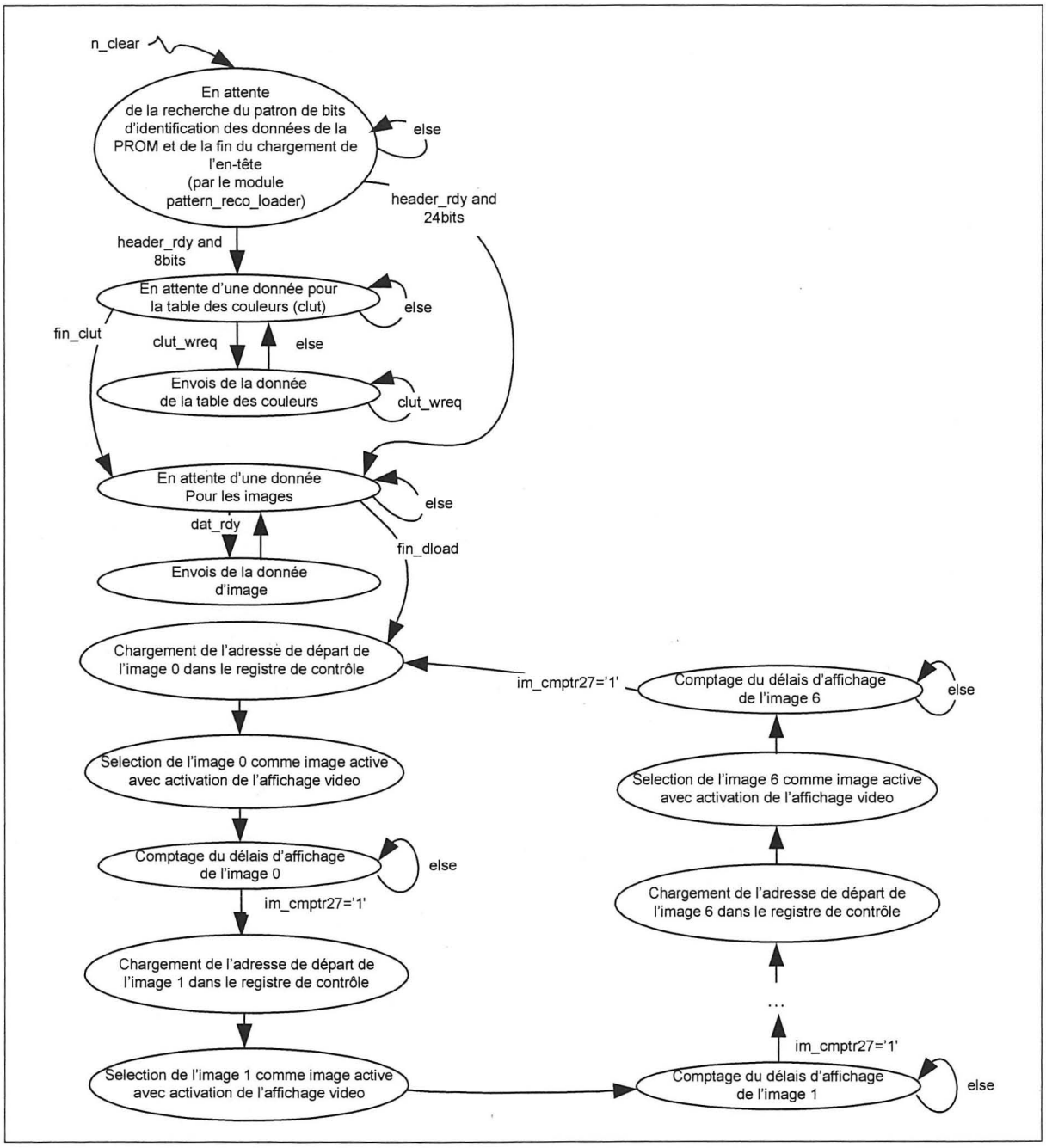

Figure 6.52 Diagramme d'état détaillé du module load_fsm.

Pour plus d'explications sur le fonctionnement du module load_fsm, vous pouvez aussi vous référer sur la section suivante qui constitue la simulation fonctionnelle du module.

6.10.6.2 Résultats des simulations du module load_fsm

Voici la simulation fonctionnelle du module load_fsm :

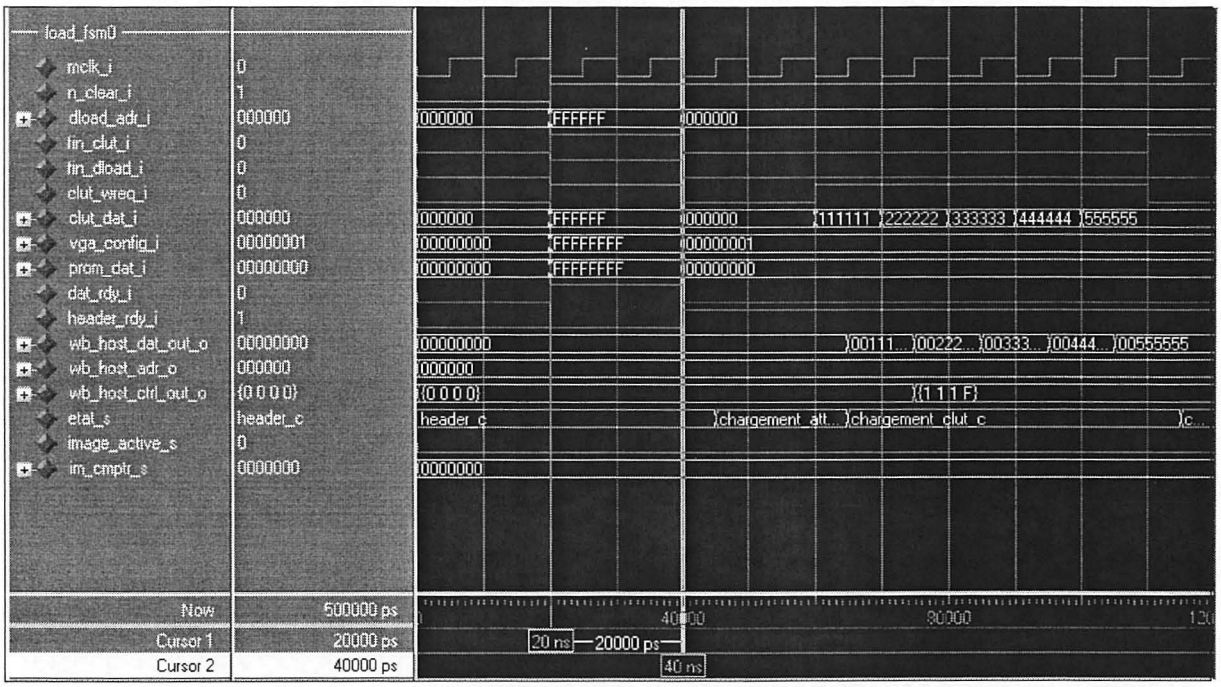

Figure 6.53 Simulation de l'initialisation du module load_fsm en mode 8 bits.

À l'initialisation de la machine, le bus Wishbone est désactivé et la machine se place à l'état header afin d'attendre la fin du chargement de l'entête par le module voisin qui le module pattern_reco_loader (voir section 6.10.3). Sur la figure précédente, le curseur numéro un pointe la remise en fonction du module après l'activation du signal n_clear. Suite à ce curseur on vérifie que le module reste bien en attente tant que le chargement de l'entête n'est pas terminé. En effet, c'est seulement au curseur numéro deux que le signal header_rdy s'active pour signifier au module que l'entête est chargée, alors le module change d'état en fonction des configurations contenues dans le signal vga_config. Dans ce premier cas, le signal vga_config indique que les images sont de type 8 bits par pixel, donc il y a une table de couleurs à charger. C'est pourquoi le module tombe à l'état chargement_attente_clut.

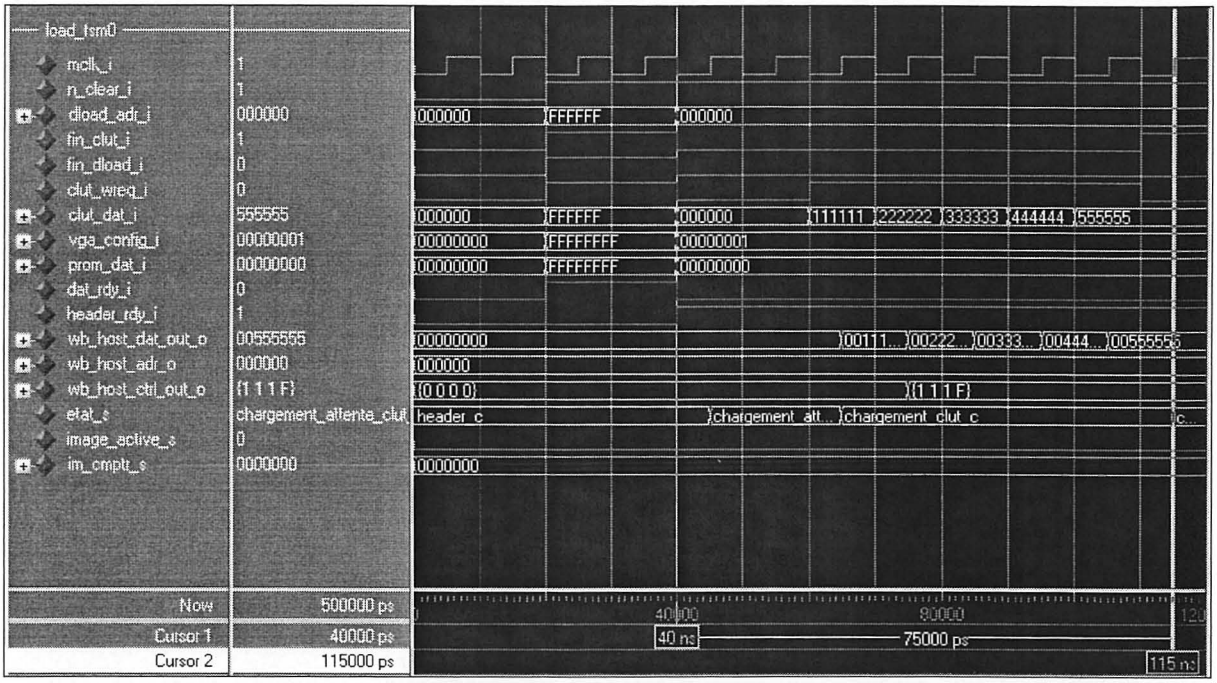

Figure 6.54 Simulation du chargement de la table des couleurs par le module load_fsm en mode 8 bits.

À partir du curseur numéro un de la figure précédente, la machine à états finis charge la table des couleurs. En pratique, pour chaque valeur de 32 bits, la machine devrait passer de l'état attente_chargement_clut à l'état chargement_clut puis revenir à l'état attente_chargement_clut puisque chacune des données à envoyer à la table des couleurs provient de la mémoire PROM qui transmet les données par un signal sériel. C'est le module pattern_reco_loader qui remet les données de 32 bits sous une forme utile et qui les transmet au module load_fsm en indiquant la validité des données par le signal dat_rdy. Donc, chaque fois que le signal dat_rdy s'active, la machine à états finis doit être en mesure de traiter la donnée à envoyer sur le bus Wishbone. Dans le banc d'essai, nous avons omis volontairement les délais d'attente entre chaque donnée à charger dans la table des couleurs. C'est pour cette raison que sur la figure précédente le signal clut_wreq reste actif pendant cinq cycles d'horloge et que l'état de la machine reste à l'état chargement_clut pendant tout ce temps, jusqu'au curseur numéro deux.

Pour envoyer des données de 24 bits sur le bus Wishbone (de 32 bits), on remplit les 8 bits les plus significatifs avec des zéros (0). Cela n'a pas beaucoup d'importance, puisque le

module clut qui mémorise la table des couleurs ne prend en considération que les 24 bits les moins significatifs de la donnée de couleur.

Une fois la table des couleurs entièrement chargée, le signal fin_clut s'active pour signifier au module load_fsm de passer au chargement des données de pixels. Le signal fin_clut est habituellement généré par le module clut_dat_gen qui fait la mise en forme des valeurs de la table des couleurs (voir section 6.10.5). La figure suivante présente le chargement des pixels par le module load_fsm.

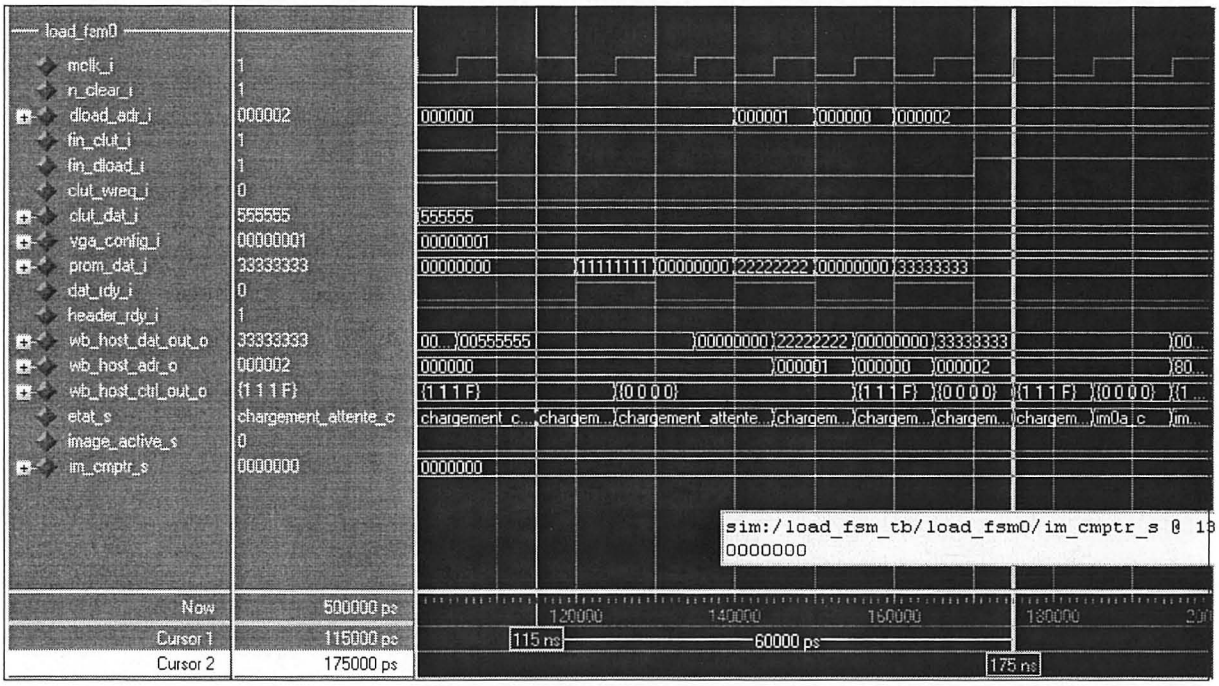

Figure 6.55 Simulation du chargement des données de pixels par le module load_fsm en mode 8 bits.

La Figure 6.55 montre au curseur numéro un le début du chargement des pixels. La machine doit osciller entre les états attente_chargement et chargement puisque l'état chargement ne dure qu'un cycle d'horloge et retourne automatiquement à l'état attente_chargement. En réalité (comme vu précédemment), le signal dat_rdy s'active environ 1 fois tous les 320 cycles d'horloge. Seulement pour réduire le temps de simulation, nous l'avons fait changer à chaque cycle d'horloge. Ceci à pour effet de pouvoir visualiser en une seule fenêtre de simulation le chargement des données pixels. Les valeurs des pixels proviennent du signal

prom_dat qui arrive directement de la PROM par l'intermédiaire du module pattern_reco_loader.

Le curseur numéro deux de la figure précédente montre l'activation du signal fin_dload qui indique que le chargement des données est terminé. La machine à états finis du module load_fsm entre alors dans la boucle d'affichage avec comme premier état : im0a.

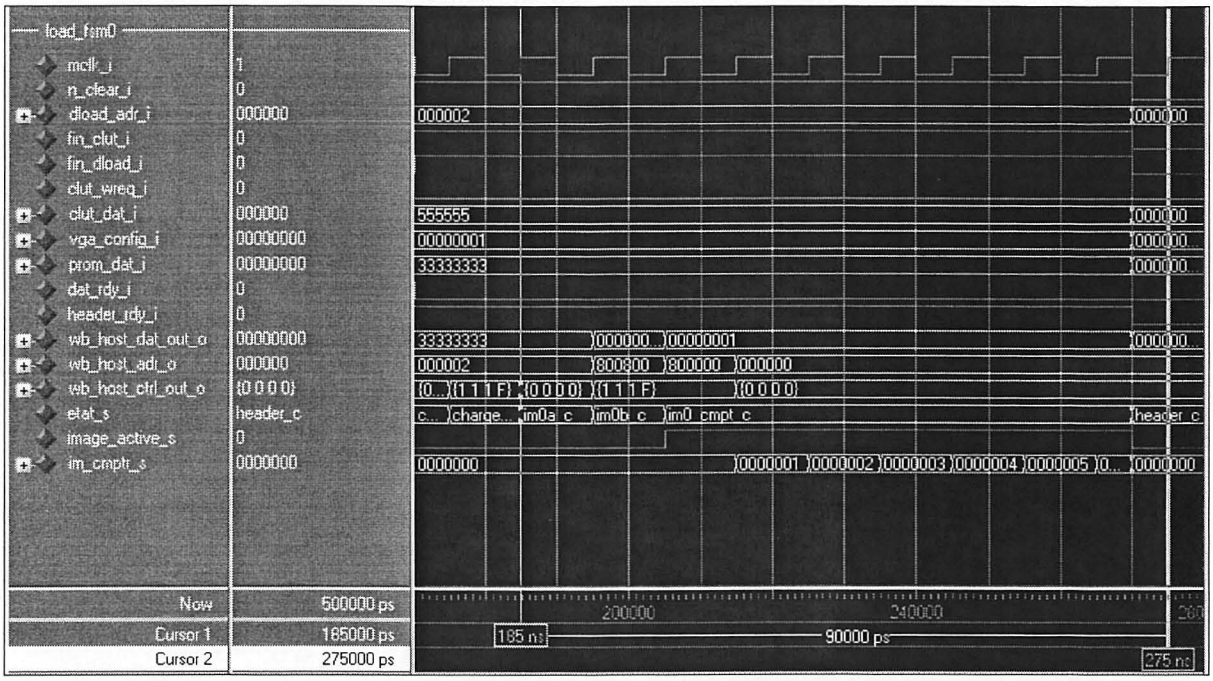

Figure 6.56 Simulation de la boucle d'affichage du module load_fsm en mode 8 bits.

Le curseur numéro un de la Figure 6.56 pointe le premier état de la boucle d'affichage. L'état im0a sert à charger l'adresse de la première image à afficher. Ensuite, l'état im0b permet d'activer l'affichage de cette image en faisant en sorte que le module ctrl_register pointe sur la bonne banque d'image et qu'il active l'affichage vidéo par l'activation du signal vdo_en (voir section 6.11.3 pour plus de détail).

Un cycle plus tard, la machine passe à l'état im0_cmpt, qui permet de temporiser l'affichage de l'image 0. Quand le compteur est terminé, la machine change d'état pour faire un changement de l'image affichée. On ne le voit pas ici puisque le temps de simulation aurait été trop grand. Ce qu'il faut savoir, c'est que pour changer l'image affichée, les étapes sont

les mêmes que pour l'affichage de l'image 0, c'est-à-dire qu'on passe par des états semblables qui se nomment : im1a, im1b et im1_cmpt. De la même manière qu'avec l'image 0, l'état im1a permet de mémoriser l'adresse de l'image 1 dans le module ctrl_register, l'état im1b permet d'activer la banque d'image adéquate pour l'affichage de l'image 1 et réitère l'activation de l'affichage vidéo. Puis l'état im1_cmpt permet de temporiser l'affichage de l'image 1.

Le processus boucle ainsi pour les images 2, 3, 4, 5 et 6 dont nous n'avons pas simulé le fonctionnement pour des raisons évidentes de temps de simulation. En effet, 10 secondes réelles à simuler demandent un temps de calcul de plusieurs heures et ne permet pas d'expliquer plus en détail le fonctionnement de la machine.

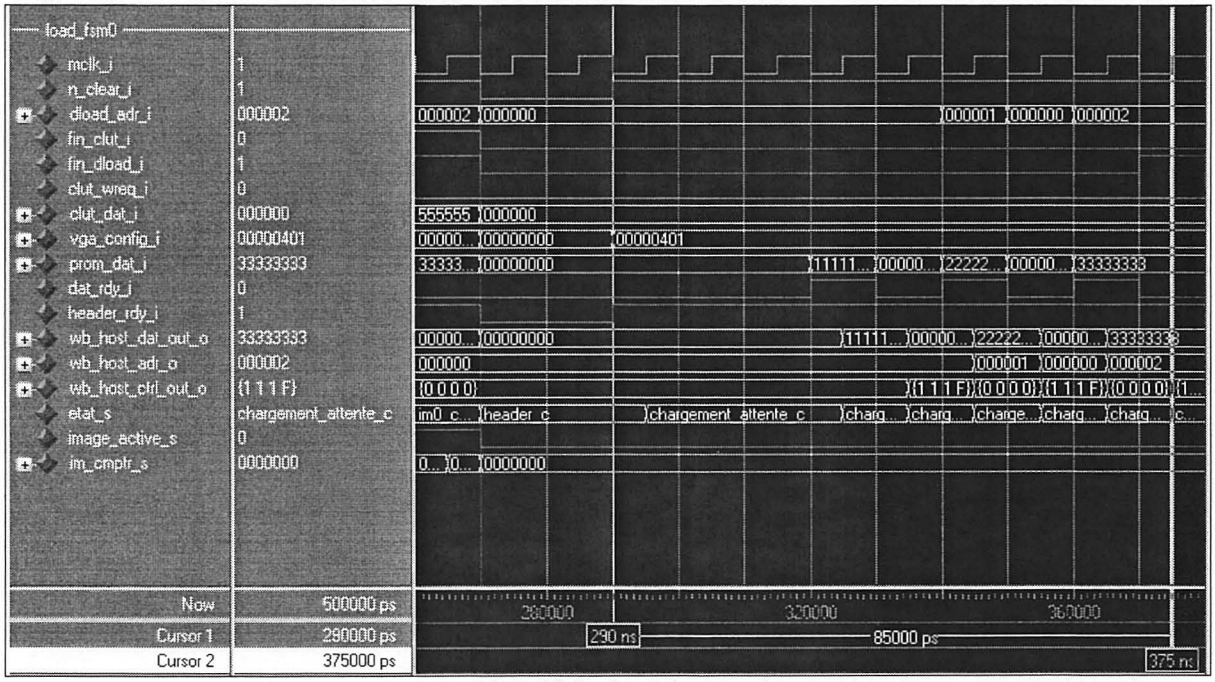

Figure 6.57 Simulation du chargement des données de pixels par le module load_fsm en mode 24 bits.

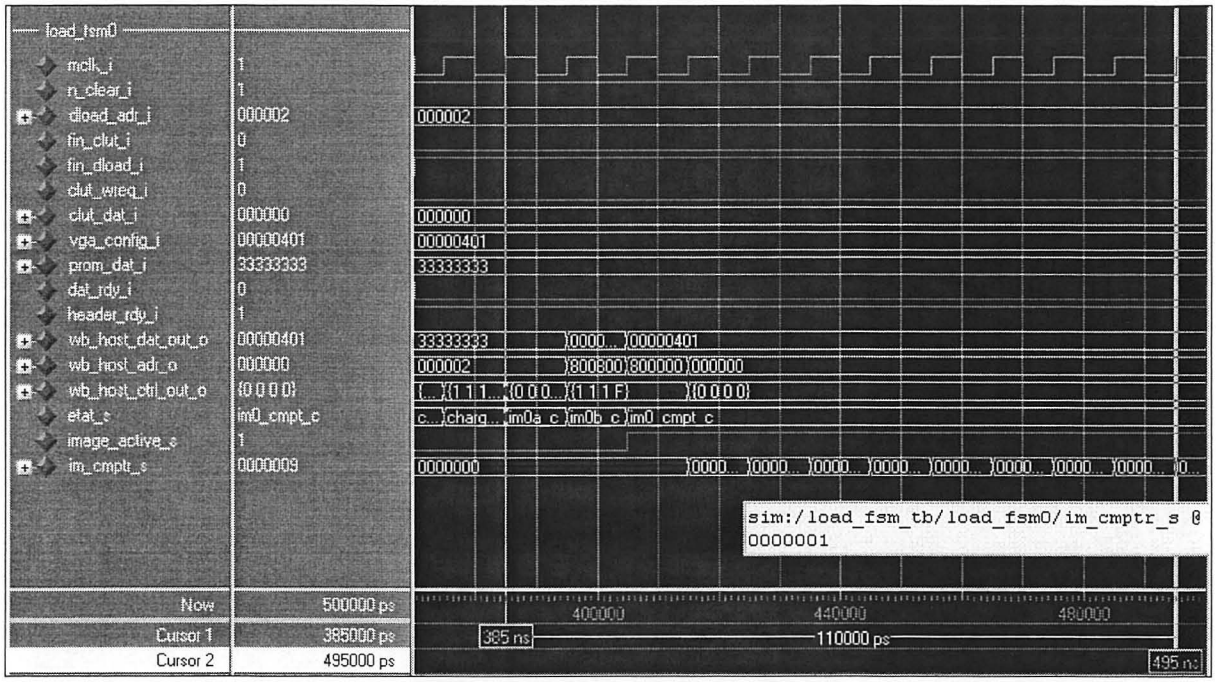

Figure 6.58 Simulation de la boucle d'affichage du module load_fsm en mode 24 bits.

Les deux dernières figures montrent le fonctionnement de la machine pour le chargement d'image de 24 bits par pixel. On observe dans ces deux images que le chargement ne diffère que d'un aspect, il n'y a pas de table des couleurs à charger, ce qui fait que la machine passe directement de l'état header à l'état chargement_attente dès que le signal header_rdy s'active. On observe ce changement au curseur numéro un de la Figure 6.57. Pour le reste du fonctionnement de la machine en mode 24 bits, tout reste semblable au fonctionnement en mode 8 bits.

6.10.7 Description du module dload_adr_gen (dload_adr_gen.vhd)

Voici le symbole du module dload_adr_gen :

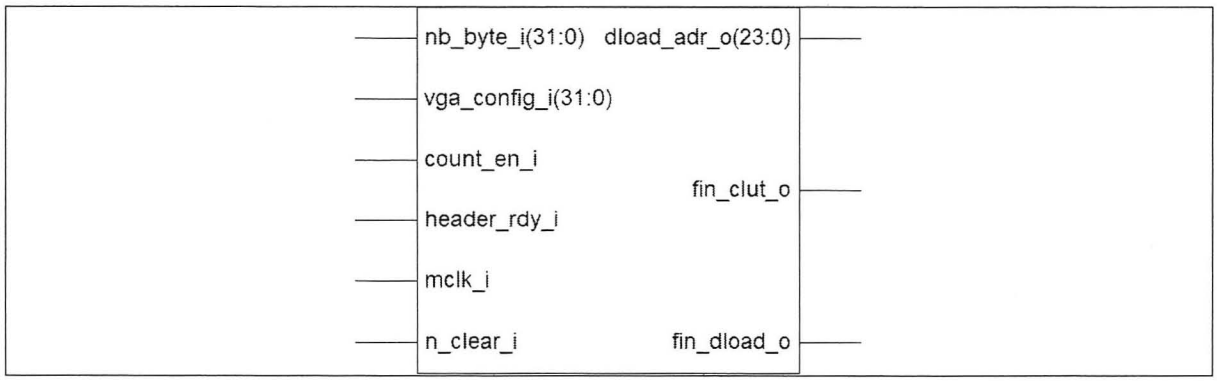

Figure 6.59 Symbole du module dload_adr_gen.

Avant de décrire le fonctionnement du module dload_adr_gen, voici un tableau de ses entrées et de ses sorties :

Tableau 6.18

Description des entrées et des sorties du module dload_adr_gen

Nom	Entrée/ Sortie	Description
mclk	E	Signal d'horloge maîtresse, voir les sections 6.7 et 6.7.4 pour plus de détails.
n_clear	E	Signal de réinitialisation générale du système. Voir la section 6.9 pour plus de détails.
count_en	E	Signal qui indique au module dload_adr_gen quand générer des valeurs d'adresse. Ce signal est généré dans le module bootloader par un processus. Il est actif uniquement lorsque l'entête des données dans la PROM est terminé.
header_rdy	E	Signal qui indique lorsque l'entête des données de la PROM est entièrement lu. Ce signal est généré par le module pattern_reco_loader.
nb_bytes[32]	E	Signal qui indique au bootloader combien d'octets devront être chargés. Cette valeur est sur 32 bits et se situe immédiatement après le patron de bits qui identifie l'emplacement des données dans la PROM.
vga_config[32]	E	Signal qui contient plusieurs paramètres de configuration du contrôleur VGA. Ce signal est envoyé au registre de contrôle ctrl_reg qui est à l'adresse 0h800000 du bus Wishbone et qui se situe dans le module top0/vga0/ctrl_register0. Pour des détails sur le signal ctrl_reg du module ctrl_register, voir la section 6.11.3.
dload_adr[24]	S	Signal d'adresse généré par le module dload_adr_gen servant à identifier l'emplacement où doit se placer la donnée dans la mémoire vidéo.

fin_dload	S	Signal qui indique que toutes les données d'images ainsi que tous les paramètres provenant de la PROM ont été chargées dans la mémoire vidéo (le contrôleur vidéo peut donc être activé). Ce signal met fin au chargement par le module bootloader. Ce signal est généré par le module bootloader0/dload_adr_gen0.
fin_clut	S	Signal qui indique que le chargement de la table des couleurs (module clut) est terminé.

6.10.7.1 Description du fonctionnement du module dload_adr_gen

Ce module contient le compteur nécessaire pour générer l'adresse où les données chargées doivent être placées, ce compteur se situe dans le sous-module adr_counter.

Le sous-module fin_dload_gen est aussi un compteur, il permet de savoir exactement lorsque le chargement des données par le module bootloader est terminé.

Le module dload_adr_gen contient aussi un processus interne qui permet d'identifier quand le chargement de la table des couleurs est terminé dans le cas où on utilise un mode d'affichage 8 bits. Voici un schéma bloc du module dload_adr_gen et de ses sous-modules :

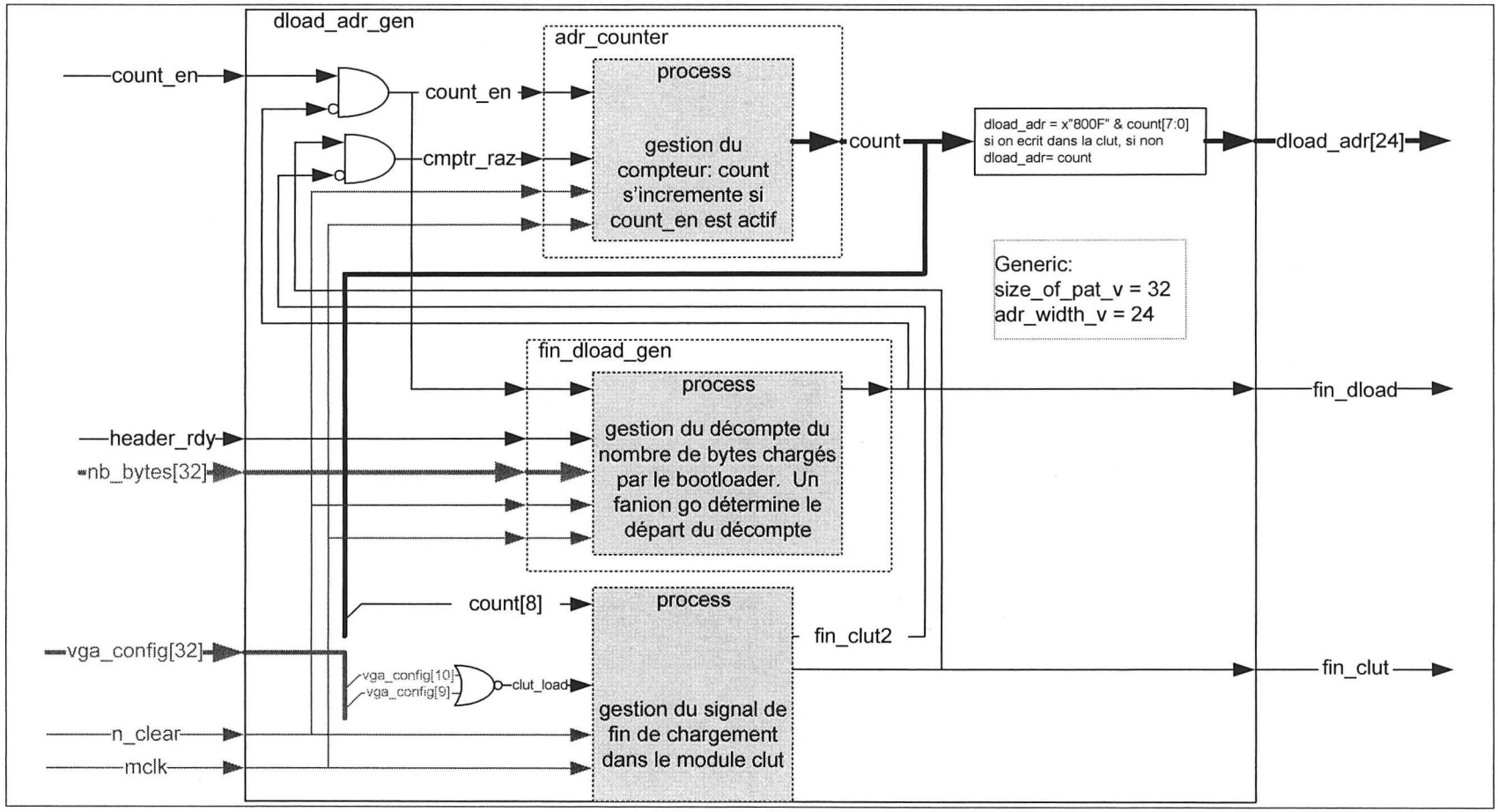

Figure 6.60 Schéma du module dload_adr_gen.

6.10.7.2 Résultats des simulations du module dload_adr_gen

Voici la simulation fonctionnelle du module dload_adr_gen :

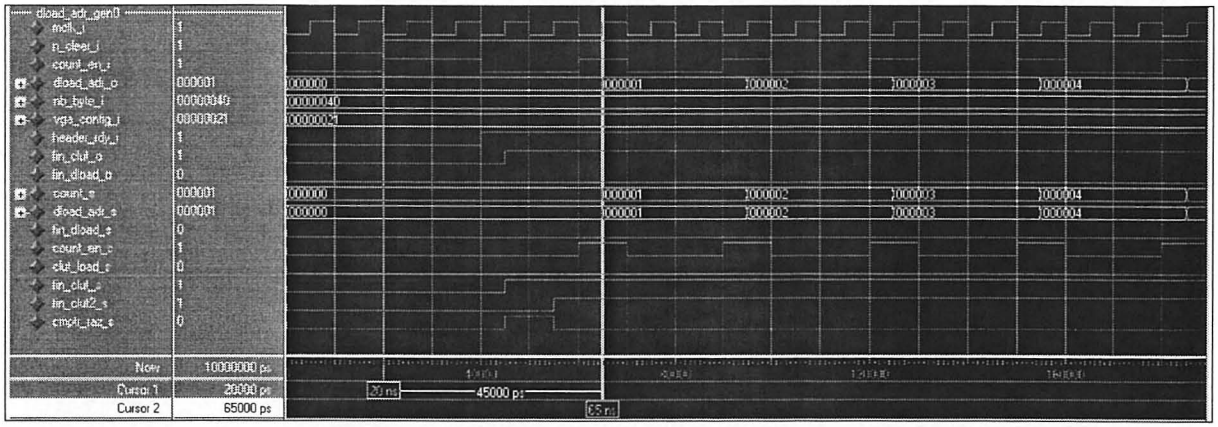

Figure 6.61 Simulation de l'initialisation du module dload_adr_gen en mode 24 bits.

La figure précédente montre l'initialisation du module dload_adr_gen. Au départ, les signaux header_rdy et fin_clut sont inactifs, ce qui fait qu'aucune adresse ne peut être générée, même si le signal count_en s'active. Ce dernier servant à permettre l'incrémentation de l'adresse lorsque le module est en fonction. Justement, le module entre en fonction lorsque le signal header_rdy s'active (voir la section 6.10.3 sur le module pattern_reco_loader). A ce moment, si le signal vga_config indique que les images sont de format 8 bpp, une table des couleurs doit être chargée (ce qui est expliqué plus bas). Autrement, le signal fin_clut s'active automatiquement et la génération des adresses mémoire commence de manière synchrone avec le signal count_en. Le curseur numéro deux de l'image précédente illustre bien cette information. Le format d'image spécifié par le signal vga_config est de 24 bpp (bit 5 = 1 et bit 4 = 0), ce qui fait qu'il n'y a pas de table de couleurs à charger.

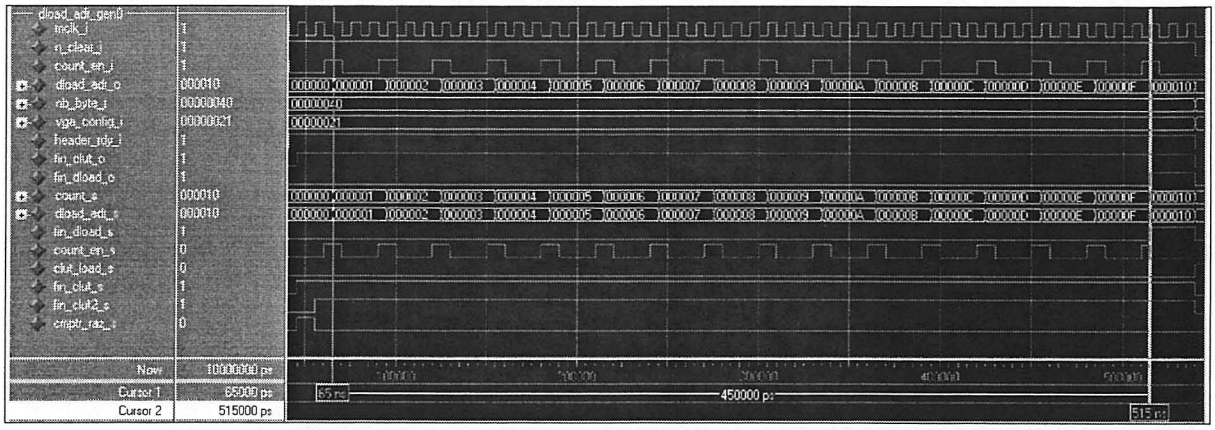

Figure 6.62 Simulation de la génération d'adresse par le module dload_adr_gen en mode 24 bits.

La figure précédente illustre un comptage normal consécutif des adresses. Il est à noter que la manière de compter les adresses dépend beaucoup de la mémoire vidéo utilisée. Dans notre cas, chaque case mémoire possède 32 bits, ce qui fait que l'incrémentation se fait par pas de un (1). On peut voir sur cette image que le signal nb_bytes = 0x40, ce qui fait que les adresses générées se termine à 0x10 puisque lorsque cette adresse est générée, il y a 4 * 0x10 octets, ce qui donne bien 0x40 octets. Le curseur numéro deux de la Figure 6.62 montre bien qu'à la génération de l'adresse 0x10, le signal fin_dload s'active, signifiant ainsi la fin de la génération d'adresse par le module dload_adr_gen. On réinitialise ensuite le module afin de faire la simulation en mode 8 bpp.

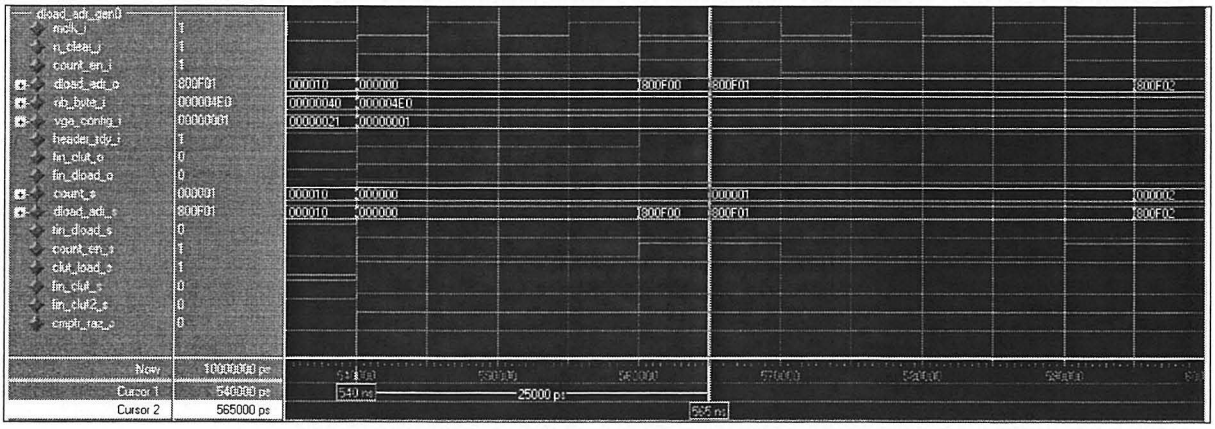

Figure 6.63 Simulation de la réinitialisation du module dload_adr_gen pour la vérification du mode 8 bits.

Pour le mode 8 bpp, la table des couleurs doit être chargée en premier, c'est pourquoi l'adresse générée au départ, suite au chargement de l'entête (ou activation du signal header_rdy), commence par la valeur 0x800F00. Cette adresse permet de viser le module ctrl_register qui est la passerelle pour écrire dans la table des couleurs. La plage d'adresse pour écrire dans la table des couleurs est de 0x800F00 à 0x800FFF, ce qui donne effectivement une plage de 256 données, donc 256 couleurs. La Figure 6.63 montre le curseur numéro deux au temps de génération de l'adresse de départ de la table des couleurs 0x800F00. Il est à noter que le système est en mode 8 bpp lorsque les bits 4 et 5 du signal vga_config sont à zéro.

La figure suivante montre tout le chargement de la table des couleurs. Chaque valeur d'adresse est incrémentée de un à partir de la valeur 0x800F00. On voit sur cette figure au curseur numéro deux que le signal fin_clut s'active dès que les adresses de la tables ont toutes été générées.

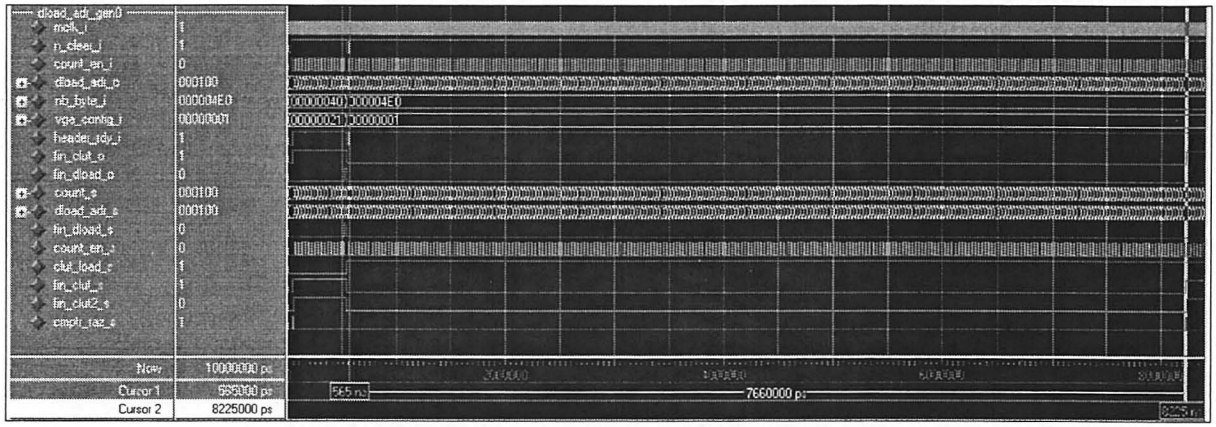

Figure 6.64 Simulation de la génération des adresses de la table des couleurs par le module dload_adr_gen en mode 8 bits.

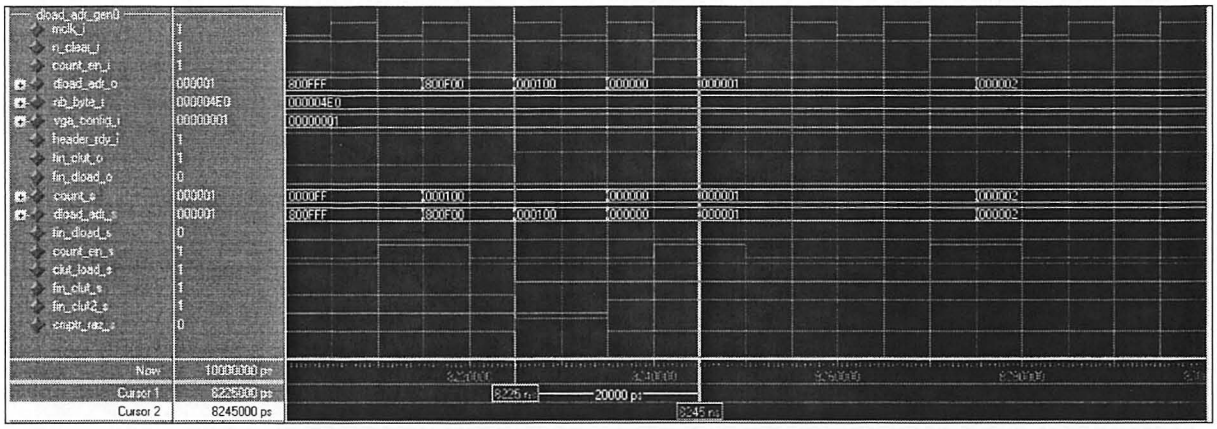

Figure 6.65 Simulation du passage de la génération d'adresse de table des couleurs à celle d'adresses mémoire par le module dload_adr_gen en mode 8 bits.

La figure précédente a été placée ici pour montrer une particularité du module dload_adr_gen à la fin du chargement de la table des couleurs en mode 8 bpp. En effet, lorsque la table des couleurs a été entièrement chargée, le compteur d'adresse doit être remis à zéro. Pour cela, le module à besoin de trois cycles d'horloge. Ces trois cycles sont ceux qui précèdent le curseur numéro deux de la figure précédente.

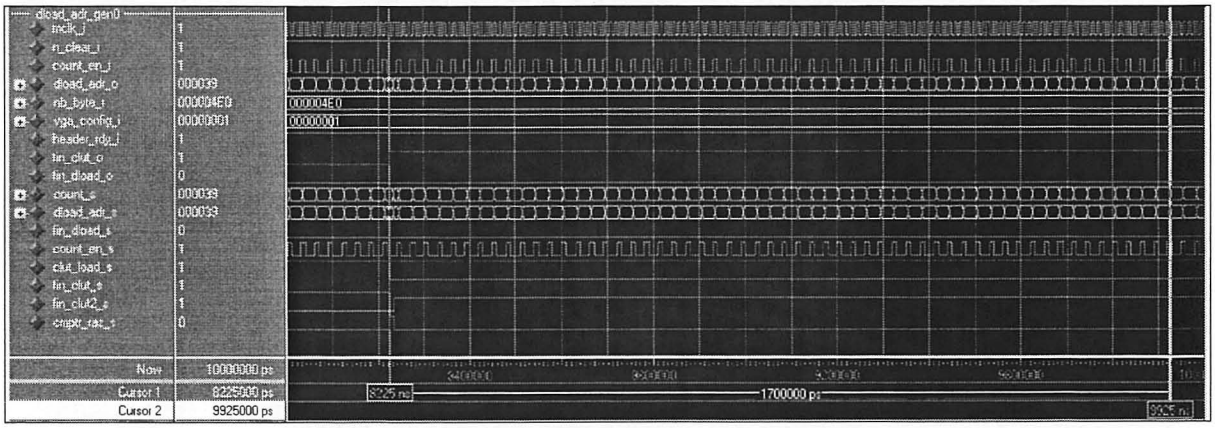

Figure 6.66 Simulation de la génération des adresses mémoire du chargement des données pixel par le module dload_adr_gen en mode 8 bits.

La figure précédente est pour montrer qu'aussitôt le chargement de la table des couleurs terminé (en mode 8 bpp) le module compteur qui génère les adresses se réinitialise et recommence à compter et fonctionne de la même façon qu'en mode 24 bpp dont on a vu le fonctionnement plus haut.

6.10.7.3 Description du module adr_counter (adr_counter.vhd)

Voici le symbole du module adr_counter :

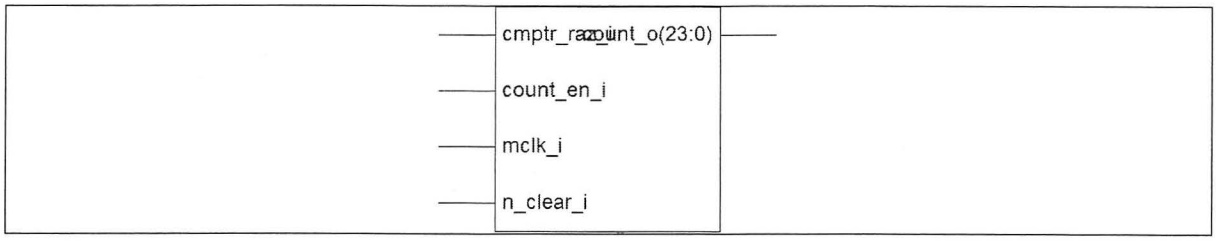

Figure 6.67 Symbole du module adr_counter.

Avant de décrire le fonctionnement du module adr_counter, voici un tableau de ses entrées et de ses sorties :

Tableau 6.19

Description des entrées et des sorties du module adr_counter

Nom	Entrée/ Sortie	Description
mclk	E	Signal d'horloge maîtresse, voir les sections 6.7 et 6.7.4 pour plus de détails.
n_clear	E	Signal de réinitialisation générale du système. Voir la section 6.9 pour plus de détails.
count_en	E	Signal qui indique au module adr_counter quand générer des valeurs d'adresse. Ce signal est généré dans le module bootloader par un processus. Il est actif uniquement lorsque l'entête des données dans la PROM est terminée et il se désactive à la fin du chargement des données.
cmptr_raz	E	Signal de remise à zéro du compteur d'adresse. Il s'active par une impulsion d'un cycle d'horloge, uniquement lorsque le signal fin_clut s'active. Ceci permet de remettre les adresses à zéro pour envoyer les données d'images ensuite à partir de l'adresse zéro dans la mémoire vidéo.
count[24]	S	Signal qui indique la valeur du compteur et qui est par le fait même l'adresse que l'on cherche à générer.

6.10.7.3.1 Description du fonctionnement du module adr_counter

En regardant le schéma de la Figure 6.60, on peut expliquer facilement que le composant fin_dload_gen est un compteur qui démarre lorsque le signal d'entrée count_en s'active et que le signal cmptr_raz est inactif. Le compteur génère donc les adresses de chargement du bootloader qui sont envoyées vers la mémoire vidéo.

6.10.7.3.2 Résultats des simulations du module adr_counter

Voici la simulation fonctionnelle du module adr_counter :

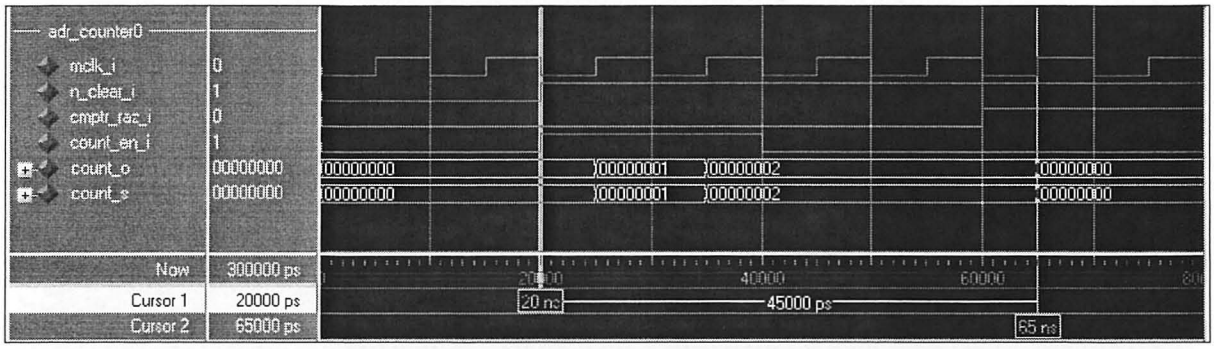

Figure 6.68 Simulation de l'initialisation du module adr_counter.

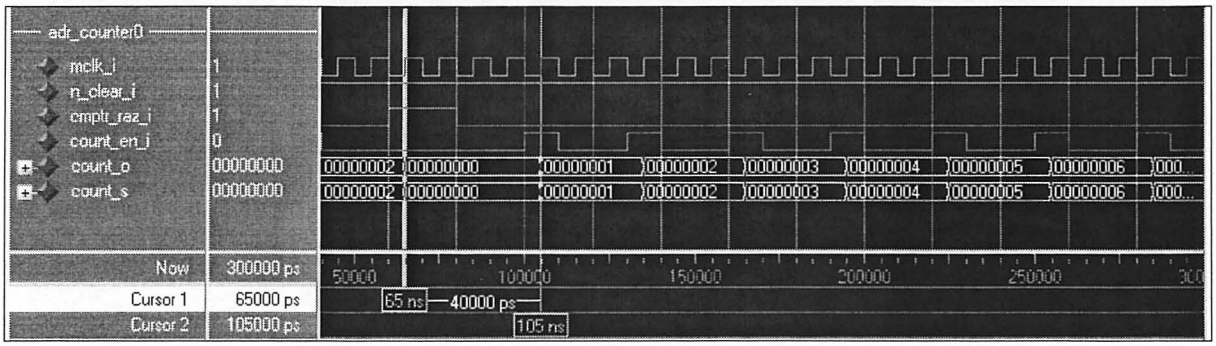

Figure 6.69 Simulation de la génération d'adresses par le module adr_counter.

La Figure 6.68 montre les deux manières de remettre le module à zéro. La première manière est montrée par le curseur numéro un qui est le signal de remise à zéro global du système. La deuxième manière est pointée par le curseur numéro deux et est effectuée par le signal cmptr_raz qui permet de remettre le compteur à zéro lorsque le chargement de la table des couleurs est terminée en mode d'affichage 8 bpp.

La Figure 6.69 montre le compteur en fonctionnement normal. Il s'incrémente de un chaque fois que le signal count_en est actif, comme le montre l'exemple du curseur numéro deux de cette figure.

6.10.7.4 Description du module fin_dload_gen (fin_dload_gen.vhd)

Voici le symbole du module fin_dload_gen :

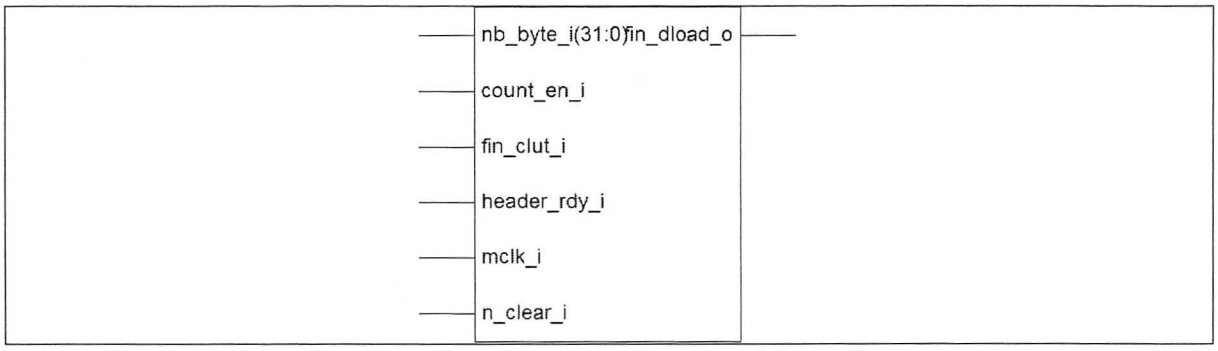

Figure 6.70 Symbole du module fin_dload_gen.

Avant de décrire le fonctionnement du module fin_dload_gen, voici un tableau de ses entrées et de ses sorties :

Tableau 6.20

Description des entrées et des sorties du module fin_dload_gen

Nom	Entrée/ Sortie	Description
mclk	E	Signal d'horloge maîtresse, voir les sections 6.7 et 6.7.4 pour plus de détails.
n_clear	E	Signal de réinitialisation générale du système. Voir la section 6.9 pour plus de détails.
count_en	E	Signal qui indique au module adr_counter quand générer des valeurs d'adresse. Ce signal est généré dans le module bootloader par un processus. Il est actif uniquement lorsque l'entête des données dans la PROM est terminée et il se désactive à la fin du chargement des données.
header_rdy	E	Signal qui indique lorsque l'entête des données de la PROM est entièrement lu. Ce signal est généré par le module pattern_reco_loader
nb_bytes[32]	E	Signal qui indique au bootloader combien d'octets devront être chargés. Cette valeur est sur 32 bits et se situe immédiatement après le patron de bits qui identifie l'emplacement des données dans la PROM.
fin_dload	S	Signal qui indique que toutes les données d'images ainsi que tous les paramètres provenant de la PROM ont été chargées dans la mémoire vidéo (le contrôleur vidéo peut donc être activé). Ce signal met fin au chargement par le module bootloader. Ce signal est généré par le module bootloader0/dload_adr_gen0.

6.10.7.4.1 Description du fonctionnement du module fin_dload_gen

Cette unité permet de faire le décompte des octets chargés en mémoire vidéo et de déterminer la fin du chargement des données à partir du nombre spécifié par le signal nb_bytes[32]. Cet élément commence le décompte dès que l'entête de la PROM est terminé, ce qui est annoncé par le signal header_rdy. Le schéma bloc du module est inséré dans celui de l'entité dload_adr_gen à la Figure 6.60.

Le décompte se fait par pas de 4 octet lors du chargement des données de pixels en mémoire et par pas de 3 octets lors du chargement de la table de couleurs dans le cas d'un mode d'affichage de 8 bpp.

6.10.7.4.2 Résultats des simulations du module fin_dload_gen

Voici la simulation fonctionnelle du module fin_dload_gen :

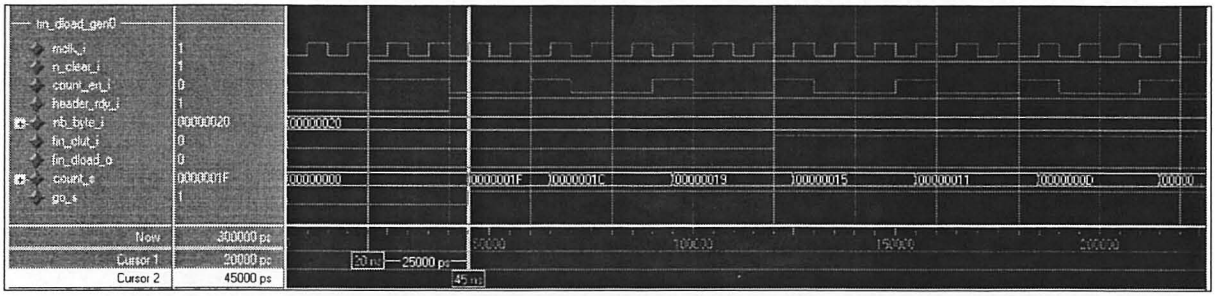

Figure 6.71 Simulation de l'initialisation du module fin_dload_gen.

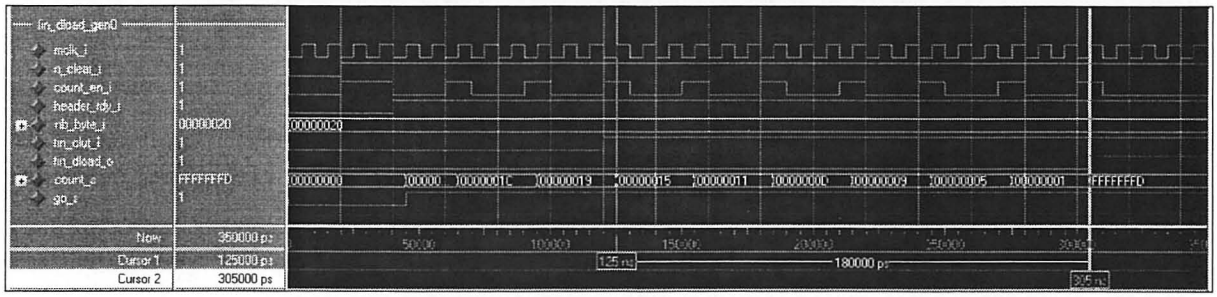

Figure 6.72 Simulation du décompte du nombre d'octets restant à charger effectué par le module fin_dload_gen.

La Figure 6.71 montre la mise à zéro du compteur count lors de l'activation du signal n_clear. Elle montre aussi sont initialisation à la valeur de nb_bytes lorsque le signal header_rdy s'active. À ce moment, au curseur numéro deux de cette figure, le compteur est prêt à effectuer son décompte pour identifier la fin du chargement des données.

La Figure 6.72 montre le décompte du compteur count. Avant l'activation du signal fin_clut (curseur numéro un de la figure), le décompte s'effectue par pas de 3. Suite à l'activation du signal fin_clut, on est donc en mode chargement des données en mémoire, donc le décompte se fait par pas de 4. Aussitôt que le compteur passe sous la valeur zéro, le bit le plus significatif du compteur count s'active et active par le fait même le signal fin_dload, ce qui est pointé par le curseur numéro deux de cette figure.

6.11 Description du module VGA (vga.vhd)

Voici le symbole du module VGA :

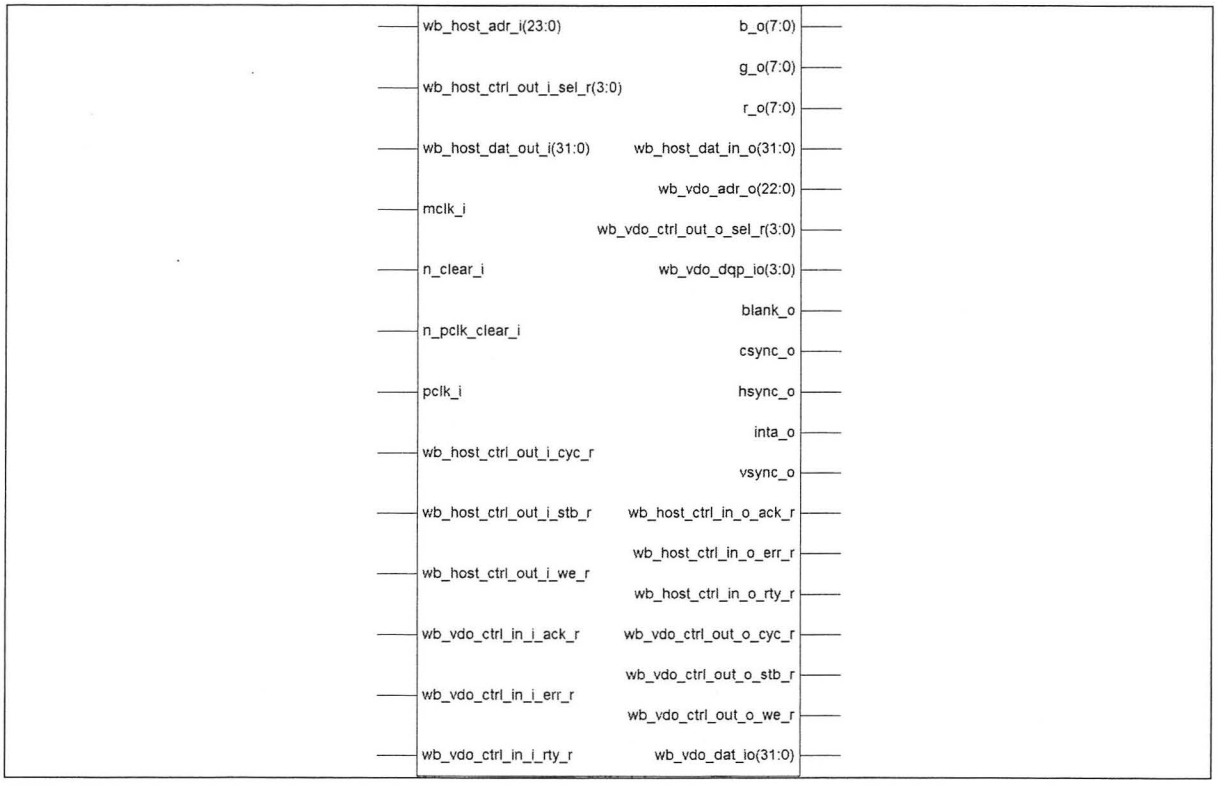

Figure 6.73 Symbole du module VGA.

Avant de décrire le fonctionnement du module VGA, voici un tableau de ses entrées et de ses sorties :

Tableau 6.21

Description des entrées et des sorties du module VGA

Nom	Entrée/ Sortie	Description
mclk	E	Signal d'horloge maîtresse, voir les sections 6.7 et 6.7.4 pour plus de détails.
pclk	E	Signal d'horloge pixel. Voir les sections 6.7 et 6.7.3 pour plus de détails.
n_clear	E	Signal de réinitialisation générale du système. Voir la section 6.9 pour plus de détails.
inta	S	Sortie globale du signal d'interruption.
wb_host_dat_in[32]	S	Sortie des données du bus Wishbone vers le module host (bootloader). Ce signal n'est pas vraiment utilisé en ce moment, mais pourra dans le futur permettre une lecture de l'état des registres (module ctrl_register) du contrôleur vidéo.
wb_host_adr[24]	E	Signal d'adresse du bus Wishbone du Host. Le maître est le module bootloader et l'esclave est le module VGA.
wb_host_dat_out[32]	E	Signal de donnée du bus Wishbone du Host. Le maître est le module bootloader et l'esclave est le module VGA.
wb_host_ctrl_out[7]	E	Groupe de signaux de contrôle du bus wishbone host. Ce groupe de signaux inclut entre autres les signaux cyc et stb qui permettent d'entamer une transaction sur le bus. Le signal sel permet de savoir la taille de la transaction. Pour plus de détails sur le fonctionnement du bus Wishbone, référez-vous au document [23] Herveille, 2002.
wb_host_ctrl_in[3]	S	Entrée de contrôle du bus Wishbone pour le module host (bootloader). C'est un groupe de signaux qui permet au module esclave de répondre au maître. Ce groupe de signaux comprend les signaux ack, err et rty. Le signal ack est le seul utilisé présentement et permet de dire au maitre que la transaction est acceptée.
wb_vdo_adr[23]	S	Signal d'adresse Wishbone entre le contrôleur vidéo (module VGA) et la mémoire vidéo (mémoire ZBT).
wb_vdo_dat[32]	E/S	Signal de données bidirectionnel Wishbone entre le contrôleur vidéo (module VGA) et la mémoire vidéo (mémoire ZBT).
wb_vdo_dqp[4]	E/S	Signal des bits de parité bidirectionnel Wishbone entre le contrôleur vidéo (module VGA) et la mémoire vidéo (mémoire ZBT).
wb_vdo_ctrl_out[7]	S	Groupe de signaux de contrôle du bus wishbone vidéo (entre le contrôleur vidéo et la mémoire ZBT). Ce groupe de signaux inclut entre autres les signaux cyc et stb qui permettent d'entamer une transaction sur le bus. Le signal sel permet de savoir la taille de la transaction. Pour plus de détails sur le fonctionnement du bus Wishbone, référez-vous au document [23] Herveille, 2002.
wb_vdo_ctrl_in[3]	E	Entrée de contrôle du bus Wishbone pour le contrôleur vidéo (maître). C'est un groupe de signaux qui permet au module esclave (module wb2zbt_wraper) de répondre au maître. Ce groupe de signaux comprend les signaux ack, err et rty. Le signal ack est le seul utilisé présentement et permet de dire au maître que la transaction est acceptée.
r[8]	S	Signal de donnée de la couleur rouge vers le convertisseur vidéo
g[8]	S	Signal de donnée de la couleur vert vers le convertisseur vidéo
b[8]	S	Signal de donnée de la couleur bleu vers le convertisseur vidéo

hsync	S	Signal de synchronisation vidéo horizontal vers le convertisseur vidéo
vsync	S	Signal de synchronisation vidéo vertical vers le convertisseur vidéo
csync	S	Signal de synchronisation vidéo composite vers le convertisseur vidéo
blank	S	Signal de synchronisation vidéo qui identifie les sections visibles de l'image. Ce signal se dirige vers le convertisseur vidéo

6.11.1 Description du fonctionnement du module VGA

Le contrôleur VGA est structuré de manière à s'interfacer facilement avec un processeur d'un côté et avec une mémoire vidéo de l'autre côté. Cependant, le contrôleur est présentement relié à un module plus simple qu'un processeur : le bootloader (section 6.10). Ce bootloader permet une communication simple avec le contrôleur VGA pour le chargement de données en mémoire vidéo et aussi pour la configuration du contrôleur VGA.

L'unité VGA possède trois fonctions principales : la mémorisation des paramètres de configuration, la gestion du chargement et de la mise en forme des pixels et la génération des signaux de synchronisation pour l'affichage VGA. Voici le schéma bloc du module top dont le module VGA fait partie :

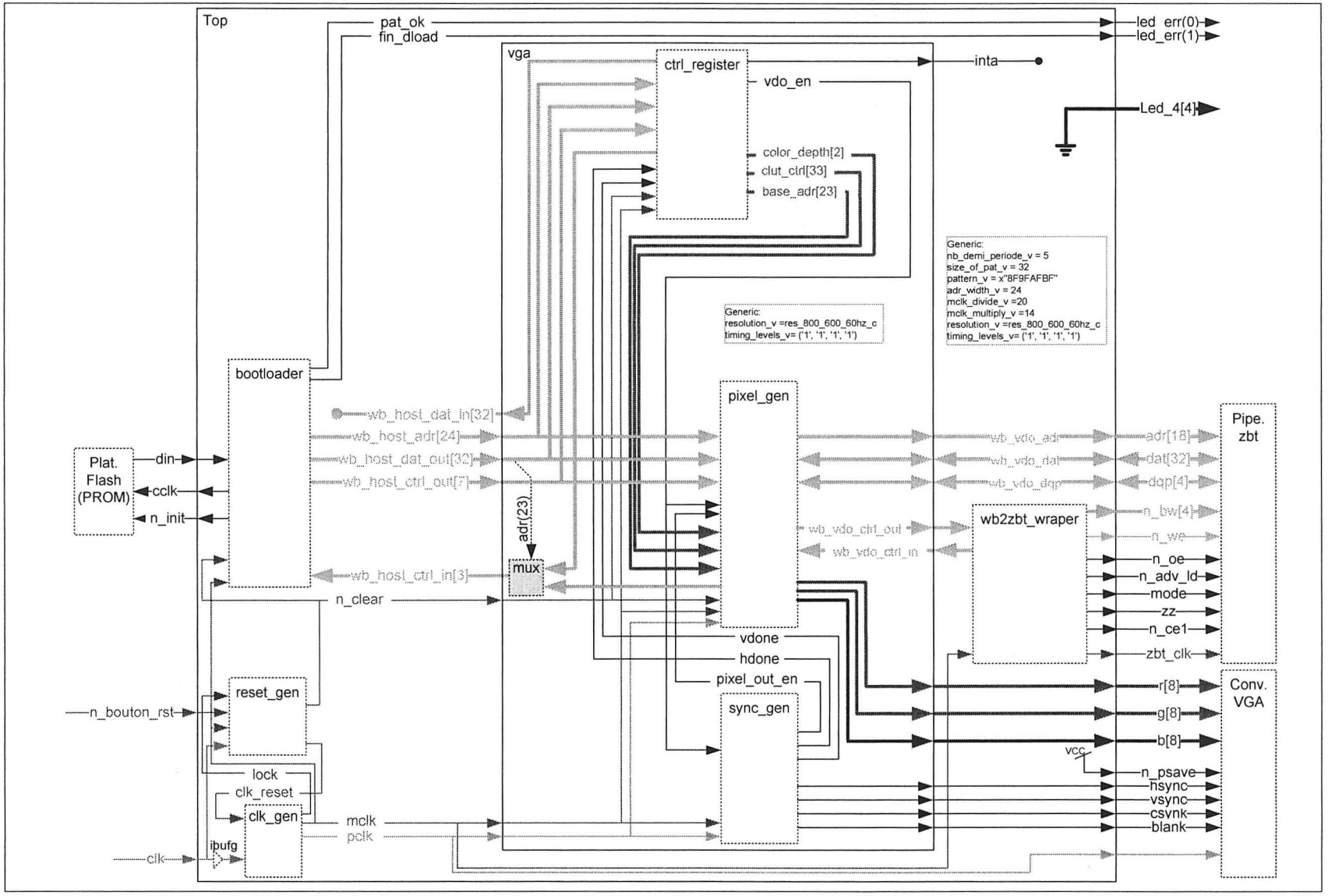

Figure 6.74 Schéma du module top avec le module VGA inclus.

Le sous-module ctrl_register permet de mémoriser dans ses registres tous les paramètres d'affichage. Il communique ces paramètres aux autres sous-modules en permanence. On peut modifier ces paramètres en accédant au bus wishbone host à partir de l'adresse 0x800000. Les détails de configurations sont donnés à la section sur le module ctrl_register (Section 6.11.3).

Le sous-module pixel_gen est l'unité qui permet de gérer le trafic des pixels sur les deux bus Wishbone. Il permet donc le transfert des données du bootloader vers la mémoire vidéo par les accès wishbone du host dont les adresses sont inférieures à 0x800000. Autrement, lorsque l'affichage est activé par le signal vdo_en, le module pixel_gen permet de lire les données des pixels en paquet de 32 bits de la mémoire vidéo et de les convertir en données de couleurs RGB vers la sortie du convertisseur VGA. Pour plus de détails sur ce module, référez-vous à la section 6.11.5.

Le sous-module sync_gen permet de générer les signaux de contrôle du port VGA vers le convertisseur VGA. Ces signaux doivent absolument respecter les temps avec précision, c'est pour cette raison qu'une horloge spéciale nommée pclk cadencée exactement à la fréquence de sortie des pixels permet de faire le décompte de chacun des délais de synchronisme à l'aide de compteurs et de machines à états finis. Les valeurs de ces compteurs, et donc de ces délais, sont configurables par l'intermédiaire de variables générique du module top. Une table des valeurs de compteurs est donnée au

Tableau 5.1 de la section 5.1.3. Le module sync_gen permet de déterminer le début et la fin d'une image. Pour plus de détail sur cette unité, référez-vous à la section 6.11.3.2.

6.11.2 Résultats des simulations du module VGA

Voici la simulation fonctionnelle du module VGA :

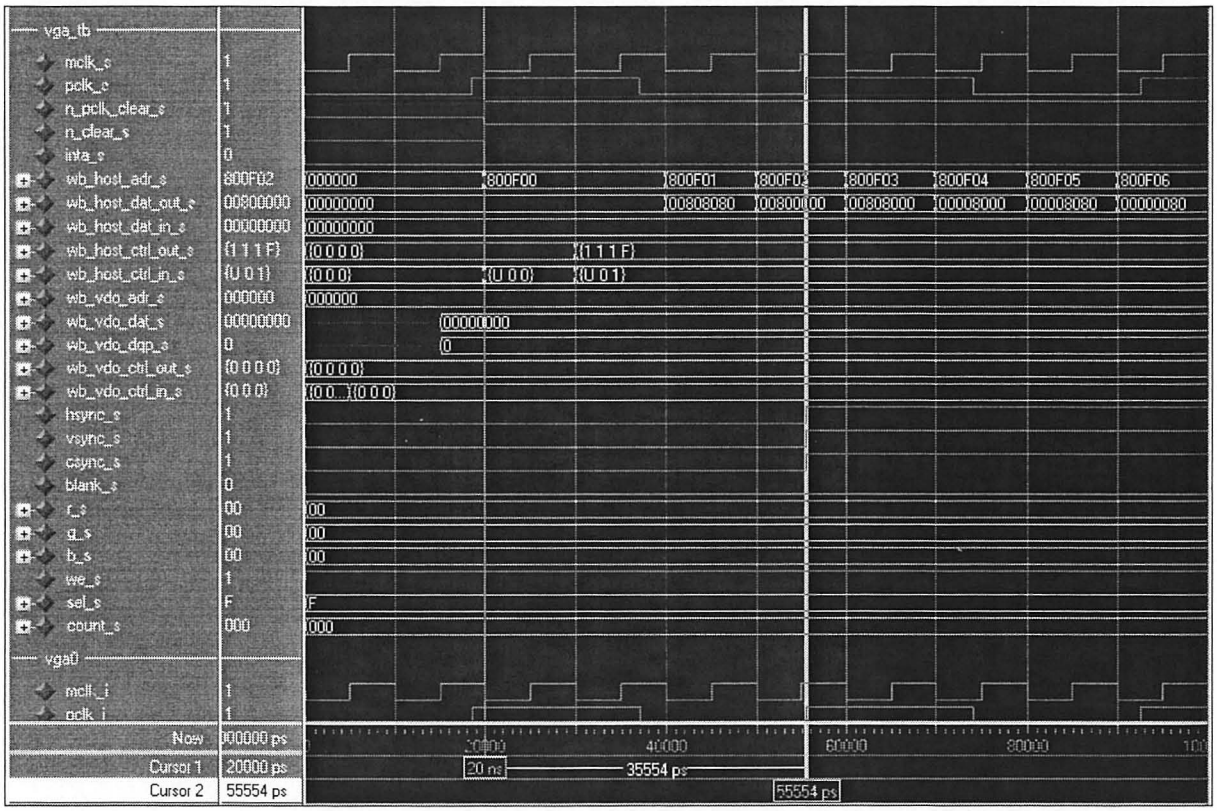

Figure 6.75 Simulation de l'initialisation du module VGA.

La Figure 6.75 montre l'initialisation du module VGA. On remarque que le module se remet en fonction après deux cycles d'horloge suivant la relâche du signal n_clear. Ce moment débute le chargement de la table des couleurs dans le module clut puisque les adresses placées sur le bus wishbone sont de 0x800F00 et plus. Le curseur numéro 2 montre que les signaux de synchronisation (hsync, vsync, csync et blank) se mettent en attente du début de l'affichage de manière synchrone avec l'horloge pixel pclk.

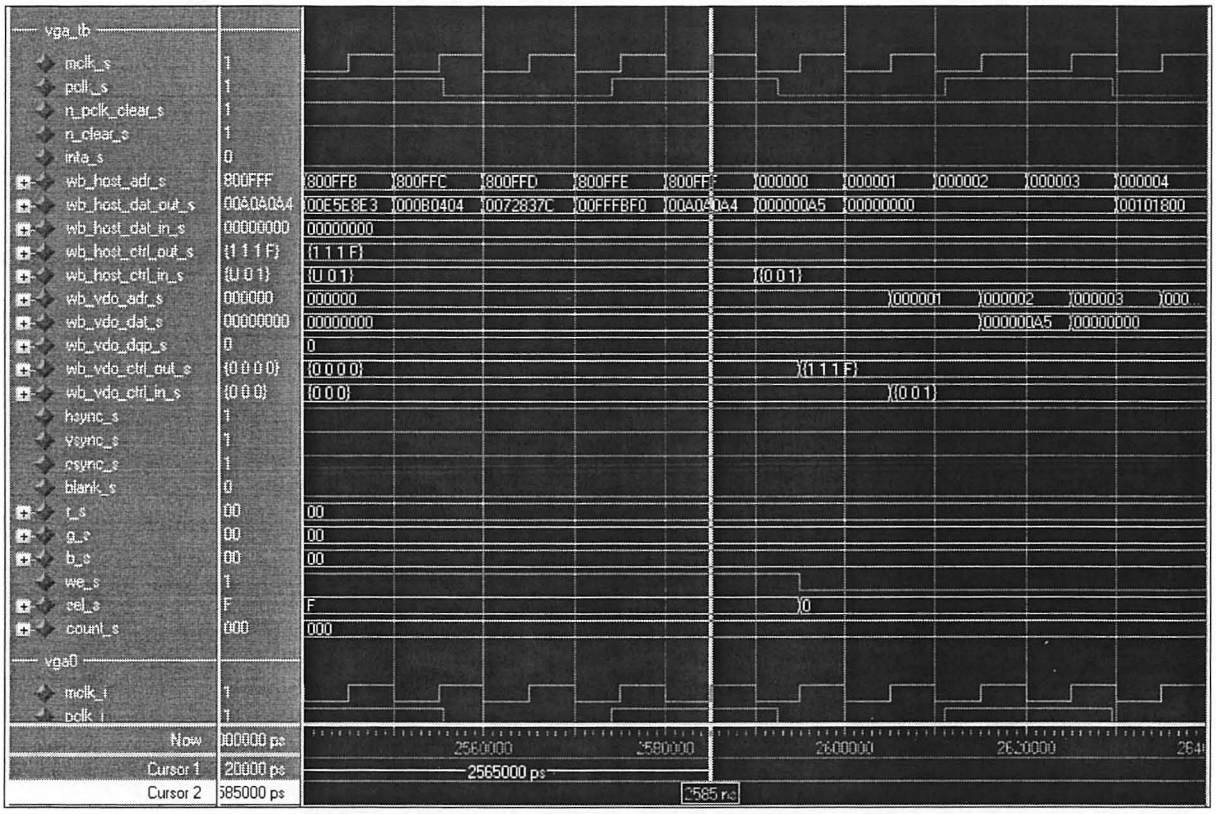

Figure 6.76 Simulation de la fin du chargement de la table des couleurs du module VGA.

La Figure 6.76 montre au curseur numéro 2 la fin du chargement des couleurs dans la table des couleurs du module clut. Le chargement des données d'image est immédiatement débuté à la suite du curseur, soit à l'adresse 0x000000 avec la donnée 0x000000A5. On peut observer le passage de la donnée et de l'adresse de l'entrée du bus wishbone host vers le bus wishbone vidéo. On remarque aussi sur cette figure et entre les deux curseurs de la Figure 6.77 que la donnée est décalée de deux cycles d'horloge par rapport à l'adresse sur le bus wishbone vidéo afin de synchroniser l'écriture dans la mémoire ZBT qui est fabriquée en pipeline. Ce pipeline demande de placer l'adresse et les signaux de contrôle de la mémoire en avance de deux cycles par rapport à la donnée.

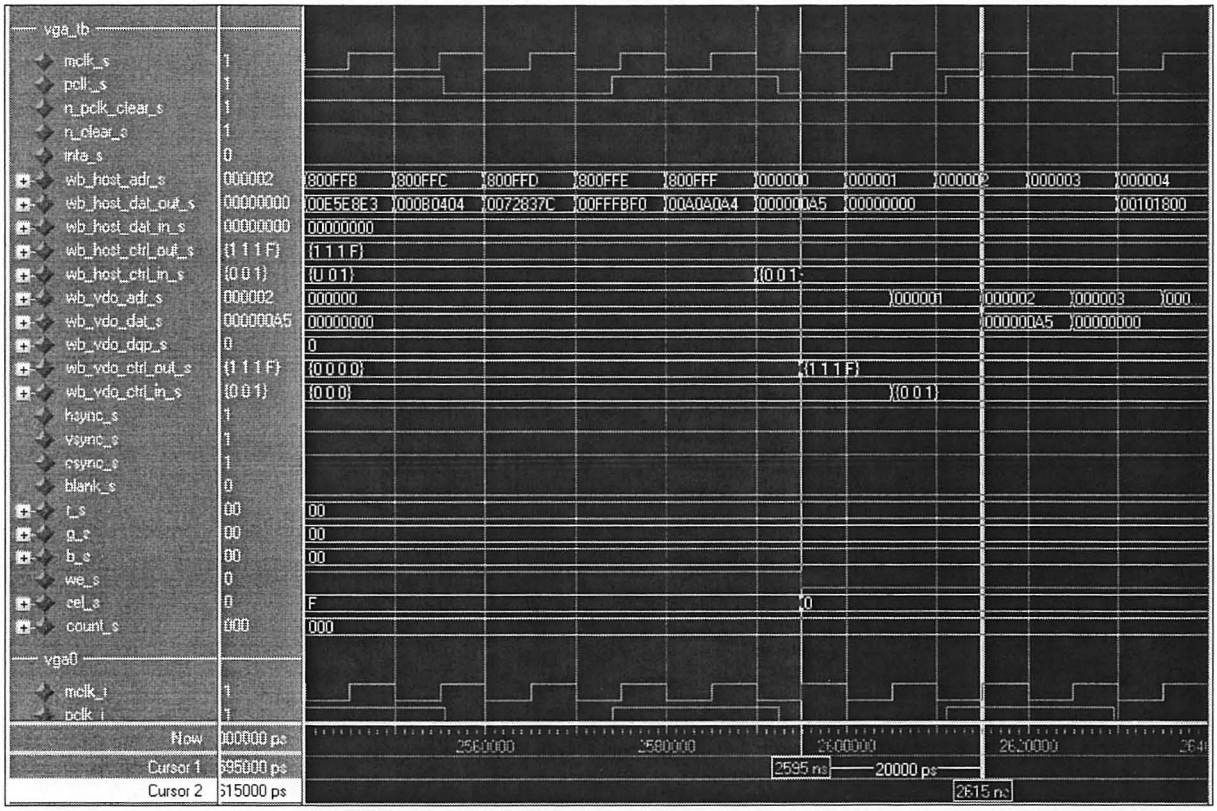

Figure 6.77 Simulation du module VGA lors du chargement de la mémoire vidéo et démonstration de son pipeline.

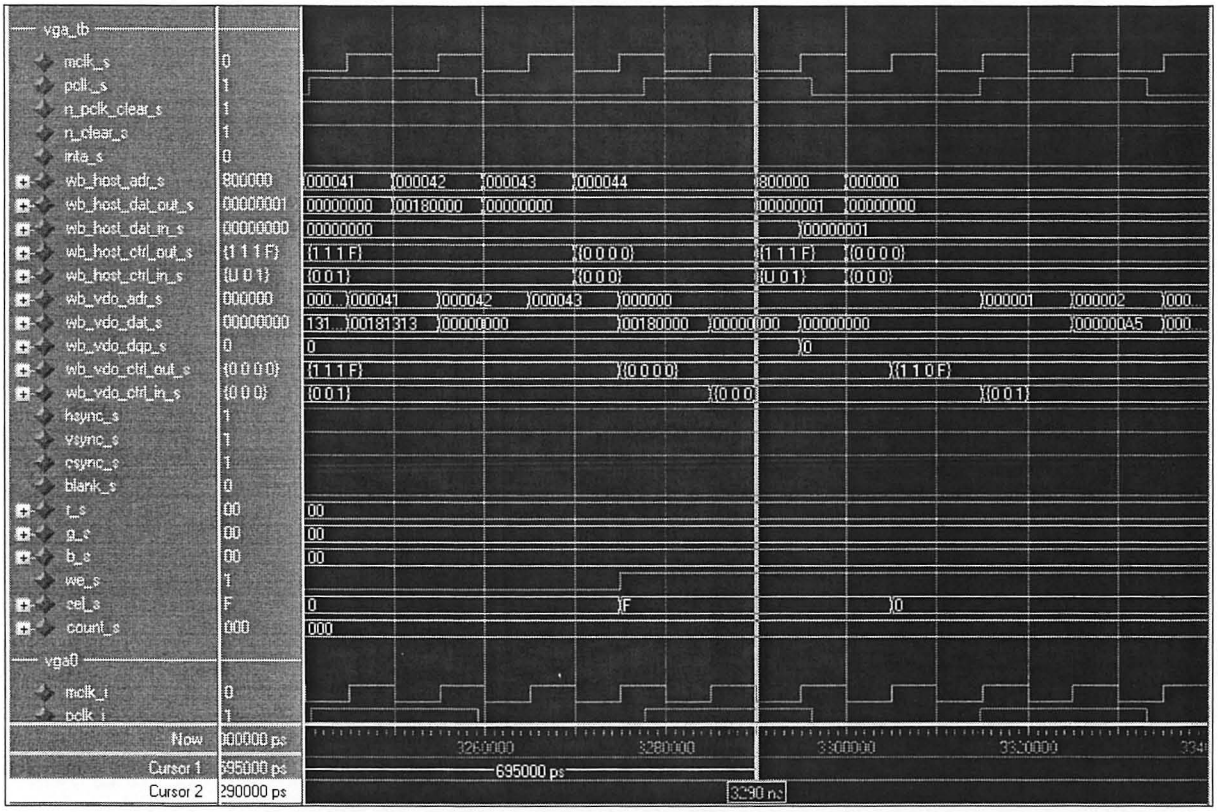

Figure 6.78 Simulation de l'activation de l'affichage vidéo sur le module VGA.

Le curseur numéro deux de la Figure 6.78 montre la donnée à envoyer au module VGA pour activer l'affichage en mode 8 bits par pixel. On doit placer cette donnée à l'adresse 0x800000 afin de la sauvegarder dans le module ctrl_register. La valeur 0x00000001 semble simple, mais elle contient deux champs importants. Le bit 0 contient la valeur « 1 » qui est la valeur donnée au signal d'activation vidéo nommé vdo_en. Les bits 4 et 5 possèdent la valeur « 00 » qui spécifie la profondeur des couleurs qui est dans ce cas de 8 bits par pixels. Pour plus de détails sur le module ctrl_register, référez-vous à la section 6.11.3.

La Figure 6.79 montre ce qui se passe immédiatement après l'activation de l'affichage. Le module fifo se remplit de 8 données de 32 bits, c'est ce qu'on voit entre les deux curseurs de l'image. On devrait s'attendre à 16 données plutôt que 8 puisque le module fifo a une profondeur de 16. Par contre, l'arrêt du remplissage de ce fifo s'arrête dès l'atteinte de la moitié de sa profondeur maximale afin d'éliminer le risque de dépassement lié au décalage de la réception des données provenant du pipeline de la mémoire vidéo.

Dès l'apparition de la première donnée à la sortie du fifo, le module pixel_processor (expliqué à la section 6.11.5.5) exécute le traitement de cette donnée en fonction de la profondeur des pixels (8 ou 24 bits par pixel) et l'achemine au module dclk_fifo (section 6.11.5.6). Tout ce traitement s'effectue dans l'intervalle de temps délimité par les deux curseurs de la Figure 6.79. On voit bien le premier pixel prêt à être affichée tout de suite après le curseur numéro deux sur les signaux r, g et b. Ces trois signaux étant synchronisés avec l'horloge pixel pclk, il est normal de les voir prendre leur première valeur au front montant de celle-ci.

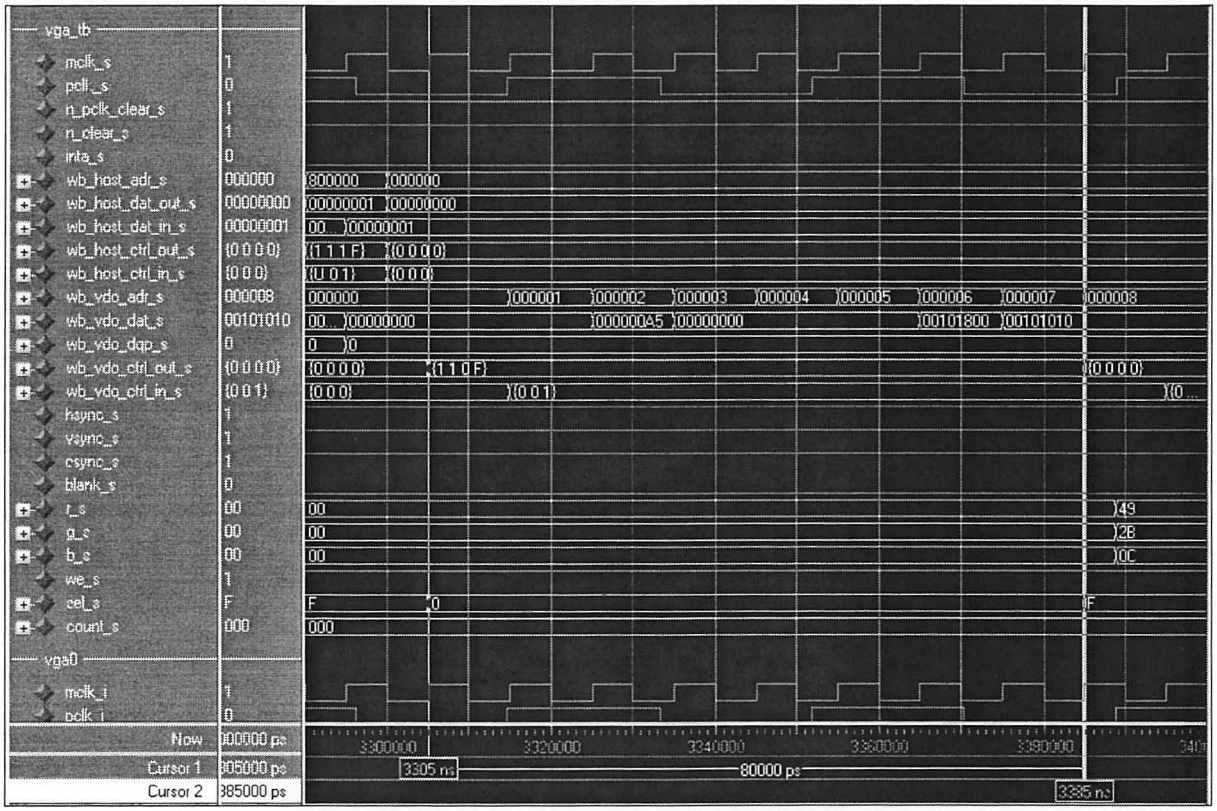

Figure 6.79 Simulation de la lecture des pixels en mémoire vidéo et du remplissage des fifos dans le module VGA.

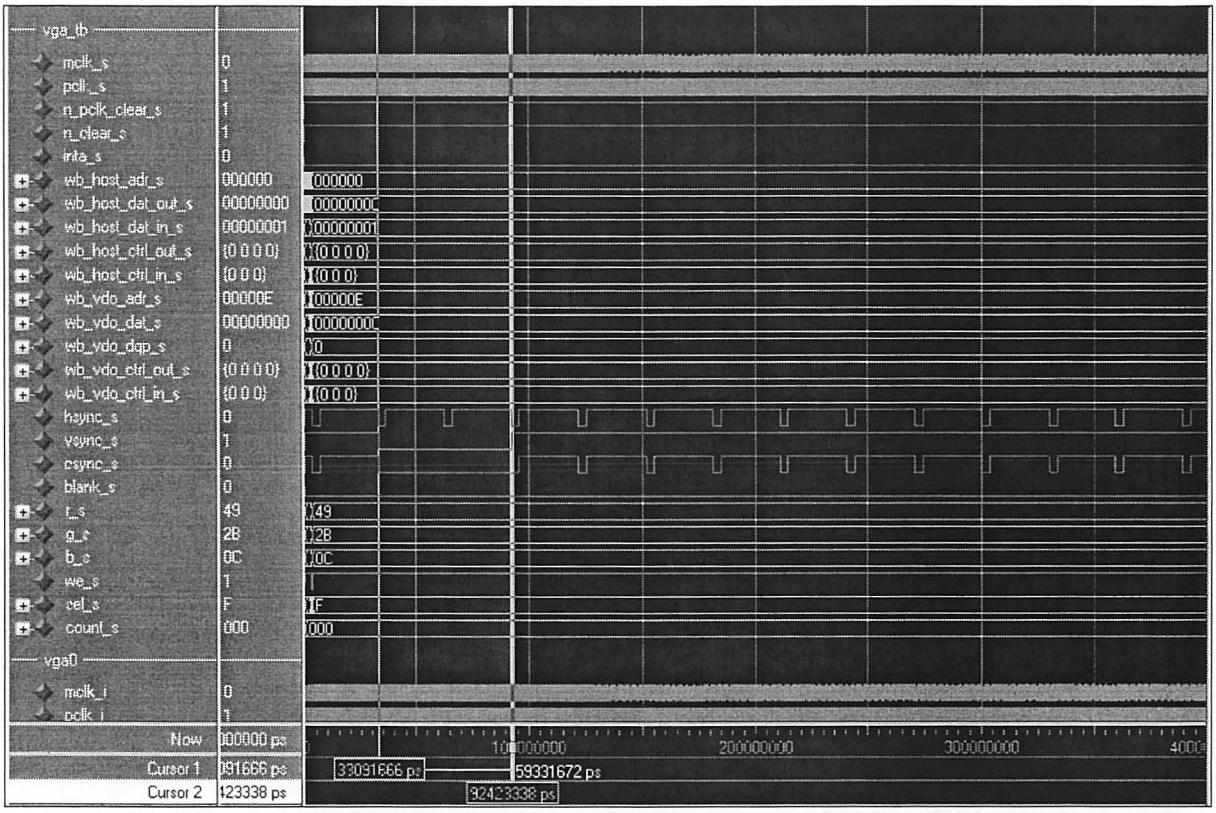

Figure 6.80 Simulation de la génération des signaux sync par le module VGA.

En changeant l'échelle d'affichage de la simulation du système VGA, on observe la génération des signaux de synchronisation vsync, hsync et csync. La Figure 6.80 montre que le signal hsync effectue un premier cycle complet avant de permettre au signal vsync de s'activer au curseur numéro un, permettant ainsi le début du cycle d'affichage de l'image. Les deux curseurs de cette image délimitent les deux cycles de balayages demandés par la résolution de l'image pour la temporisation du signal vsync (référez-vous au

Tableau 5.1 pour les valeurs des champs de temporisation pour la résolution 800x600 à 60 Hz).

Cette figure montre aussi que les signaux sync sont actif bas. Il est possible de les faire fonctionner dans les deux polarités en changeant la variable générique timing_levels. Les valeurs par défaut forcent les signaux sync actifs bas et le signal blank actif haut puisque c'est ce qui est le plus populaire sur les écrans conventionnels. À ce sujet, on remarque sur la figure que le signal blank est inactif puisque le champ de temporisation gate n'est pas encore atteint.

La Figure 6.81 montre, entre les deux curseurs, le champ de temporisation vertical back_porch qui comptabilise 31 cycles de balayage. Suite à cela, le champ gate vertical débute (au curseur numéro deux) et le signal blank s'active. Le signal blank est l'intersection (porte logique « and ») entre les champs de temporisation gate vertical et horizontal. Cette figure ne permet pas de voir les premiers pixels évoluer aux sorties r, g et b puisque son échelle d'affichage est trop grande, mais on remarque un changement de valeurs vis-à-vis le curseur numéro deux.

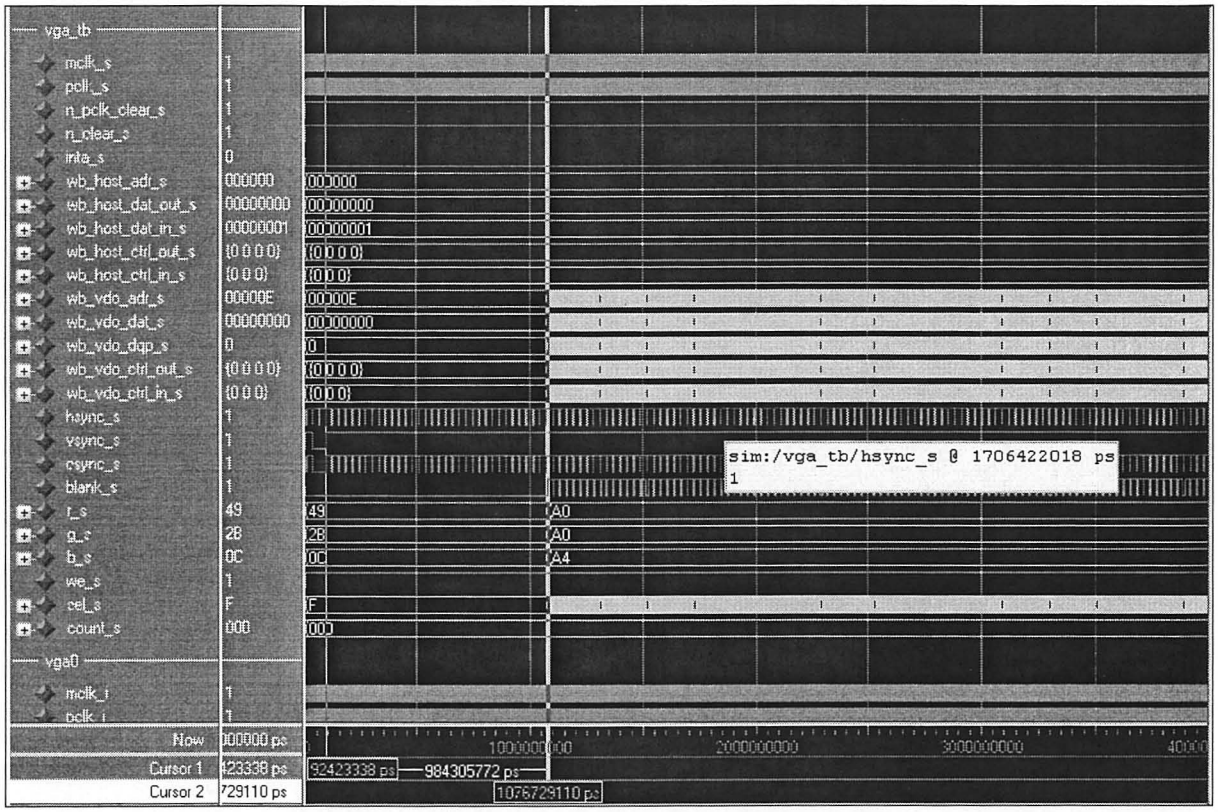

Figure 6.81 Simulation du back_porch et du début de l'activation du signal blank par le module VGA.

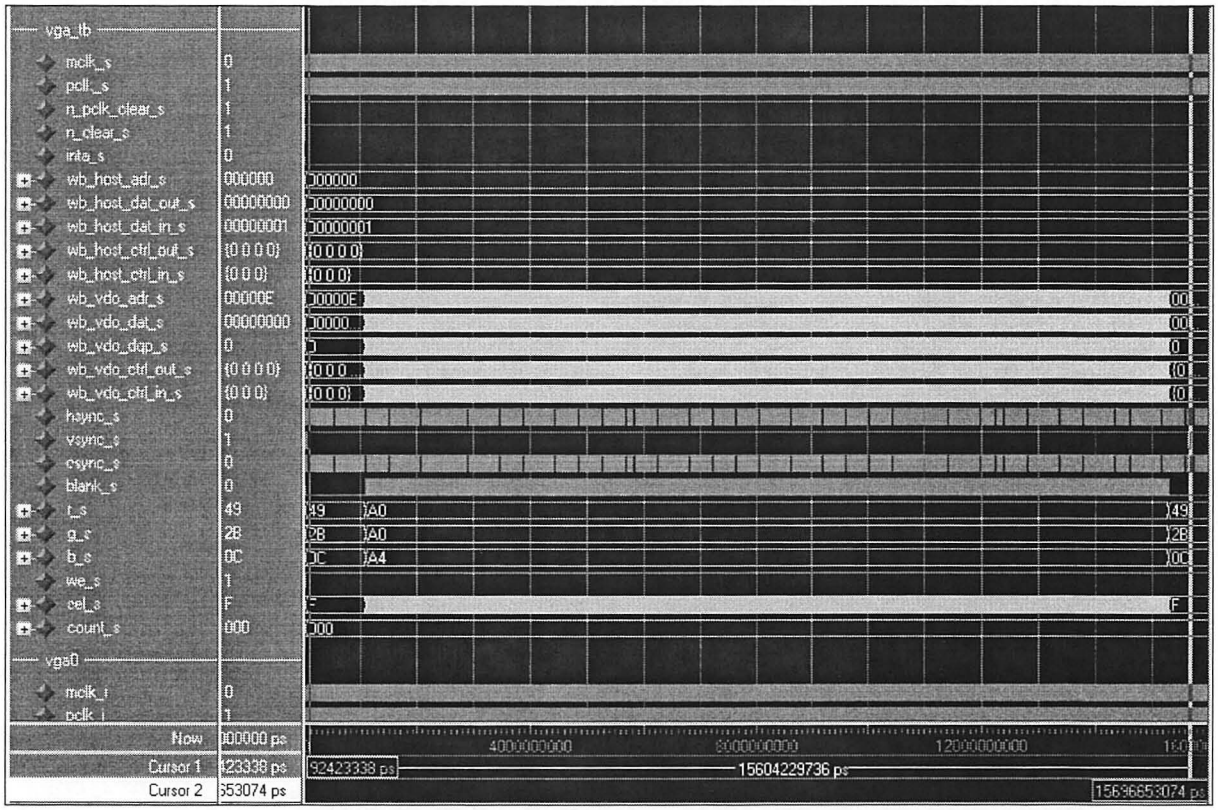

Figure 6.82 Simulation complète d'un sync à l'autre du module VGA.

La Figure 6.82 montre l'affichage complet d'une image entre les deux curseurs, d'un sync vertical à l'autre. Tout près du curseur numéro deux, on peut voir que l'image se prépare pour un autre affichage en rechargeant les valeurs du premier pixel aux sorties r, g et b. Le temps de balayage montré par les deux curseurs est de 15,6 ms ce qui ne correspond pas à la valeur nominale pour un taux de rafraichissement de 60 Hz. Ceci est dû au fait que la fréquence pixel pclk n'est pas tout à fait conforme à la valeur nominale de 25,175 MHz. Selon les paramètres de génération de l'horloge pclk, la fréquence générée de celle-ci est de 100 MHz * 8/ 31 = 25,81 MHz, ce qui correspond à la valeur la plus proche qui permet de faire fonctionner l'affichage. Donc le balayage de l'image est un peu plus rapide que la norme. La différence n'est pas assez significative pour empêcher l'écran de fonctionner, mais lorsque la fréquence est différente de la norme, il est parfois nécessaire de modifier les valeurs de compteur pour assurer le bon fonctionnement de l'affichage.

6.11.3 Description du module ctrl_register (ctrl_register.vhd)

Voici le symbole du module ctrl_register :

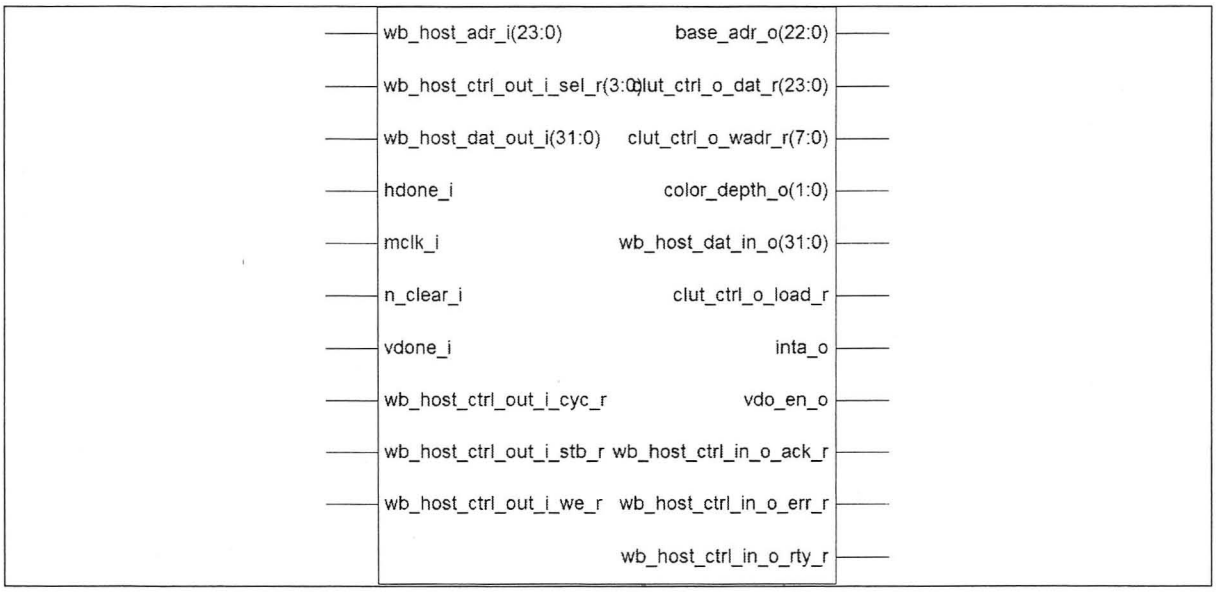

Figure 6.83 Symbole du module ctrl_register.

Avant de décrire le fonctionnement du module ctrl_register, voici un tableau de ses entrées et de ses sorties :

Tableau 6.22

Description des entrées et des sorties du module ctrl_register

Nom	Entrée/ Sortie	Description
mclk	E	Signal d'horloge maîtresse, voir les sections 6.7 et 6.7.4 pour plus de détails.
n_clear	E	Signal de réinitialisation générale du système. Voir la section 6.9 pour plus de détails.
wb_host_dat_out[32]	E	Signal de donnée du bus Wishbone du Host. Le maître est le module bootloader et l'esclave est le module VGA (et ses sous-modules, dont le module ctrl_register fait partie).
wb_host_adr[24]	E	Signal d'adresse du bus Wishbone du Host. Le maître est le module bootloader et l'esclave est le module VGA (et ses sous-modules, dont le module ctrl_register fait partie).
wb_host_ctrl_out[7]	E	Groupe de signaux de contrôle du bus wishbone host. Ce groupe de signaux inclut entre autres les signaux cyc et stb qui permettent d'entamer une transaction sur le bus. Le signal sel permet de savoir la taille de la transaction. Pour plus de détails sur le fonctionnement du bus Wishbone, référez-vous au document [23] Herveille, 2002.

wb_host_ctrl_in[3]	S	Entrée de contrôle du bus Wishbone pour le module host (bootloader). C'est un groupe de signaux qui permet au module esclave de répondre au maître. Ce groupe de signaux comprend les signaux ack, err et rty. Le signal ack est le seul utilisé présentement et permet de dire au maître que la transaction est acceptée.
hdone	E	Signal qui indique la fin d'une ligne d'affichage. Ce signal est généré par le module sync_gen.
vdone	E	Signal qui indique la fin de l'affichage d'une image. Ce signal est généré par le module sync_gen.
vdo_en	E	Signal général qui indique si le contrôleur vidéo est activé ou non, ce signal provient du module ctrl_register et correspond au bit ctrl_reg(0).
color_depth[2]	S	Signal qui spécifie le nombre de bits d'encodage de l'image : • - 00 pour un encodage 8 bits • - 01 pour un encodage 16 bits (pas encore implémenté) • - 1x pour un encodage 24 bits
base_adr[23]	S	Adresse de départ de l'image sélectionnée. Deux banques d'images sont disponibles dans le module ctrl_register. Pour modifier l'image affichée, il suffit d'inscrire son index dans le registre approprié, soit ctrl_reg[5 :4]. On accède à ce registre à l'adresse 0x800000.
clut_ctrl[33]	S	Groupe de signaux qui permettent de contrôler le chargement de la table des couleurs (module clut) par le module ctrl_register. Ce groupe contient les signaux : • - load : signal d'activation de chargement d'une donnée • - adr[8] : emplacement de la donnée à charger dans la table • - dat[24] : données de 24 bits à chargée
wb_host_dat_in[32]	S	Sortie des données du bus Wishbone vers le module host (bootloader). Ce signal n'est pas vraiment utilisé en ce moment, mais pourra dans le futur permettre une lecture de l'état des registres (module ctrl_register) du contrôleur vidéo.
inta	S	Sortie globale du signal d'interruption.

6.11.3.1 Description du fonctionnement du module ctrl_register

On comprend aisément le rôle du module ctrl_register par son nom. Il permet en effet de mémoriser les paramètres de contrôles qui servent à configurer tous les modules internes du contrôleur VGA. On peut voir un schéma bloc du module dans la figure suivante :

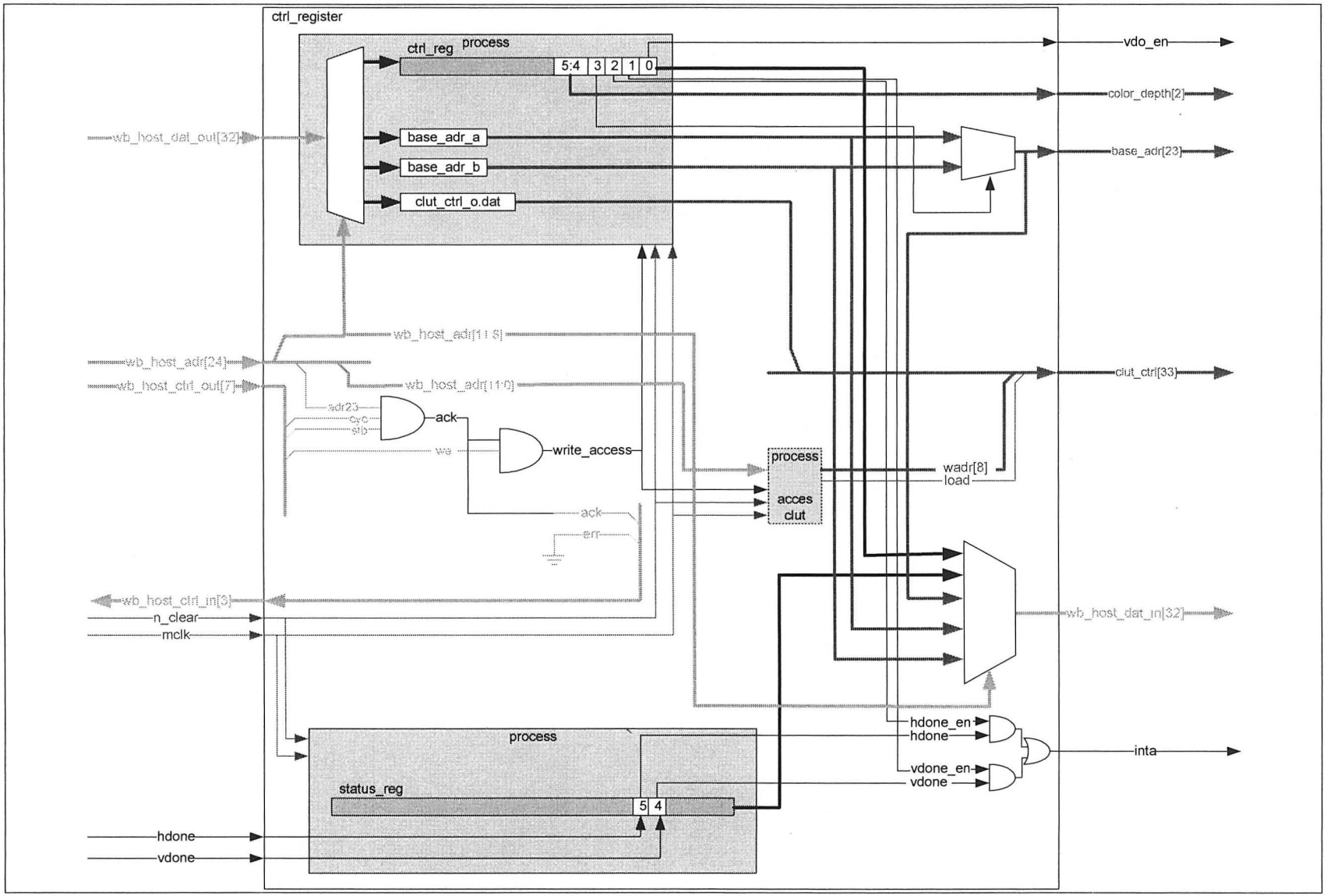

Figure 6.84 Schéma du module ctrl_register.

Les registres de contrôle sont accessibles à partir de l'adresse 0x800000 sur le bus wishbone host. Les bits 11 à 8 de l'adresse correspondent au registre visé. Pour faciliter la compréhension de ce module, voici un tableau des registres avec leur adresse d'accès et les champs qu'ils contiennent :

Tableau 6.23

Description des registres internes du module ctrl_register

Nom	adresse d'accès	Description
ctrl_reg	0x800000	Ce registre est le registre de configuration de base, il est accessible en lecture et en écriture et possède les champs suivants : • ctrl_reg_s(5 downto 4) : précision des couleurs 00 = 8 bits 01 = 16 bits 1x = 24 bits • ctrl_reg_s(3) : banque d'image active 0 = base_adr_a_s 1 = base_adr_b_s • ctrl_reg_s(2) : activation de l'interruption de fin de course horizontale • ctrl_reg_s(1) : activation de l'interruption de fin de course verticale • ctrl_reg_s(0): activation du signal vidéo (vdo_en)
status_reg	0x800100	Registre accessible en lecture seulement qui permet de connaitre l'état de la machine. • status_reg(5) = hdone • status_reg(4) = vdone
base_adr_a	0x800800	Ce registre contient l'adresse de départ de la première image en banque, tous les 32 bits sont utilisés. Ce registre est accessible en lecture et en écriture.
base_adr_b	0x800900	Ce registre contient l'adresse de départ de la deuxième image en banque, tous les 32 bits sont utilisés. Ce registre est accessible en lecture et en écriture.
clut	0x800Fxx	Plage d'adresse réservée pour l'accès à la table des couleurs (module clut) en écriture. Dans ce cas-ci, les bits d'adresse 7 à 0 deviennent l'emplacement de la donnée à placer dans la clut. Seul les 24 bits les moins significatifs du bus de données seront considérés.

6.11.3.2 Résultats des simulations :

Voici la simulation fonctionnelle du module ctrl_register :

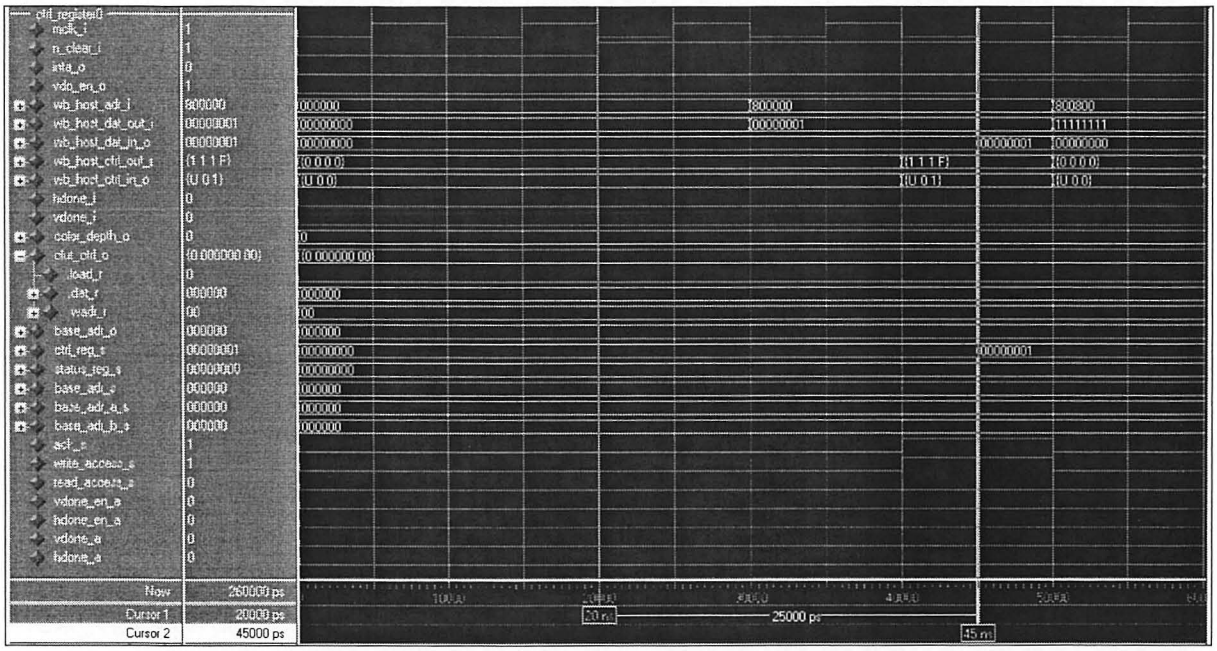

Figure 6.85 Simulation de l'initialisation du module ctrl_register.

La figure précédente montre au curseur numéro un, la réinitialisation de tous les registres. Tous sont forcés à la valeur zéro, ce qui désactive automatiquement le contrôleur vidéo et efface les valeurs d'adresse des images en mémoire vidéo.

Au curseur numéro deux de cette même figure, on place la valeur 0x00000001 à l'adresse 0x800000, ce qui met le bit le moins significatif du registre ctrl_reg actif et donc le signal vdo_en s'active par le fait même. À ce moment précis, le contrôleur vidéo entre en fonction. Le registre ctrl_reg dicte aussi le mode d'affichage par l'intermédiaire des bits 4 et 5. Leur valeur est « 00 » ce qui nous place en mode 8 bpp.

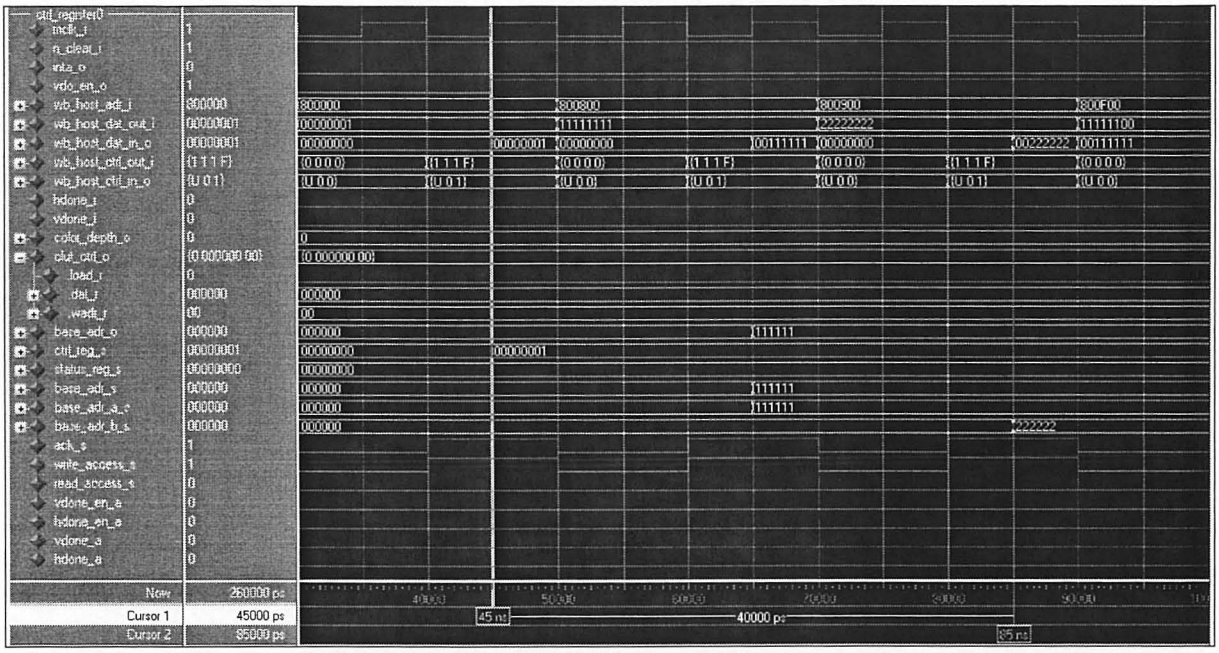

Figure 6.86 Simulation de la modification des adresses de base dans le module ctrl_register.

La Figure 6.86 permet de visualiser ce qui se passe lors du changement des adresses de base des images. Le module ctrl_register possède deux registres qui permettent d'enregistrer l'adresse de départ d'une image dans la mémoire vidéo, il est donc possible de changer d'image rapidement en sélectionnant une adresse de base ou l'autre. C'est pour cette raison qu'on affirme que le système possède deux banques d'image. La banque d'image active est spécifiée par le bit 3 du registre ctrl_reg. Dans le cas de cette figure, l'adresse de base sélectionnée est celle du registre base_adr_a. C'est que lorsqu'on modifie la valeur de ce registre, cette même valeur se retrouve aussi sur le signal base_adr qui prend en tout temps la valeur de l'adresse de base sélectionnée par le registre ctrl_reg.

Pour modifier la valeur de l'adresse de base a, on accède à l'adresse 0x800800 du bus Wishbone host et pour modifier l'adresse de base b, on accède à l'adresse 0x800900. Le curseur numéro deux de la figure précédente montre un changement de l'adresse de base b dans le registre base_adr_b.

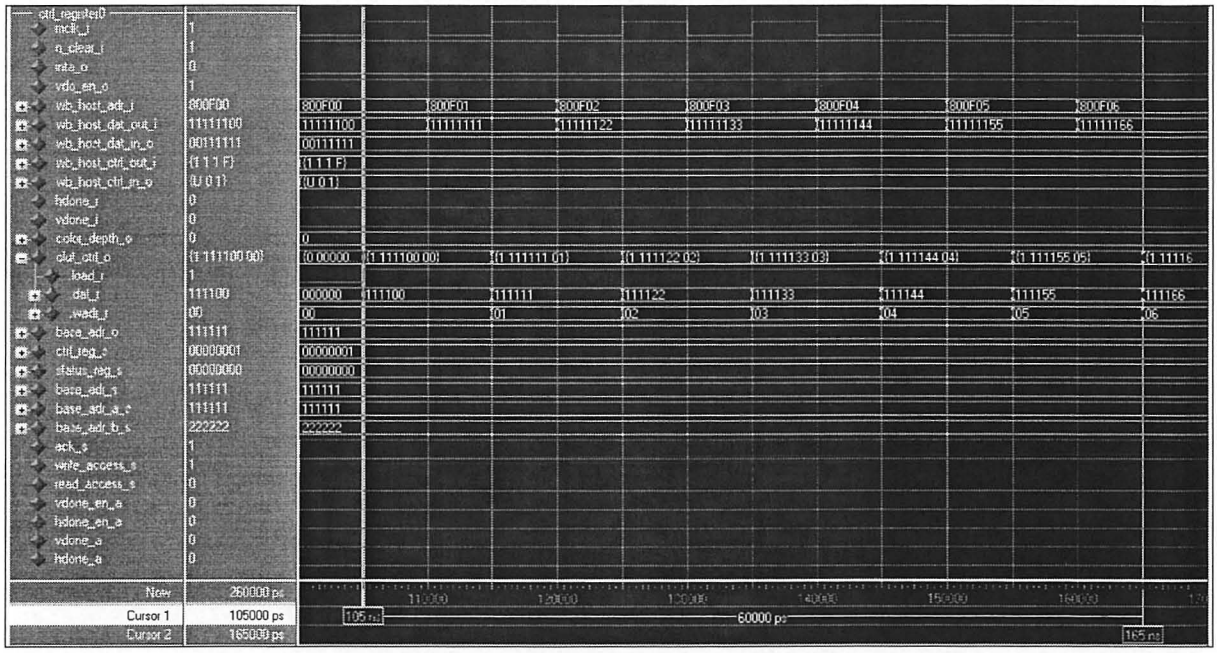

Figure 6.87 Simulation du chargement de la table des couleurs par le module ctrl_register.

La table des couleurs n'est pas contenue dans le module ctrl_register, mais son remplissage se fait par son intermédiaire. On peut écrire dans la table des couleurs à la plage d'adresses de 0x800F00 à 0x800FFF. La Figure 6.87 montre le chargement des premières valeurs de couleur dans la table des couleurs. Au curseur numéro un, on pointe l'adresse 0x800F00 sur le bus wishbone host, ce qui permet d'écrire la valeur des 24 bits les moins significatifs du signal des données du bus wishbone au premier emplacement de la table des couleurs. Ce même curseur montre en effet le signal clut_ctrl.load actif, le signal clut_ctrl.wadr = 0x00 et le signal clut_ctrl.dat = 0x111100. Ces valeurs sont le transfert des valeurs de couleur du bus wishbone vers le signal groupé clut_ctrl.

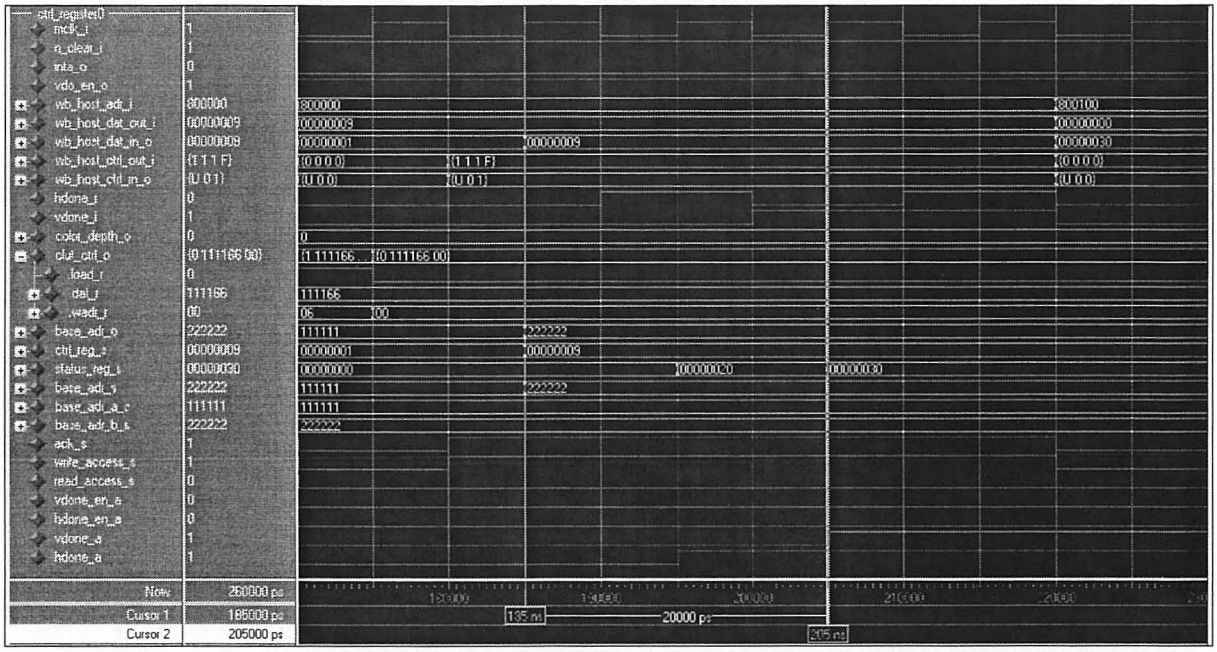

Figure 6.88 Simulation du changement de la banque d'image active dans le module ctrl_register.

Le curseur numéro un de la figure précédente montre l'activation de la banque d'image b en plaçant le bit numéro 3 du registre ctrl_reg à 1. Le signal de sortie base_adr prend aussitôt la valeur du registre base_adr_b.

Le curseur numéro deux de cette même figure expose le fonctionnement des bits d'interruptions. Le bit 5 du registre status_reg s'active dès que le signal hdone s'active. Le bit 4 du registre status_reg s'active dès que le signal vdone s'active. Le registre status_reg demeure ainsi tant que sa valeur n'a pas été lue. La lecture du registre status_reg est montrée sur la figure suivante au curseur numéro un. On y voit très bien la valeur du registre status_reg se remettre à zéro à ce moment.

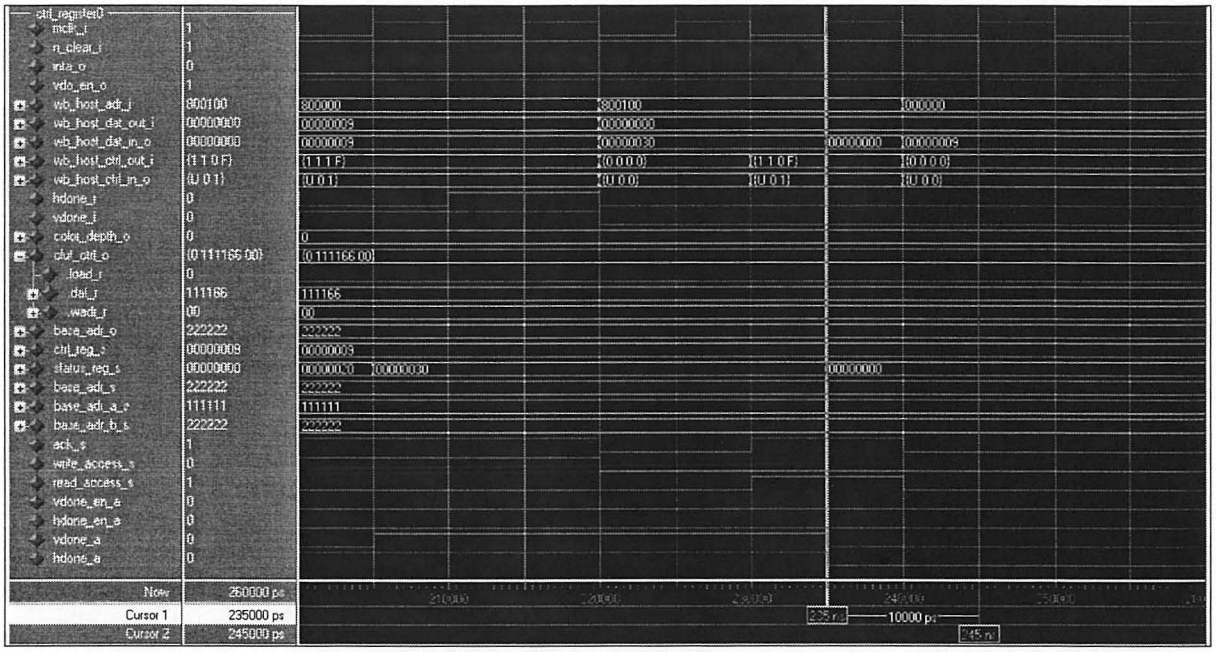

Figure 6.89 Simulation de la lecture du registre status_reg dans le module ctrl_register.

Pour accéder à ce registre, on doit utiliser l'adresse 0x800100 et activer le bus wishbone host en lecture. Il est impossible d'écrire dans le registre status_reg.

6.11.4 Description du module sync_gen (sync_gen.vhd)

Voici le symbole du module sync_gen :

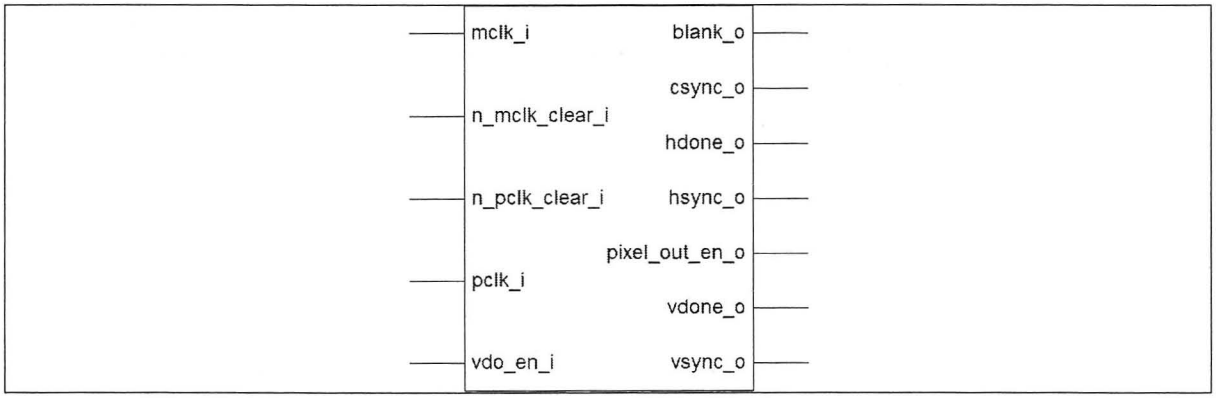

Figure 6.90 Symbole du module sync_gen.

Avant de décrire le fonctionnement du module sync_gen, voici un tableau de ses entrées et de ses sorties :

Tableau 6.24

Description des entrées et des sorties du module sync_gen

Nom	Entrée/ Sortie	Description
mclk	E	Signal d'horloge maîtresse, voir les sections 6.7 et 6.7.4 pour plus de détails.
pclk	E	Signal d'horloge des pixels. Il est généré par le module pclk_gen (section 6.7.3).
vdo_en	E	Signal général qui indique si le contrôleur vidéo est activé ou non, ce signal provient du module ctrl_register et correspond au bit ctrl_reg(0).
pixel_out_en	S	Signal qui indique au contrôleur vidéo de sortir un nouveau pixel. Ce signal contrôle donc la lecture des pixels dans le module dclk_fifo.
hdone	S	Signal qui indique la fin d'une ligne d'affichage. Ce signal est généré par le module sync_gen.
vdone	S	Signal qui indique la fin de l'affichage d'une image. Ce signal est généré par le module sync_gen.
hsync	S	Signal de synchronisation vidéo horizontal vers le convertisseur vidéo.
vsync	S	Signal de synchronisation vidéo vertical vers le convertisseur vidéo.
csync	S	Signal de synchronisation vidéo composite vers le convertisseur vidéo.
blank	S	Signal de synchronisation vidéo qui identifie les sections visibles de l'image. Ce signal se dirige vers le convertisseur vidéo.

6.11.4.1 Description du fonctionnement du module sync_gen

Le module sync_gen est celui de la génération des signaux de synchronisme de l'affichage. Il sert en effet à activer les signaux qui permettent d'identifier avec exactitude le moment pour afficher les pixels. Il sert à marquer le début et la fin de chaque ligne (horizontale) ainsi que le début et la fin de chaque image (verticale). Le module sync_gen sert aussi à faire le passage d'un domaine d'horloge à un autre. Tous les compteurs internes sont synchrones sur le signal d'horloge pclk, alors que les signaux hdone et vdone qui identifient la fin de l'affichage d'un axe (horizontal ou vertical) se doivent de se synchroniser sur l'horloge maîtresse afin d'être propagés dans le reste du système VGA qui est synchrone à l'horloge maîtresse. Pour ce faire, il est nécessaire de passer ces signaux dans un double registre (double buffered) afin de réduire les métastabilités. Voici un schéma bloc de l'élément sync_gen avec ses sous-modules :

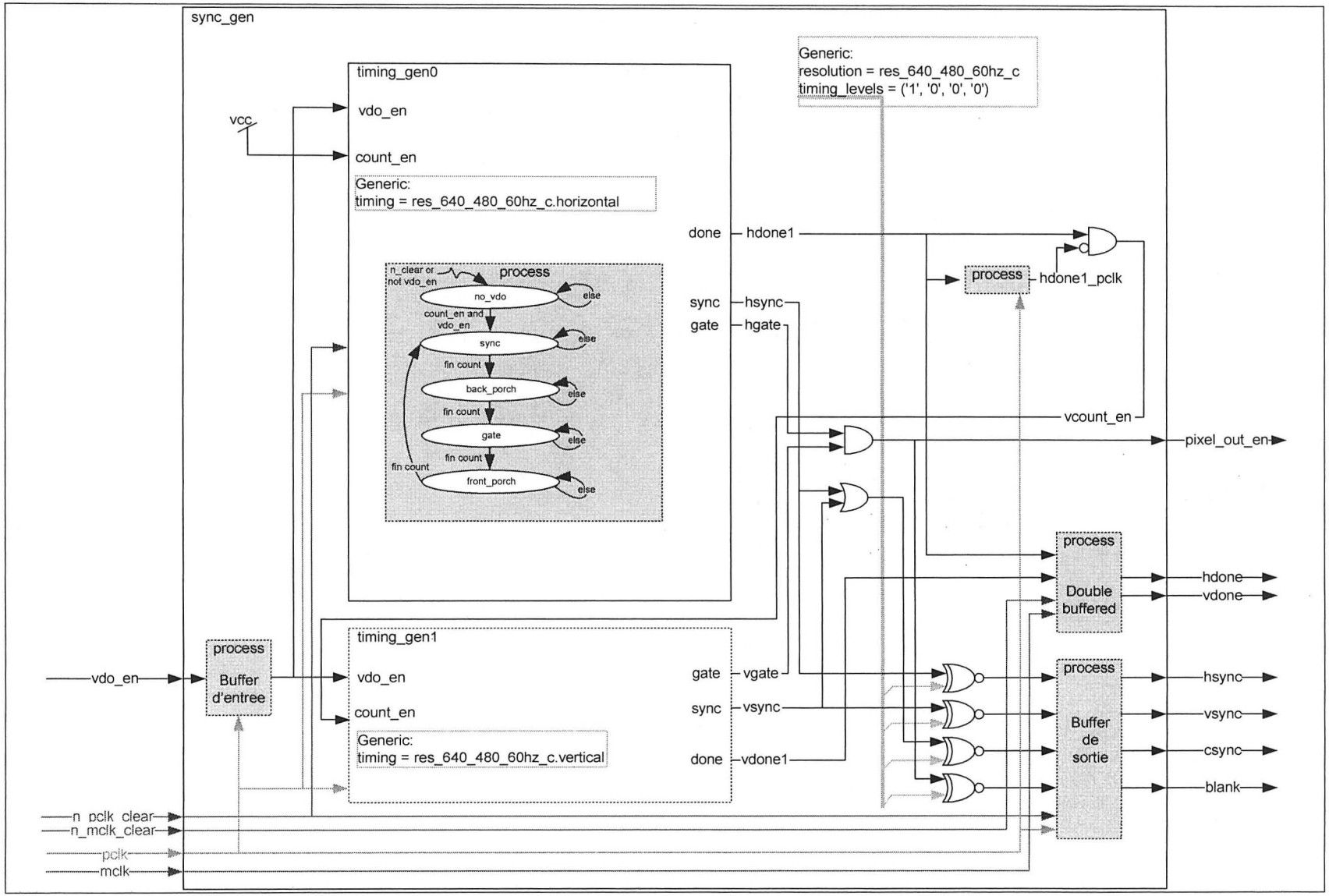

Figure 6.91 Schéma du module sync_gen.

En voyant cette figure on peut comprendre que le module sync_gen permet de faire les décomptes des délais de chaque signaux de synchronisation et que ce même module est constitué de deux autres sous unités qui englobent chaque composant pour les deux axes vertical et horizontal. Le module timing_gen0 génère les signaux de l'axe horizontal et le module timing_gen1 génère les signaux de l'axe vertical.

Le module sync_gen permet aussi de modifier la polarité des signaux VGA. La variable générique timing_levels par défaut configure les signaux VGA comme actif bas, sauf pour le signal blank qui est actif haut. C'est pour cette raison que la variable générique est assignée à la valeur ('1','0','0','0').

6.11.4.2 Résultats des simulations du module sync_gen

Voici la simulation fonctionnelle du module sync_gen :

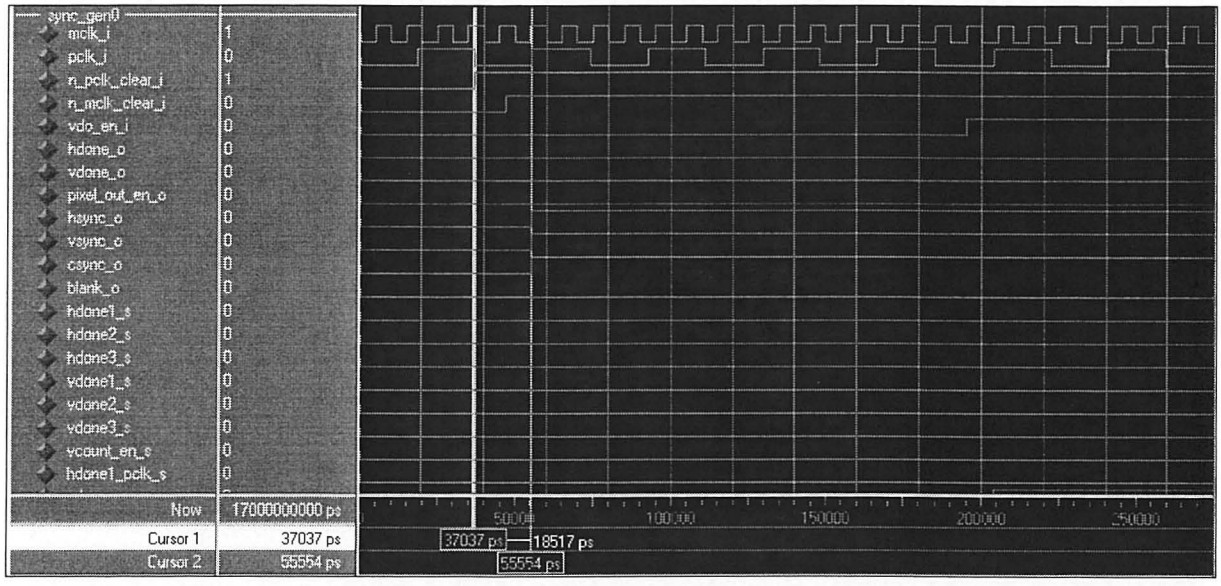

Figure 6.92 Simulation de l'initialisation du module sync_gen.

Le module possède deux signaux de remise à zéro puisqu'il utilise deux domaines d'horloge. Pour le moment, uniquement le signal n_mclk_clear est utilisé car il est possible d'initialiser le système dans un état connu sans avoir recours à un autre signal de remise à zéro. Par contre pour rendre la remise à zéro plus complète et générale lors de la complexification du

système, il deviendra primordial de générer un signal de reinitialisation concordant aux deux domaines d'horloge à la fois.

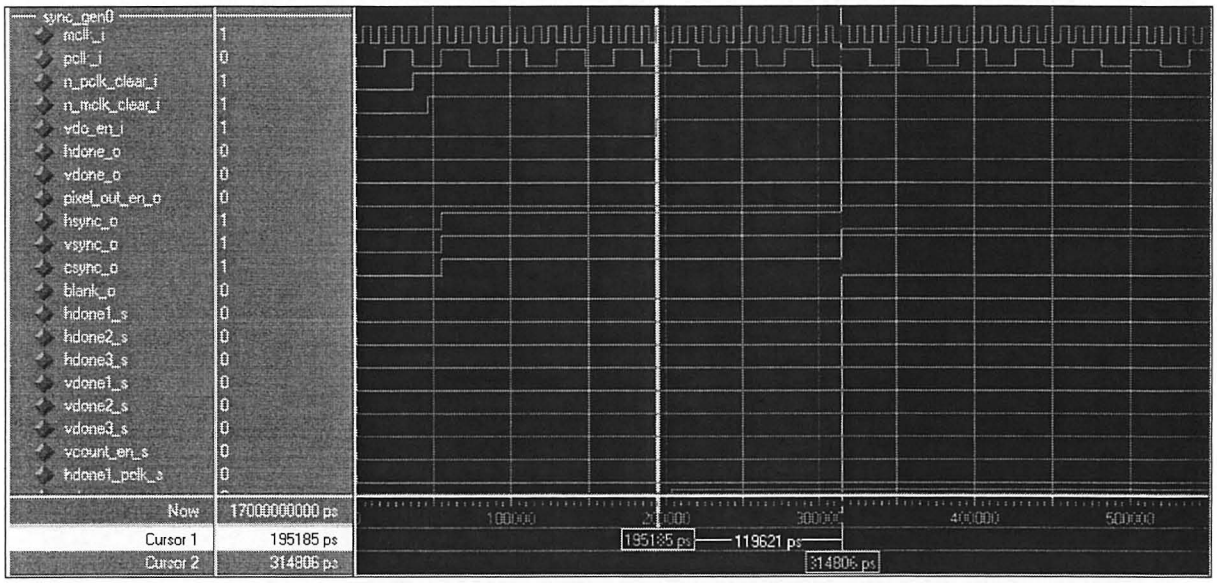

Figure 6.93 Simulation de l'effet de l'activation du système vidéo sur le module sync_gen.

En activant le système vidéo, les compteurs et la machine à états finis des deux modules timings_gen se mettent en branle. C'est pour cette raison que sur la Figure 6.93, à quatre cycles de pclk plus loin du curseur numéro un, on a le signal hsync que s'active. Ceci est du au changement de domaines d'horloges.

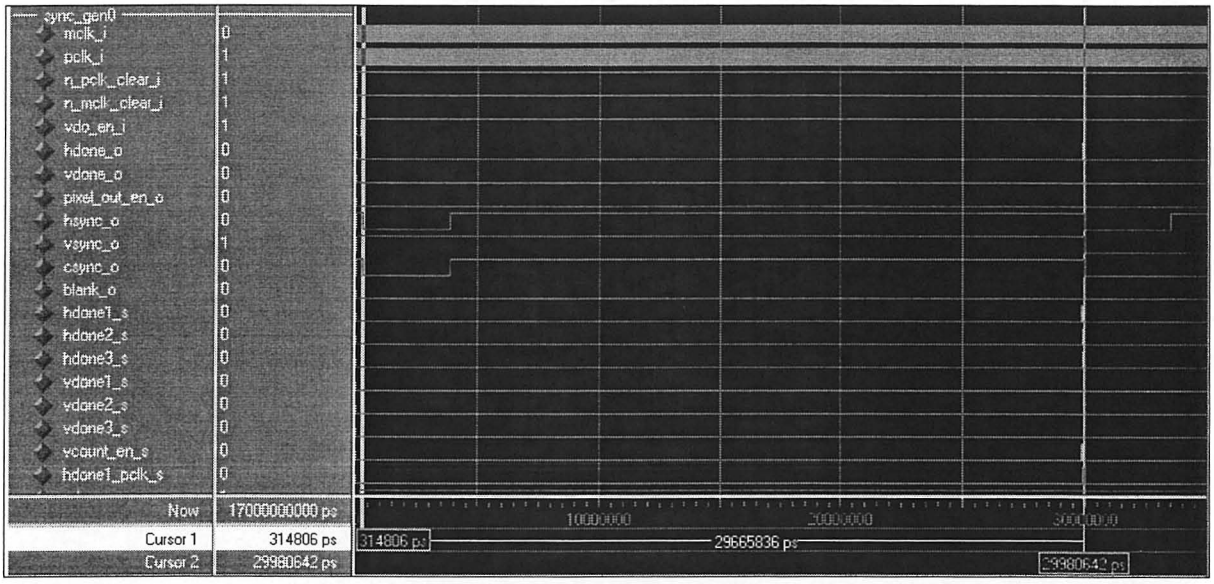

Figure 6.94 Simulation d'une durée complète de l'affichage d'une ligne d'image par le module sync_gen.

La figure précédente montre la génération du signal hsync et le délai entre deux signaux hsync consécutifs. On mesure un délai de 29,665 836 µs, la période de pclk étant de 37,037 ns, on obtient 801 pulsations. Selon les configurations de la simulation qui est à la résolution 640x480 à 60Hz, le nombre de pulsations par ligne doit être la somme hsync + front_porch + hgate + back_porch = 96+48+640+16 = 800. Les valeurs des compteurs pour les différentes résolutions sont données au

Tableau 5.1.

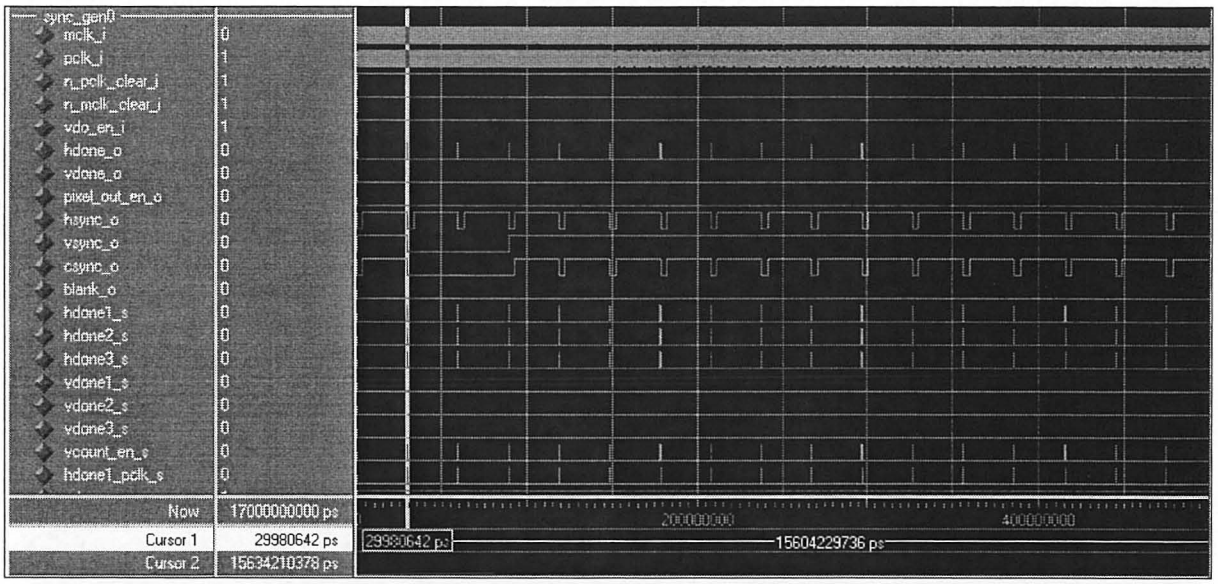

Figure 6.95 Simulation du début de l'affichage d'une image par le module sync_gen.

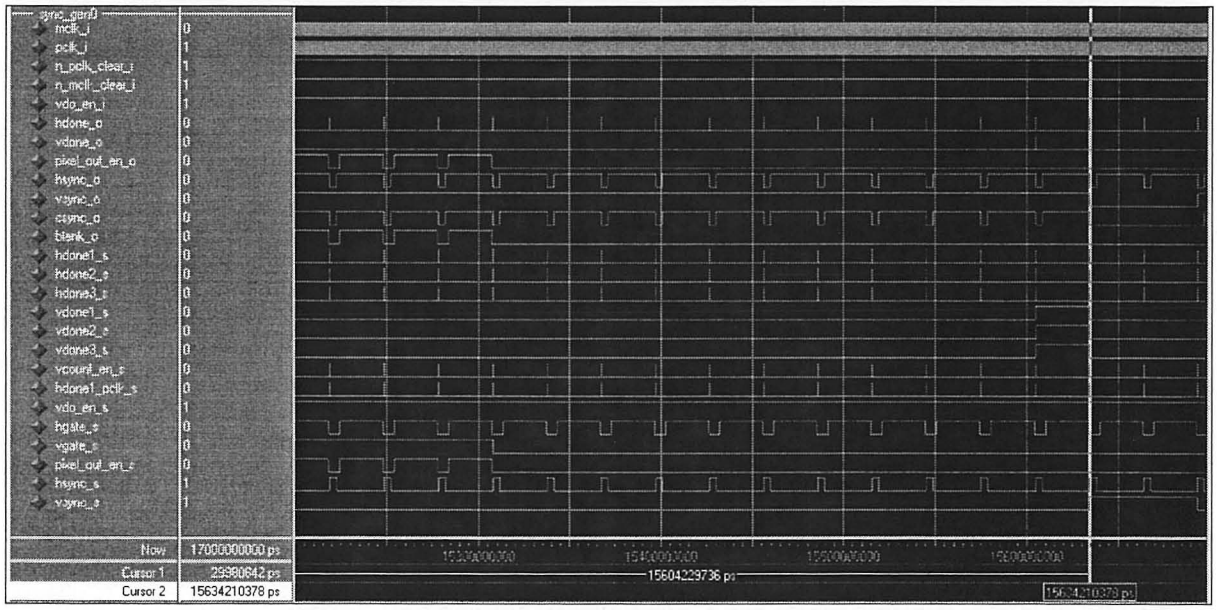

Figure 6.96 Simulation de la fin de l'affichage d'une image par le module sync_gen.

La Figure 6.95 et la Figure 6.96 montrent le début et la fin de l'affichage d'une image. On voit le curseur numéro un sur la première figure et le curseur numéro deux sur la deuxième. Le délai qui est mesuré ici correspond au délai que de l'affichage d'une image. Ce délai est donc de 15,604 ms, ce qui correspond effectivement au taux de rafraichissement de 60 Hz car

1/ 60 = 16,666 ms. Il y a une légère différence qui est due à la génération imprécise de la fréquence d'horloge pixel (voir la section 6.7.3), mais aussi parce que la simulation ne permet pas des pas de calculs plus petits que 1 ps (faible influence sur l'erreur tout de même).

6.11.4.3 Description du module timing_gen (timing_gen.vhd)

Voici le symbole du module timing_gen :

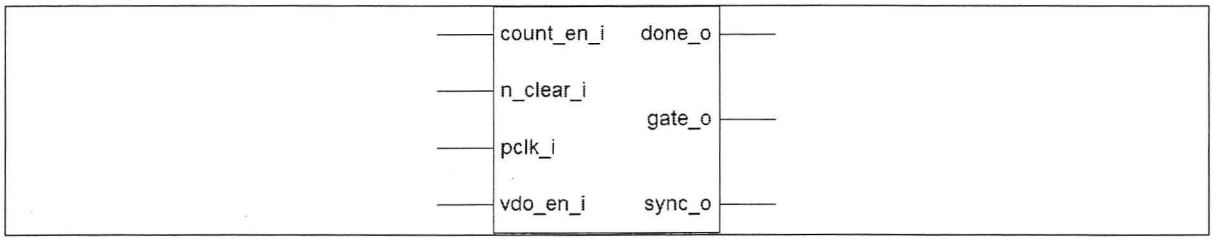

Figure 6.97 Symbole du module timing_gen.

Avant de décrire le fonctionnement du module timing_gen, voici un tableau de ses entrées et de ses sorties :

Tableau 6.25

Description des entrées et des sorties du module timing_gen

Nom	Entrée/ Sortie	Description
pclk	E	Signal d'horloge des pixels. Il est généré par le module pclk_gen (section 6.7.3).
count_en	E	Signal qui permet d'activer les compteurs. Dans le cas des compteurs pour les signaux de synchronisation horizontaux, ce signal est toujours actif. Par contre dans le cas des compteurs pour les signaux de synchronisation verticaux, ce signal s'active sur le signal hdone qui provient de la sortie done du compteur horizontal (deux modules timing_gen en cascade)
vdo_en	E	Signal général qui indique si le contrôleur vidéo est activé ou non, ce signal provient du module ctrl_register et correspond au bit ctrl_reg(0).
sync	S	Signal qui marque le début d'une ligne de pixel ou le début d'une image (horizontal ou vertical respectivement)
gate	S	Signal qui identifie la portion des pixels visibles à l'écran.
done	S	Signal qui indique la fin d'une ligne ou d'une image (horizontal ou vertical respectivement)

6.11.4.3.1 Description du fonctionnement du module timing_gen

Le module timing_gen est une machine à états finis (voir Figure 6.91 pour le diagramme d'état de la machine) qui permet de décompter les délais pour l'activation des différents signaux d'un axe d'affichage (horizontal ou vertical). En tout, quatre décomptes sont nécessaires pour générer les signaux de synchronisme d'un axe d'affichage. Ces signaux sont les suivants : sync qui est le signal qui identifie le début de l'axe, gate est le signal qui identifie la plage visible de l'affichage sur cet axe et done est le signal qui identifie la fin de l'affichage de l'axe.

La machine à états finis du module timings_gen permet de générer consécutivement les quatre décomptes nécessaires à l'aide d'un seul compteur. Ceci permet de sauver énormément d'espace en termes de bascules puisque le compteur est réutilisé plutôt que d'utiliser quatre compteurs différents en cascade. Le premier décompte est celui qui calcule le temps d'activation du signal sync. En d'autres mots, le signal sync doit rester actif toute la durée de ce premier décompte. Le deuxième décompte permet de calculer le délai entre le signal sync et le signal gate (back_porch). Le troisième décompte calcule le temps d'activation du signal gate. Le dernier décompte permet de calculer le délai entre le signal gate et le prochain signal sync (front_porch).

Pour démarrer les compteurs, le signal vdo_en doit être activé ainsi que le signal count_en. Ensuite, la machine change d'état à la fin de chacun des compteurs lorsque le signal count_en est aussi actif. Le signal done est généré au dernier compte du dernier état (front_porch).

6.11.4.3.2 Résultats des simulations du module timing_gen

Voici la simulation fonctionnelle du module timings_gen :

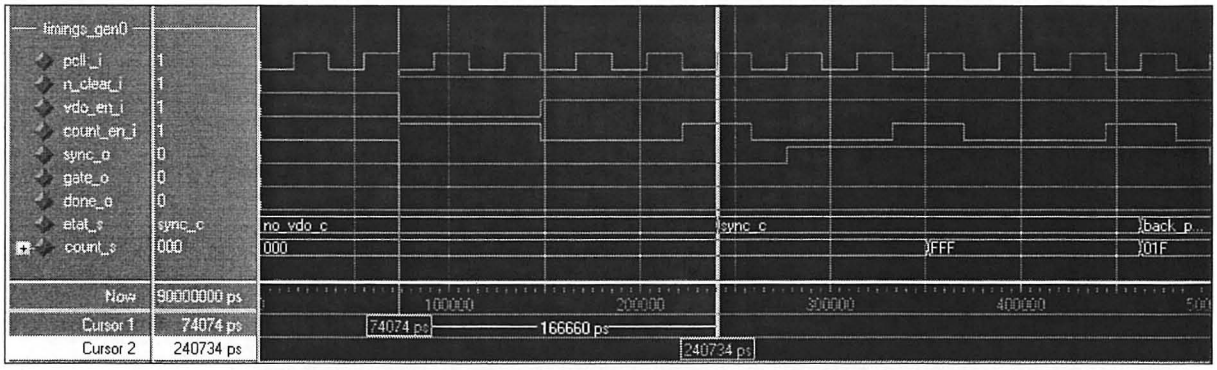

Figure 6.98 Simulation de l'initialisation du module timings_gen.

Lors de l'initialisation du module timings_gen, la machine à états finis est à l'état no_vdo. Le compteur s'active uniquement lorsque le signal vdo_en et le signal count_en s'activent simultanément. La résolution de l'image simulée ici est de 640x480 à 60 Hz et nous ne simulons que la partie verticale, donc la valeur du compteur sync est « 2 ». Nous utilisons un décompteur qui permet de minimiser au maximum le nombre de portes logiques internes puisque le signal de fin de course du compteur est en fait le bit le plus significatif du signal count. Lorsque le signal arrive à la valeur 0 moins 1, c'est-à-dire 0xFFF en complément vrai, c'est le seul temps ou le bit le plus significatif est actif et c'est aussi ce bit qui détermine la fin du compte. Pour cette raison, on initialise le compteur toujours à la valeur de décompte désiré moins 2. Pourquoi « - 2 », c'est parce que le compteur passe par deux valeurs « de trop », soit le 0x000 et 0xFFF, ce qui fait qu'on doit le partir en « avance » pour s'assurer que le nombre de compte soit exacte. Pour le compteur vertical sync, par exemple, la valeur du compte doit être de 2, alors selon l'équation, on démarre le décompte à la valeur 2-2 = 0. C'est pour cette raison que le décompte de la Figure 6.98 au curseur numéro deux démarre à la valeur 0x000.

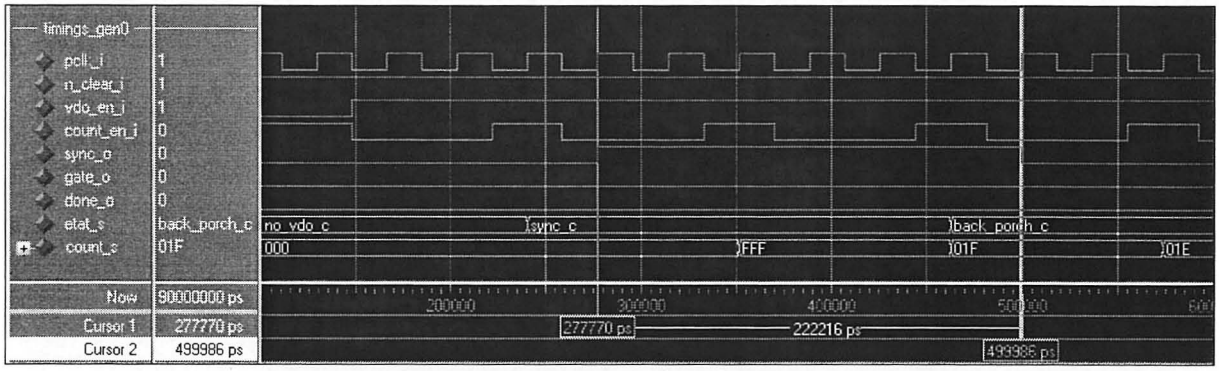

Figure 6.99 Simulation du décompte pour la génération du signal sync par le module timings_gen.

La figure précédente montre que le signal sync est actif pour les deux pulsations de compteur généré durant l'état sync. La sortie sync est synchrone avec l'horloge pixel.

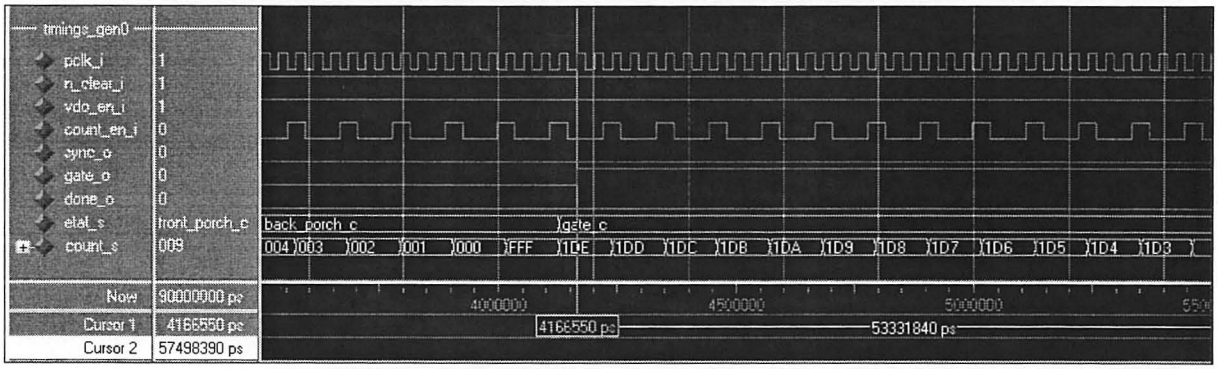

Figure 6.100 Simulation du début de la génération du signal gate par le module timings_gen.

À la figure précédente, on voit très bien que l'état back_porch se termine avec le compte 0xFFF et l'état gate débute avec la valeur 480 moins 2 en hexadécimal, ce qui donne bien 0x1DE. Le signal gate s'active un cycle d'horloge plus loin que l'état puisqu'il est généré une fois que l'état a été activé (voir le code VHDL de la machine pour plus de compréhension).

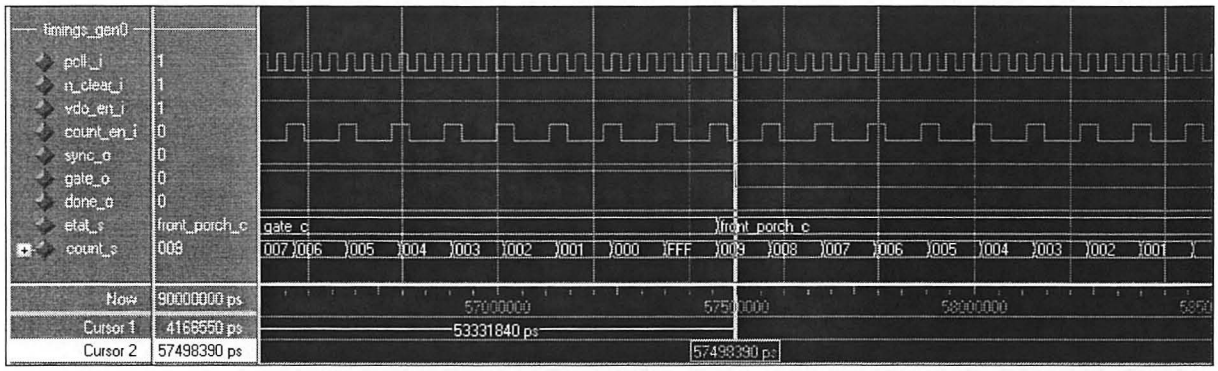

Figure 6.101 Simulation de la fin de la génération du signal gate par le module timings_gen.

La figure précédente montre la fin du signal gate qui correspond bien à la fin du décompte avec la valeur 0xFFF. Le signal gate est la fenêtre temporelle de visibilité des pixels sur l'image.

La Figure 6.102 montre le signal done qui est généré au dernier compte du dernier état. La durée du signal done est déterminée par l'intervalle entre les deux dernières activations du signal count_en, ce qui permet d'identifier adéquatement la dernière valeur du décompte. On remarque aussi sur cette prochaine figure, que le décompte se réinitialise pour générer un autre signal sync. C'est un autre affichage d'image qui débute.

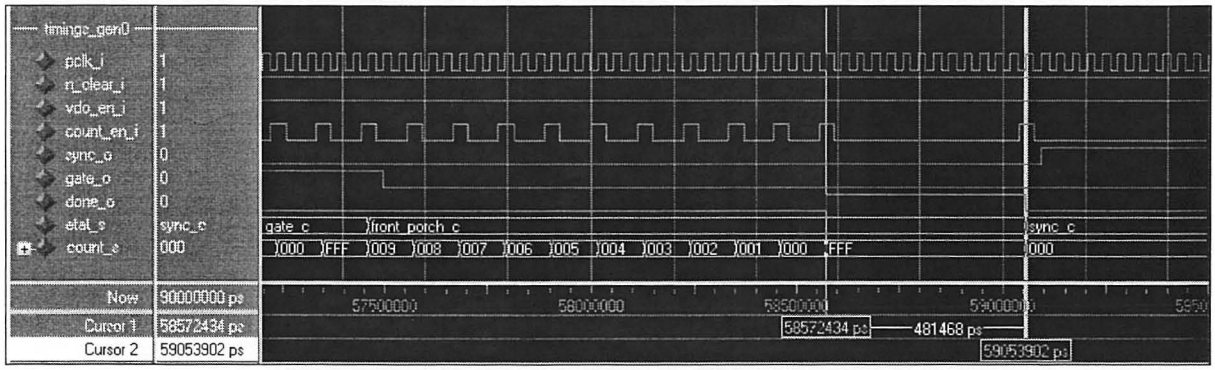

Figure 6.102 Génération du signal done à la fin du dernier décompte.

Afin de s'assurer d'une simulation plus proche de la réalité, le signal d'entrée count_en est pulsé de manière à mettre le module timing_gen en attente à chaque compte. La fréquence

d'horloge choisie est 27,175 MHz afin de reproduire la fréquence d'affichage VGA pour une résolution de 640x480 pixels avec un taux de rafraichissement de 60 images par seconde.

6.11.5 Description du module pixel_gen (pixel_gen.vhd)

Voici le symbole du module pixel_gen :

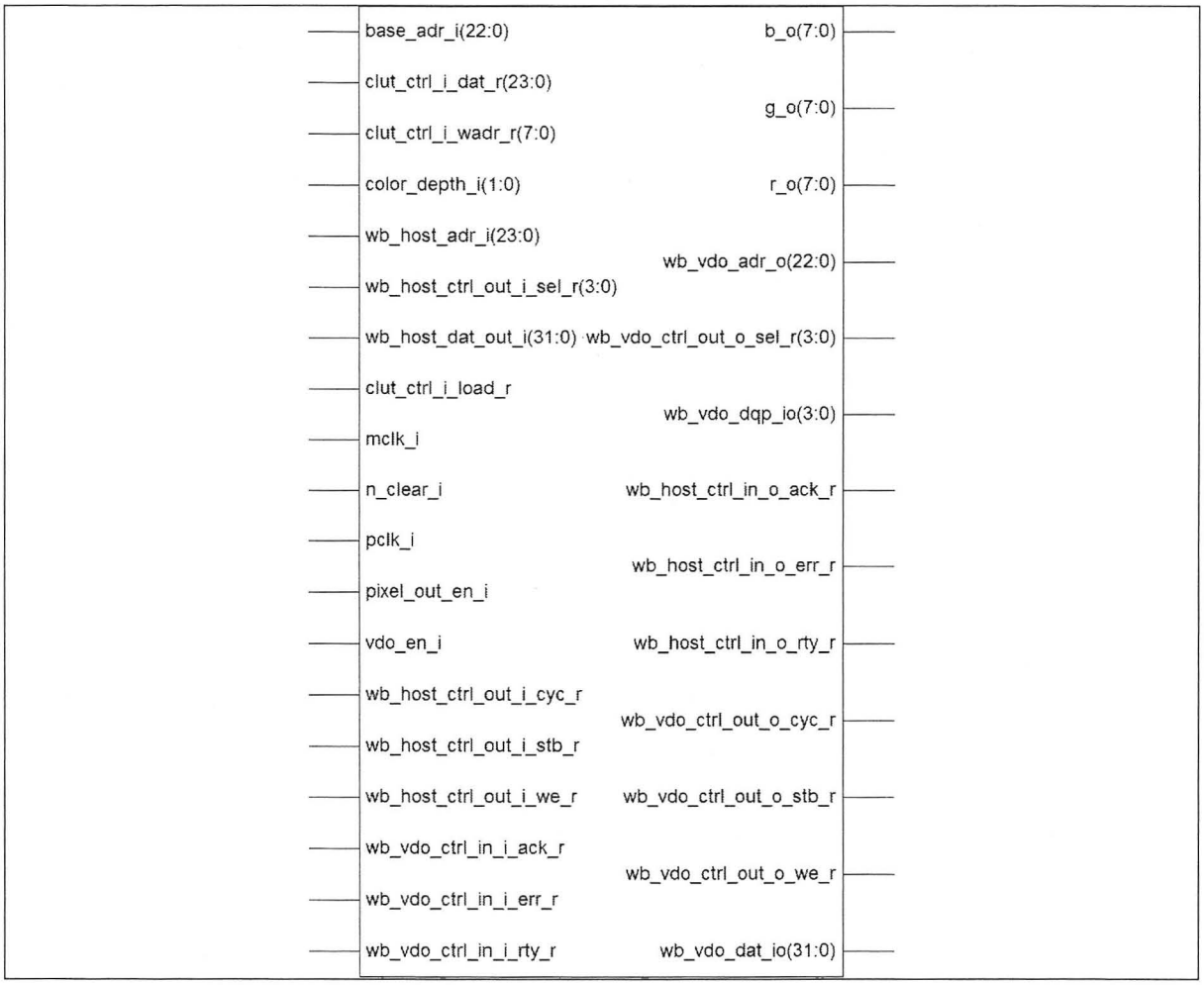

Figure 6.103 Symbole du module pixel_gen.

Avant de décrire le fonctionnement du module pixel_gen, voici un tableau de ses entrées et de ses sorties :

Tableau 6.26

Description des entrées et des sorties du module pixel_gen

Nom	Entrée/ Sortie	Description
mclk	E	Signal d'horloge maîtresse, voir les sections 6.7 et 6.7.4 pour plus de détails.
pclk	E	Signal d'horloge des pixels. Il est généré par le module pclk_gen (section 6.7.3).
n_clear	E	Signal de réinitialisation générale du système. Voir la section 6.9 pour plus de détails.
vdo_en	E	Signal général qui indique si le contrôleur vidéo est activé ou non. Il provient du module ctrl_register et correspond au bit ctrl_reg(0).
wb_host_dat_out[32]	E	Signal de donnée du bus Wishbone du Host. Le maître est le module bootloader et l'esclave est le module VGA (et ses sous-modules, dont le module pixel_gen fait partie).
wb_host_adr[24]	E	Signal d'adresse du bus Wishbone du Host. Le maître est le module bootloader et l'esclave est le module VGA (et ses sous-modules, dont le module pixel_gen fait partie)
wb_host_ctrl_out[7]	E	Groupe de signaux de contrôle du bus wishbone host. Ce groupe de signaux inclut entre autres les signaux cyc et stb qui permettent d'entamer une transaction sur le bus. Le signal sel permet de savoir la taille de la transaction. Pour plus de détails sur le fonctionnement du bus Wishbone, référez-vous au document [23] Herveille, 2002.
wb_host_ctrl_in[3]	S	Entrée de contrôle du bus Wishbone pour le module host (bootloader). C'est un groupe de signaux qui permet au module esclave de répondre au maître. Ce groupe de signaux comprend les signaux ack, err et rty. Le signal ack est le seul utilisé présentement et permet de dire au maître que la transaction est acceptée.
base_adr[23]	E	Adresse de départ de l'image sélectionnée. Deux banques d'images sont disponibles en mémoire vidéo. Pour modifier l'image affichée, il suffit d'inscrire son index dans le registre approprié (module ctrl_register), soit ctrl_reg(3). On accède à ce registre à l'adresse 0x800000.
color_depth[2]	E	Signal qui spécifie le nombre de bits d'encodage de l'image : • 00 pour un encodage 8 bits; • 01 pour un encodage 16 bits (pas encore implémenté); • 1x pour un encodage 24 bits.
clut_ctrl[33]	E	Groupe de signaux qui permettent de contrôler le chargement de la table des couleurs (module clut) par le module ctrl_register. Ce groupe contient les signaux : • load : signal d'activation de chargement d'une donnée; • adr[8] : emplacement de la donnée à charger dans la table; • dat[24] : données de 24 bits à charger.
pixel_out_en	E	Signal qui indique au contrôleur vidéo de sortir un nouveau pixel. Ce signal contrôle donc la lecture des pixels dans le module dclk_fifo.
wb_vdo_adr[23]	S	Signal d'adresse Wishbone entre le contrôleur vidéo (module VGA) et la mémoire vidéo (mémoire ZBT).
wb_vdo_dat[32]	E/S	Signal de données bidirectionnel Wishbone entre le contrôleur vidéo (module VGA) et la mémoire vidéo (mémoire ZBT).
wb_vdo_dqp[4]	E/S	Signal des bits de parité bidirectionnel Wishbone entre le contrôleur vidéo (module VGA) et la mémoire vidéo (mémoire ZBT).
wb_vdo_ctrl_out[7]	S	Groupe de signaux de contrôle du bus wishbone vidéo (entre le contrôleur vidéo et la mémoire ZBT). Ce groupe de signaux inclut entre autres les

		signaux cyc et stb qui permettent d'entamer une transaction sur le bus. Le signal sel permet de savoir la taille de la transaction. Pour plus de détails sur le fonctionnement du bus Wishbone, référez-vous au document [23] Herveille, 2002.
wb_vdo_ctrl_in[3]	E	Entrée de contrôle du bus Wishbone pour le contrôleur vidéo (maître). C'est un groupe de signaux qui permet au module esclave (module wb2zbt_wraper) de répondre au maître. Ce groupe de signaux comprend les signaux ack, err et rty. Le signal ack est le seul utilisé présentement et permet de dire au maître que la transaction est acceptée.
r[8]	S	Signal de donnée de la couleur rouge vers le convertisseur vidéo
g[8]	S	Signal de donnée de la couleur vert vers le convertisseur vidéo
b[8]	S	Signal de donnée de la couleur bleu vers le convertisseur vidéo

6.11.5.1 Description du fonctionnement du module pixel_gen

Cette unité est le cœur du traitement des pixels. Plusieurs modules y sont placés en série afin d'assurer la continuité du flot de données. Voici un schéma bloc du module pixel_gen avec ses sous-modules :

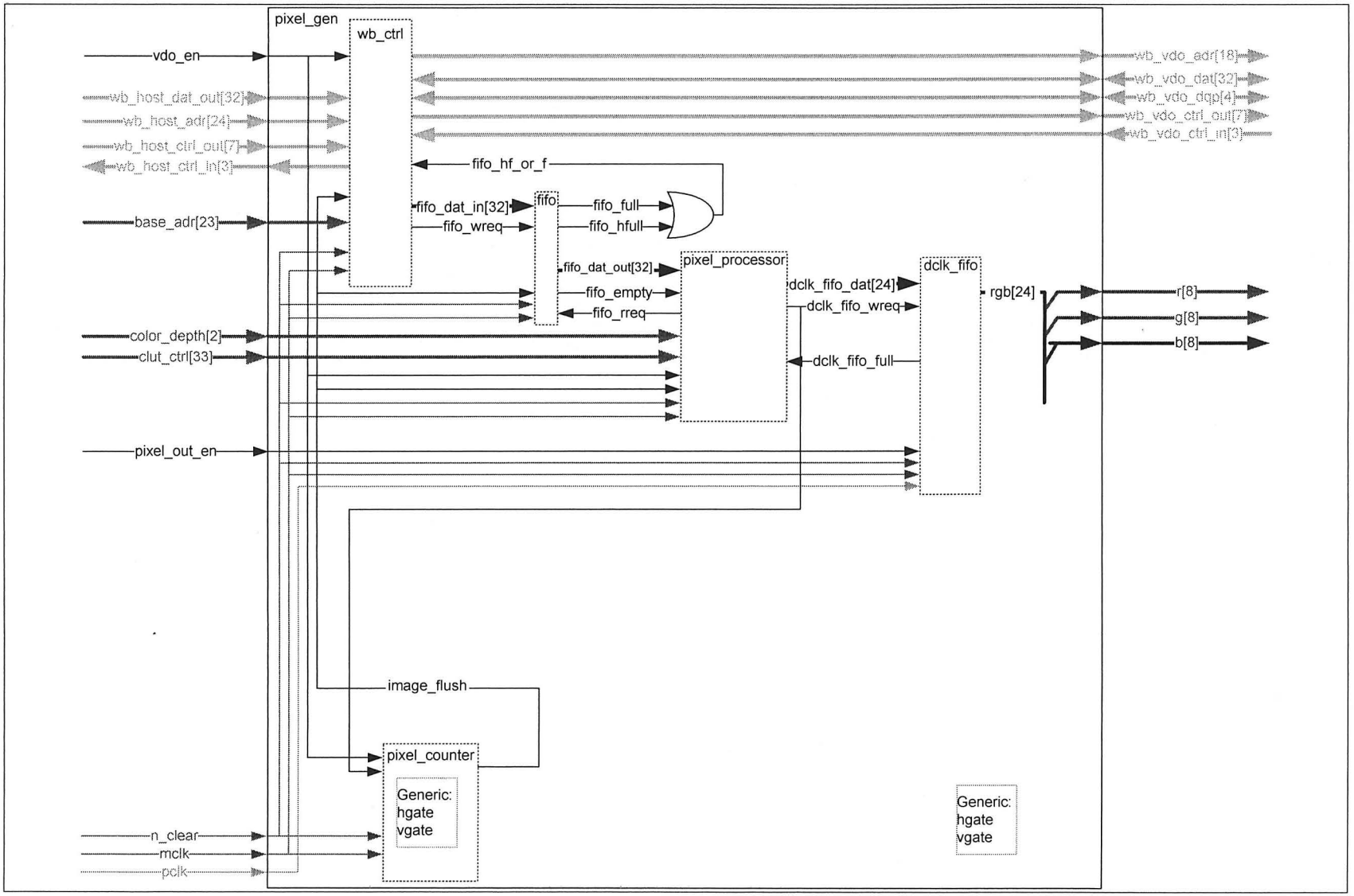

Figure 6.104 Schéma du module pixel_gen.

Le module wb_ctrl permet de faire la gestion des deux bus Wishbone. Il permet de récupérer les données provenant du module host (bootloader) et de les transférer dans la mémoire vidéo. Il permet aussi de charger automatiquement les pixels un après l'autre lors de l'affichage d'une image. Ces pixels sont ensuite envoyés dans le module fifo (section 6.11.5.4) afin d'assurer un flot de données continue lors de traitements sur ces pixels. Le module fifo fournit ensuite les données de pixel sans formatage au module pixel_processor (section 6.11.5.5).

En effet, le module pixel_processor sert à générer des pixels de 24 bits en tout temps à sa sortie. Par contre son entrée (fifo_dat_out) de données pixel qui provient du fifo peut prendre plusieurs configurations même si la taille de cette entrée est fixe à 32 bits. Par exemple, en mode d'image 8 bits, le signal fifo_dat_out comprend 4 pixels de 8 bits pour un total de 32 bits. En mode d'image 24 bits, le signal fifo_dat_out comprend, par exemple, un pixel de 24 bits juxtaposé avec 8 bits qui proviennent d'un autre pixel voisin. De cela, il est nécessaire de manipuler les données de 32 bits reçus et de générer des pixels de taille fixe de 24 bits en sortie du module pixel_processor.

Les données de 24 bits ainsi formatées sont ensuite acheminées vers le module dclk_fifo (section 6.11.5.6) qui est le module de sortie ainsi que le fifo permettant de passer du domaine d'horloge rapide, soit mclk l'horloge maîtresse, au domaine d'horloge plus lent, soit pclk l'horloge de l'afficheur vidéo.

Le module pixel_counter (section 6.11.5.7) permet de savoir exactement quand l'image termine son cycle d'affichage afin de générer un signal image_flush, ce qui permet de recommencer tout le processus de chargement des pixels à partir de l'adresse du premier pixel.

6.11.5.2 Résultats des simulations du module pixel_gen

Voici la simulation fonctionnelle du module pixel_gen :

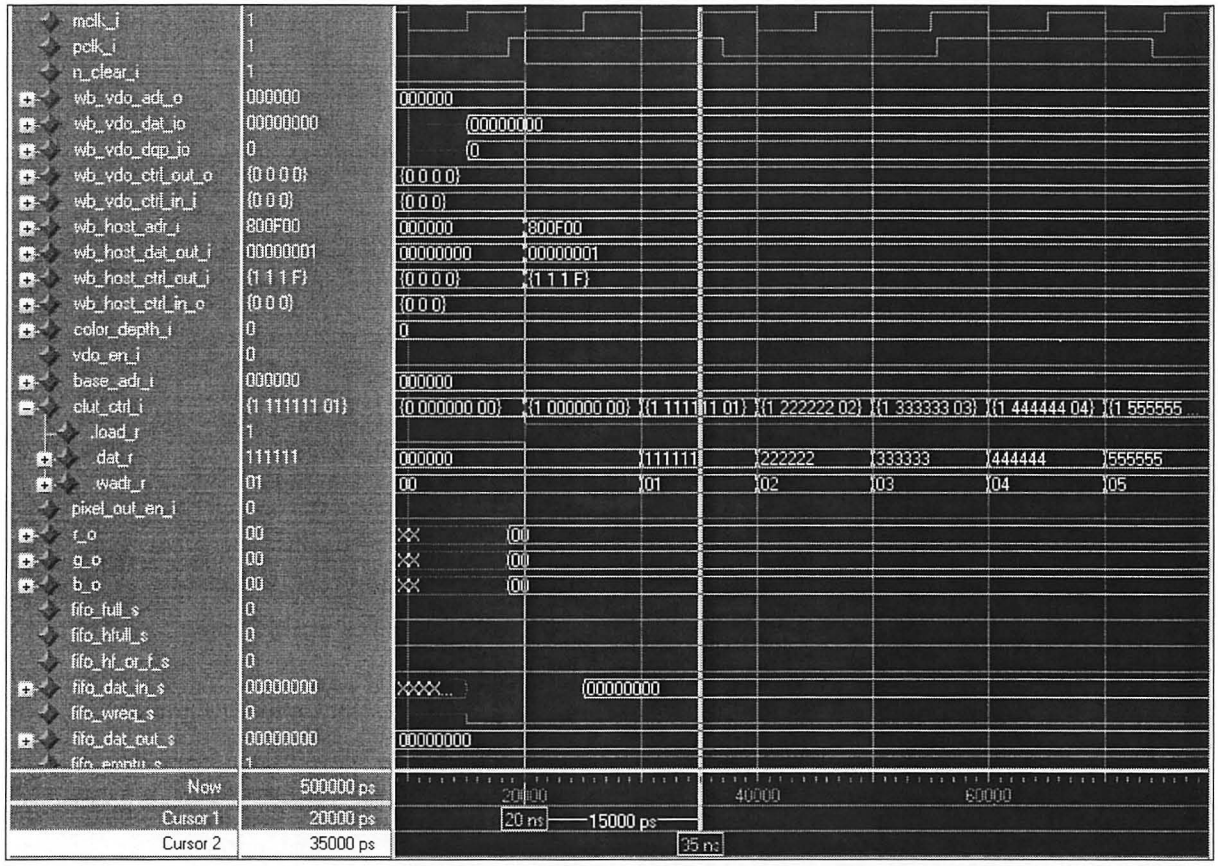

Figure 6.105 Simulation de l'initialisation du module pixel_gen.

La figure précédente permet de voir l'initialisation du module pixel_gen avant le curseur numéro un. À partir de ce curseur, le signal n_clear se désactive et le module pixel_gen se met à charger la table des couleurs puisque le signal clut_ctrl.load s'active et que le bus d'adresse Wishbone host pointe à une adresse supérieure à 0x800F00. Le chargement de la table des couleurs passe par le module ctrl_register qui est voisin au module pixel_gen. C'est ensuite ce dernier module qui active l'écriture de la table par l'intermédiaire du groupe de signaux clut_ctrl. Ce groupe de signaux est une entrée des modules imbriqués pixel_processor et clut. Le module clut est la table des couleurs proprement dite.

La figure suivante montre toute la simulation du chargement de la table des couleurs. On voit bien le signal clut_ctrl.load actif entre les deux curseurs de cette figure.

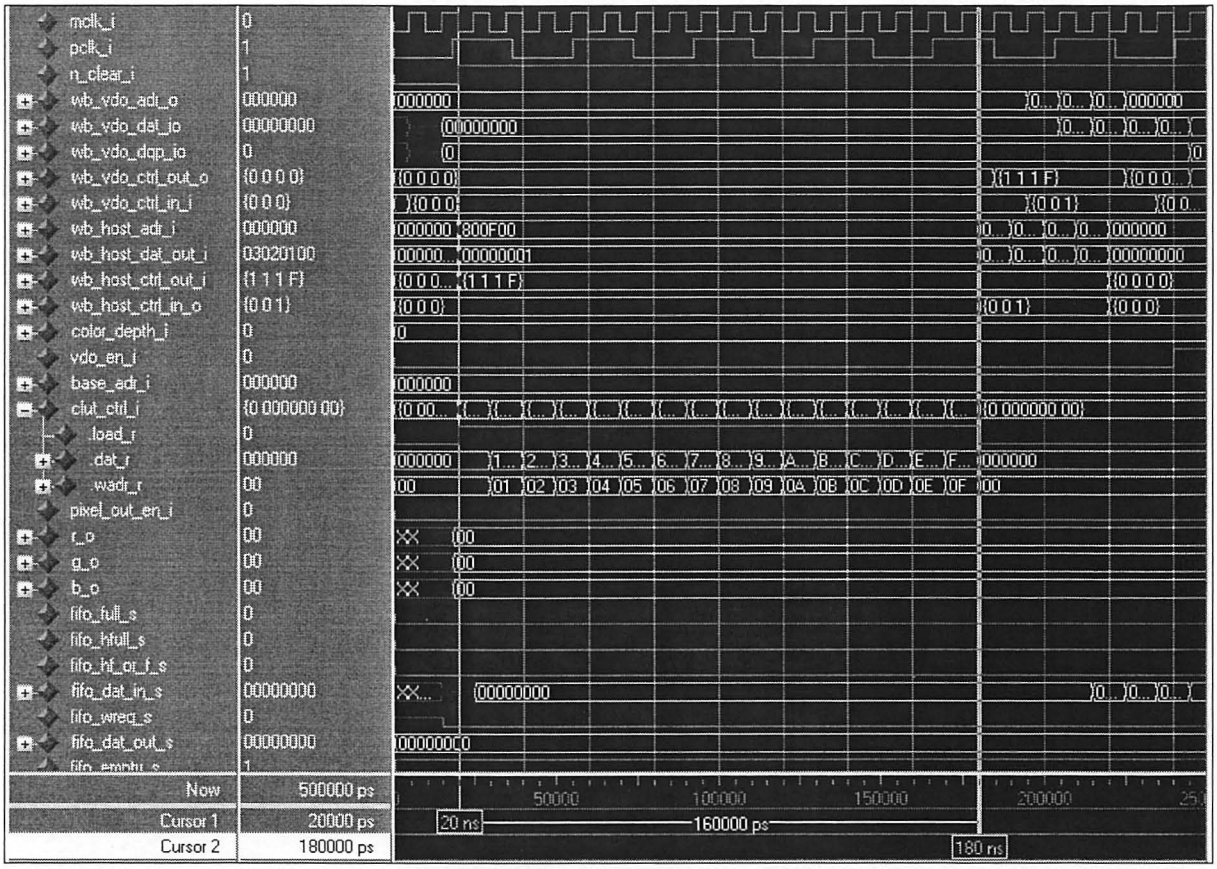

Figure 6.106 Simulation du chargement de la table des couleurs par le module pixel_gen.

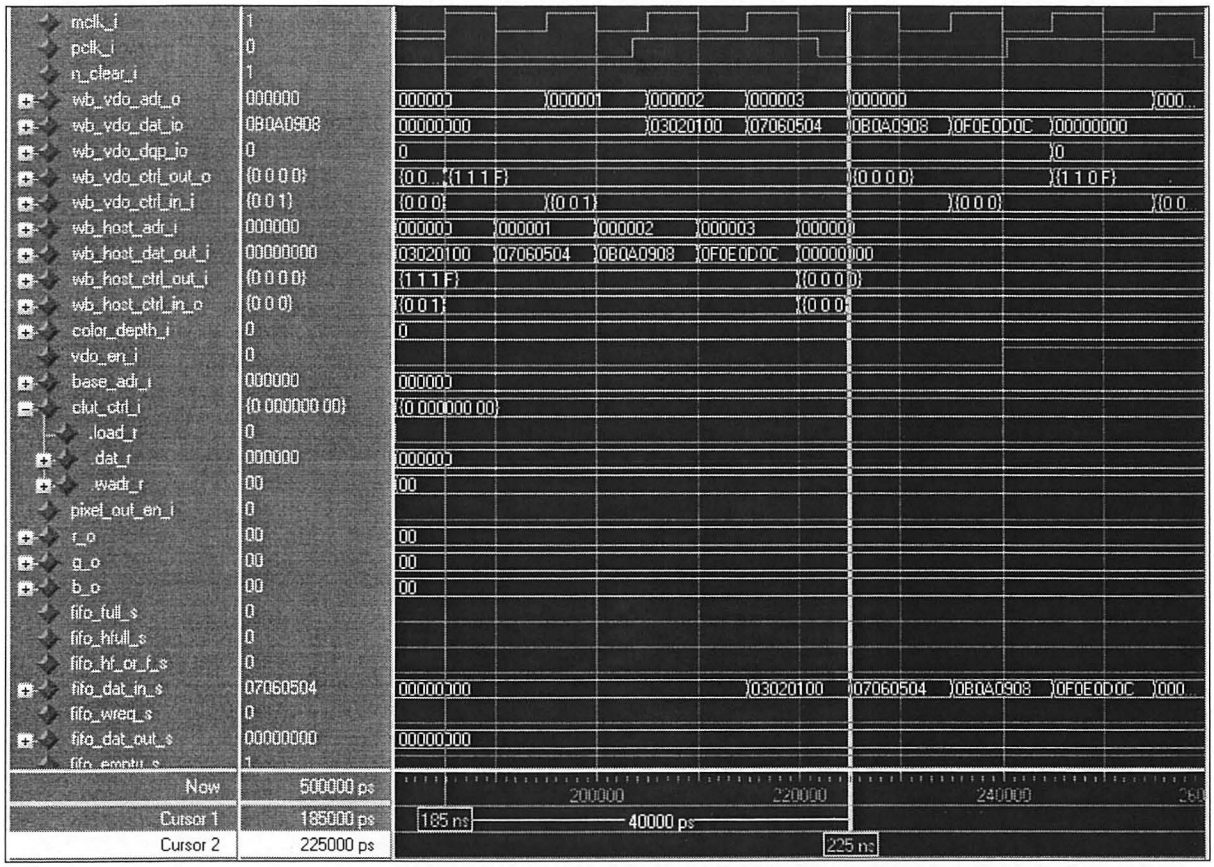

Figure 6.107 Simulation du chargement de la mémoire vidéo par le module pixel_gen.

La Figure 6.107 montre le chargement de la mémoire vidéo par le module pixel_gen. Le chargement est contrôlé par le module host du bus Wishbone host (bootloader). La plage d'adresse du chargement de la mémoire vidéo est de 0x000000 à 0x7FFFFF. Le bus wishbone vidéo contient en effet un bit d'adresse de moins que le bus Wishbone host puisque le bus Wishbone host permet aussi d'accéder au module ctrl_register à partir de l'adresse 0x800000.

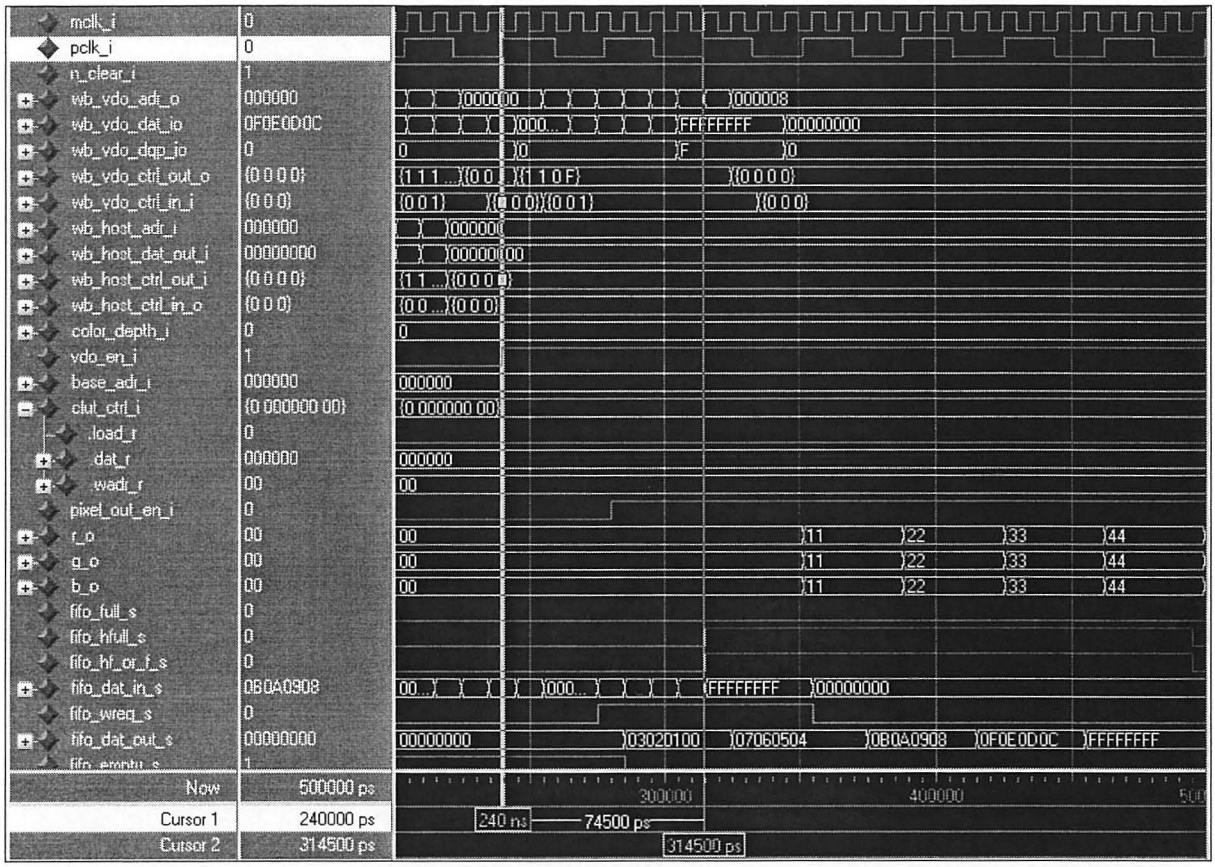

Figure 6.108 Simulation du module pixel_gen en mode affichage normal.

La Figure 6.108 montre le début de l'affichage d'une image pixel par pixel. On voit le remplissage graduel des modules fifo, pixel_precessor et dclk_fifo. Le curseur numéro un montre l'activation du signal vdo_en. Suite à cette activation, le module wb_ctrl commence à accéder à la mémoire vidéo en lecture pour lire chaque pixel de l'image active (à partir de l'adresse de base) de manière consécutive. Lorsqu'une donnée pixel provenant de la mémoire ZBT est prête (la donnée est en retard de deux cycles sur l'adresse et les signaux de contrôles), le signal fifo_wreq active l'écriture de cette donnée dans le module fifo. Un cycle plus tard, cette même donnée est accessible au module pixel_processor qui la traite et qui l'envoie à son tour au module dclk_fifo. Le curseur numéro deux de cette figure montre la sortie de la première donnée du module dclk_fifo après l'activation du signal vdo_en. On dénombre 8 cycles d'horloge maîtresse entre les deux curseurs.

6.11.5.3 Description du module wb_ctrl (wb_ctrl.vhd)

Voici le symbole du module wb_ctrl :

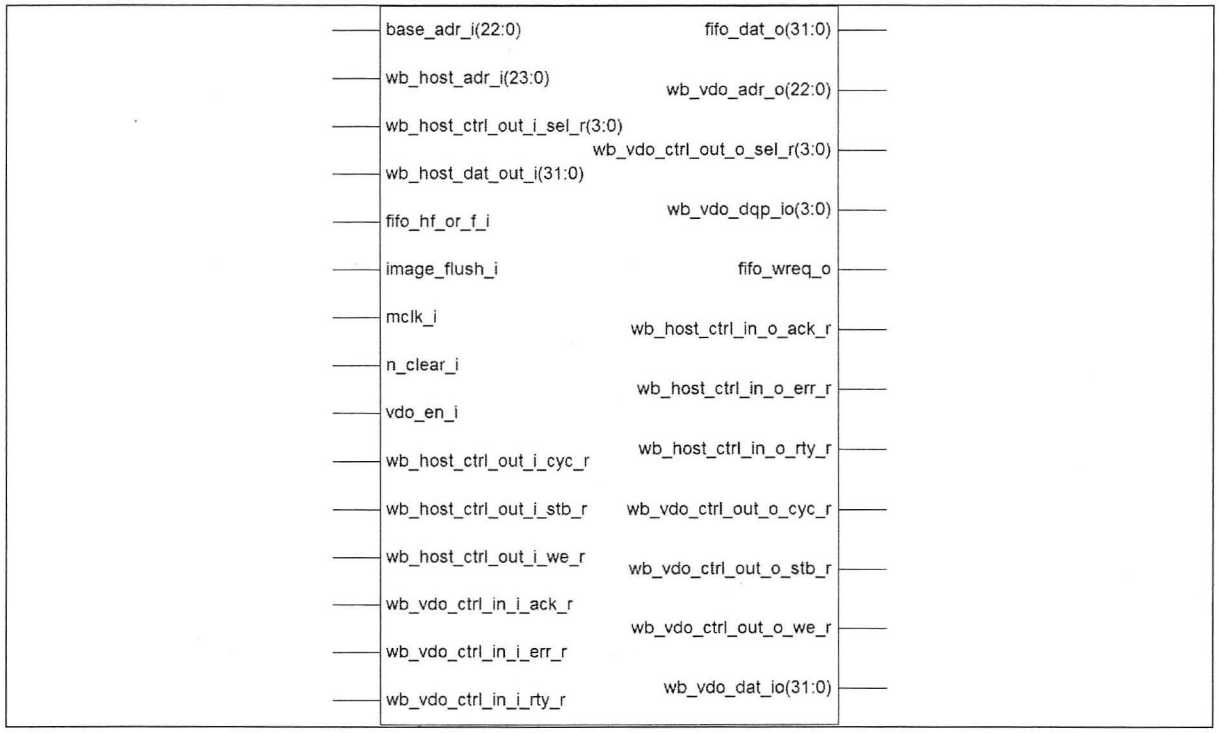

Figure 6.109 Symbole du module wb_ctrl.

Avant de décrire le fonctionnement du module wb_ctrl, voici un tableau de ses entrées et de ses sorties :

Tableau 6.27

Description des entrées et des sorties du module wb_ctrl

Nom	Entrée/ Sortie	Description
mclk	E	Signal d'horloge maîtresse, voir les sections 6.7 et 6.7.4 pour plus de détails.
n_clear	E	Signal de réinitialisation générale du système. Voir la section 6.9 pour plus de détails.
image_flush	E	Signal qui indique que tous les tampons doivent être vidés lorsque tous les pixels d'une image ont été transférés au module dclk_fifo. Ce signal est généré par le module pixel_counter

wb_host_ctrl_out[7]	E	Groupe de signaux de contrôle du bus wishbone host. Ce groupe de signaux inclut entre autres les signaux cyc et stb qui permettent d'entamer une transaction sur le bus. Le signal sel permet de savoir la taille de la transaction. Pour plus de détails sur le fonctionnement du bus Wishbone, référez-vous au document [23] Herveille, 2002.
wb_host_adr[24]	E	Signal d'adresse du bus Wishbone du Host. Le maître est le module bootloader et l'esclave est le module VGA (et ses sous-modules dont le module wb_ctrl fait partie).
wb_host_dat_out[32]	E	Signal de donnée du bus Wishbone du Host. Le maître est le module bootloader et l'esclave est le module VGA (et ses sous-modules, dont le module wb_ctrl fait partie).
wb_host_ctrl_in[3]	S	Entrée de contrôle du bus Wishbone pour le module host (bootloader). C'est un groupe de signaux qui permet au module esclave de répondre au maître. Ce groupe de signaux comprend les signaux ack, err et rty. Le signal ack est le seul utilisé présentement et permet de dire au maitre que la transaction est acceptée.
vdo_en	E	Signal général qui indique si le contrôleur vidéo est activé ou non, ce signal provient du module ctrl_register et correspond au bit ctrl_reg(0).
base_adr[23]	E	Signal d'adresse de départ de l'image sélectionnée. Deux banques d'images sont disponibles en mémoire vidéo. Pour modifier l'image affichée, il suffit d'inscrire son index dans le registre approprié (module ctrl_register), soit bit ctrl_reg(3). On accède à ce registre à l'adresse 0x800000 du bus Wishbone host.
fifo_hf_or_f	E	signal qui rassemble les signaux du module fifo : fifo_full et fifo_hfull. Il permet de mettre en arrêt la lecture des pixels en mémoire à partir du moment où le module fifo est à moitié plein.
wb_vdo_adr[23]	S	Signal d'adresse Wishbone entre le contrôleur vidéo (module VGA) et la mémoire vidéo (mémoire ZBT).
wb_vdo_dat[32]	E/S	Signal de données bidirectionnel Wishbone entre le contrôleur vidéo (module VGA) et la mémoire vidéo (mémoire ZBT).
wb_vdo_dqp[4]	E/S	Signal des bits de parité bidirectionnel Wishbone entre le contrôleur vidéo (module VGA) et la mémoire vidéo (mémoire ZBT).
wb_vdo_ctrl_out[7]	S	Groupe de signaux de contrôle du bus wishbone vidéo (entre le contrôleur vidéo et la mémoire ZBT). Ce groupe de signaux inclut entre autres les signaux cyc et stb qui permettent d'entamer une transaction sur le bus. Le signal sel permet de savoir la taille de la transaction. Pour plus de détails sur le fonctionnement du bus Wishbone, référez-vous au document [23] Herveille, 2002.
wb_vdo_ctrl_in[3]	S	Ce groupe de signaux contient tous les signaux de contrôle du bus wishbone reçus par le maître (qui est dans ce cas-ci le module VGA). On y trouve donc les signaux suivants : • ack : ce signal averti le maître que la requête a été prise en charge par l'esclave; • err : signal d'erreur de transmission sur le bus; • rty : signal qui averti le maître de réessayer sa requête plus tard (utilisé dans le cas où l'esclave serait incapable de répondre immédiatement.
fifo_wreq	S	Signal qui indique au module fifo de mémoriser une donnée. Ce signal provient du module wb_ctrl
fifo_dat_in[32]	S	Signal des données pixel qui proviennent de la mémoire vidéo et qui passent par le module wb_ctrl en direction du module fifo.

6.11.5.3.1 Description du fonctionnement du module wb_ctrl

Le module wb_ctrl est un module important du contrôleur VGA car c'est lui que dirige le trafic des données entre les deux bus Wishbone. Il permet l'arbitrage des accès mémoire par le module host (bootloader). C'est lui aussi qui fait la lecture automatique des données des pixels en mémoire lors de l'affichage d'une image. Voici un schéma bloc du système :

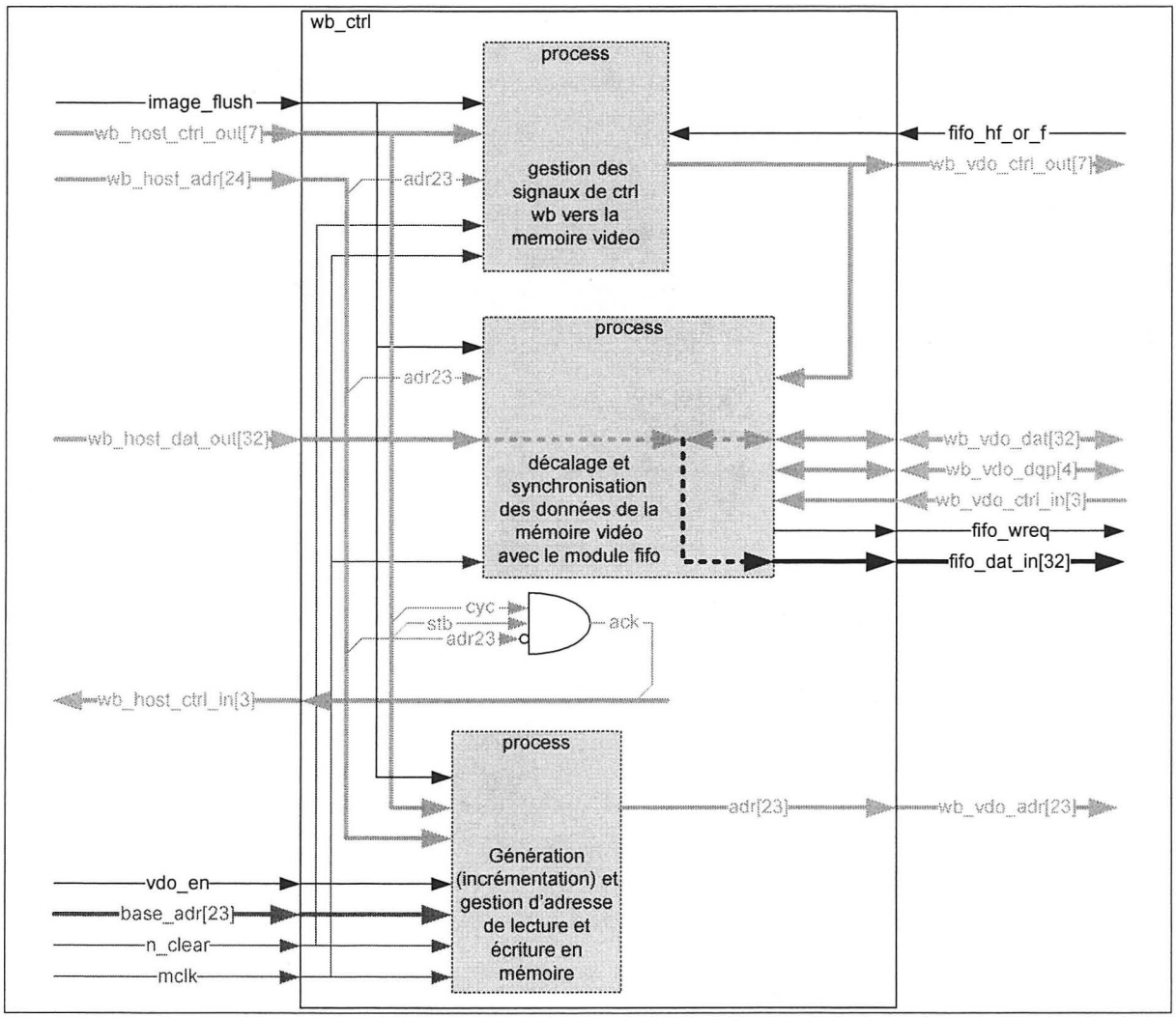

Figure 6.110 Schéma du module wb_ctrl.

Le processus du haut de la figure précédente permet de gérer les signaux de contrôle vers la mémoire vidéo. Il permet entre autres d'activer une lecture automatique en mode affichage ou de donner le contrôle au module host lorsque ce dernier en fait la requête.

Le processus qui se trouve au centre du schéma permet de bien synchroniser les échanges entre la mémoire vidéo et le reste du système puisque la mémoire vidéo possède un pipeline qui décale les données de deux cycles d'horloges en retard sur l'adresse. Il faut en tenir compte autant en lecture qu'en écriture. Référez-vous au document [10] Cypress, 2004, pour en savoir plus sur le fonctionnement de la mémoire ZBT.

Le processus du bas permet de générer les adresses d'accès à la mémoire vidéo (signal wb_vdo_adr) en commençant toujours par l'adresse de base de l'image sélectionnée dans le registre de contrôle (voir la section 6.11.3). L'adresse de la mémoire vidéo peut aussi provenir directement du module host lorsque celui-ci en fait la requête.

6.11.5.3.2 Résultats des simulations du module wb_ctrl

Voici la simulation fonctionnelle du module wb_ctrl :

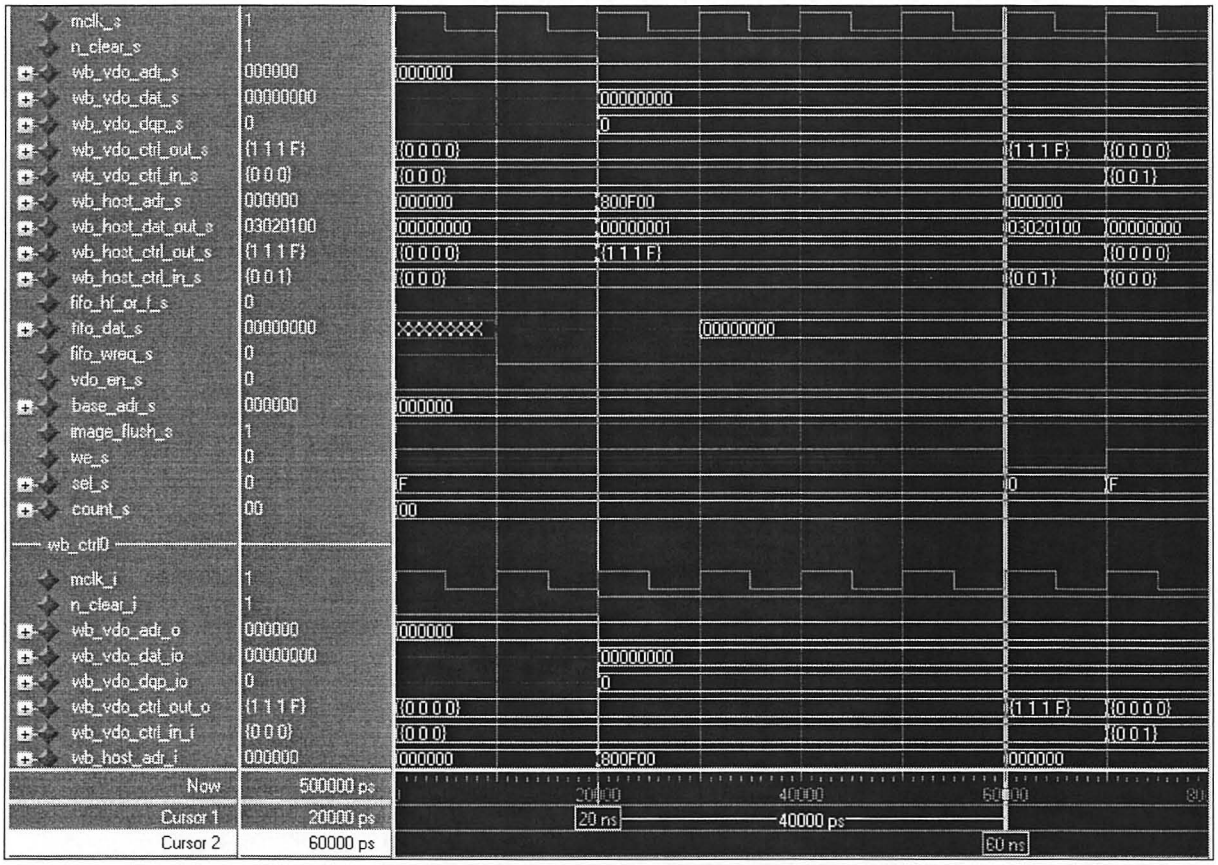

Figure 6.111 Simulation de l'initialisation du module wb_ctrl.

La figure précédente montre l'initialisation du module wb_ctrl. Le module se remet en fonction au curseur numéro un. Suite à ce curseur, on vérifie l'inactivité du module lorsque celui-ci n'est pas accédé, c'est-à-dire lorsque signal wb_host_adr possède une adresse au-delà de la valeur 0x7FFFFF (dans ce cas-ci, la valeur est de 0x800100). Le curseur numéro deux de cette figure montre le début du chargement de la mémoire vidéo par le module host (bootloader).

Le module wb_ctrl permet de décaler la donnée de 2 cycles d'horloge par rapport à l'adresse afin d'envoyer les données convenablement à la mémoire ZBT. On peut voir cela à la figure suivante au curseur numéro un. On active l'écriture en mémoire et la donnée apparait sur le bus (signal wb_vdo_dat) deux cycles d'horloge plus loin. Pour la simulation, on a bien séparé chaque écriture afin de bien identifier le fonctionnement, mais il serait possible

d'envoyer des données en rafale, donc à chaque cycle, sans problème. Il suffit de s'assurer d'avoir deux cycles d'horloge entre l'adresse et la donnée lors des accès mémoire.

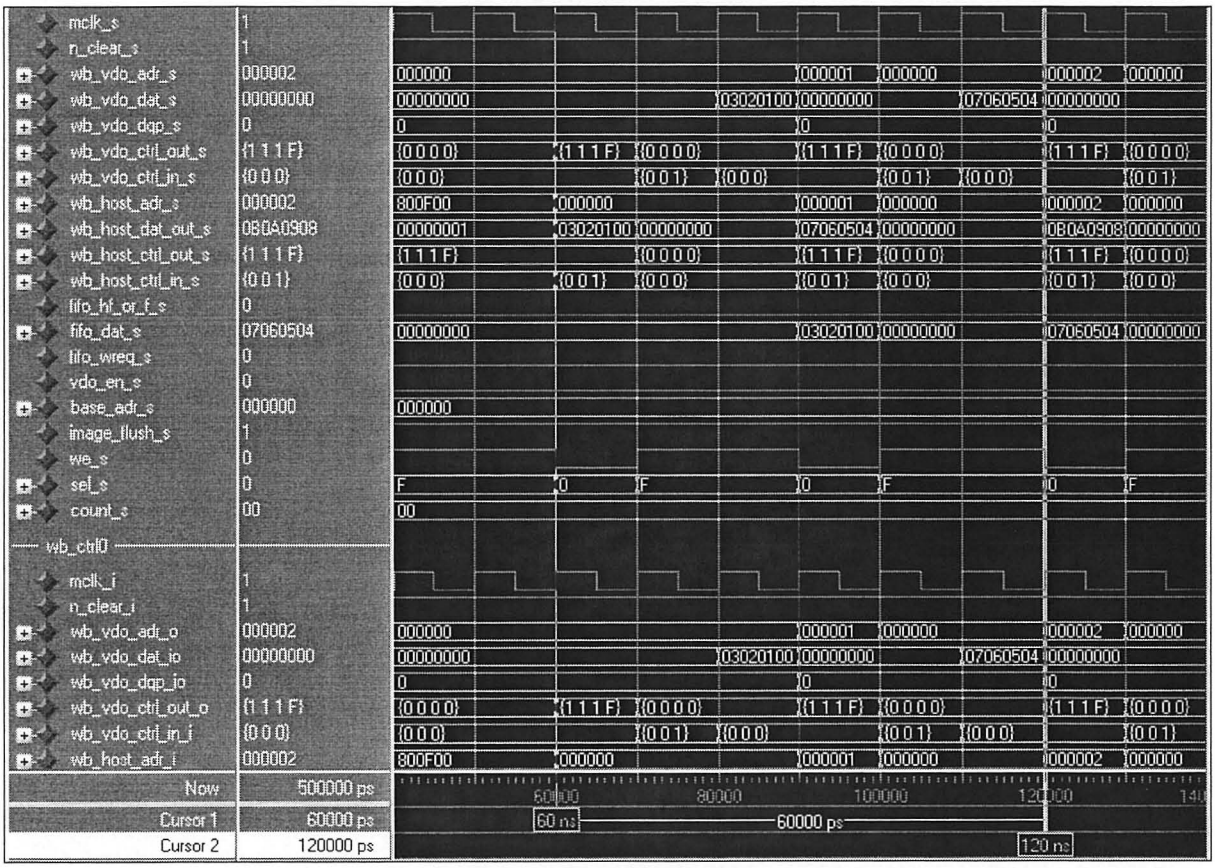

Figure 6.112 Simulation du chargement de la mémoire vidéo par le module wb_ctrl.

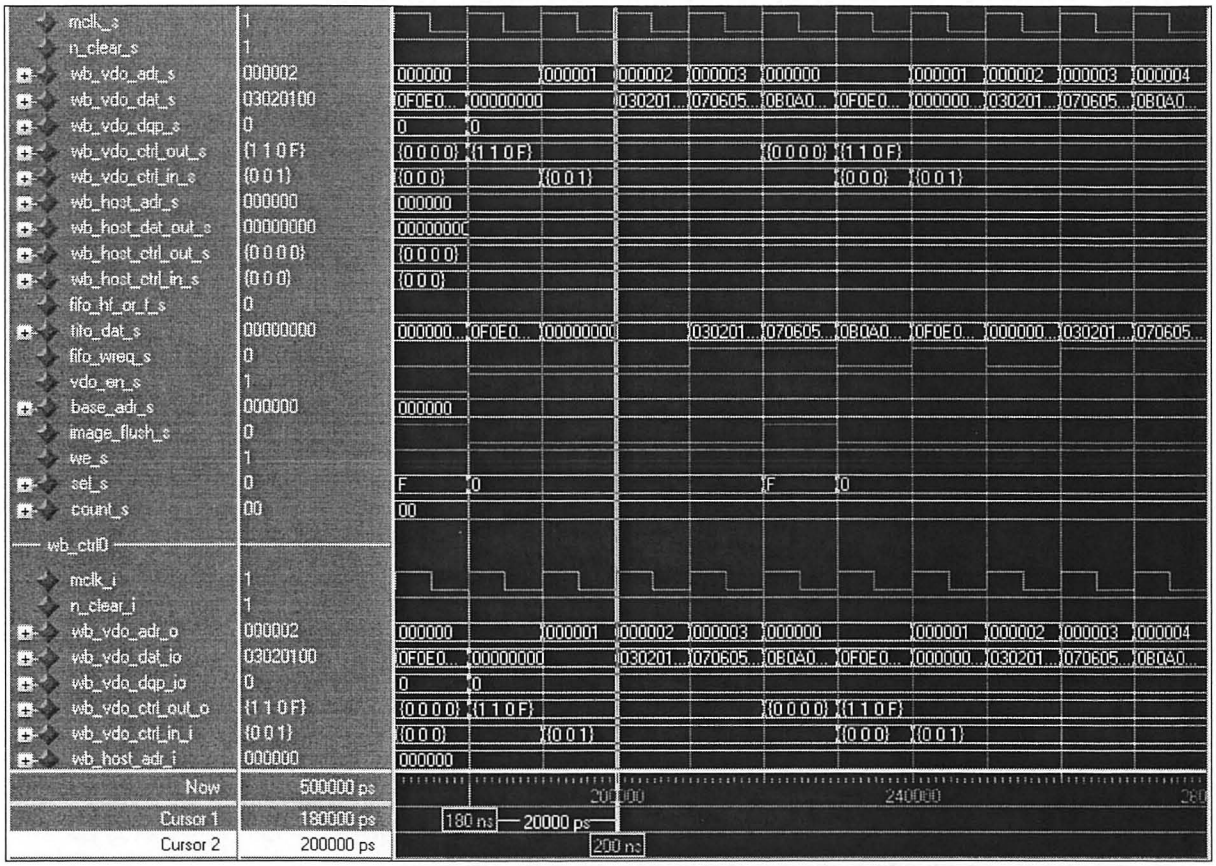

Figure 6.113 Simulation de la lecture en mémoire vidéo par le module wb_ctrl.

La figure précédente montre la lecture en mémoire par le module wb_ctrl. Les deux curseurs montrent le décalage de deux cycles d'horloge entre l'adresse et la donnée. Chaque donnée reçue est envoyée directement au module fifo qui est voisin au module wb_ctrl. La lecture en mémoire est suspendue lorsque le module fifo est à moitié plein. Nous n'avons pas d'exemple ici, mais lorsque cela arrive, il reste suffisamment d'espace dans le fifo pour permettre deux écritures provenant de la mémoire (puisque la donnée arrive toujours deux cycles plus tard).

À peu près au centre de l'image, le signal image_flush s'active durant un cycle d'horloge afin de vérifier que le module wb_ctrl recommence bien sa lecture depuis le début lorsque cet événement se produit. On remarque que l'adresse est réinitialisée à 0x000000 et que la lecture en mémoire recommence.

6.11.5.4 Description du module fifo (fifo.vhd)

Voici le symbole du module fifo :

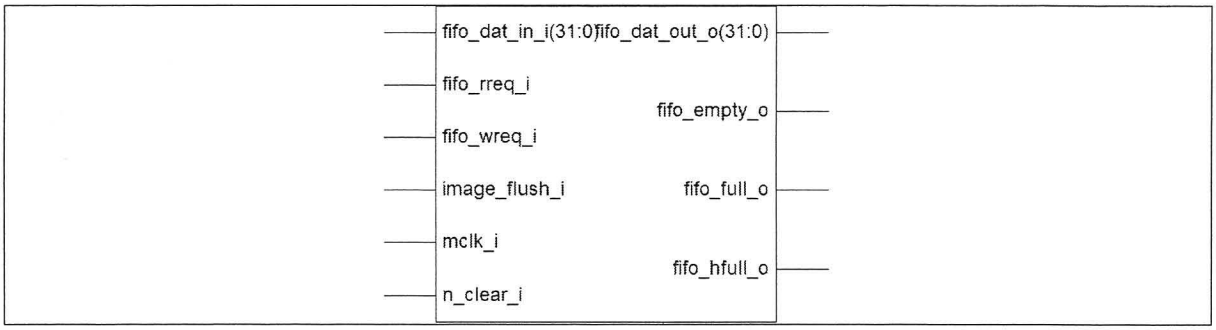

Figure 6.114 Symbole du module fifo.

Avant de décrire le fonctionnement du module fifo, voici un tableau de ses entrées et de ses sorties :

Tableau 6.28

Description des entrées et des sorties du module fifo

Nom	Entrée/ Sortie	Description
mclk	E	Signal d'horloge maîtresse, voir les sections 6.7 et 6.7.4 pour plus de détails.
n_clear	E	Signal de réinitialisation générale du système. Voir la section 6.9 pour plus de détails.
fifo_dat_in[32]	E	Signal des données pixel qui proviennent de la mémoire vidéo et qui passent par le module wb_ctrl en direction du module fifo.
fifo_wreq	E	Signal qui indique au module fifo de mémoriser une donnée. Ce signal provient du module wb_ctrl.
image_flush	E	Signal qui indique que tous les tampons doivent être vidés lorsque tous les pixels d'une image ont été transférés au module dclk_fifo. Ce signal est généré par le module pixel_counter.
fifo_rreq	E	Signal qui signifie au module fifo de faire sortir une donnée vers le module pixel_processor. Ce signal provient du module pixel_processor.
fifo_dat_out[32]	S	Signal de données de sortie vers le module pixel_processor.
fifo_full	S	Signal qui indique que le module fifo est plein.
fifo_hfull	S	Signal qui indique que le module fifo est à moitié plein.
fifo_empty	S	Signal qui indique que le module fifo est vide. Ce signal indique au module pixel_processor qu'aucune donnée n'est disponible, ce qui le place en attente.

6.11.5.4.1 Description du fonctionnement du module fifo

Le module fifo est un registre qui fonctionne selon le principe : premier entré, premier sorti. Il est organisé sous forme d'un ensemble de registres. On peut écrire dans le registre pointé par la valeur d'un compteur wptr et on peut lire le registre pointé par le pointeur rptr. Ces pointeurs permettent de savoir si les registres sont vides ou pleins en comparant leur valeur. Voici un schéma bloc du module fifo :

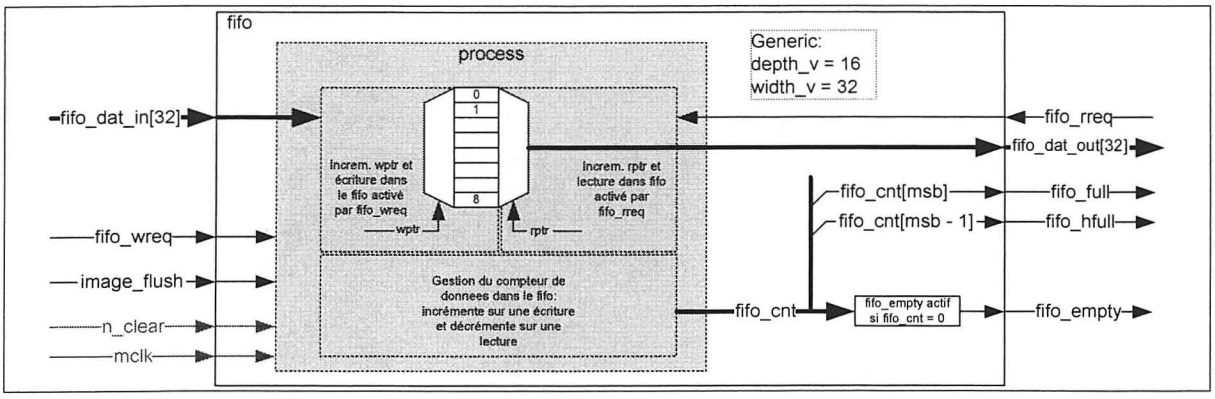

Figure 6.115 Schéma du module fifo.

Ce module sert de tampon entre les modules wb_ctrl et pixel_processor afin de s'assurer que le flot des données en sortie du contrôleur VGA n'est pas brisé. Il permet de laisser suffisamment de temps au module pixel_processor pour qu'il fasse son formatage approprié des pixels.

6.11.5.4.2 Résultats des simulations du module fifo

Voici la simulation fonctionnelle du module fifo :

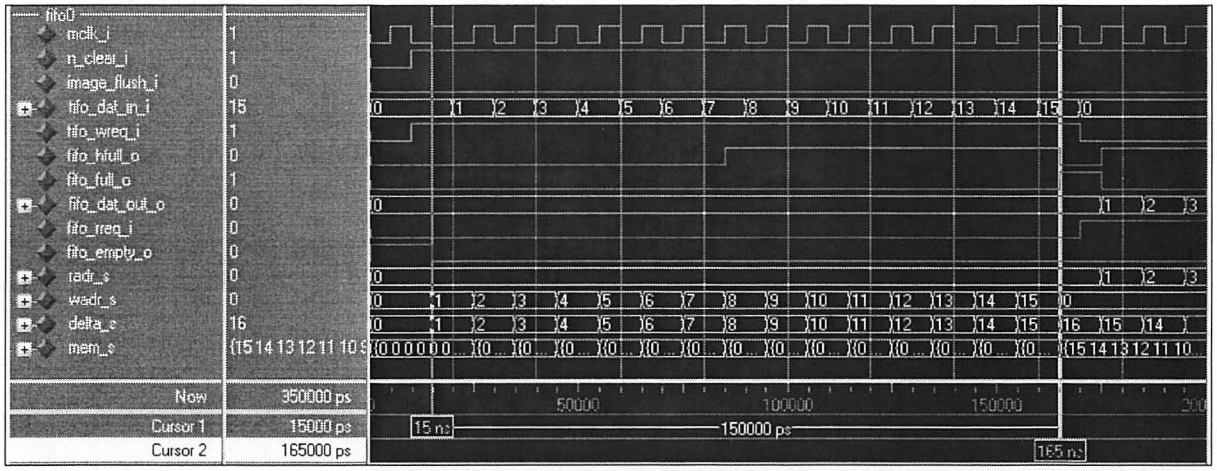

Figure 6.116 Simulation de l'écriture dans le module fifo.

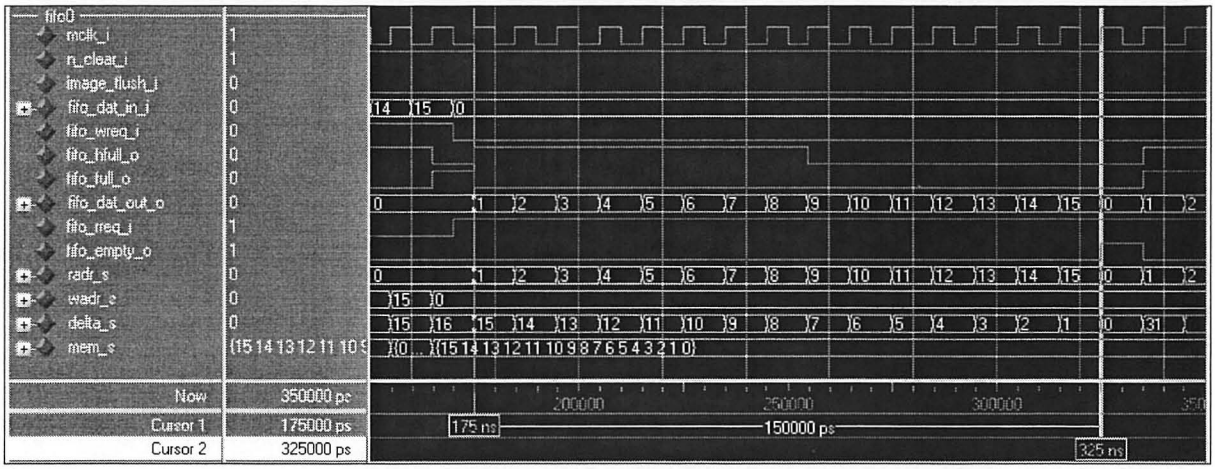

Figure 6.117 Simulation de la lecture dans le module fifo.

La Figure 6.116 montre l'écriture dans le module fifo. Le signal delta permet de voir qu'il n'y a pas de lecture et d'écriture simultanée puisque le delta augmente jusqu'à son maximum. À ce moment, le fifo est plein et le signal fifo_full s'active de manière synchrone (voir le curseur numéro deux).

La Figure 6.117 montre la lecture dans le module fifo. Le module fifo étant plein, chaque lecture dans celui-ci permet d'incrémenter l'adresse de lecture et fait en sorte que la valeur du signal delta diminue. Le signal fifo_full se désactive donc suite à la première lecture (au curseur numéro un) et après 15 autres données lues, le fifo est vide ce qui fait activer le signal fifo_empty au curseur numéro deux. Il est à noter qu'il n'y a pas de protection sur le dépassement, ni en lecture, ni en écriture pour la simple raison de simplifier

l'implémentation. Il faut donc s'assurer de bien gérer les événements sur les signaux fifo_full et fifo_empty pour s'assurer du bon fonctionnement du fifo.

6.11.5.5 Description du module pixel_processor (pixel_processor.vhd)

Voici le symbole du module pixel_processor :

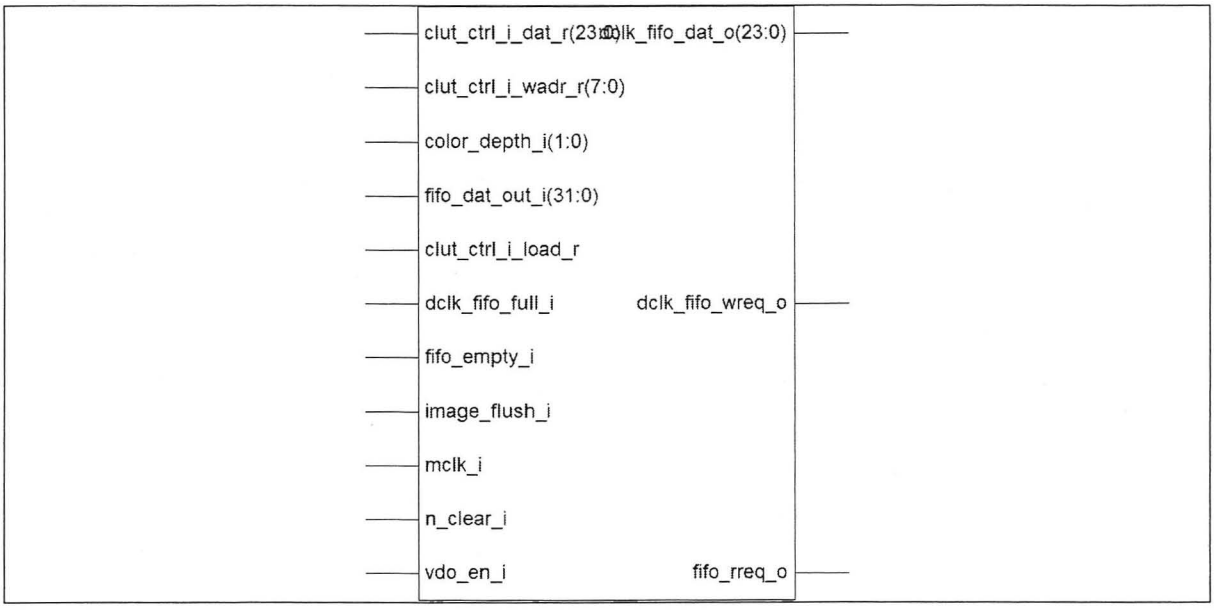

Figure 6.118 Symbole du module pixel_processor.

Avant de décrire le fonctionnement du module pixel_processor, voici un tableau de ses entrées et de ses sorties :

Tableau 6.29

Description des entrées et des sorties du module pixel_processor

Nom	Entrée/ Sortie	Description
mclk	E	Signal d'horloge maîtresse, voir les sections 6.7 et 6.7.4 pour plus de détails.
n_clear	E	Signal de réinitialisation générale du système. Voir la section 6.9 pour plus de détails.

clut_ctrl[33]	E	Groupe de signaux qui permettent de contrôler le chargement de la table des couleurs (module clut) par le module ctrl_register. Ce groupe contient les signaux : • load : signal d'activation de chargement d'une donnée; • adr[8] : emplacement de la donnée à charger dans la table; • dat[24] : données de 24 bits à chargée.
fifo_rreq	S	Signal qui signifie au module fifo de faire sortir une donnée vers le module pixel_processor. Ce signal provient du module pixel_processor.
fifo_dat_out[32]	E	Signal de données de sortie du module fifo vers le module pixel_processor.
color_depth[2]	E	Signal qui spécifie le nombre de bits d'encodage de l'image : • 00 pour un encodage 8 bits • 01 pour un encodage 16 bits (pas encore implémenté) • 1x pour un encodage 24 bits
vdo_en	E	Signal général qui indique si le contrôleur vidéo est activé ou non, ce signal provient du module ctrl_register et correspond au bit ctrl_reg(0).
fifo_empty	E	Signal qui indique que le module fifo est vide. Ce signal indique au module pixel_processor qu'aucune donnée n'est disponible, ce qui le place en attente.
image_flush	E	Signal qui indique que tous les tampons doivent être vidés lorsque tous les pixels d'une image ont été transférés au module dclk_fifo. Ce signal est généré par le module pixel_counter.
dclk_fifo_full	E	Signal qui indique que le module dclk_fifo est plein, ce qui permet au module pixel_processor de se mettre en attente.
dclk_fifo_dat[24]	S	Signal qui constitue les données de pixel qui sont envoyées au module dclk_fifo avant d'être affichées.
dclk_fifo_wreq	S	Signal qui indique au module dclk_fifo et au pixel_counter qu'un pixel valide est chargé dans le module dclk_fifo.

6.11.5.5.1 Description du fonctionnement du module pixel_processor

L'unité pixel_processor est celui qui permet au contrôleur VGA d'extraire les valeurs des pixels qui proviennent de la mémoire et de les mettre en forme pour que toutes les couleurs qui sortent vers le convertisseur VGA soient de 24 bits RGB. Chaque mode de couleur fonctionne différemment. Voici le fonctionnement de chaque mode implémenté.

En mode 8 bits, les pixels emmagasinés dans la mémoire vidéo sont encodés sur 8 bits, ce qui donne accès à 256 couleurs différentes. Chacune de ces couleurs sont mémorisées dans le sous-module clut lors du chargement par le bootloader (uniquement si le format des images sont configurées en mode 8 bits). Donc, pour chaque octet chargé de la mémoire, on doit lire la couleur associée dans la table des couleurs et c'est cette couleur de 24 bits qui est envoyée en sortie.

Pour le mode d'affichage 24 bits, les pixels sont placés en mémoire consécutivement. Lors de la lecture en mémoire des données pixel, 32 bits sont lus à la fois afin de diminuer la bande passante utilisée. Il est donc nécessaire de reformater les pixels en 24 bits et de garder les octets supplémentaires pour reconstituer le prochain pixel. Par exemple, si on lit en mémoire les octets de couleur B2, R1, G1, B1 (ce qui donne 32 bits) Le pixel de 24 bits R1G1B1 est envoyé vers le convertisseur VGA, mais l'octet B2 est gardé de côté et sera jumelé au prochain mot de 32 bits lut de la mémoire, soit le mot G3, B3, R2, G2. Le pixel R2G2B2 sera donc formé et envoyé vers le convertisseur VGA et les octets G3, B3 seront mis de côté pour le prochain pixel et ainsi de suite. C'est un peu le même principe qui est utilisé pour remplir la mémoire PROM et qui est expliqué à la section 6.10.5. Voici une figure qui explique l'ordonnancement des pixels.

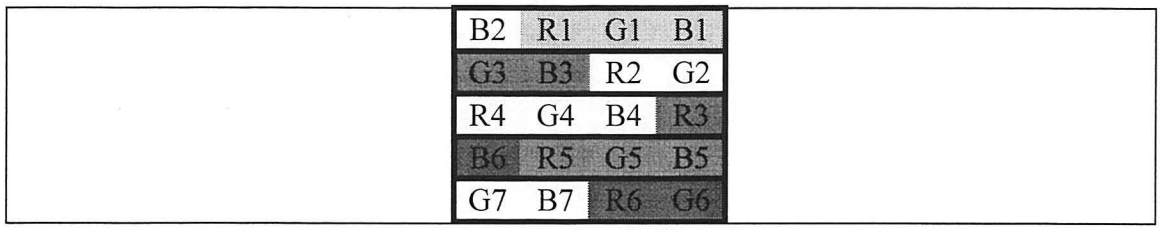

Figure 6.119 Organisation des données dans la mémoire vidéo.

Cette manière de fonctionner est très efficace pour la lecture et l'écriture des données en mémoire car elle permet une utilisation de 25% de moins d'espace et aussi de réduire de 25% le nombre d'accès mémoire, ce qui libère de la bande passante de la mémoire.

Voici un schéma bloc du module pixel_processor :

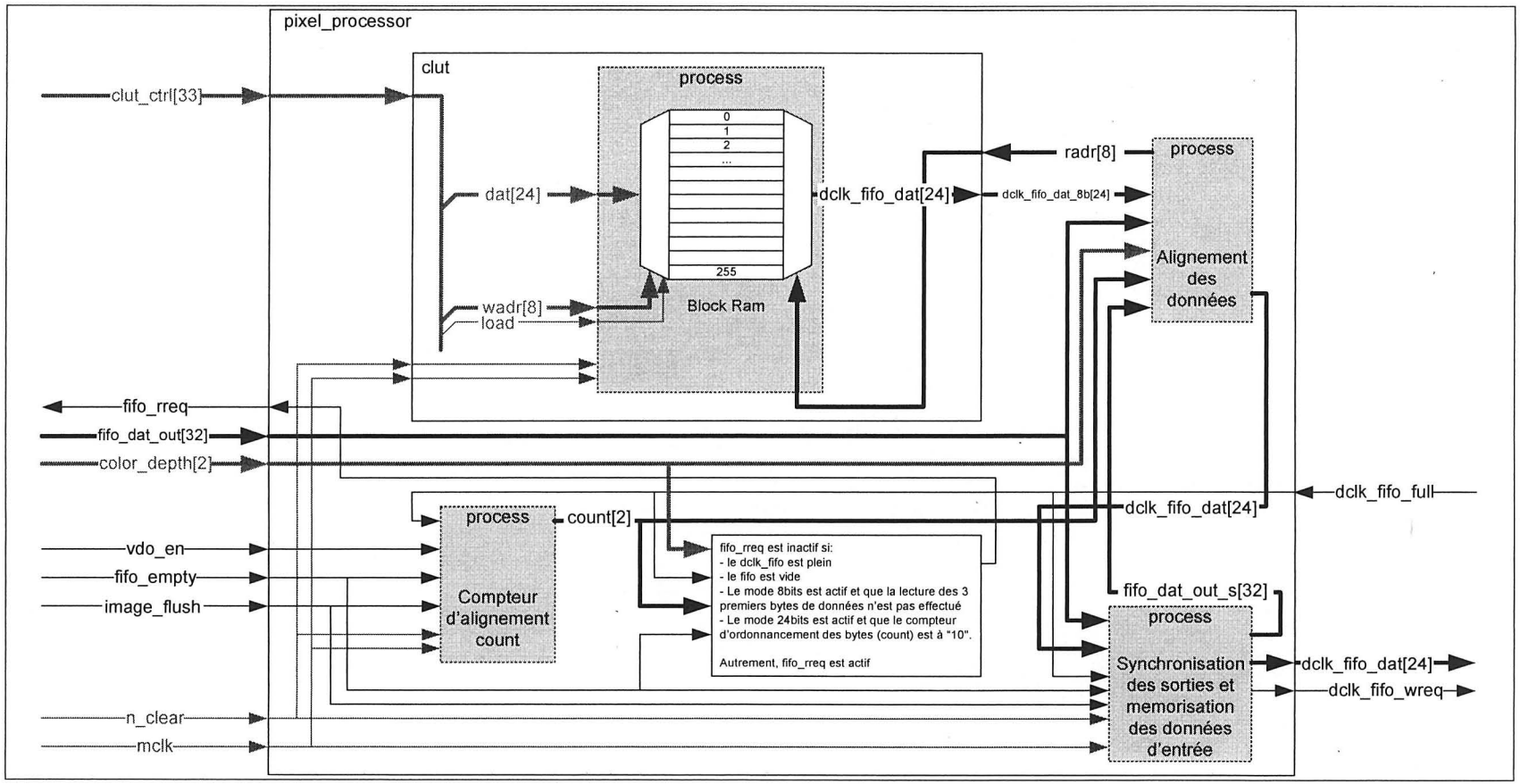

Figure 6.120 Schéma du module pixel_processor.

Sur la figure précédente, on voit le sous-module clut qui constitue la table des couleurs utilisée en mode 8 bits uniquement et qui est expliqué à la section suivante. On observe aussi trois processus qui permettent de remettre les pixels en 24 bits RGB.

Le processus du bas à gauche sur la figure précédente sert de compteur pour identifier les octets qui constituent le pixel présent et aussi les octets à mettre de côté lors d'un affichage en mode 24 bpp (uniquement). Pour mieux comprendre l'utilité du compteur, on peut se référer à la Figure 6.119. La première ligne doit être synchronisée avec la valeur 0 du compteur afin de lire les 3 octets les moins significatifs du mot. La deuxième ligne doit être synchronisée avec la valeur 1 du compteur afin de concaténer les deux octets les moins significatifs du mot avec l'octet le plus significatif du mot précédent. La troisième ligne permet de lire deux pixels de 24 bits. Donc, le compteur doit passer ses deux dernières valeurs, 2 et 3, sur ce mot. Lorsque le compteur est à 2, le pixel est formé du bit le moins significatif du mot concaténé avec les deux octets les plus significatifs du mot précédent. Lorsque le compteur est à 3, le pixel est formé des 3 octets les plus significatifs du mot présent. Le processus recommence ainsi continuellement à chaque trois mot, ou autrement dit, à chaque quatre pixels.

En mode 8 bpp, le compteur identifie simplement l'octet à utiliser comme pixel. Il sert en quelque sorte de pointeur de pixel sur le mot lu du module fifo.

Le processus du haut à droite permet la concaténation des données pixels. Le dernier processus en bas à droite de la figure précédente sert à la synchronisation des sorties vers le convertisseur VGA.

6.11.5.5.2 Résultats des simulations du module pixel_processor

Voici la simulation fonctionnelle du module pixel_processor :

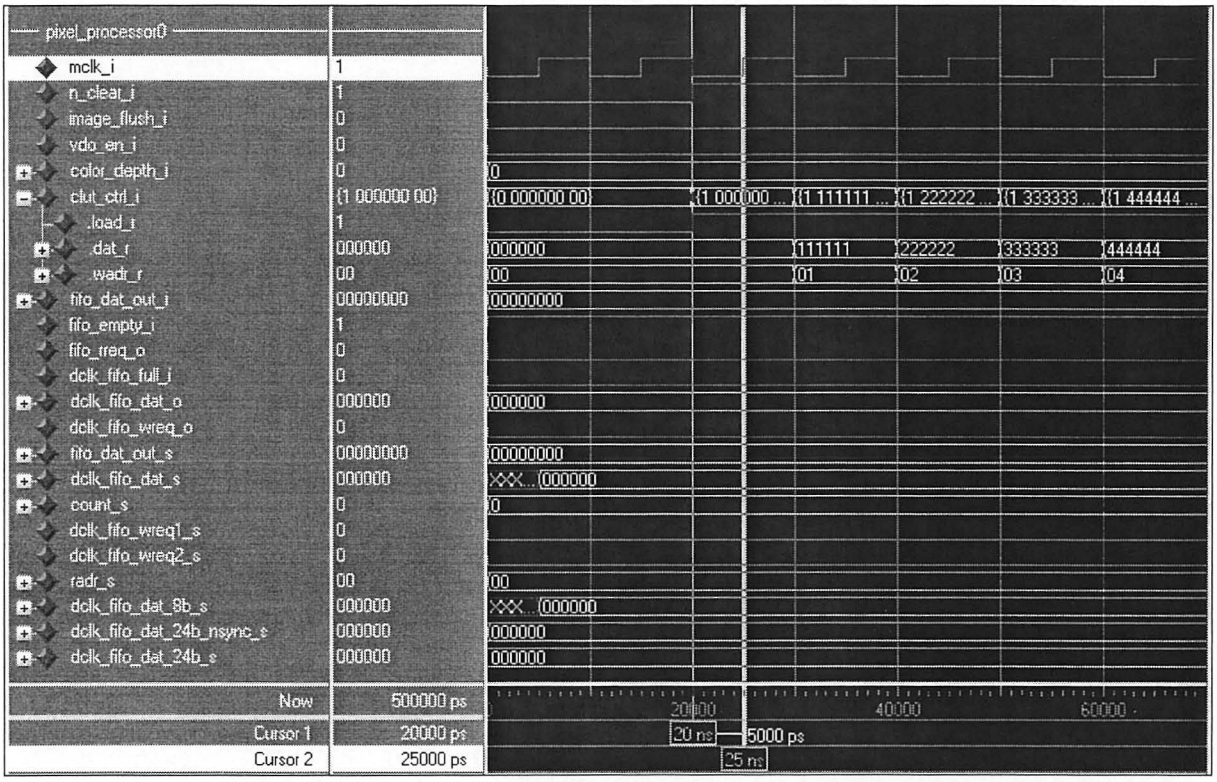

Figure 6.121 Simulation de l'initialisation du module pixel_processor.

La figure précédente montre l'initialisation du module pixel_processor et le début du chargement de la table des couleurs. Tous les signaux internes du module sont remis à zéro lors de l'initialisation. Au curseur numéro un, on désactive le signal n_clear et on active le signal clut_ctrl.load afin de commencer le chargement de la table des couleurs. La table contient 256 valeurs de 24 bits, ce qui demande huit bits sur le signal wadr pour accéder à chacune des couleurs contenu dans la table. La figure suivante montre le chargement des 16 premières valeurs de la table entre les deux curseurs. Suite à cela, on active la vidéo en activant le signal vdo_en.

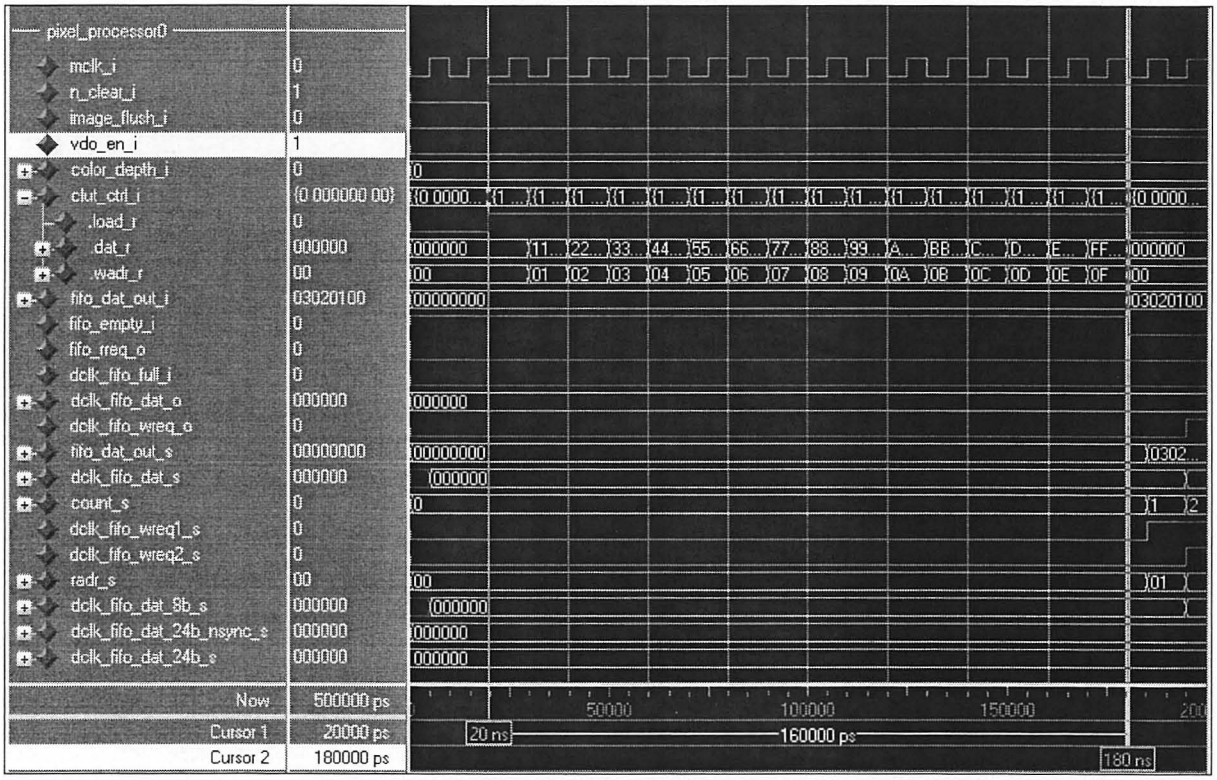

Figure 6.122 Simulation du chargement de la table des couleurs du module pixel_processor.

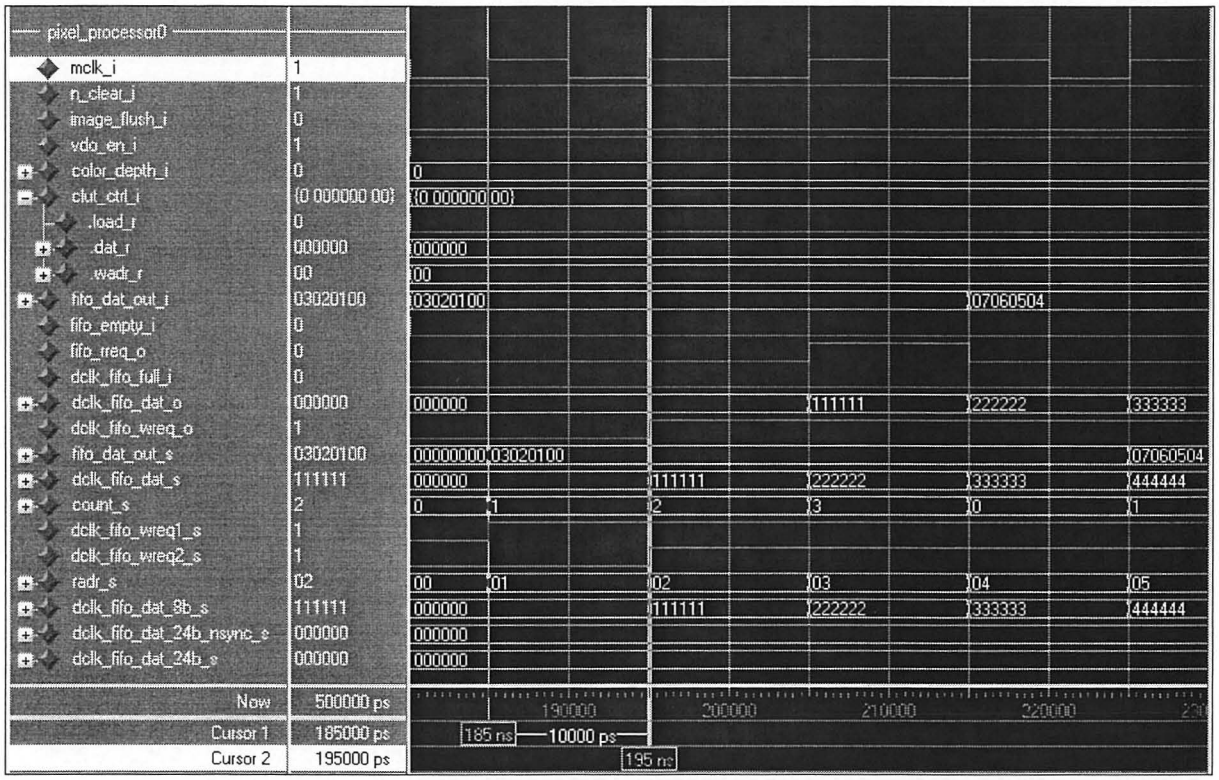

Figure 6.123 Simulation de la mise en forme des pixels par le module pixel_processor en mode 8 bpp.

Suite à l'activation de la vidéo en mode 8 bpp, soit au curseur numéro un de la Figure 6.123, le module attend qu'il y ait une donnée valide à l'entrée fifo_dat_out. Si le signal fifo_empty est inactif, c'est qu'une donnée est valide à cette entrée, comme c'est le cas ici. Le module passe donc chaque octet de ce mot de 32 bits dans la table des couleurs, ce qui retourne une valeur de 24 bits pour chaque octet. Ce 24 bits est le pixel à afficher et est acheminé en sortie sur le signal dclk_fifo_dat vers le module dclk_fifo. Le compteur count permet d'identifier l'octet à traiter. La valeur 0 pointe sur l'octet le moins significatif du signal fifo_dat_out alors que la valeur 3 pointe sur l'octet le plus significatif. Il est à noter qu'à la valeur 3 du compteur count, le signal fifo_rreq s'active lorsque le fifo n'est pas vide (signal fifo_empty).

La figure suivante montre que le processus se poursuit lorsque les quatre octets sont traités. Une nouvelle donnée est lue du module fifo et le compteur count est remis à zéro pour un nouveau traitement octet par octet.

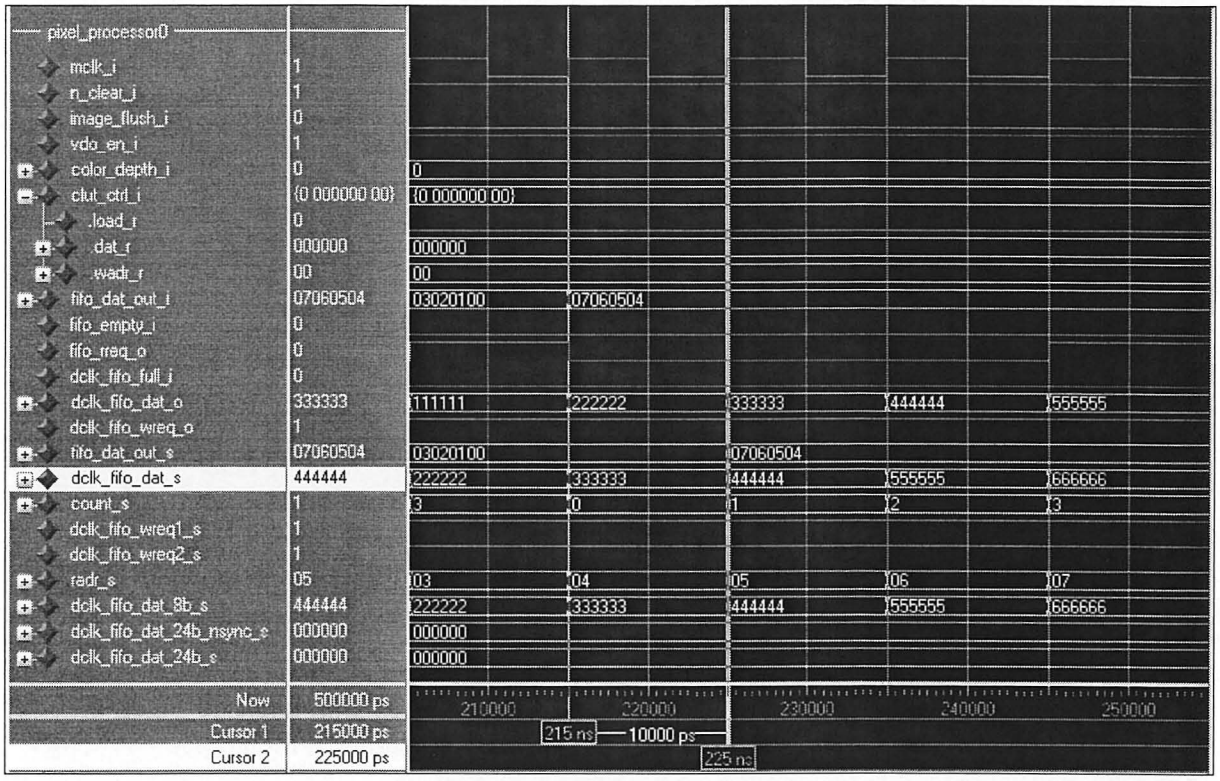

Figure 6.124 Simulation de la mise en forme des pixels par le module pixel_processor en mode 8 bpp, la suite.

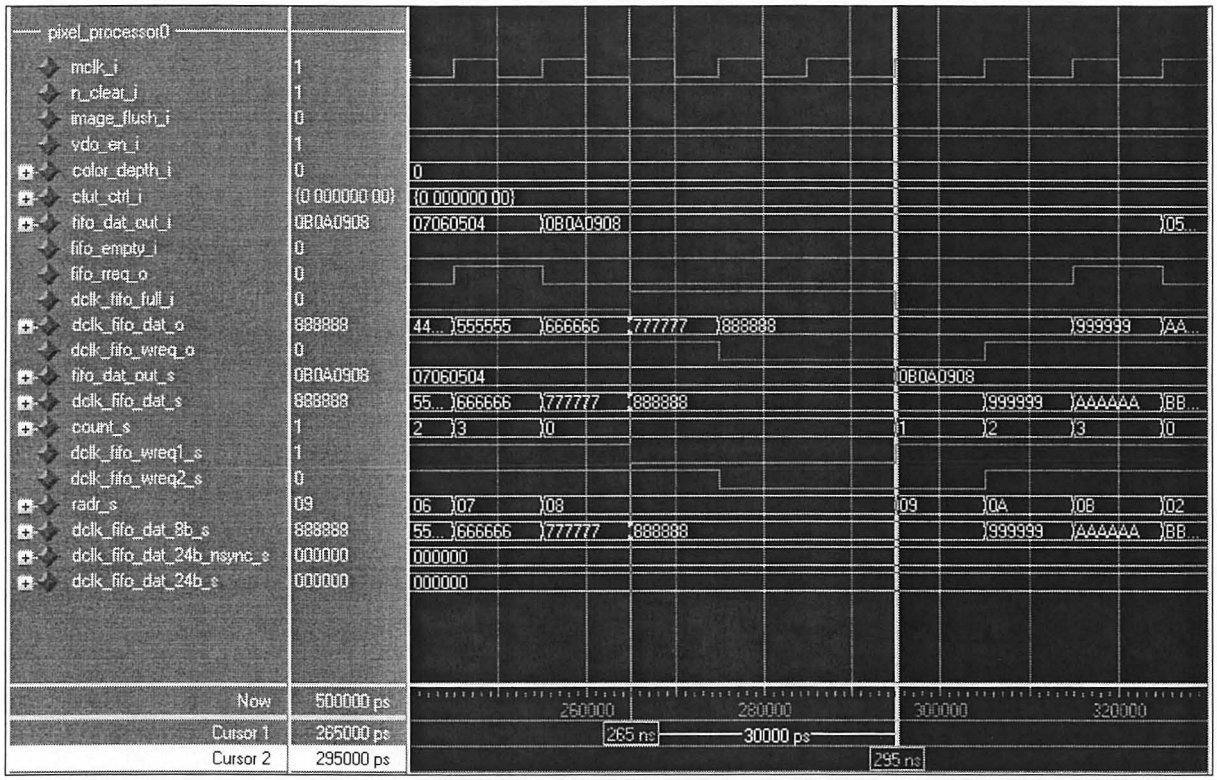

Figure 6.125 Simulation de l'effet du signal dclk_fifo_full sur le fonctionnement du module pixel_processor.

La Figure 6.125 montre que si le signal dclk_fifo_full s'active, le traitement des données est interrompu pour laisser le temps au module dclk_fifo de se vider. Ainsi, étant donné que le module dclk_fifo est lu de manière beaucoup plus lente que tout le reste de la chaine de traitement des pixels, les modules fifo et pixel_processor sont souvent en attente de cette manière, ce qui permet de libérer la mémoire vidéo pour d'autres tâches éventuelles.

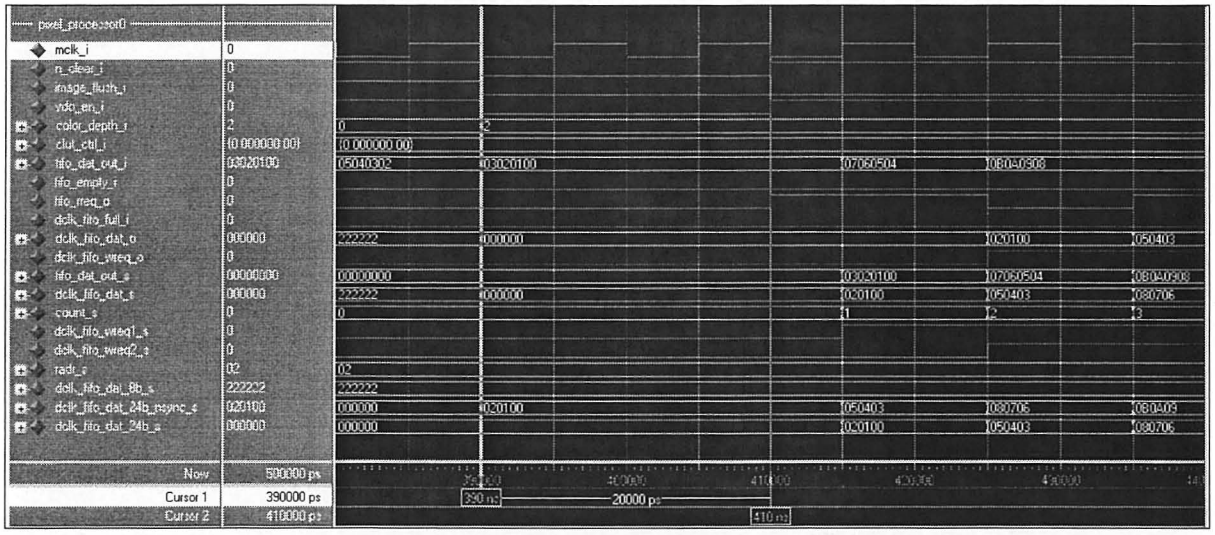

Figure 6.126 Simulation de la réinitialisation du module pixel_processor en mode 24 bpp.

La figure précédente montre la réinitialisation du module pixel_processor en mode 24 bpp. Lors de la remise à zéro, le signal vdo_en se doit d'être désactiver pour rendre la simulation réaliste. Le curseur numéro un montre l'activation du signal n_clear et le curseur numéro deux montre l'activation du signal vdo_en.

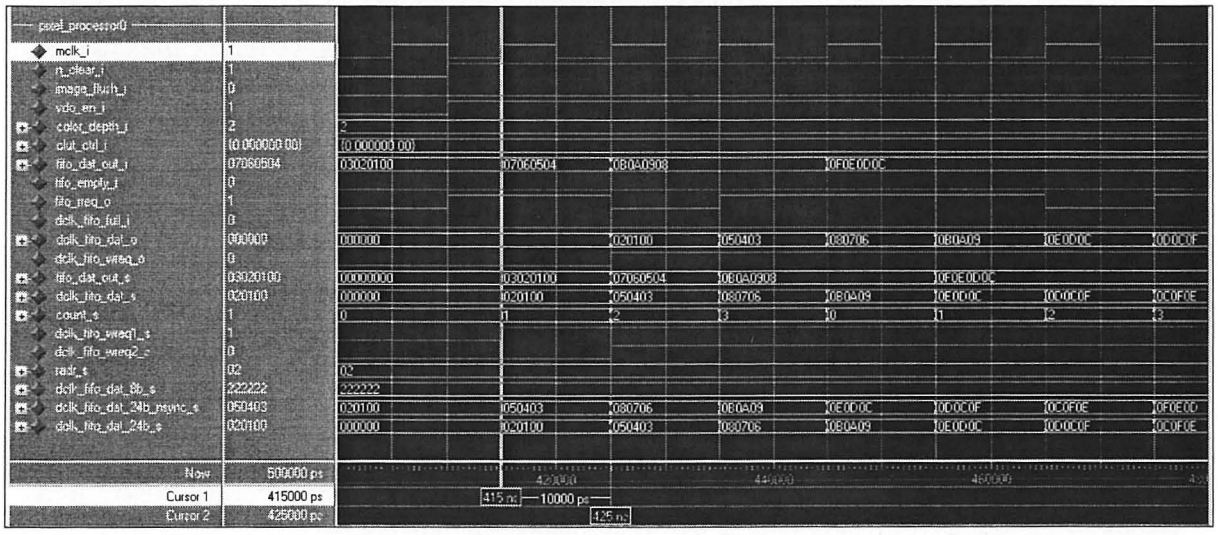

Figure 6.127 Simulation du traitement des pixels par le module pixel_processor en mode 24 bpp.

Si on regarde les signaux fifo_dat_out et dclk_fifo_dat de la Figure 6.127, on observe exactement la remise en forme des données de 24 bits tel que l'explique la section 6.11.5.5.1. Les données de 24 bits sortent 2 cycles derrière l'entrée à cause des registres nécessaires au traitement. Le curseur numéro un montre que lorsque le compteur est à la valeur 0, le registre interne dclk_fifo_dat mémorise les trois octets les moins significatifs de l'entrée fifo_dat_out. Lorsque le compteur count est à la valeur 1, le registre interne dclk_fifo mémorise les deux octets les moins significatifs de l'entrée concaténés avec l'octet le plus significatif du registre de mémorisation de l'entrée fifo_dat_out. Le processus continue ainsi comme expliqué à la section précédente.

6.11.5.5.3 Description du module clut (clut.vhd)

Voici le symbole du module clut :

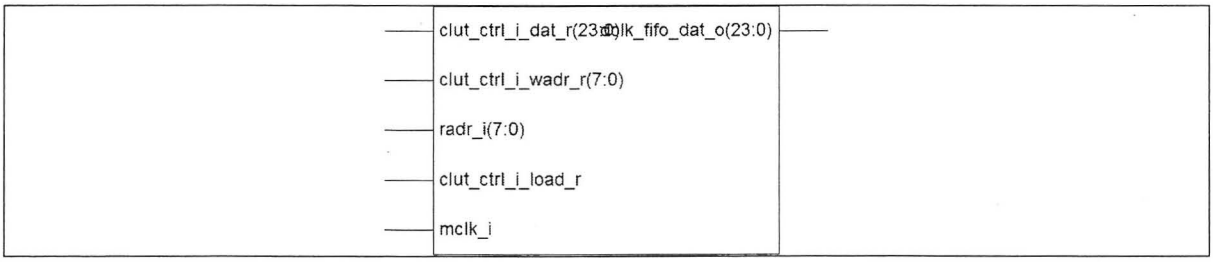

Figure 6.128 Symbole du module clut.

Avant de décrire le fonctionnement du module clut, voici un tableau de ses entrées et de ses sorties :

Tableau 6.30

Description des entrées et des sorties du module clut

Nom	Entrée/ Sortie	Description
mclk	E	Signal d'horloge maîtresse, voir les sections 6.7 et 6.7.4 pour plus de détails.
n_clear	E	Signal de réinitialisation générale du système. Voir la section 6.9 pour plus de détails.

clut_ctrl[33]	E	Groupe de signaux qui permettent de contrôler le chargement de la table des couleurs (module clut) par le module ctrl_register. Ce groupe contient les signaux : • load : signal d'activation de chargement d'une donnée; • adr[8] : emplacement de la donnée à charger dans la table; • dat[24] : données de 24 bits à chargée.
dclk_fifo_dat[24]	S	Signal qui constitue les données de pixel qui sont envoyées au module dclk_fifo avant d'être affichées.
radr[8]	E	Signal de la couleur encodée sur 8bits utilisée lors de l'affichage d'image en mode 8 bits.

6.11.5.5.3.1 Description du fonctionnement du module clut

Selon la Figure 6.120, on observe facilement que ce module est uniquement un petit groupe de 256 registres de 24 bits chacun qui servent de table des couleurs pour le mode d'affichage 8 bits. Le groupe de signaux clut_ctrl permet de remplir la table des couleurs. Le signal radr permet d'identifier la valeur de la table qui doit se présenter à la sortie dclk_fifo_dat.

6.11.5.5.3.2 Résultats des simulations du module clut

Voici la simulation fonctionnelle du module clut :

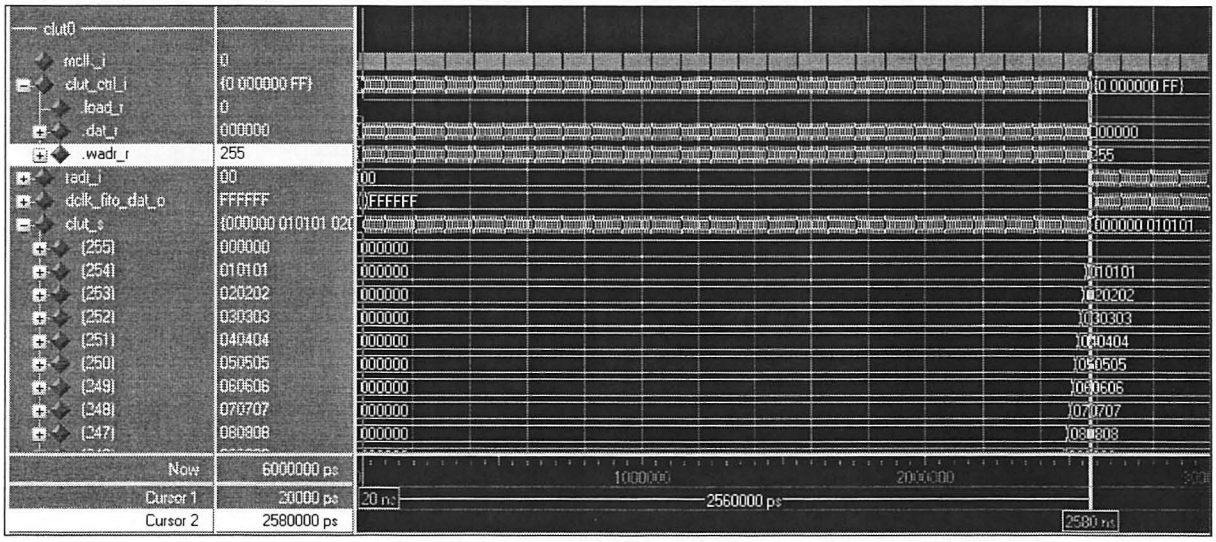

Figure 6.129 Vue d'ensemble de la simulation du module clut.

Comme vous pouvez le constater sur la figure précédente, il est inutile de visualiser l'ensemble de la simulation puisque cela ne nous donne aucune information pertinente sur le fonctionnement du module. Nous montrons plutôt dans les deux figures suivantes que les valeurs qui ont été enregistrées dans la table peuvent être lues par la suite sans problème.

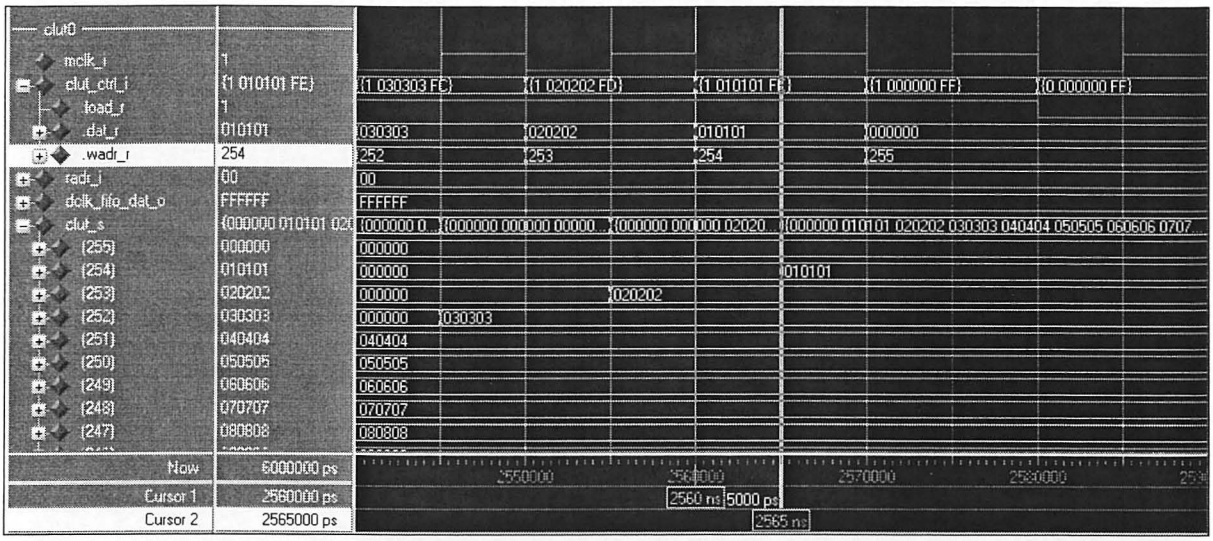

Figure 6.130 Simulation du remplissage de la table des couleurs dans le module clut.

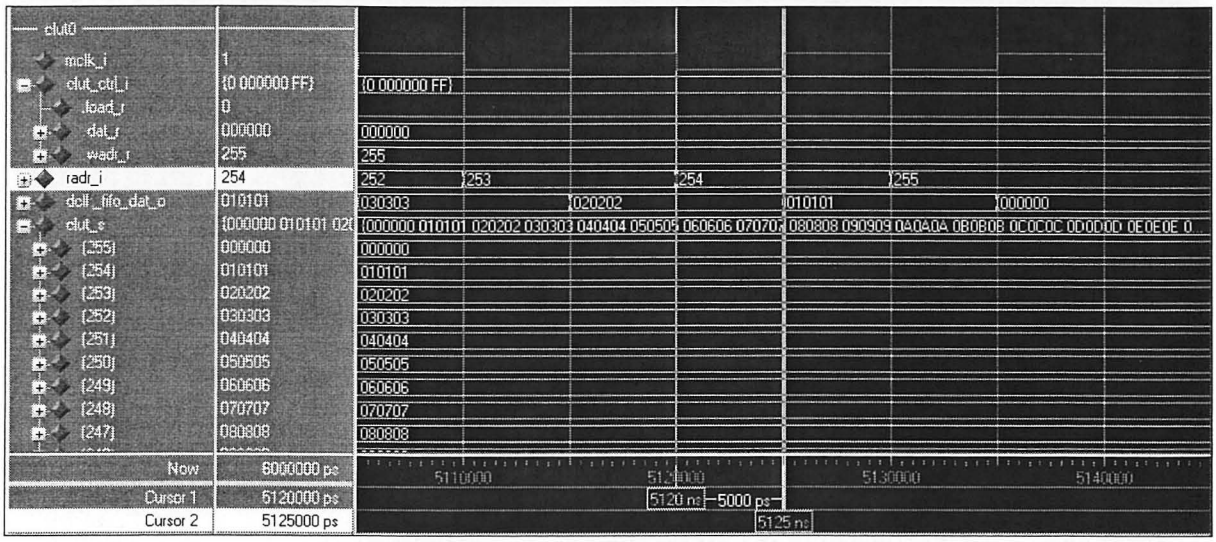

Figure 6.131 Simulation de la lecture de la table des couleurs dans le module clut.

À la Figure 6.130, on montre au curseur numéro deux l'écriture de la valeur 0x010101 à l'adresse 254 et sur la Figure 6.131 on montre la lecture de cette même valeur à cette même adresse. Le module clut est uniquement une table de 256 registres de 24 bits.

6.11.5.6 Description du module dclk_fifo (dclk_fifo.vhd)

Voici le symbole du module dclk_fifo :

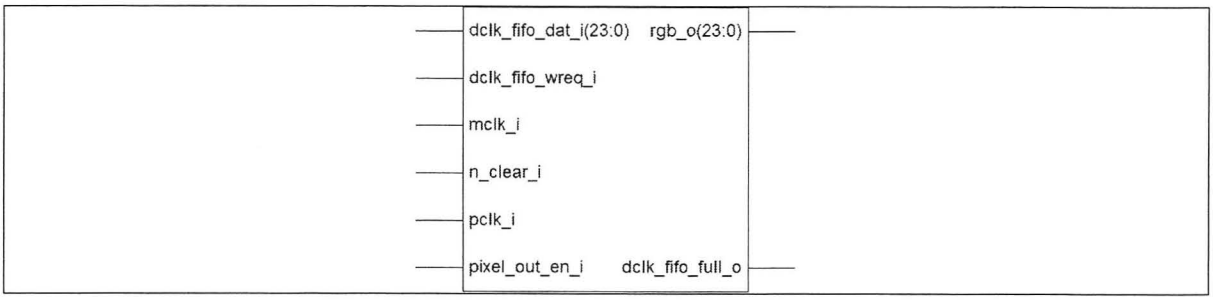

Figure 6.132 Symbole du module dclk_fifo.

Avant de décrire le fonctionnement du module dclk_fifo, voici un tableau de ses entrées et de ses sorties :

Tableau 6.31

Description des entrées et des sorties du module dclk_fifo

Nom	Entrée/ Sortie	Description
mclk	E	Signal d'horloge maîtresse, voir les sections 6.7 et 6.7.4 pour plus de détails.
pclk	E	Signal d'horloge des pixels. Il est généré par le module pclk_gen (section 6.7.3).
n_clear	E	Signal de réinitialisation générale du système. Voir la section 6.9 pour plus de détails.
dclk_fifo_dat[24]	E	Signal qui constitue les données de pixel qui sont envoyées au module dclk_fifo avant d'être affichées.
dclk_fifo_full	S	Signal qui indique que le module dclk_fifo est plein, ce qui permet au module pixel_processor de se mettre en attente.
dclk_fifo_wreq	E	signal qui indique au dclk_fifo et au pixel_counter qu'un pixel valide est chargé dans le module dclk_fifo.
pixel_out_en	E	Signal qui indique au contrôleur vidéo de sortir un nouveau pixel. Ce signal contrôle donc la lecture des pixels dans le module dclk_fifo.
rgb[24]	S	Sortie des couleurs groupées en un seul signal de 24 bits vers le convertisseur VGA.

6.11.5.6.1 Description du fonctionnement du module dclk_fifo

Le module dclk_fifo est un groupe de registres fifo qui permet d'écrire les données d'entrée en synchronisation avec l'horloge maîtresse mclk et de lire les données de sortie en

synchronisation avec l'horloge pixel pclk. Ce module sert à s'assurer que le flot de pixels en sortie est continu à la sortie vers le convertisseur VGA. Voici un schéma bloc du module dclk_fifo :

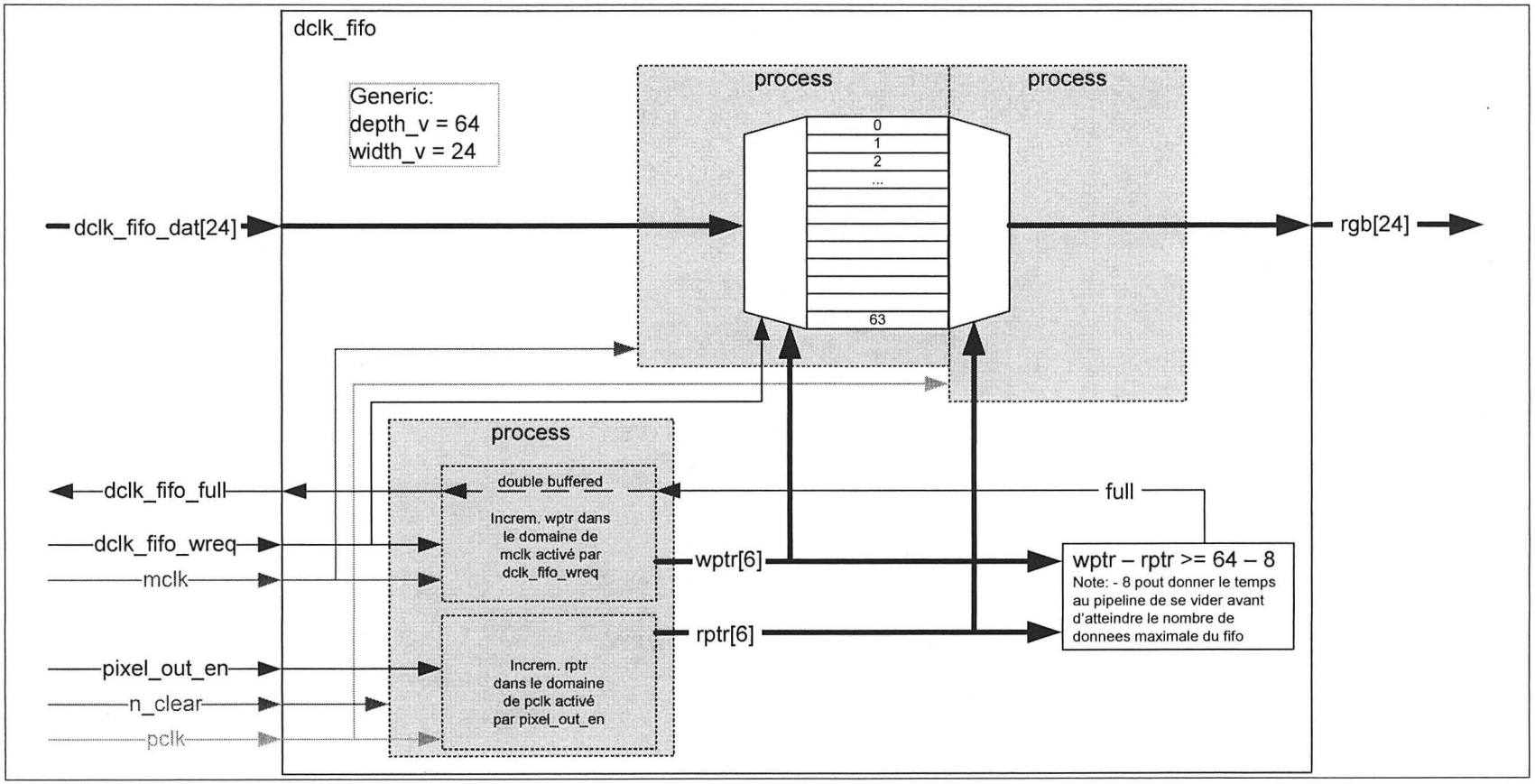

Figure 6.133 Schéma du module dclk_fifo.

Les deux processus du haut servent au contrôle des registres du fifo. Celui de gauche est synchrone sur l'horloge maîtresse mclk et permet l'écriture dans les registres et celui de droite est un multiplexeur synchrone sur l'horloge pixel pclk. Le processus du bas sur la figure précédente est celui qui permet de gérer les pointeurs de lecture et d'écriture dans les registres. Le pointeur d'écriture wptr est synchrone sur l'horloge maîtresse mclk et le pointeur de lecture rptr est synchrone sur l'horloge pixel pclk.

6.11.5.6.2 Résultats des simulations du module dclk_fifo

Voici la simulation fonctionnelle du module dclk_fifo :

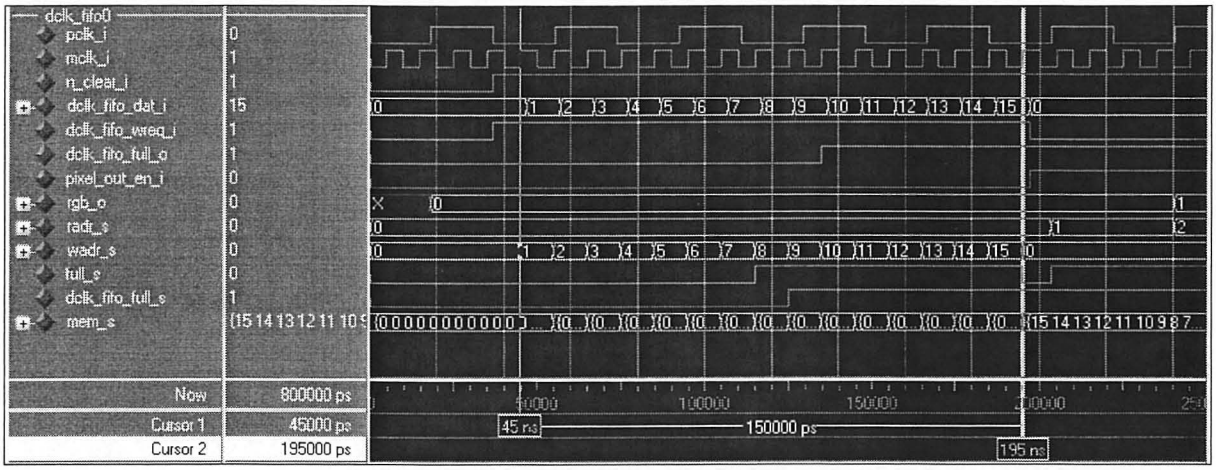

Figure 6.134 Simulation du remplissage du module dclk_fifo avec l'horloge d'entrée mclk.

La figure ci-haut montre que l'écriture dans le module dclk_fifo se fait de manière synchrone avec l'horloge maîtresse mclk.

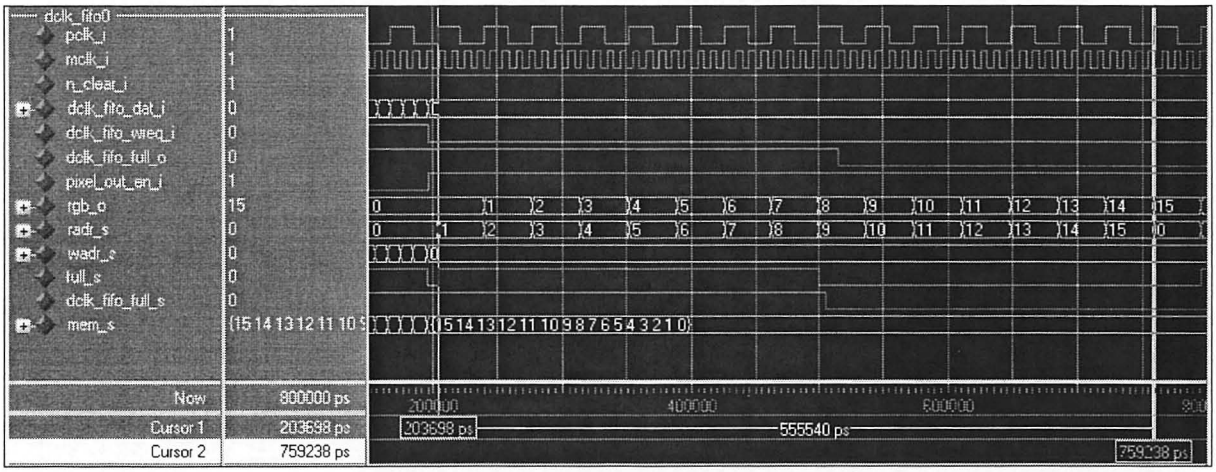

Figure 6.135 Simulation de la lecture dans le module dclk_fifo avec l'horloge de sortie pclk.

La figure ci-haut montre que la lecture dans le module dclk_fifo se fait de manière synchrone avec l'horloge pixel pclk. Le module dclk_fifo est en effet un ensemble de registres « first-in-first-out » avec deux horloges de synchronisation. Le module est codé de manière à ce que sa synthèse utilise un « block ram » dans le FPGA.

6.11.5.7 Description du module pixel_counter (pixel_counter.vhd)

Voici le symbole du module pixel_counter :

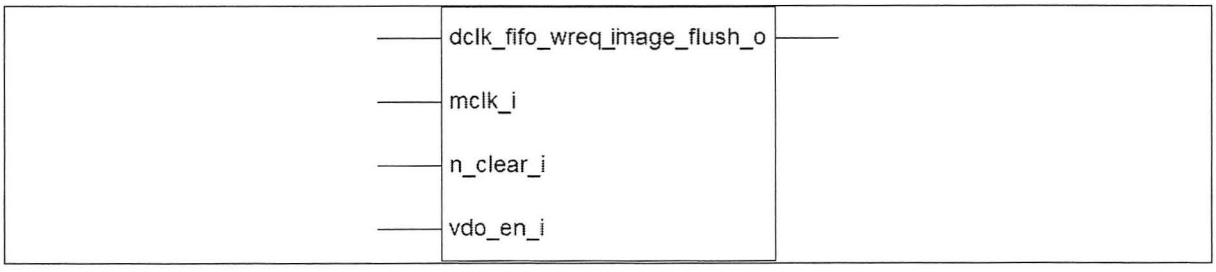

Figure 6.136 Symbole du module pixel_counter.

Avant de décrire le fonctionnement du module pixel_counter, voici un tableau de ses entrées et de ses sorties :

Tableau 6.32

Description des entrées et des sorties du module pixel_counter

Nom	Entrée/ Sortie	Description
mclk	E	Signal d'horloge maîtresse, voir les sections 6.7 et 6.7.4 pour plus de détails.
n_clear	E	Signal de réinitialisation générale du système. Voir la section 6.9 pour plus de détails.
vdo_en	E	Signal général qui indique si le contrôleur vidéo est activé ou non, ce signal provient du module ctrl_register et correspond au bit ctrl_reg(0).
dclk_fifo_wreq	E	Signal qui indique aux modules dclk_fifo et pixel_counter qu'un pixel valide est chargé dans le module dclk_fifo.
image_flush	S	Signal qui indique que tous les tampons doivent être vidés lorsque tous les pixels d'une image ont été transférés au module dclk_fifo. Ce signal est généré par le module pixel_counter.

6.11.5.7.1 Description du fonctionnement du module pixel_counter

L'unité pixel_counter est un double compteur qui permet d'identifier précisément le dernier pixel de l'image qui sera chargé dans le module dclk_fifo. Ceci permet de vider tous les registres intermédiaires qui précèdent le module dclk_fifo afin de recommencer une nouvelle image. Voici le schéma bloc du module pixel_counter :

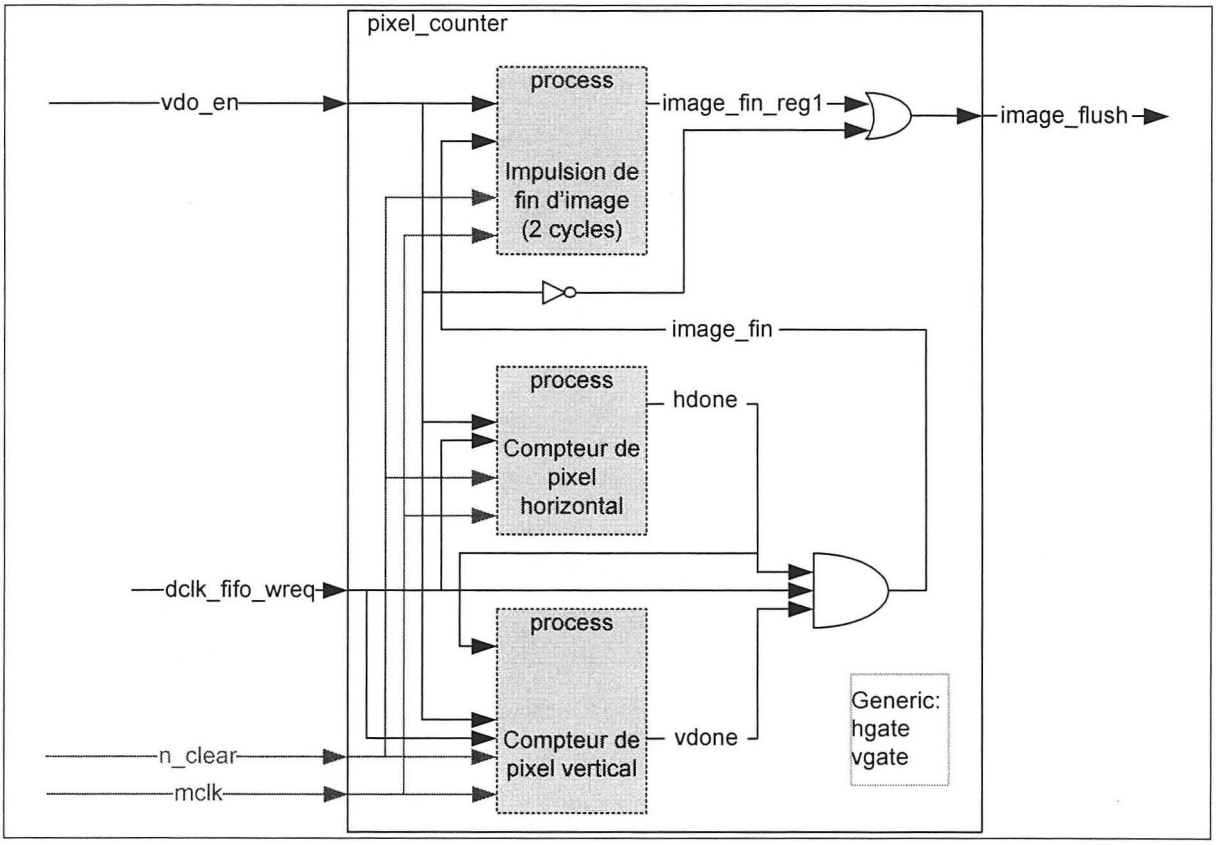

Figure 6.137 Schéma du module pixel_counter.

Le processus du haut sur l'image précédente permet de générer une impulsion de deux cycles d'horloge sur le signal image_flush pour s'assurer de tenir compte du pipeline de la mémoire vidéo. Ce signal permet de remettre à zéro toute la chaine de registres nécessaire au traitement et à la mise en forme des pixels avant l'affichage.

Les deux processus du bas permettent de générer les compteurs de pixels des deux axes, vertical et horizontal, et de générer un signal à la fin de l'affichage de chacun des axes. Lorsque les deux axes sont terminés à la fois, c'est qu'on a terminé l'image.

6.11.5.7.2 Résultats des simulations du module pixel_counter

Voici la simulation fonctionnelle du module pixel_counter :

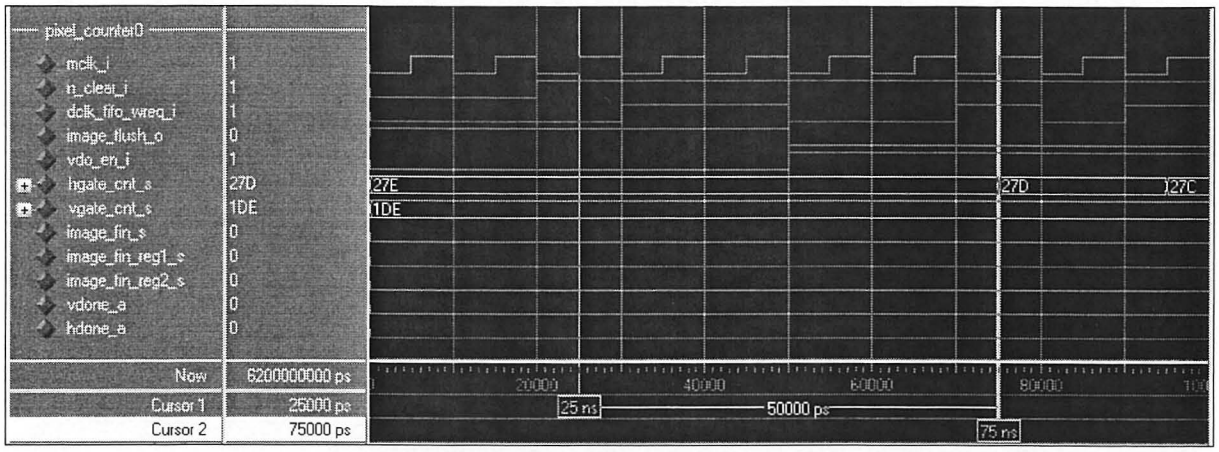

Figure 6.138 Simulation de l'initialisation du module pixel_counter.

La figure ci-haut montre l'initialisation des deux compteurs, horizontal et vertical, lors de l'activation du signal n_clear. Au curseur numéro deux, le signal vdo_en s'active, alors les compteurs, horizontal et vertical, se mettent en marche. Sur cette image on ne voit que le compteur horizontal fonctionner, mais si vous vous référez à l'image suivante, on voit le compteur vertical décrémenter au curseur numéro deux, soit à chaque fois que le compteur horizontal termine un compte complet.

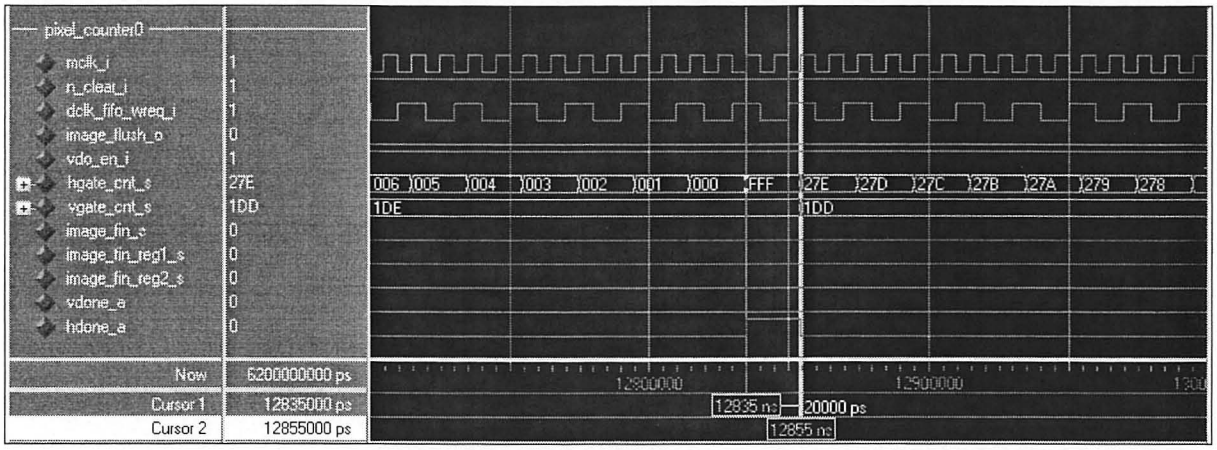

Figure 6.139 Simulation du décomptage vertical et de la fin d'un compte horizontal par le module pixel_counter.

A chaque fin de compte de un ou l'autre des compteurs, un signal done est généré, ce qui permet d'identifier la fin de l'affichage ou plutôt la fin de la lecture de l'image en mémoire vidéo.

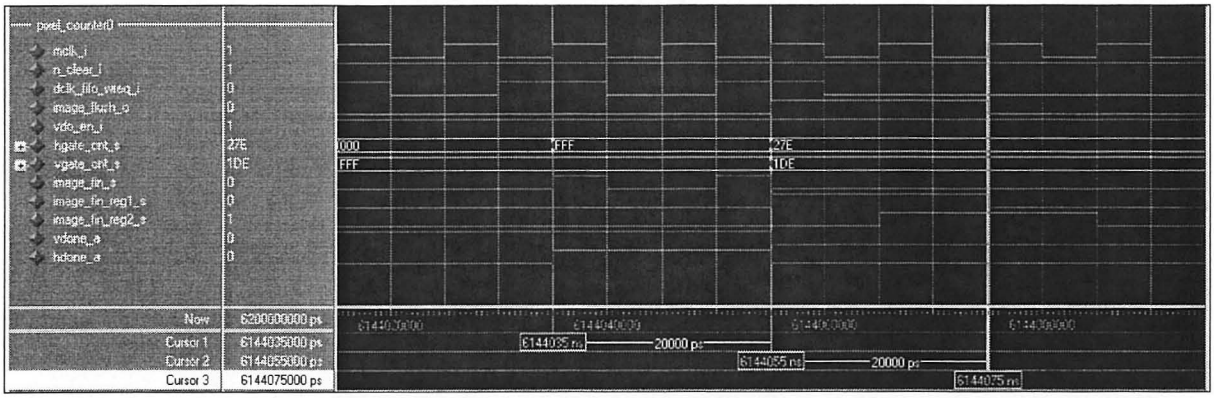

Figure 6.140 Simulation de la fin des deux compteurs avec génération du signal image_flush par le module pixel_counter.

En effet, le curseur numéro un de la Figure 6.140 montre que les signaux vdone et hdone sont simultanément actifs. C'est donc que tous les pixels ont été chargés de la mémoire vidéo. Le signal image_flush est donc généré avec l'activation d'un troisième signal simultané : le signal dclk_fifo_wreq. Ce dernier valide la mémorisation du dernier pixel dans le module dclk_fifo. Les pixels subséquents qui ont été mémorisés dans la chaine de registres (modules fifo et pixel_precessor) peuvent donc être réinitialisés par le signal image_flush afin de permettre un nouvel affichage.

6.12 Conclusion

Cette section termine le chapitre le plus technique de ce document. C'est dans ce dernier que l'on fait les différents liens entre les modules du système. La hyérarchie décrite à la section 6.1 est rigoureusement respectée afin de facilité la recherche d'information à travers le document. Ce chapitre explique dans le détail comment interragissent les signaux pour donner le fonctionnement escompté. L'explication théorique du code liée à une description du fonctionnement par le biais de chronogrammes fonctionnels et de shémas permet de mettre en lumière les détails de fonctionnalité, autant au niveau temporel que logique. Le CHAPITRE 6 permet donc de décrire correctement le module bootloader et le module vga qui sont les pièces maitresses du système d'AVN.

Le module bootloader permet de faire la préparation des données mémoire avant la mise en marche de l'affichage. Il est assez complèxe, mais permet de charger facilement les données d'image en mémoire vidéo et aussi de charger la banque de couleurs lors d'un affichage d'images en mode 8 bits. Il peut être remplacé par un module incluant un processeur puisque l'interface de communication avec le module vga est implémentée par un bus wishbone.

Le module vga est contrôlé par quelques commandes simples qui permettent par exemple : l'activation ou la désactivation de l'affichage, le changement de la banque d'image active, etc. Lorsque le module vga est en fonction d'affichage, il charge lui-même ses données d'image de la mémoire vidéo et achemine les données pixel une à la suite de l'autre et au bon moment afin de créer le balayage de couleur adéquat sur l'écran d'affichage.

Ces deux modules important font partie d'un module qui les englobent et qui les connectent. Ce module, nommé top (voir section 6.5) est la section VHDL en entier du projet d'AVN. C'est d'ailleur à partir de ce module que tous les autres découlent et c'est spécifiquement dans le but de comprendre son fonctionnement totalement que ce chapitre à été composé.

CHAPITRE 7

EXPLICATION DES ÉTAPES DE MISE EN MARCHE DU CONTRÔLEUR VIDÉO NUMÉRIQUE À PARTIR DES FICHIERS SOURCE ET VÉRIFICATION DU FONCTIONNEMENT PRATIQUE À L'AIDE D'UN ÉCRAN LCD

Ce chapitre est consacré à l'explication de toutes les étapes nécessaires de la création du projet dans le logiciel ISE 8.2i jusqu'au fonctionnement final du contrôleur vidéo numérique dans un FPGA.

7.1 Création du projet dans le logiciel ISE 8.2i de Xilinx

Pour créer un nouveau projet dans le logiciel ISE 8.2i de Xilinx, il suffit de démarrer le logiciel et de dérouler le menu « FILE » puis choisir « New Project... ». La fenêtre de la figure suivante apparaît.

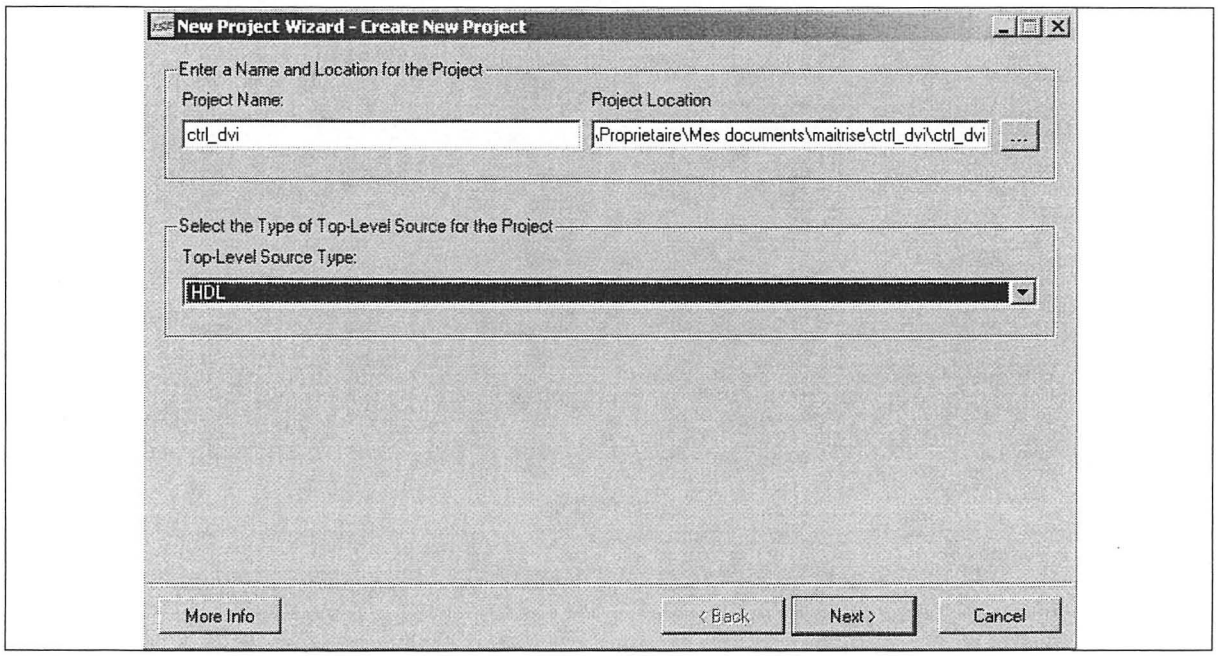

Figure 7.1 Fenêtre de création d'un nouveau projet dans le logiciel ISE 8.2i.

Cette fenêtre comprend trois champs importants à remplir correctement. Le champ « Project Name » contient le nom du projet à créer, dans ce cas-ci, on le nomme « ctrl_dvi ». Le

champ « Project Location » contient le chemin d'accès qui comprend tout le dossier de projet. Le champ « Top Level Source Type » doit pointer sur l'option « HDL » puisque tout le projet est écrit en langage VHDL. Une fois tous les champs correctement remplis, appuyez sur le bouton « Next ».

La fenêtre suivante apparaît :

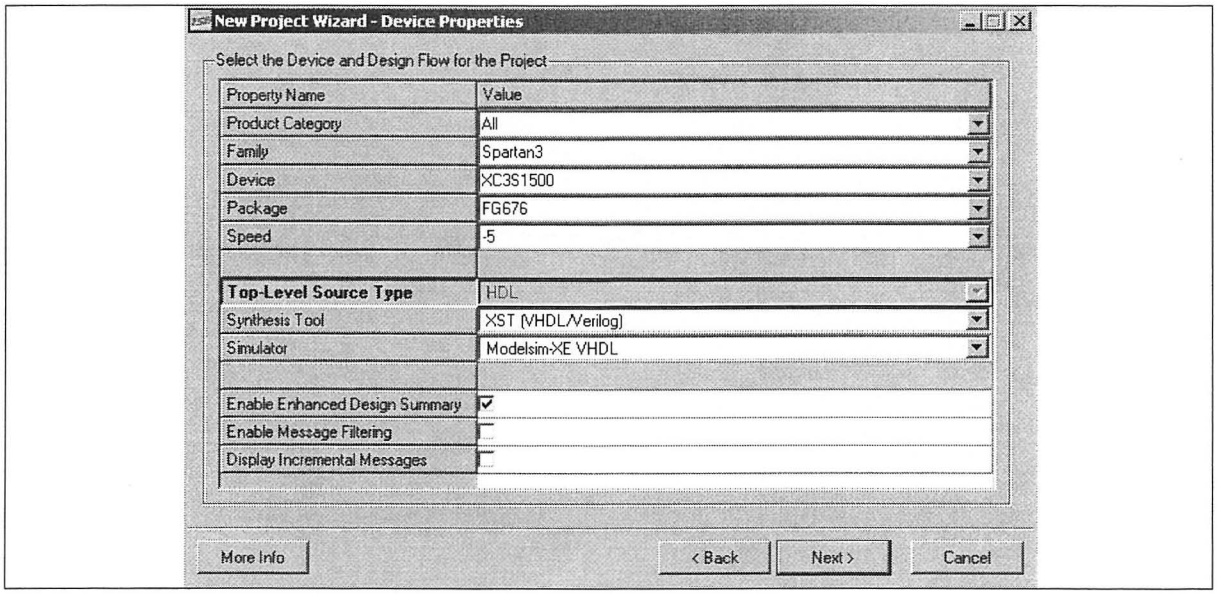

Figure 7.2 Fenêtre de sélection de la technologie utilisée lors de la création d'un nouveau projet dans le logiciel ISE 8.2i.

Afin d'utiliser la plateforme de développement SP305 de Xilinx sans problème, vous devez faire les sélections telles qu'elles sont présentées à la Figure 7.2 et appuyer sur le bouton « Next ». Pour les fenêtres subséquentes, appuyer sur les boutons « Next », « Next » et « Finish » consécutivement pour terminer la création du projet. Nous n'ajoutons pas les sources par cette interface puisque nous voulons les ajouter au préalable dans le dossier de projet.

Placez le dossier « RTL » contenant les fichiers sources VHDL dans le dossier de projet. Une fois cette étape terminée, retournez dans le logiciel ISE 8.2i de Xilinx et double-cliquez sur la commande « Add Existing Source » dans la zone des « Processes ». Une fenêtre de

dialogue vous permet de sélectionner toutes les sources nécessaires au projet. Les deux images qui suivent permettent d'identifier les fichiers sources à ajouter.

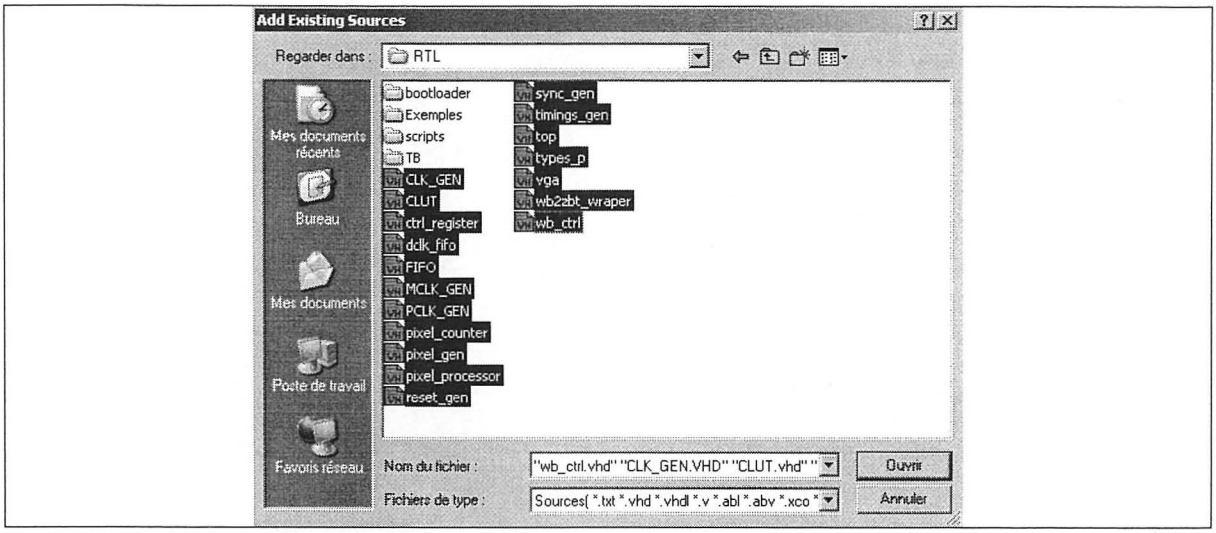

Figure 7.3 Fenêtre de dialogue pour l'ajout des fichiers source du contrôleur VGA dans un projet du logiciel ISE 8.2i.

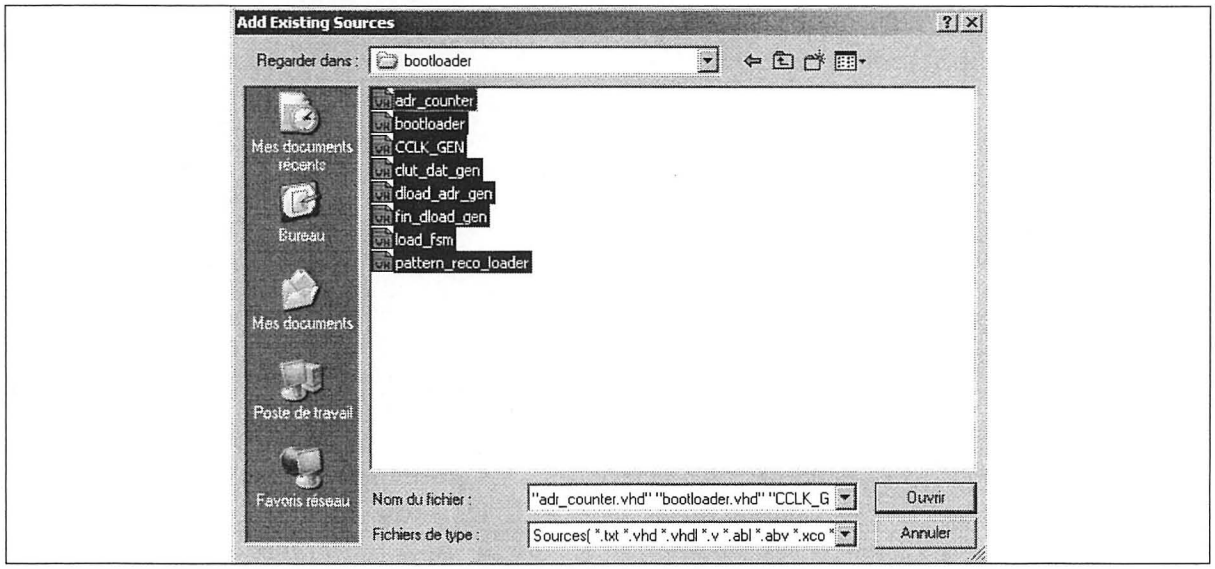

Figure 7.4 Fenêtre de dialogue pour l'ajout des fichiers source du module bootloader dans un projet du logiciel ISE 8.2i.

Une fois la sélection des fichiers effectuée, appuyez sur le bouton « Ouvrir ». Cela vous amènera à une fenêtre de dialogue de ce genre :

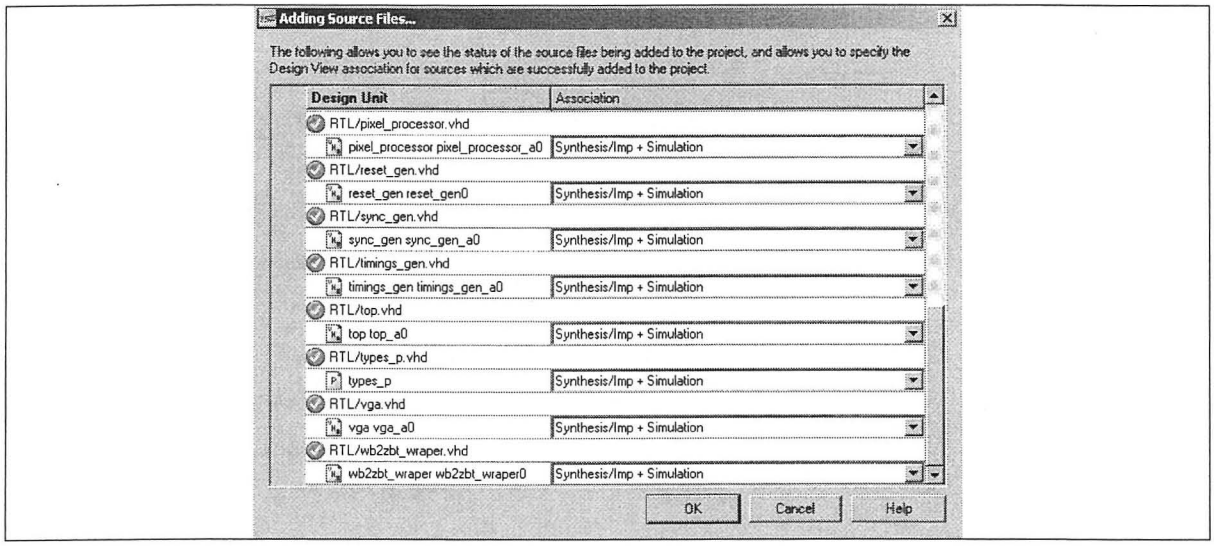

Figure 7.5 Fenêtre de dialogue pour le choix du type de module utilisé spécifique à chaque fichier source ajouté au projet du logiciel ISE 8.2i.

Normalement, Le logiciel détecte automatiquement qu'un module est un banc d'essai, une librairie ou un module descriptif de fonctionnement. Il suffit donc d'appuyer sur le bouton « OK » pour terminer l'ajout des fichiers source. Si vous devez ajouter des fichiers qui ne font pas partie de la synthèse, comme dans le cas de certains fichiers faisant partie d'un banc d'essai par exemple, vous devez changer l'association pour « Simulation Only ». Lorsque tous les fichiers source nécessaires à l'implémentation du FPGA ont été ajoutés, vous pouvez passer à l'étape suivante.

7.2 Configuration, fichiers de contraintes et synthèse du projet

Comme première configuration, nous utilisons les configurations par défaut du logiciel, sauf pour une chose : la hiérarchie des modules doit être maintenue. Pour cela, vous devez sélectionner le menu properties du menu contextuel au-dessus de l'icône « synthesize - XST » Comme montré à l'image qui suit.

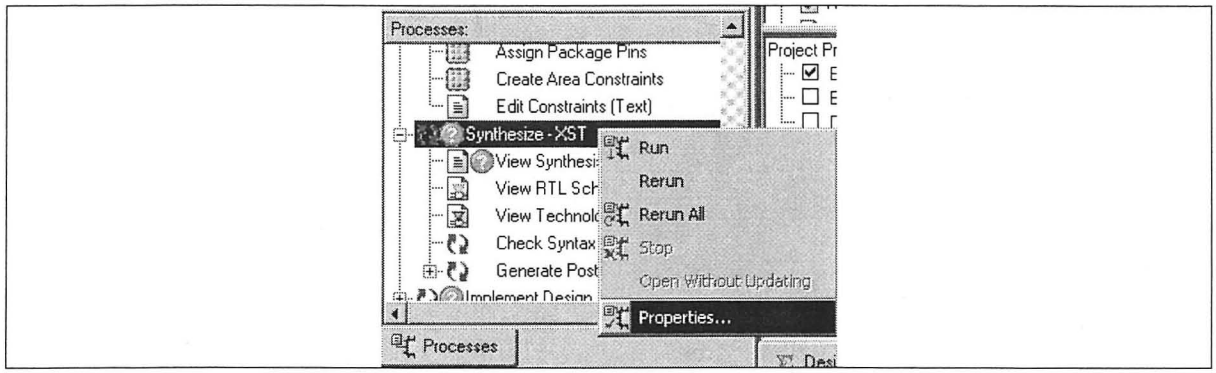

Figure 7.6 Menu contextuel des propriétés de synthèse du logiciel ISE 8.2i.

Ceci vous fait apparaître la fenêtre des propriétés de synthèse. Assurez-vous que l'option « keep hierarchie » est à « Yes » dans le niveau d'affichage « Advanced » comme démontré dans la figure ci-bas :

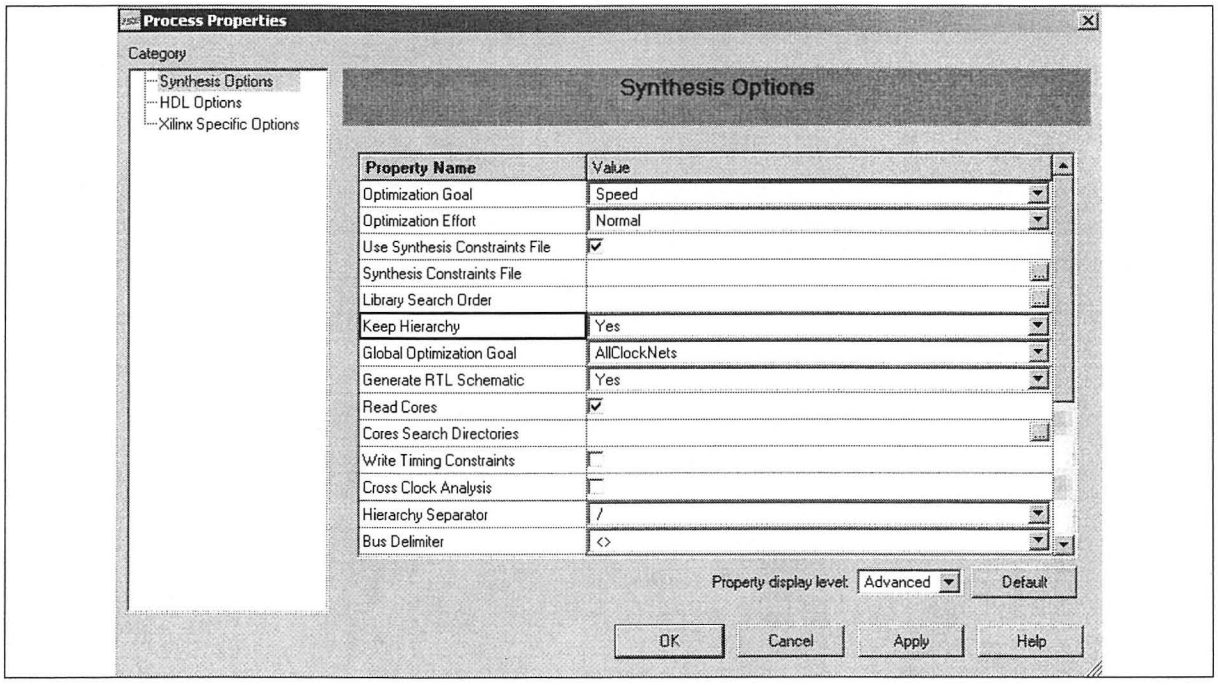

Figure 7.7 Fenêtre de dialogue des propriétés de synthèse du logiciel ISE 8.2i.

De plus, un fichier de contraintes nommé top.ucf doit être ajouté au projet pour que celui-ci fonctionne correctement. Deux solutions s'offrent à vous pour ce fichier de contraintes. La première solution est d'insérer le fichier qui se trouve dans le dossier du projet déjà fonctionnel de ce contrôleur vidéo. L'autre option est d'en créer un à partir du début. Les

contraintes d'association des numéros de broches de la composante FPGA avec les signaux d'entrée et de sortie du système sont déjà incorporées dans le fichier top.vhd. Tout ce qu'il reste à faire est de contraindre le placement des modules à l'intérieur du FPGA. Pour ce faire, double-cliquez sur « Create Area Constraints » de la section « Processes » comme montré sur l'image suivante.

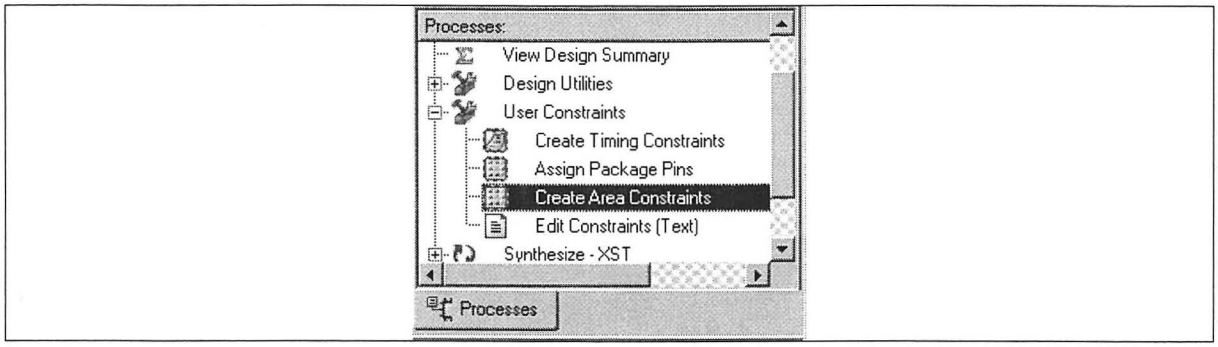

Figure 7.8 Section Processes avec l'icône Create Area Constraints sélectionné dans le logiciel ISE 8.2i.

Suite à cette action, la fenêtre du logiciel PACE apparaît une fois que la synthèse du code a été faite (il est à noter que la synthèse doit être faite en gardant la hiérarchie des modules). Vous devez alors placer les composantes du projet de manière à les garder les plus rapprochées possible tout en les gardant aussi très proches des leurs entrées et sorties respectives Ceci donne dans notre cas un placement qui ressemble à celui de la fenêtre qui suit :

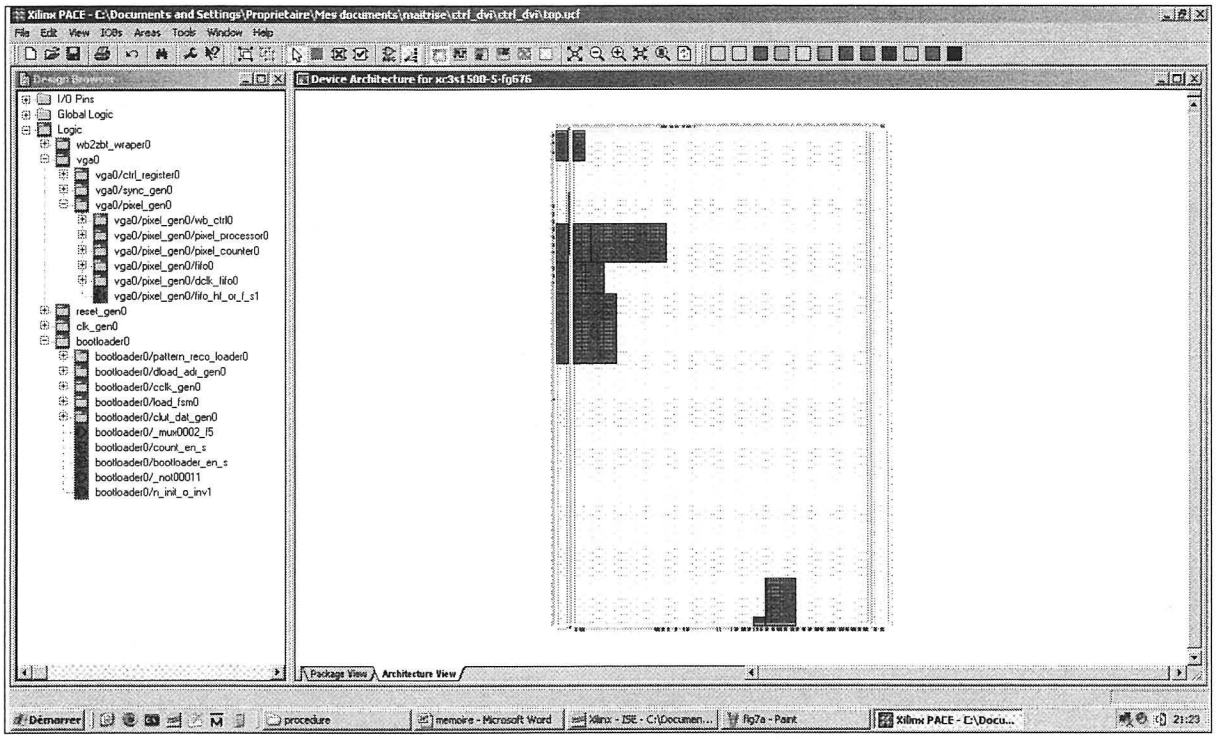

Figure 7.9 Placement des composantes du projet à l'aide du logiciel PACE.

A chaque fois que vous effectuez une sauvegarde ou que vous quittez PACE, le fichier top.ucf est remis à jour avec les nouvelles contraintes de placement des composantes. Dans le cas de la Figure 7.9, les contraintes générées sont les suivantes :

```
#PACE: Start of PACE Area Constraints
INST "clk_gen0/mclk_gen0/dcm0" LOC = "DCM_X0Y0";
INST "clk_gen0/pclk_gen0/dcm0" LOC = "DCM_X0Y1";
INST "vga0/pixel_gen0/dclk_fifo0/Mram_mem_s1" LOC = "RAMB16_X0Y13";
INST "vga0/pixel_gen0/pixel_processor0/clut0/Mram_clut_s1" LOC = "RAMB16_X0Y12";

AREA_GROUP "AG_bootloader0" RANGE = SLICE_X0Y85:SLICE_X17Y68;
INST "bootloader0" AREA_GROUP = "AG_bootloader0";
AREA_GROUP "AG_vga0/ctrl_register0" RANGE = SLICE_X0Y93:SLICE_X7Y86;
INST "vga0/ctrl_register0" AREA_GROUP = "AG_vga0/ctrl_register0";
INST "vga0/pixel_gen0/dclk_fifo0" AREA_GROUP = "AG_vga0/pixel_gen0/dclk_fifo0";
AREA_GROUP "AG_vga0/pixel_gen0/fifo0" RANGE = SLICE_X10Y103:SLICE_X33Y94;
INST "vga0/pixel_gen0/fifo0" AREA_GROUP = "AG_vga0/pixel_gen0/fifo0";
AREA_GROUP "AG_vga0/pixel_gen0/pixel_counter0" RANGE = SLICE_X8Y93:SLICE_X13Y86;
INST "vga0/pixel_gen0/pixel_counter0" AREA_GROUP = "AG_vga0/pixel_gen0/pixel_counter0";
AREA_GROUP "AG_vga0/pixel_gen0/pixel_processor0" RANGE = SLICE_X0Y103:SLICE_X9Y94;
INST "vga0/pixel_gen0/pixel_processor0" AREA_GROUP = "AG_vga0/pixel_gen0/pixel_processor0";
AREA_GROUP "AG_vga0/pixel_gen0/wb_ctrl0" RANGE = SLICE_X66Y11:SLICE_X75Y0;
```

```
INST "vga0/pixel_gen0/wb_ctrl0" AREA_GROUP = "AG_vga0/pixel_gen0/wb_ctrl0";
AREA_GROUP "AG_vga0/sync_gen0" RANGE = SLICE_X0Y127:SLICE_X7Y120;
INST "vga0/sync_gen0" AREA_GROUP = "AG_vga0/sync_gen0";
AREA_GROUP "AG_wb2zbt_wraper0" RANGE = SLICE_X62Y1:SLICE_X65Y0;
INST "wb2zbt_wraper0" AREA_GROUP = "AG_wb2zbt_wraper0";
```

Vous pouvez aussi utiliser ce placement en copiant ces contraintes directement dans le fichier top.ucf.

Une fois le placement des modules terminé, relancez la synthèse du code VHDL en sélectionnant la commande « Rerun » du menu contextuel au-dessus de l'icône « Synthesize – XST » comme sur la figure qui suit.

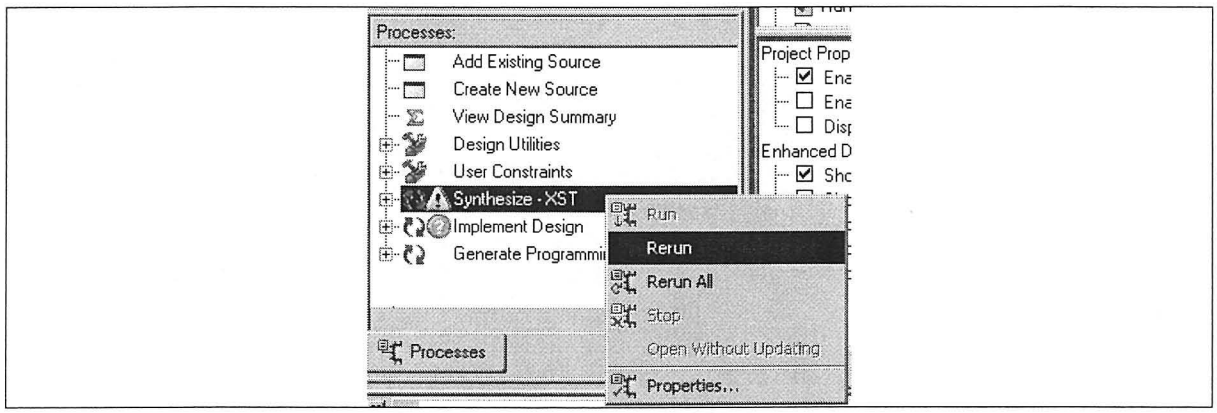

Figure 7.10 Menu contextuel pour la commande Rerun de la synthèse du logiciel ISE 8.2i.

Si vous êtes prêts à programmer le FPGA, vous pouvez passer directement à la section suivante sans activer la commande « Rerun » puisque celle-ci se fera automatiquement avec cette étape.

7.3 Placement et routage et création du fichier *.bit

Vous pouvez maintenant lancer le placement et le routage du circuit interne du FPGA ainsi que la génération du fichier de programmation *.bit. Pour effectuer ces opérations, vous n'avez qu'à double-cliquer sur l'icône « Generate Programming File » de la section « Processes » comme montré sur la figure suivante.

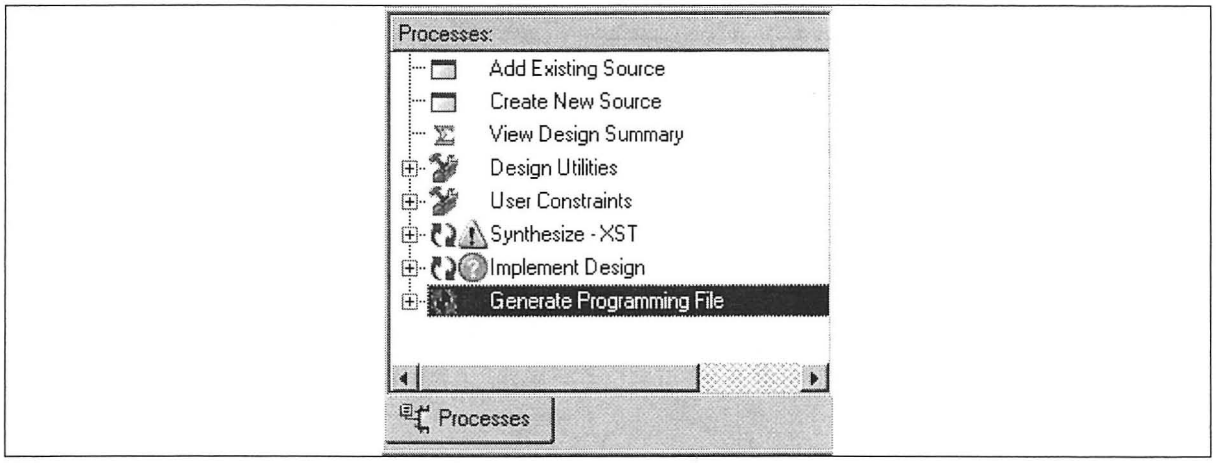

Figure 7.11 Double-clique pour générer le fichier *.bit avec le logiciel ISE 8.2i.

Le processus prend quelques minutes selon votre configuration matérielle informatique. Une fois le processus terminé, le fichier top.bit est généré directement dans le dossier de projet /ctrl_dvi/.

7.4 Génération du fichier *.mcs comprenant les données de configuration de base du FPGA

Afin d'utiliser le contrôleur vidéo, il est nécessaire de charger des images dans la mémoire vidéo. Pour faire cela, nous avons opté pour une solution simple et efficace qui nous permet de stocker les images directement à la suite de la trame de configuration du FPGA dans la mémoire de configuration PROM de la plateforme de développement. Pour faire cela, plusieurs manipulations sont présentement nécessaires, mais il sera possible dans le future d'automatiser ces étapes à l'aide d'un script. Ces étapes sont décrites dans les sections 7.4 (la présente section) et 7.5.

La première étape est de générer le fichier *.MCS de départ qui contient l'équivalent du fichier *.bit grâce à l'utilisation du logiciel Impact de Xilinx. Pour lancer Impact, déroulez la poignée marquée d'un « + » vis-à-vis l'icône « Generate Programming File » et double-cliquez sur l'icône « Configure Device (iMPACT) » tel que montré sur la figure suivante.

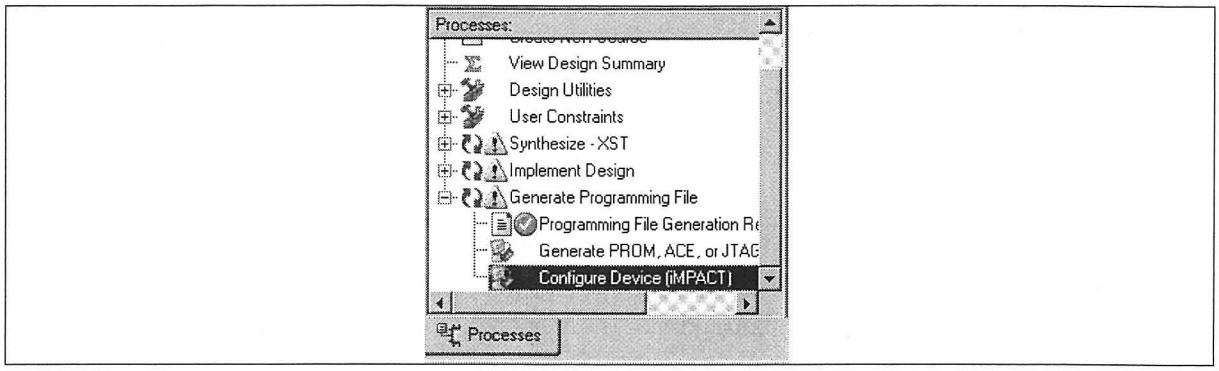

Figure 7.12 Double-clique pour lancer iMPACT avec le logiciel ISE 8.2i.

Le logiciel iMPACT est chargé et une fenêtre de départ apparaît si c'est la première fois que vous lancez iMPACT pour ce projet.

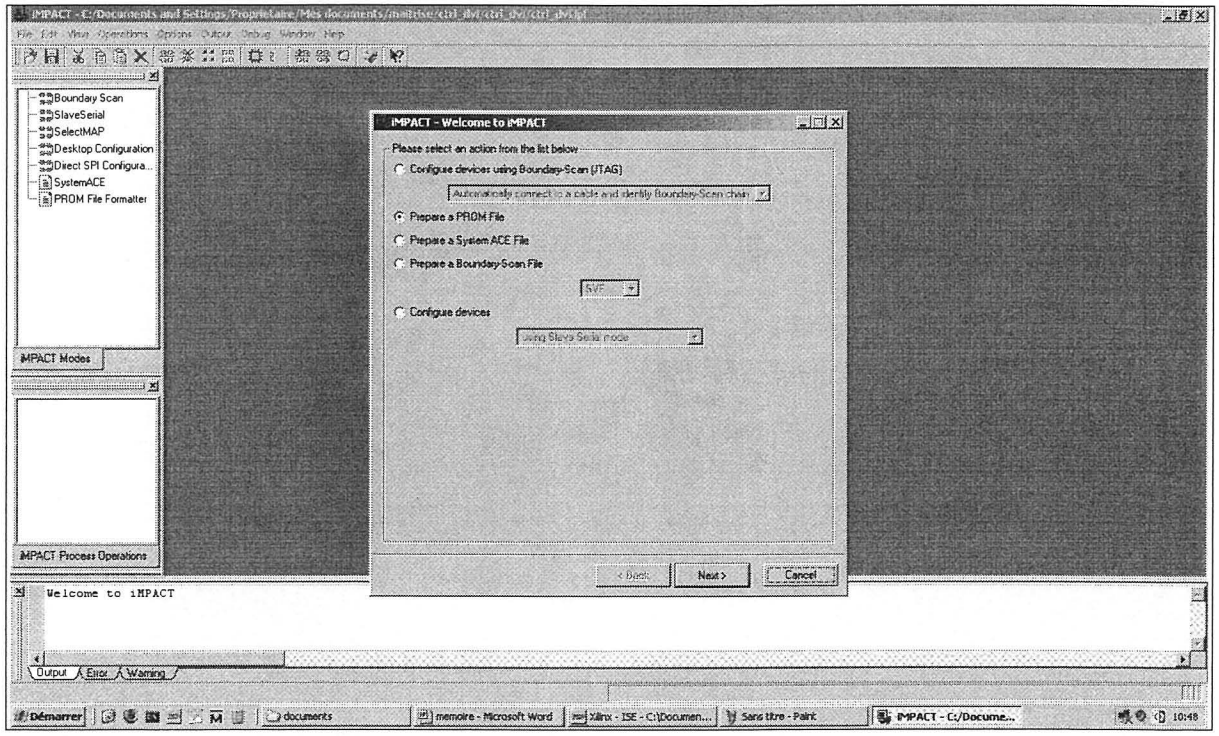

Figure 7.13 Fenêtre de départ du logiciel iMPACT.

La Figure 7.13 présente cette première fenêtre du logiciel iMPACT. Sélectionnez le bouton radio « Prepare a PROM File » et appuyez sur le bouton « Next ». Vous avez alors la fenêtre suivante qui est la fenêtre de préparation pour générer le fichier *.mcs

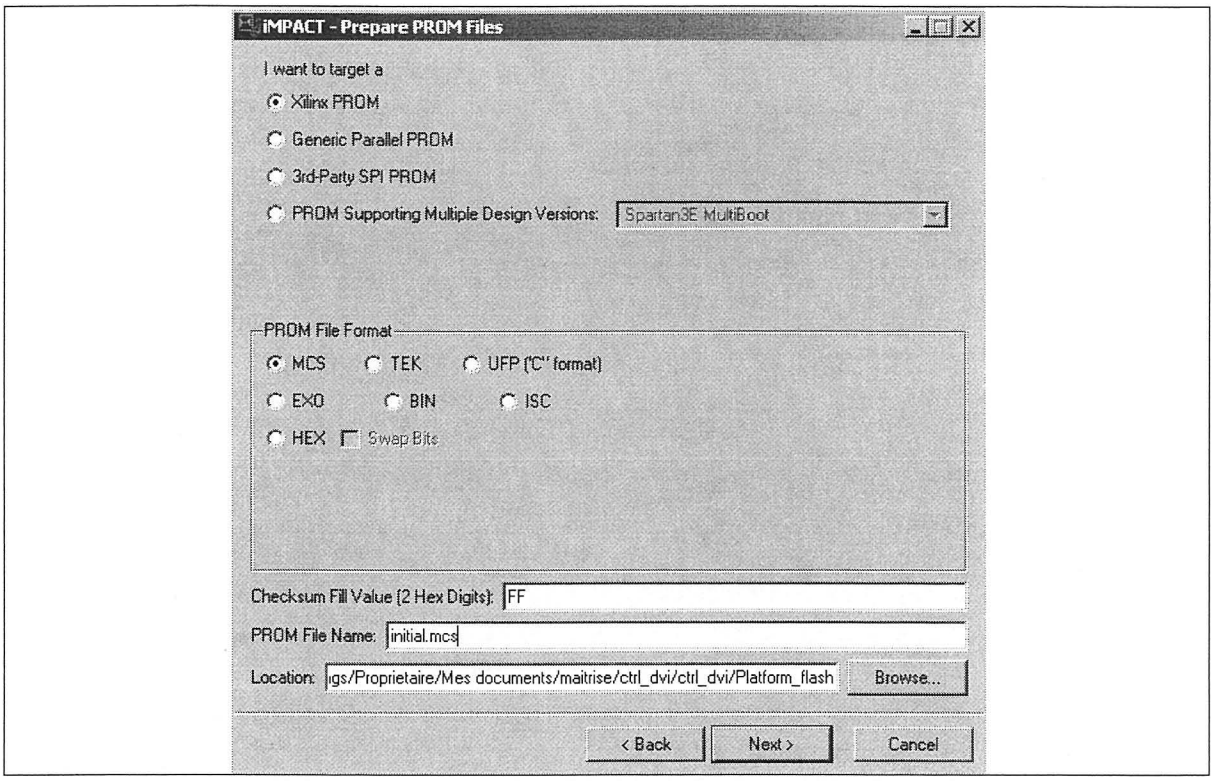

Figure 7.14 Fenêtre de préparation d'un fichier *.mcs du logiciel iMPACT.

La mémoire PROM utilisée sur la plateforme de développement est une composante fabriquée par la compagnie Xilinx. Donc, le bouton radio « Xilinx PROM » doit être sélectionné. C'est d'ailleurs celui qui est sélectionné par défaut.

Dans l'encadré « PROM File Format », le bouton radio MCS doit être sélectionné (il l'est aussi par défaut). Vous devez ensuite nommer le fichier *.mcs dans le champ « PROM File Name ». Dans notre cas, nous l'avons nommé initial.mcs pour identifier que c'est le fichier de base. Enfin, on doit identifier l'emplacement de création du fichier. Nous avons donc placé un dossier nommé « Platform_flash » à l'intérieur du dossier de projet afin d'y sauvegarder tous les fichiers relatifs à la mémoire PROM. Le chemin d'accès pour ce dossier peut être écrit à la main ou en appuyant sur le bouton « browse » puis en allant le chercher dans l'arborescence du disque dur. Une fois tous les champs remplis adéquatement, vous pouvez appuyer sur le bouton « Next » qui vous amènera à la figure suivante.

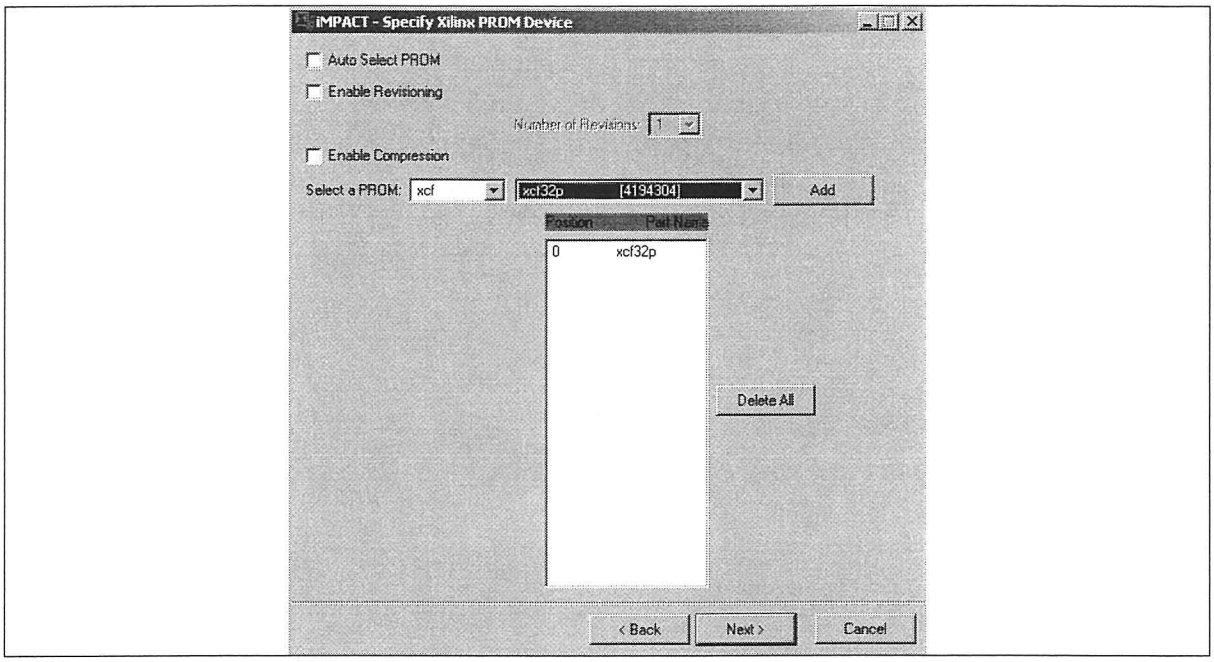

Figure 7.15 Fenêtre de sélection du type de mémoire PROM du logiciel iMPACT.

La fenêtre de dialogue de la Figure 7.15 vous permet de sélectionner la composante « xcf32p » et de l'ajouter dans la liste de mémoire à programmer en appuyant sur le bouton « Add ». Comme c'est la seule mémoire PROM de la carte de développement, c'est aussi la seule composante à ajouter. Vous pouvez faire « Next » par la suite, ce qui vous donne une dernière fenêtre de dialogue faisant état du sommaire de la génération du fichier *.mcs. Vous pouvez appuyer sur le bouton « Finish » pour en arriver à la fenêtre de la Figure 7.16 qui vous permet maintenant d'ajouter le fichier de configuration nécessaires pour le FPGA à convertir en fichier *.mcs.

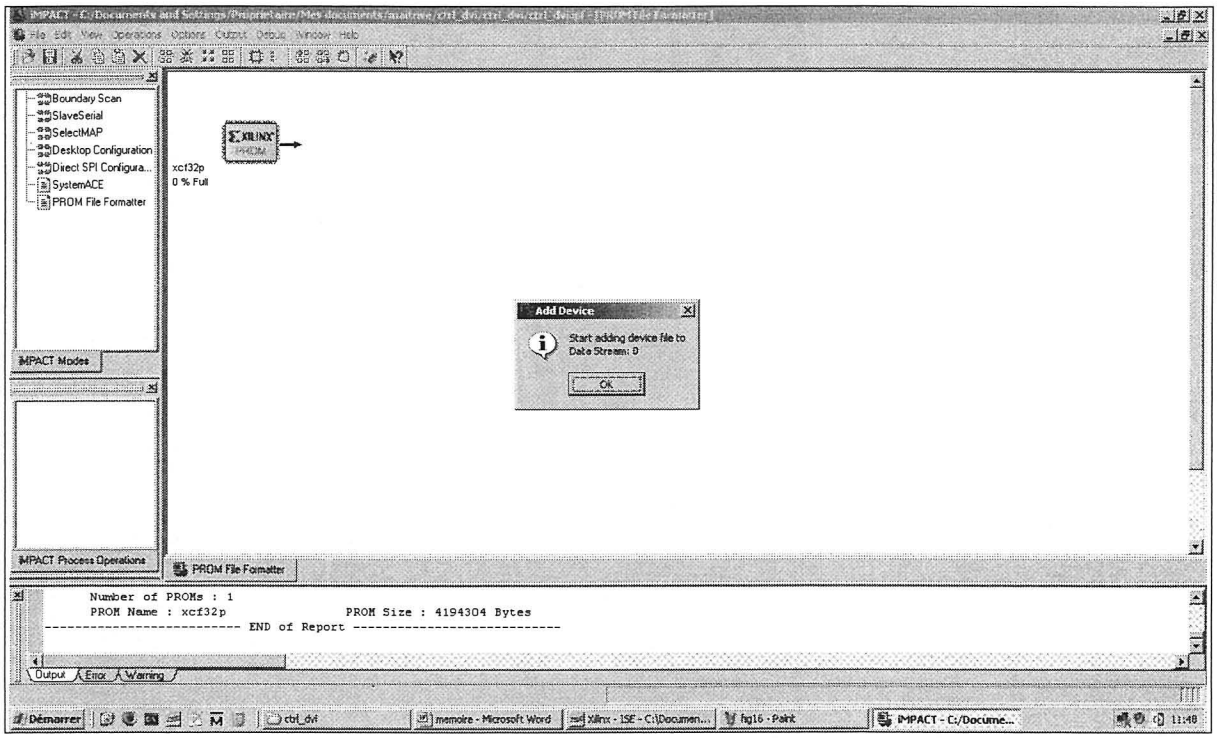

Figure 7.16 Fenêtre de sélection des fichiers de configuration du FPGA du logiciel iMPACT.

Appuyez sur le bouton « OK » dans la fenêtre « Add Device » et choisissez le fichier top.bit, qui a été généré auparavant à la section 7.3, grâce à la fenêtre de dialogue de l'image suivante. Appuyez sur le bouton « Ouvrir » pour valider la sélection.

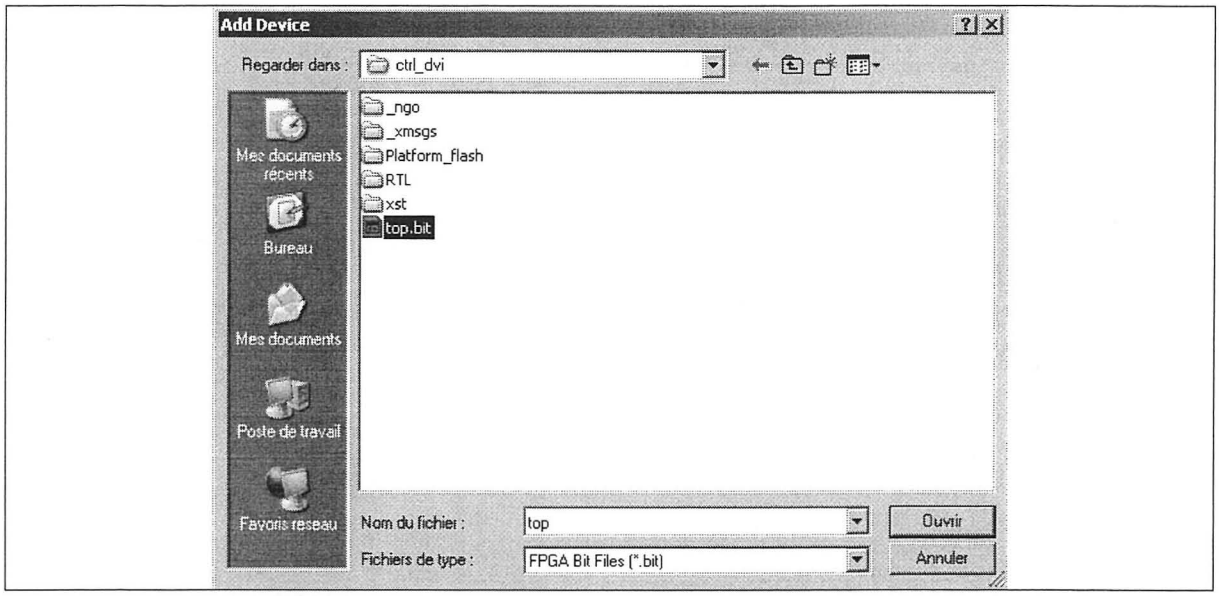

Figure 7.17 Fenêtre de sélection du fichier *.bit avec le logiciel iMPACT.

Une fenêtre de dialogue (Figure 7.18) vous demandant si vous voulez ajouter un autre fichier de configuration apparaît. Faite « No », ce qui vous donnera la Figure 7.19 qui vous montre le fichier top.bit comme faisant partie de la mémoire PROM xcf32p et prenant 15.54% de l'espace de celle-ci. Sélectionnez l'icône de la PROM pour qu'elle devienne de couleur verte et lancez la génération du fichier initial.mcs en double-cliquant sur la commande « Generate File… » dans la section iMPACT Process Operations.

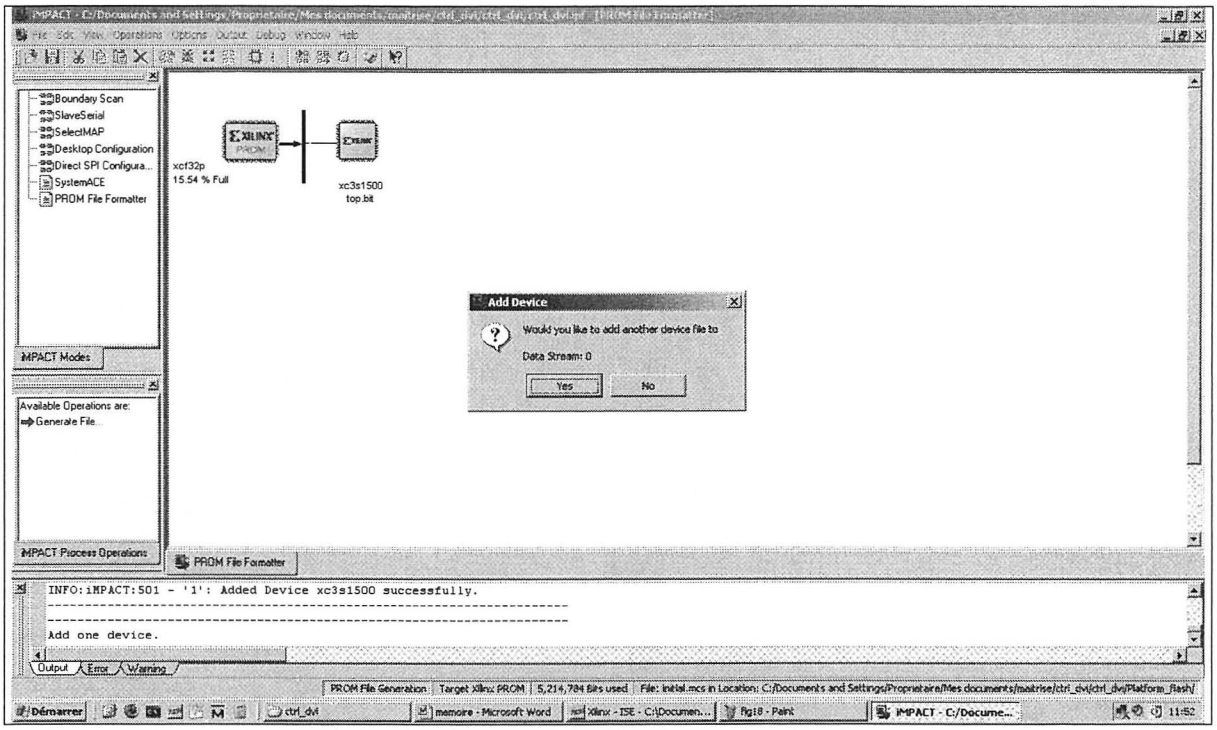

Figure 7.18 Fenêtre du logiciel iMPACT.

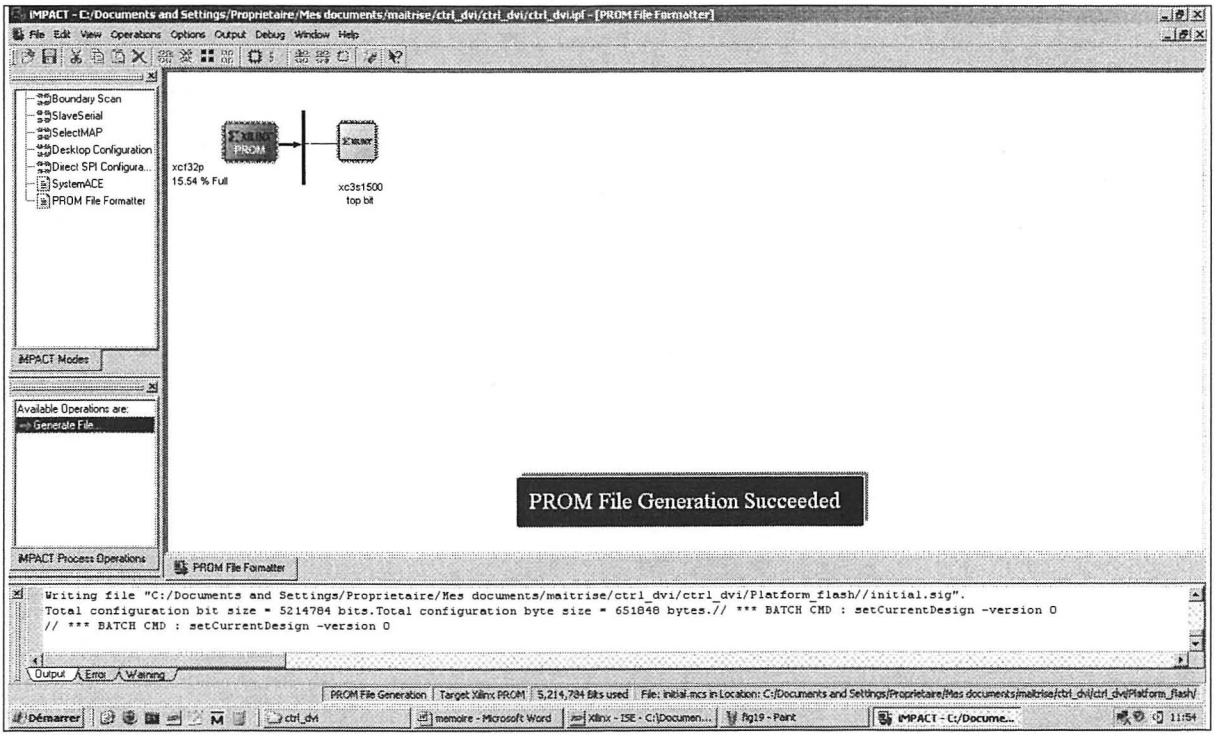

Figure 7.19 Fenêtre du logiciel iMPACT.

Lorsque le fichier est généré, un message vous l'annonce en bleu, comme montré sur l'image précédente. Vous êtes donc prêt à passer à l'étape suivante qui est d'ajouter les données d'image à ce fichier *.mcs.

7.5 Exécution des scripts pour insérer les données d'images dans le fichier *.mcs

Le fichier *.mcs est un fichier texte, ce qui permet de le traiter facilement à l'aide de scripts écrit en langage PERL. Le PERL a, quand à lui, l'avantage d'être installé par défaut avec la suite de logiciels ISE et son utilisation n'occasionne aucune manipulation supplémentaire. Vous trouverez les détails sur le fonctionnement des scripts aux sections 5.2 et 7.5.

Le premier script à exécuter est celui qui permet de convertir les images de format *.bmp en format texte. Vous devez modifier le script pour lui donner le bon fichier *.bmp à convertir. Ensuite, il suffit de lancer le script pour qu'il génère un fichier texte. Pour cela, vous devez vous placer dans le répertoire « Plateform_flash » à l'intérieur du projet et ensuite vous devez lancer la commande « xilperl bmp2txt.pl ». Pour le moment, chaque image doit être convertie de manière indépendante. Éventuellement, il sera possible d'écrire un script qui place plusieurs images simultanément dans un même fichier texte. La figure suivante montre un terminal DOS avec la dernière commande prête à s'effectuer.

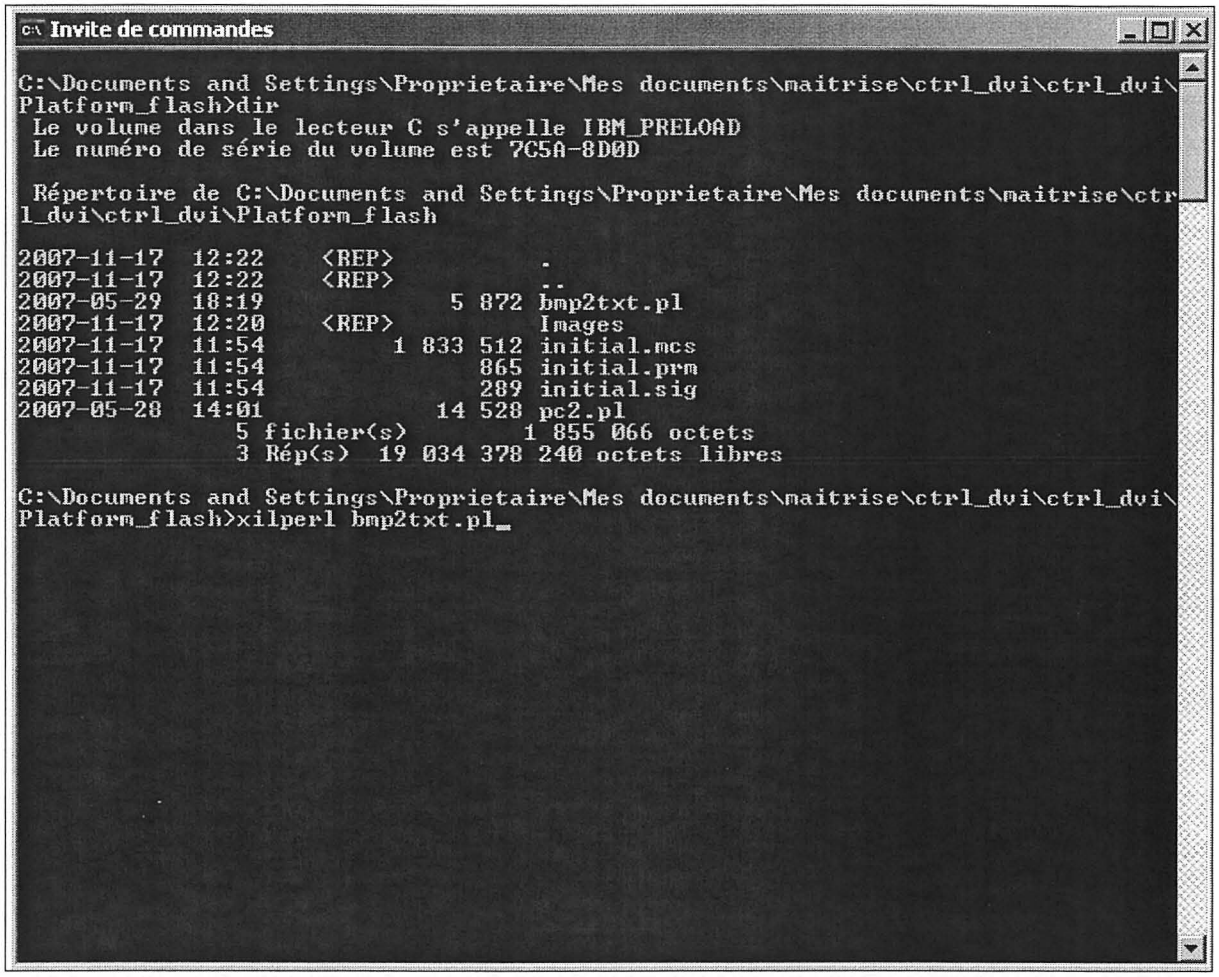

Figure 7.20 Fenêtre terminal DOS avec la commande xilperl bmp2txt.pl.

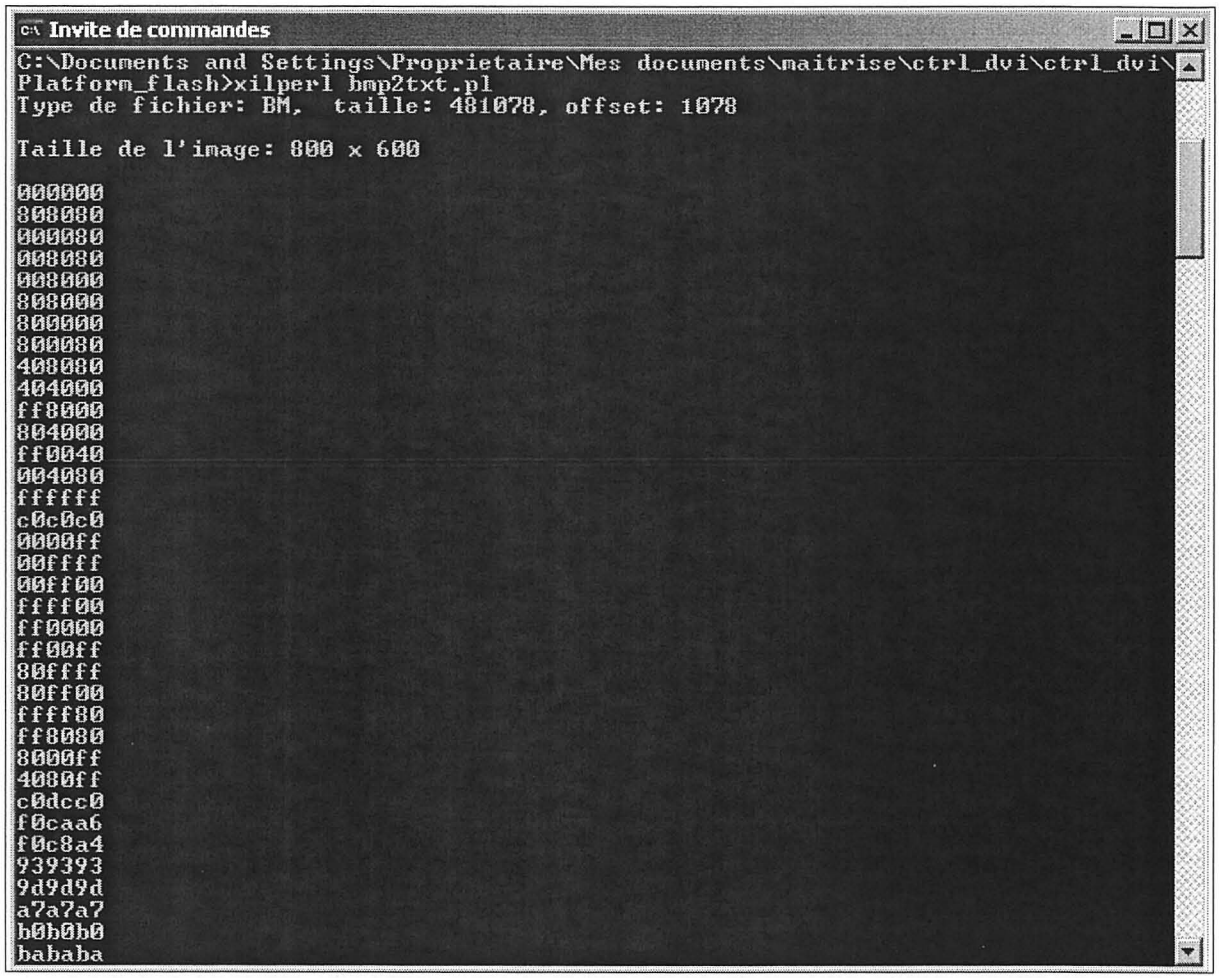

Figure 7.21 Fenêtre terminal DOS avec le résultat de l'exécution de la commande xilperl bmp2txt.pl.

La Figure 7.21 montre le début de l'affichage de la table des couleurs lors de l'exécution du script bmp2txt.pl. Cet affichage sert entre autres à vérifier l'homogénéité des tables de chaque image.

Les deux figures qui suivent, montrent le terminal DOS avant et après l'exécution du script pc2.pl qui permet d'ajouter les données d'image au fichier initial.mcs. Sur la Figure 7.22, vous pouvez voir un exemple de la commande pour exécuter ce script. Le paramètre –ps permet de spécifier la taille de la mémoire PROM (PROM Size), dans ce cas-ci, elle est de 32 Mb d'où « -ps 32 ». Le paramètre « –uf » (User File) permet de spécifier le fichier de texte qui contient les données d'image à ajouter au fichier *.mcs. Dans ce cas-ci, nous avons

utilisé « -uf A800600.txt ». Le dernier paramètre « -pf » (PROM File) sert à spécifier quel fichier *.mcs de départ doit être utilisé pour générer le fichier *.mcs final. Dans le cas de l'image qui suit, le fichier utilisé est initial.mcs qui a été généré à la section précédente et le fichier généré par la commande est nommé new_initial.mcs. Le script génère toujours un fichier dont le nom est le même que le fichier *.mcs initial avec un préfixe « new_ ».

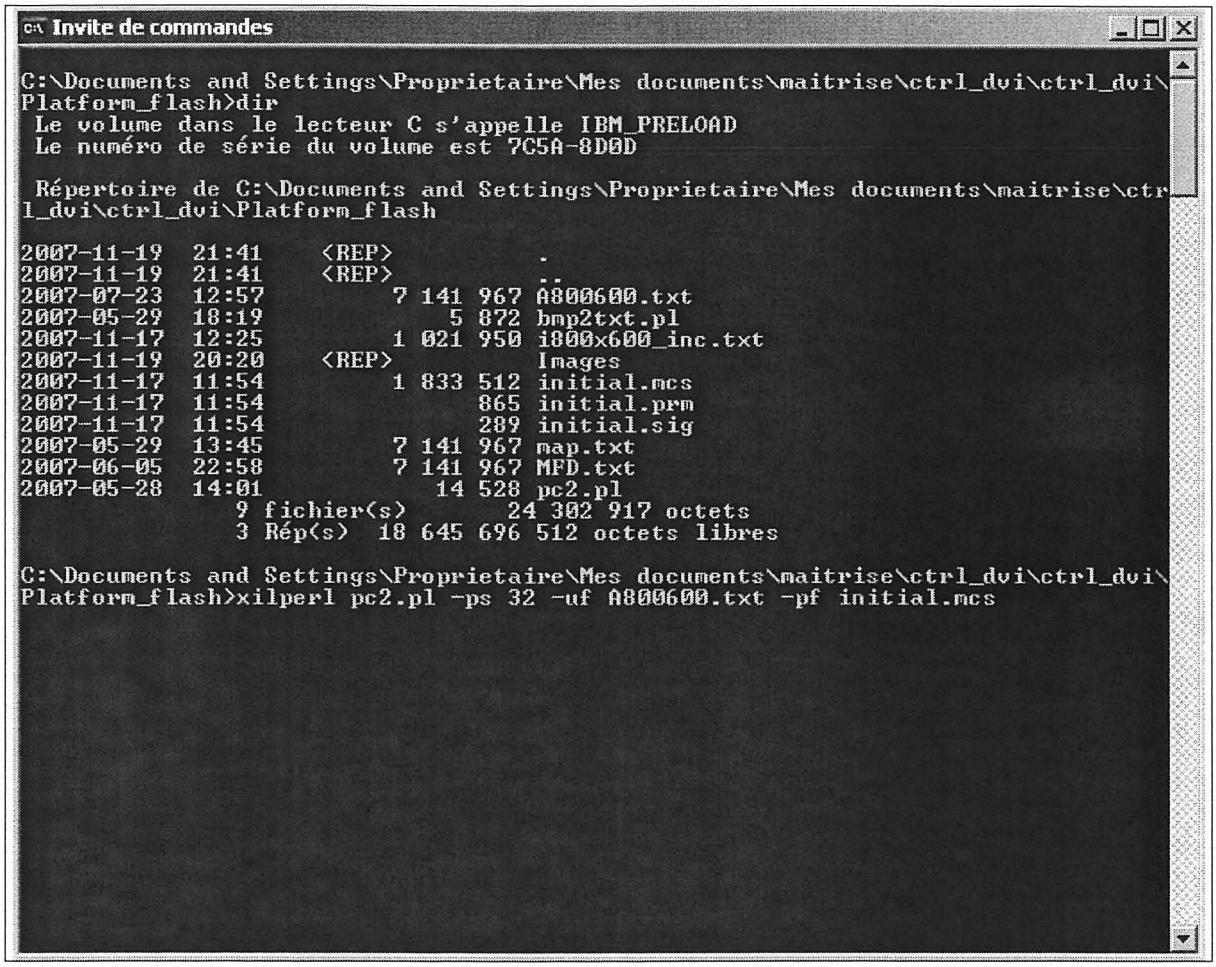

Figure 7.22 Fenêtre terminal DOS avec la commande xilperl pc2.pl.

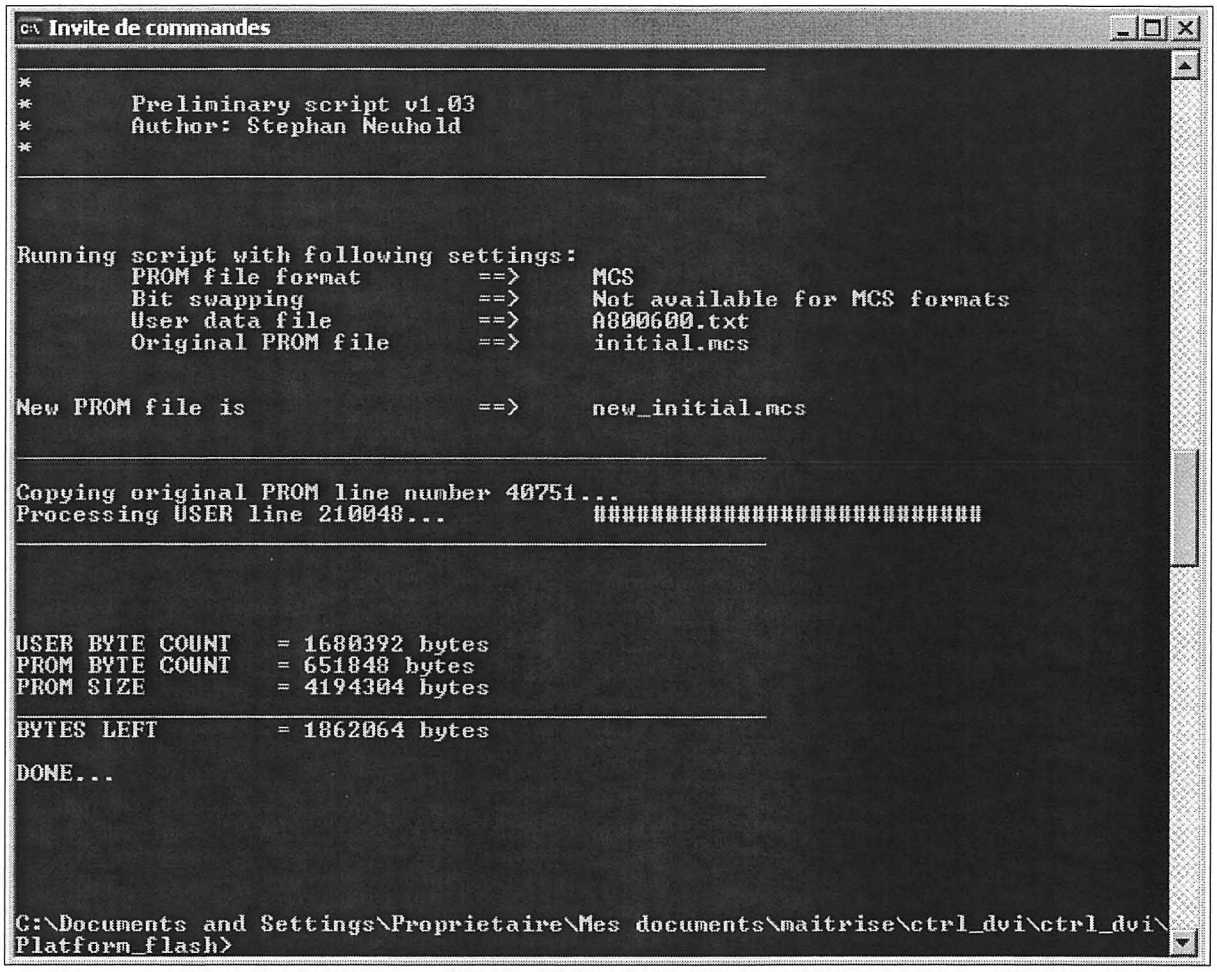

Figure 7.23 Fenêtre terminal DOS avec le résultat de l'exécution de la commande xilperl pc2.pl.

L'exécution du script pc2.pl donne quelques informations comme celle montrée sur la figure précédente. Une fois terminé, vous êtes prêt à configurer le FPGA. Cependant, il est impératif de bien brancher les composantes avant de passer à cette étape. C'est pour cette raison que la section suivante a été rédigée.

7.6 Branchement des composantes matérielles pour le bon fonctionnement du projet

Pour faire fonctionner tout le système, vous avez besoins de ce matériel :

A. Un ordinateur PC avec Windows XP ou ultérieur et qui possède un port parallèle et un port PS2;
B. Le logiciel ISE 8.2i de Xilinx préinstallé;
C. Un câble JTAG Parallel Cable IV de Xilinx;
D. Un écran VGA de plus pour tester le fonctionnement du projet;
E. La plateforme de développement SP305 de Xilinx avec son alimentation.

Les deux figures suivantes montrent le schéma de connexion des composantes.

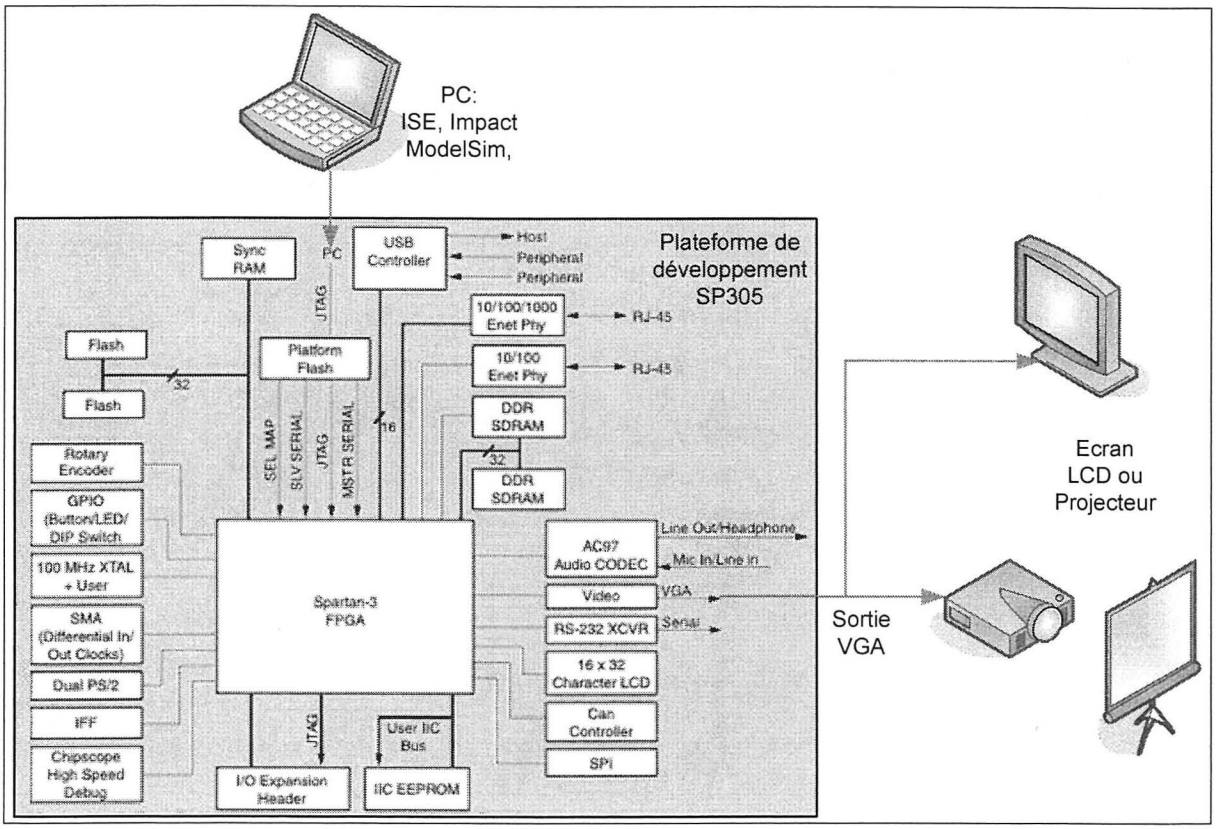

Figure 7.24 Schéma de connexion de la carte SP305 de Xilinx avec ses périphériques.

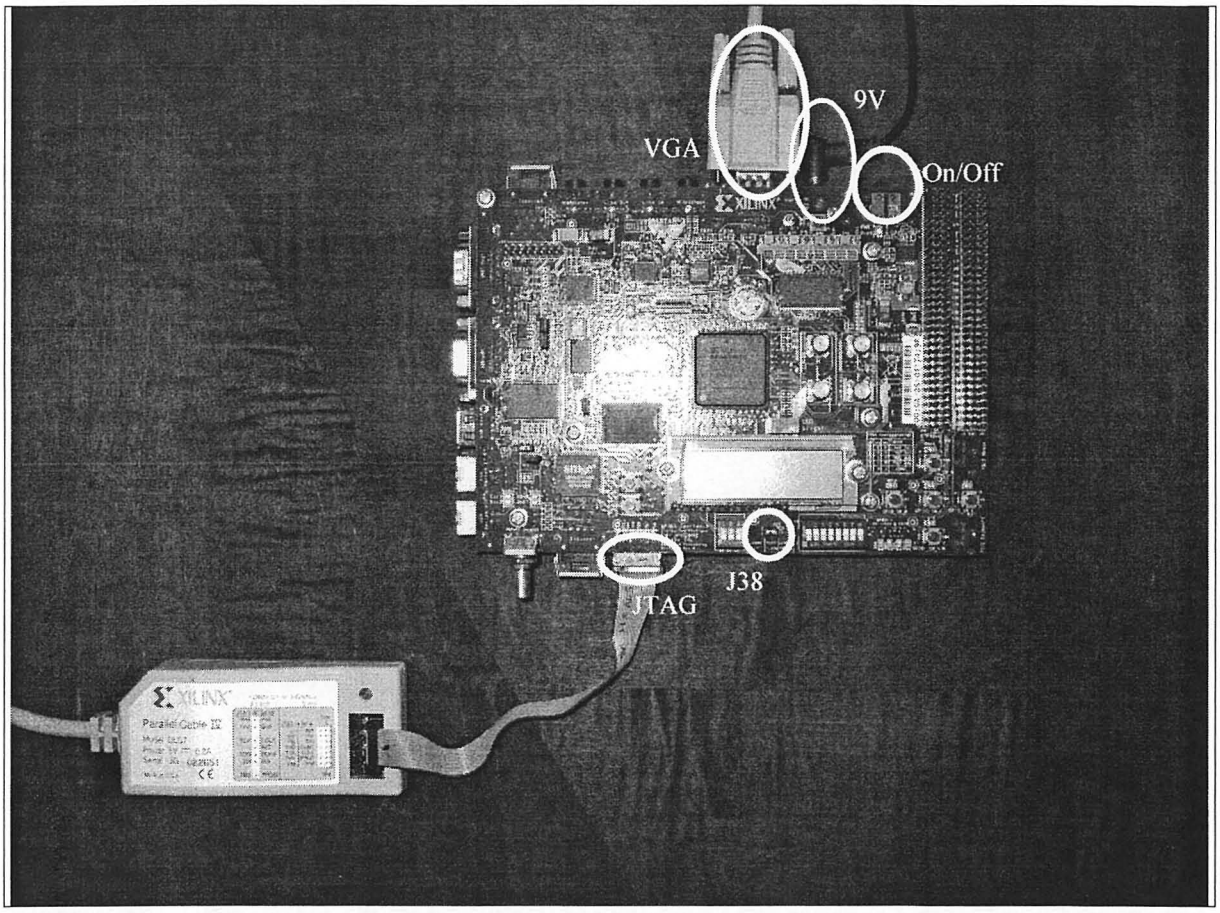

Figure 7.25 Photo du câblage de la carte SP305 de Xilinx.

L'ordinateur qui exécute la suite ISE8.2i de Xilinx doit obligatoirement posséder un port parallèle et un port PS2 afin de pouvoir brancher le câble JTAG « Parallel Cable IV » dont on voit l'une des extrémités sur la photo de la Figure 7.25. Il est possible d'utiliser d'autres technologies de câble JTAG, mais il faut bien s'assurer de la compatibilité de cette technologie avec la carte SP305 avant de procéder au changement.

Le câblage de la carte SP305 est simple; vous pouvez vous référer à la Figure 7.24 pour suivre les prochaines explications. L'ordinateur PC est relié à la carte par le câble JTAG « Parallel Cable IV ». Le câble JTAG est ensuite branché sur la carte comme le montre la Figure 7.25. Pour terminer, la carte est branchée sur un écran par un câble VGA. Nous faisons abstraction des explications pour connecter les alimentations des différent

périphériques puisque nous prenons pour acquis que les compétences de base des lecteurs de ce document sont suffisantes.

Le FPGA doit être placé en mode « Master Serial » puisque sa configuration se fera à l'aide d'un signal sériel provenant de la PROM et c'est lui qui devra générer le signal d'horloge cclk pour sa propre configuration, d'où l'appellation « Master ». Pour placer le FPGA dans ce mode, il faut placer ses trois broches M2, M1 et M0 à zéro. Ceci peut être fait simplement en plaçant tous les interrupteurs SW13 à zéro comme vous pouvez le constater à gauche du cavalier J38 sur la Figure 7.25.

7.7 Chargement de la mémoire PROM et configuration automatique du FPGA

Pour programmer le FPGA, vous devez au préalable initialiser la chaine JTAG. Sur la carte SP305, la chaine JTAG contient deux composantes dans l'ordre suivant : La PROM xcf32p et le FPGA XC3S1500. Pour initialiser le lien JTAG, activez le logiciel iMPACT et ouvrez le mode « Boundary Scan ». La figure suivante montre le mode « Boundary Scan » sélectionné.

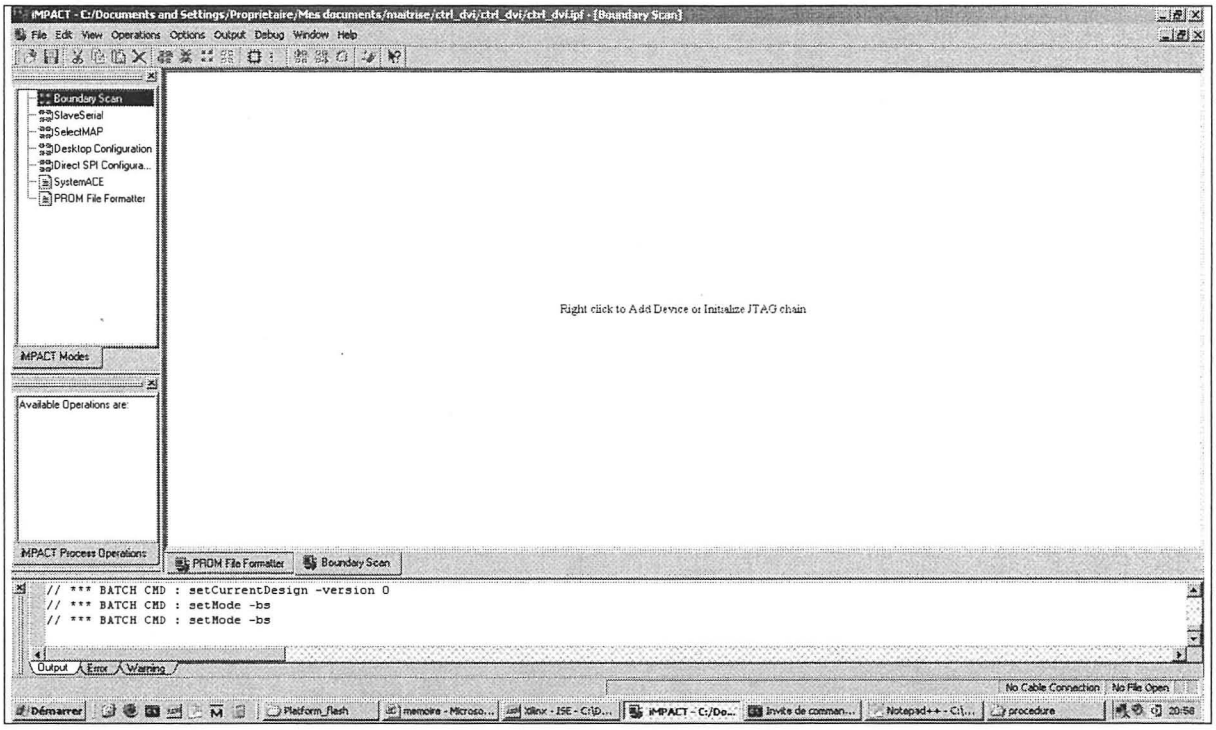

Figure 7.26 Sélection du mode « Boundary Scan » dans le logiciel iMPACT.

Par la suite, déroulez le menu contextuel de la région « Boundary Scan » et lancez la commande « Initialize Chain » comme montré à la figure suivante.

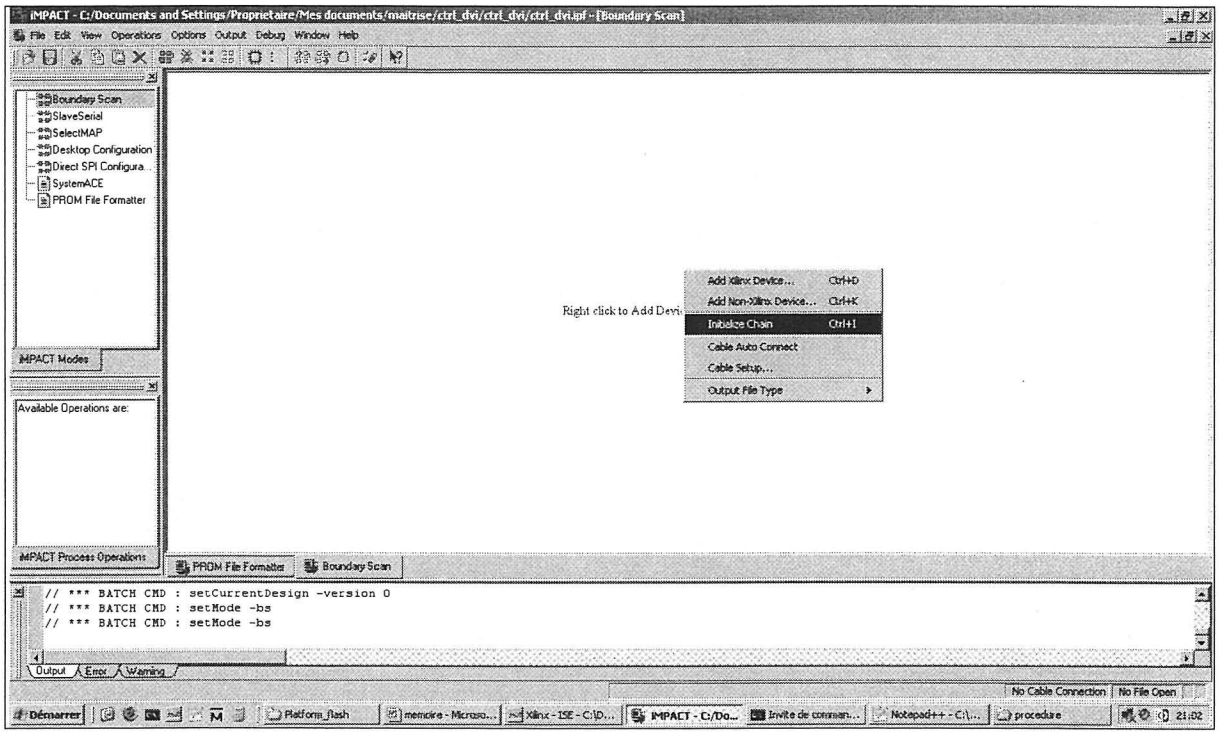

Figure 7.27 Initialisation de la chaine JTAG sur le logiciel iMPACT.

La chaine des composantes de la carte SP305 apparait alors dans la région « Boundary Scan » de la fenêtre et une fenêtre de dialogue « Assign New Configuration File » apparaît avec la composante xcf32P présélectionnée comme sur la figure suivante.

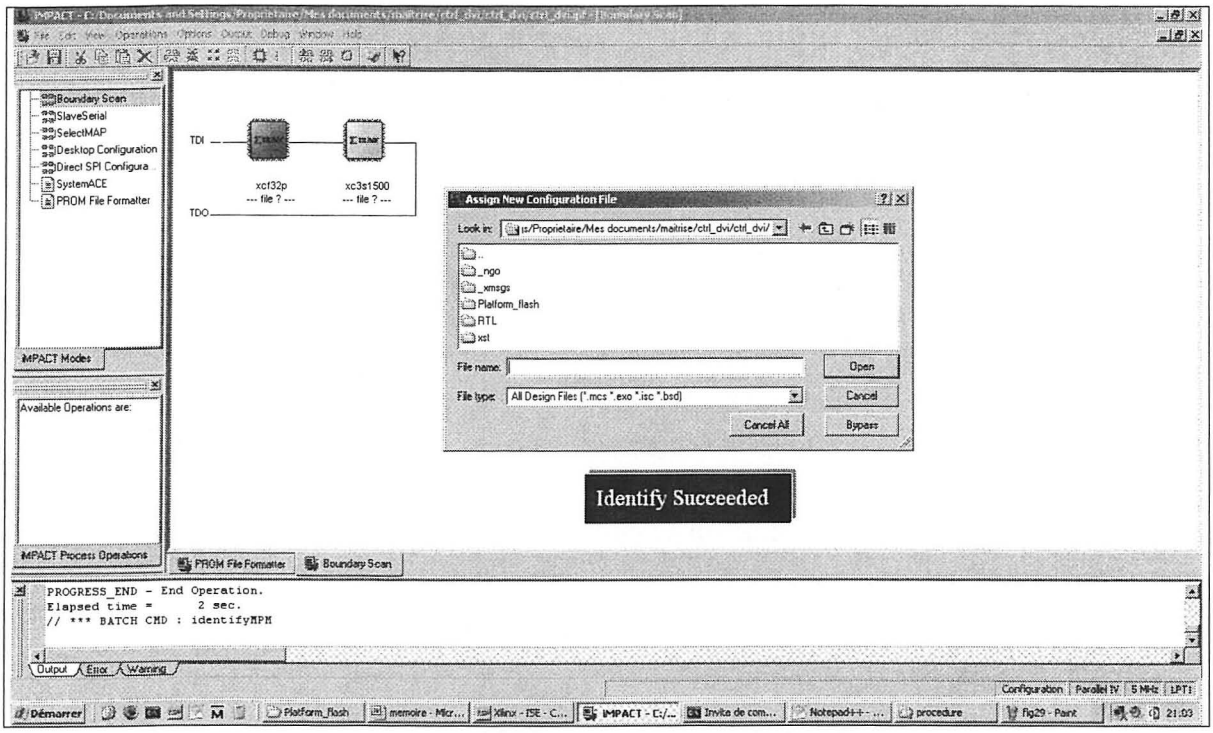

Figure 7.28 Fenêtre de dialogue pour la sélection du fichier new_initial.mcs avec le logiciel iMPACT.

Vous devez alors aller chercher le fichier new_initial.mcs que vous venez de générer et qui doit se trouver dans le dossier Plateform_flash. Appuyez sur le bouton « Open » lorsque cela est fait. Le fichier sera donc assigné à la composante xcf32p et une autre fenêtre apparaîtra pour la sélection du fichier de programmation du FPGA comme sur la Figure 7.29. Appuyez sur le bouton « bypass » pour que la programmation du FPGA soit sautée.

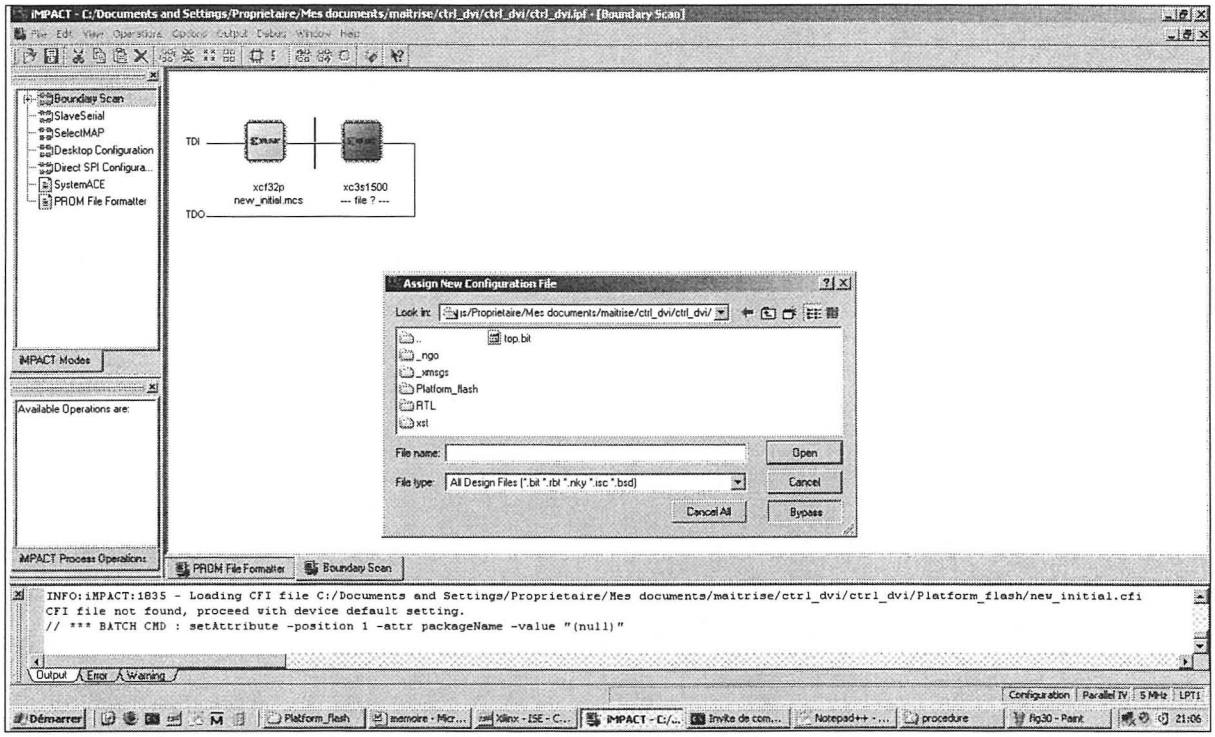

Figure 7.29 Fenêtre de dialogue pour désactiver la programmation du FPGA avec le logiciel iMPACT.

Nous sautons la programmation du FPGA à cette étape puisque celui-ci sera programmé par la mémoire PROM. Ceci nous permet d'éteindre la carte et de garder le programme sur celle-ci sans avoir à recharger le fichier à chaque mise sous tension de la carte.

Pour programmer la PROM, sélectionner l'icône qui la représente dans la chaîne et double-cliquez sur la commande « Program » dans la région « iMPACT Process Operations » du logiciel iMPACT comme la figure suivante le démontre.

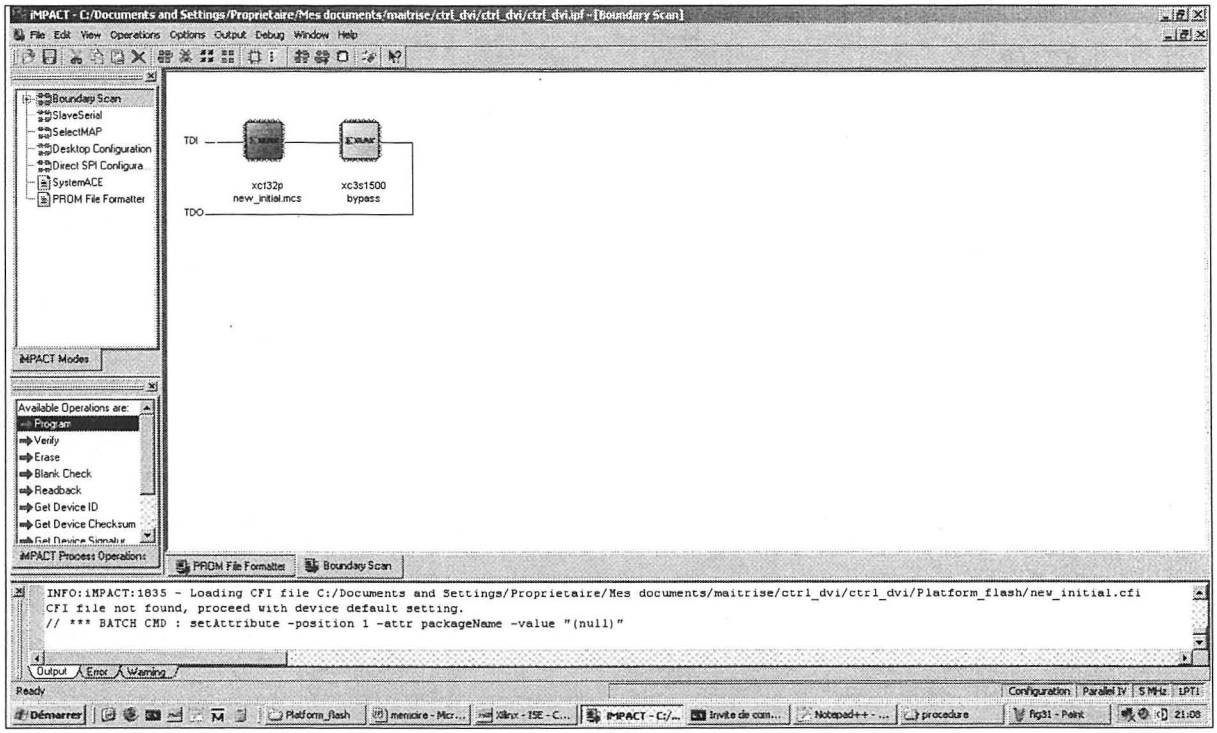

Figure 7.30 Démarrage de la commande « Program » de la mémoire PROM par le logiciel iMPACT.

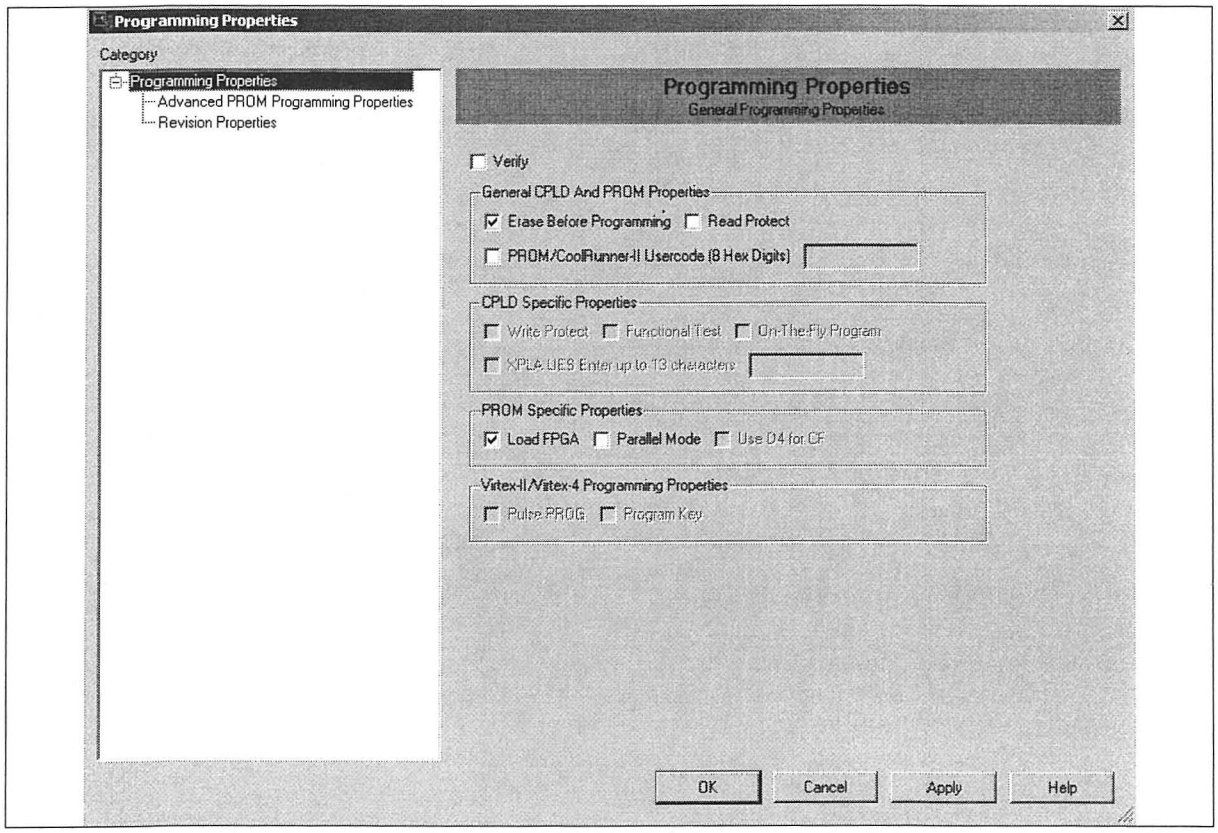

Figure 7.31 Fenêtre d'options de programmation de la mémoire PROM du logiciel iMPACT.

Au lancement de la programmation de la mémoire xcf32p, vous aurez l'apparition de la fenêtre des options de programmation de celle-ci. Sélectionner les options telles qu'elles sont montrées sur la figure précédente afin que la mémoire PROM soit effacée avant d'être reprogrammée et que le FPGA soit configuré suite au succès de la programmation de la mémoire PROM. Appuyez sur le bouton « OK » pour lancer le processus. La barre de progression de la programmation prendra un certain temps avant de commencer à avancer puisque le programme débute par l'effacement de la mémoire PROM avant de la reprogrammée. Un message vous sera affiché lorsque le processus sera terminé.

Voici maintenant ce qui se passera sur la carte SP305 à partir de la programmation de celle-ci. La figure suivante montre un schéma qui indique la programmation de la mémoire PROM par le lien JTAG entre la carte et l'ordinateur PC.

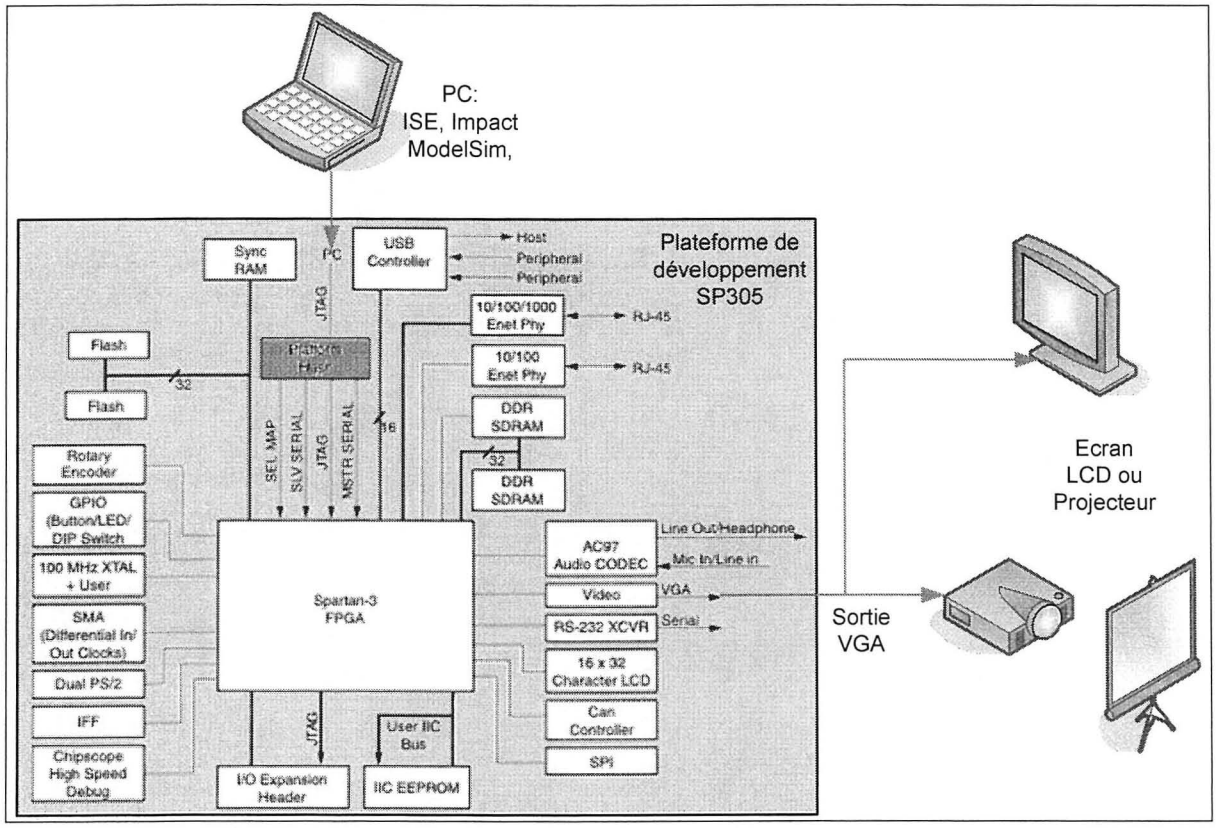

Figure 7.32 Schéma bloc montrant la programmation de la mémoire PROM sur la carte SP305.

Ensuite, la Mémoire PROM lancera la configuration du FPGA comme montré sur la figure qui suit.

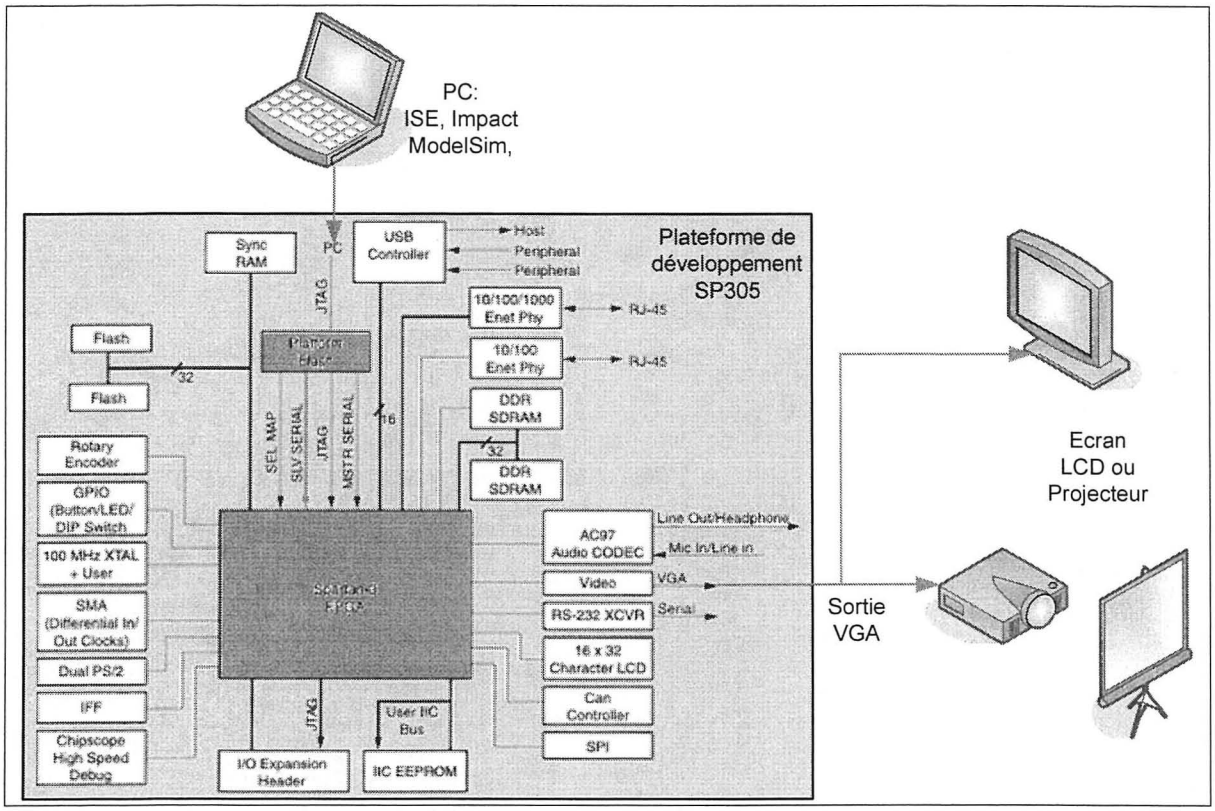

Figure 7.33 Schéma bloc de la configuration du FPGA par la mémoire PROM sur la carte SP305.

Suite à la configuration du FPGA, les DEL « DONE » et « INIT » s'allument et le module Bootloader qui se trouve maintenant dans le FPGA se met à la recherche des données d'image dans la mémoire PROM. Lorsque les données sont détectées grâce au patron de bits 0x8F9FAFBF, la DEL « err1 s'allume pour indiquer le début du chargement de la mémoire. Ceci est présenté sur la Figure 7.34 : Le FPGA charge les données de la mémoire PROM et les envois à la mémoire vidéo représentée ici par la boîte nommée « Sync RAM »

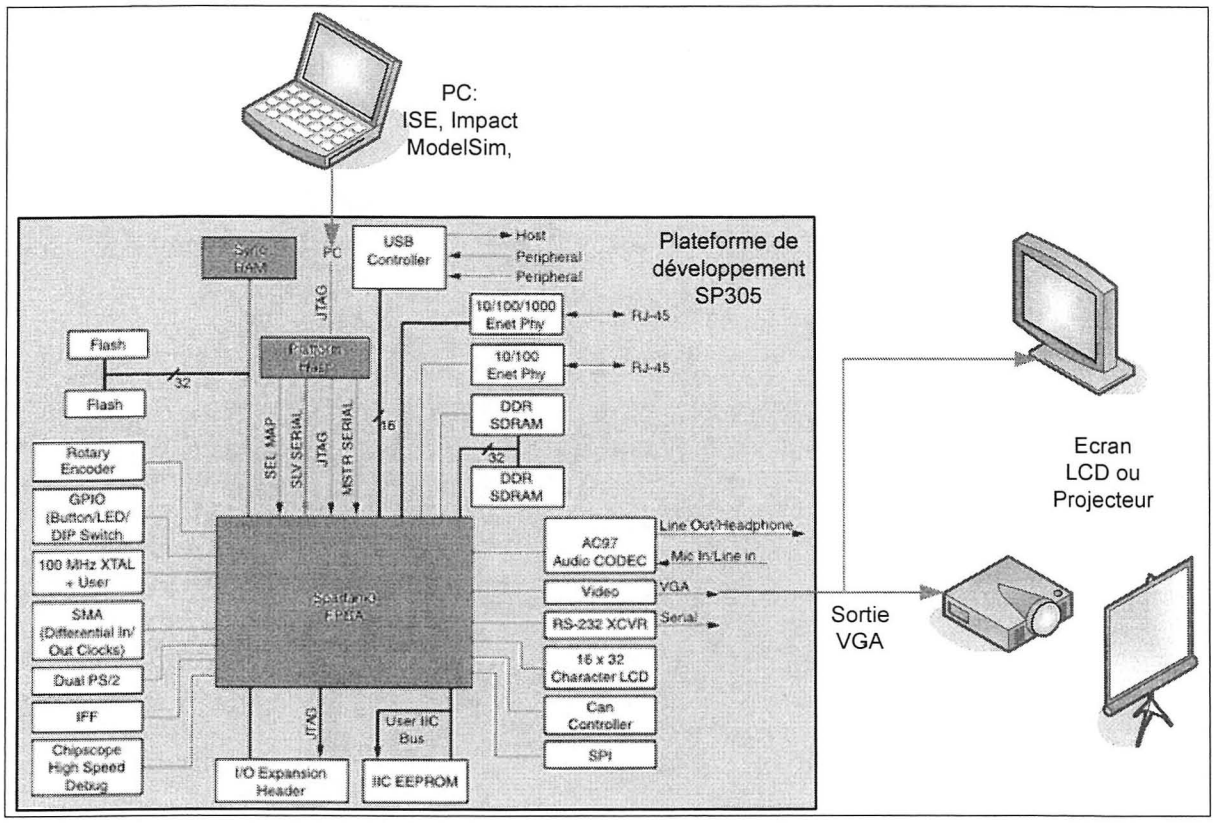

Figure 7.34 Chargement des données d'image dans la mémoire vidéo.

Lorsque le chargement de la mémoire est terminé, la DEL « Err2 » s'allume et le système d'affichage se met en action dans le FPGA. Il est maintenant possible de déconnecter le câble JTAG puisque la carte peut fonctionner en mode indépendant à partir du moment que la mémoire PROM est programmée. La figure suivante montre les composantes sollicitées pour l'affichage vidéo.

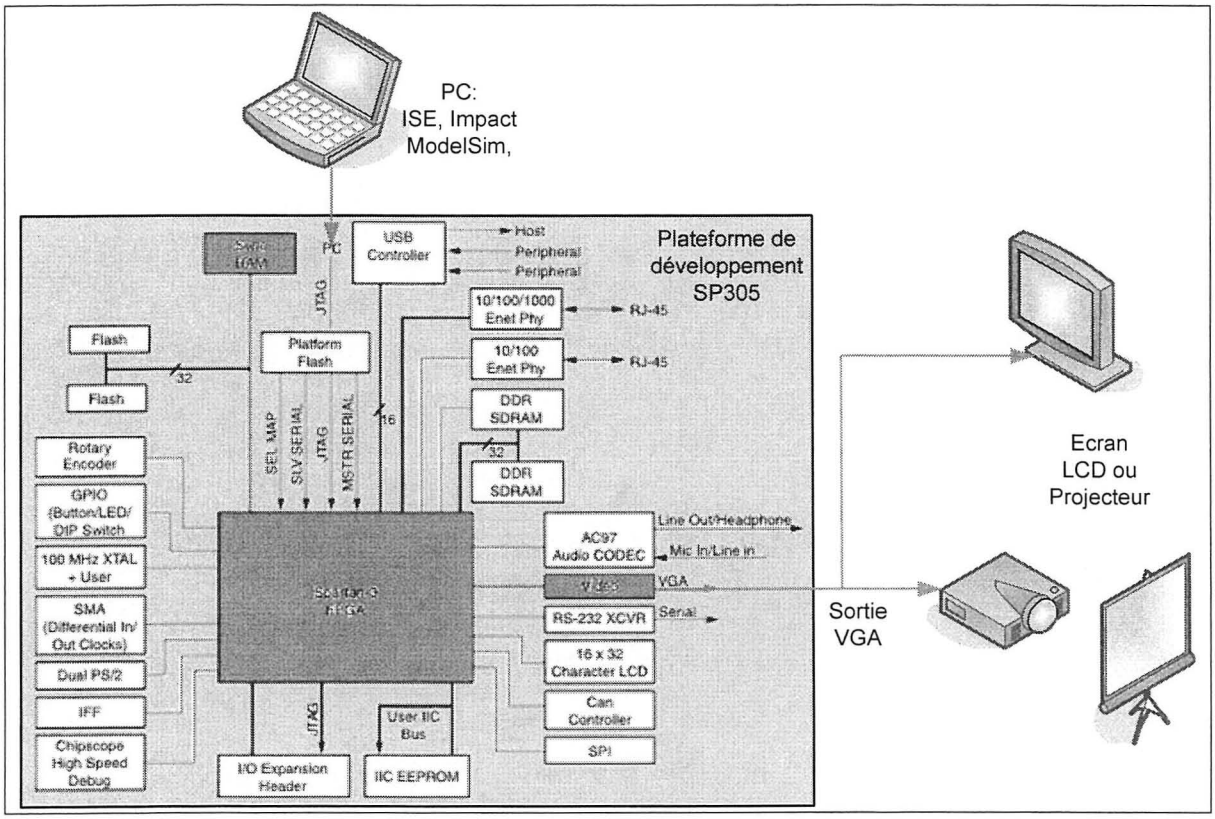

Figure 7.35 Schéma bloc du système lors de l'affichage des images sur l'écran de vérification.

La sélection des images affichées est faite par le module bootloader qui contrôle le système vidéo et qui remplace pour le moment un processeur.

Voici une photo du résultat lorsque le système est activé :

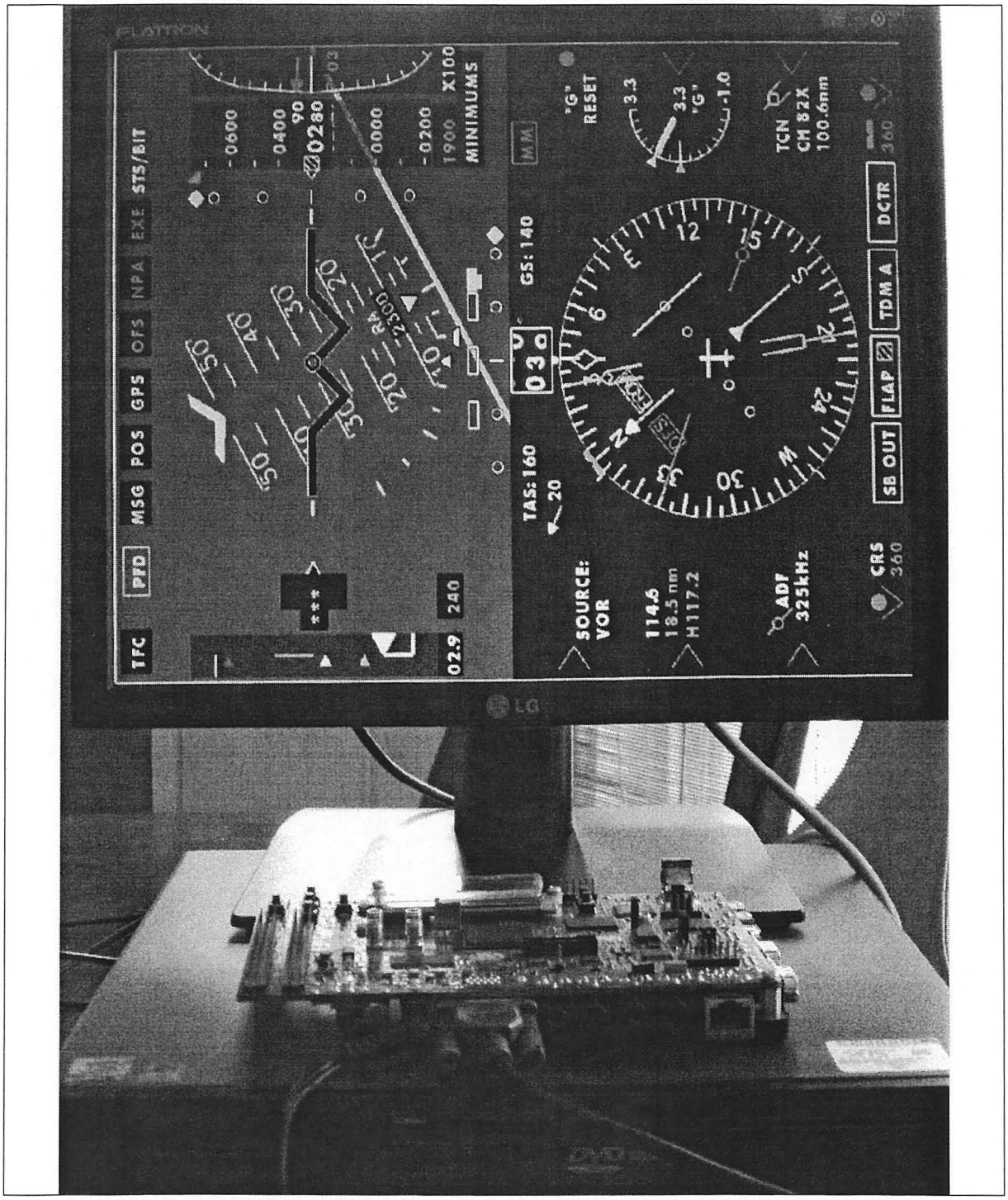

Figure 7.36 Démonstration de la fonctionnalité du système.

Sur la Figure 7.36, on montre la fonctionnalité d'affichage du système. Sur cette photo, la résolution est de 800 x 600 pixels en mode 8 bpp avec un taux de rafraichissement de 60 Hz.

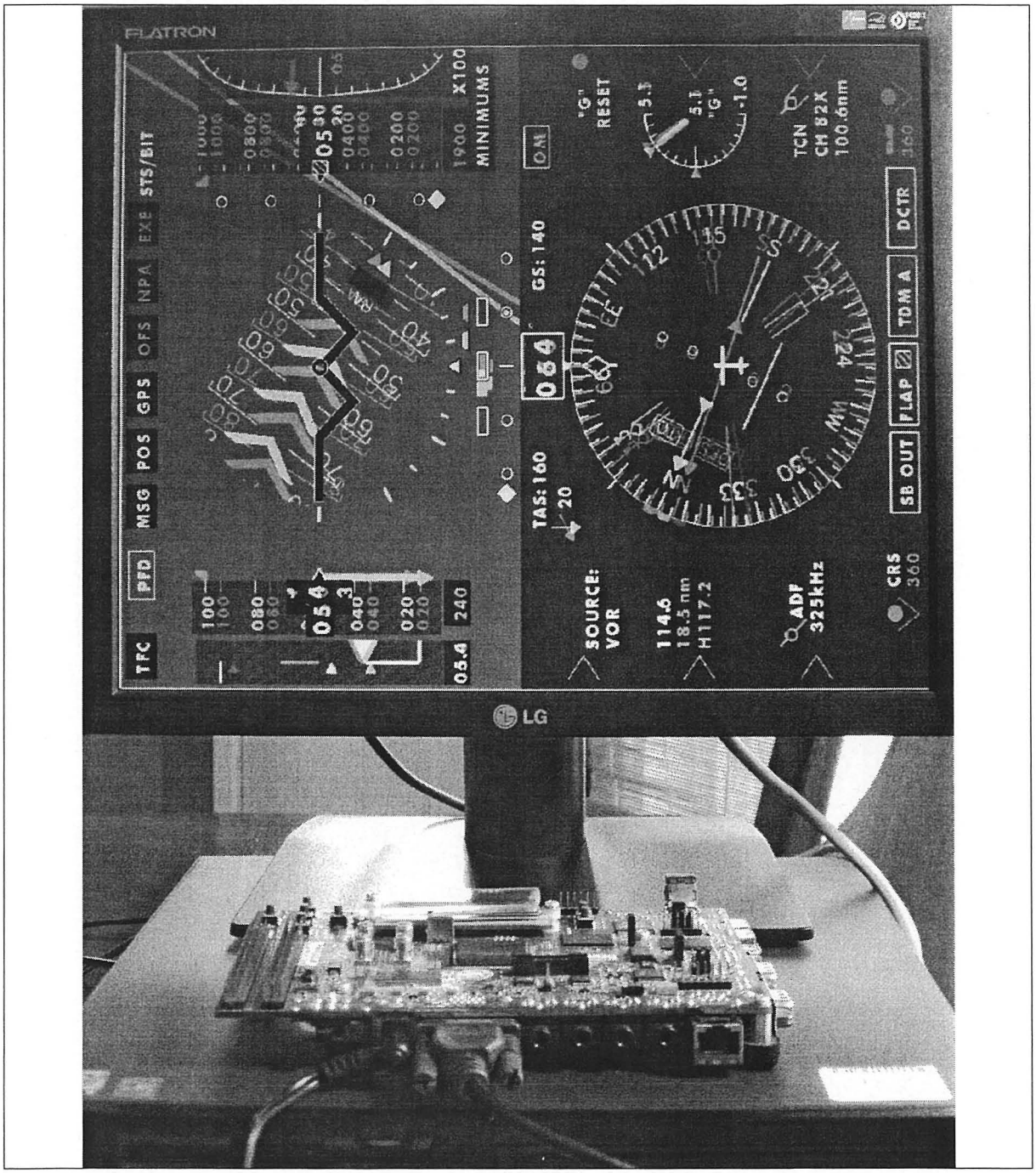

Figure 7.37 Démonstration de la fonctionnalité du système avec transition d'image.

La Figure 7.37 montre aussi la fonctionnalité du système, mais elle montre aussi le coté dynamique de l'afficheur puisque sur cette photo on voit bien que l'image est en transition vers une autre image. C'est le module bootloader qui permet de faire le changement de l'image affichée environ toute les secondes. Pour que la photo puisse prendre les deux

images en un seul clic, c'est que la transition entre les images consécutives est assez rapide pour éventuellement faire de la vidéo avec ce contrôleur.

7.8 Conclusion

Cette section conclue le CHAPITRE 7. Ce chapitre sert de guide pour mettre en marche le système d'AVN à partir du début et étape par étape. Les étapes contenues dans ce chapitre sont donc les suivantes :

A. Création du projet dans le logiciel ISE 8.2i de Xilinx;
B. Configuration, fichiers de contraintes et synthèse du projet;
C. Placement et routage et création du fichier *.bit;
D. Génération du fichier *.mcs comprenant les données de configuration de base du FPGA;
E. Exécution des scripts pour insérer les données d'images dans le fichier *.mcs;
F. Branchement des composantes matérielles pour le bon fonctionnement du projet;
G. Chargement de la mémoire PROM et configuration automatique du FPGA.

Les étapes A, B, C et D sont nécessaire uniquement la première fois que la génération du fichier de configuration est nécessaire. Par la suite, il est possible de re partir de l'étape E en utilisant toujours le même fichier de base *.mcs et de suivre les étapes subséquentes (les étapes E, F et G) afin d'insérer des images différentes et de mettre en fonction le système d'affichage numérique.

CHAPITRE 8

DISCUSSIONS ET RECOMMANDATIONS

Ce chapitre permet de situer l'état fonctionnel du système d'AVN dans son contexte de développement au travers trois sections différentes :

A. Interprétation des résultats;
B. Les problèmes rencontrés et les essais infructueux;
C. Ce qui peut être ajouté ou amélioré.

Ce chapitre se veut très important dans la mesure où il peut servir de tremplin pour le développement d'un système d'AVN plus complexe.

8.1 Interprétation des résultats

Cette section a pour but de montrer l'aboutissement du projet final. Pour chaque section, une explication des résultats permet de bien comprendre les choix technologiques qui ont permis de mener le projet à terme tout en répondant aux attentes de départs.

8.1.1 L'espace utilisé par le code synthétisé

Suite à l'utilisation de BlockRAM, il a été possible de rendre le contrôleur très efficace au niveau de l'espace qu'il occupe. Voici un tableau qui résume les résultats obtenus :

Tableau 8.1

Sommaire des ressources matérielles utilisées par le code dans le FPGA XC3S1500 donné par le logiciel ISE 8.2i de Xilinx

Logic Utilization	**Used**	**Available**	**Utilization**
Total Number Slice Registers	1,047	26,624	3%
Number used as Flip Flops	991		
Number used as Latches	56		
Number of 4 input LUTs	979	26,624	3%
Logic Distribution			
Number of occupied Slices	845	13,312	6%
Number of Slices containing only related logic	845	845	100%
Number of Slices containing unrelated logic	0	845	0%
Total Number 4 input LUTs	1,030	26,624	3%
Number used as logic	979		
Number used as a route-thru	51		
Number of bonded IOBs	111	487	22%
IOB Flip Flops	69		
Number of Block RAMs	4	32	12%
Number of GCLKs	5	8	62%
Number of DCMs	2	4	50%
Total equivalent gate count for design	292,391		
Additional JTAG gate count for IOBs	5,328		

À première vue, on remarque que la composante FPGA est sous-utilisée avec uniquement 6% des « slices » et 3% des LUT. On remarque aussi que quatre Block RAM sont inférés à cause, entre autres, de l'utilisation de fifo et aussi de la table de couleurs. Il est à noter que les statistiques du Tableau 8.1 englobent celles engendrées par le module bootloader, ce qui n'est pas négligeable étant donné la complexité de ce module. Comme première étape, il est donc facile d'affirmer que ce contrôleur prend très peu d'espace. De plus, étant donné que plusieurs optimisations peuvent encore être faites, il pourrait être possible de réduire encore la quantité de ressources utilisées par le système. La section 8.2 explique en détail ce qui peut être fait pour réduire encore l'espace utilisée et augmenter les fréquences d'opération du contrôleur vidéo.

8.1.2 La fréquence maximale atteinte de l'horloge système

Le convertisseur numérique vers analogique du port VGA permet de transmettre les pixels avec une fréquence de 50 MHz ([58] Xilinx, 2005, p. 21). Cependant, les fréquences théoriques atteignables des deux horloges générées par le système sont de 94,7 MHz pour l'horloge pixel pclk et 91,1 MHz pour l'horloge maître mclk. Ces données sont tirées du rapport de synthèse du logiciel ISE 8.2i.

8.1.3 Les résolutions maximales d'affichage atteintes par le système

En théorie, ce qui limite présentement le système est le convertisseur VGA qui est limité à une fréquence de 50 MHz. Si on se fie à cette limite physique et qu'on regarde le

Tableau 5.1, la résolution maximale atteignable est 800x600 avec rafraichissement de 75 Hz puisque la fréquence pixel nécessaire pour cette résolution est de 49,5 MHz. Cependant, le convertisseur a été poussé au-delà de ses limites nominales, puisque la résolution 1024 x 768 avec un rafraichissement de 60 Hz a été essayée avec succès. Ce qui est intéressant, c'est que la fréquence pixel demandée pour cette résolution est de 65 MHz, ce qui dépasse de 30 % la capacité fréquentielle nominale du convertisseur.

Aussi, si on se fie aux fréquences théoriquement atteignables de notre AVN, il est possible d'atteindre la résolution de 1024 x 768 avec un rafraichissement de 75 Hz. Il est permis de penser qu'en optimisant encore les délais de propagations critiques du système, il sera possible d'atteindre la résolution de 1024 x 768 avec un rafraichissement de 85 Hz. En effet, la fréquence pixel de cette résolution est de 94,5 MHz, ce qui est légèrement plus élevée que la fréquence théorique atteignable de l'horloge maître du système.

La profondeur des couleurs n'affecte pas la fréquence d'opération de l'affichage. Elle n'affecte que la bande passante du canal de communication entre le contrôleur et la mémoire vidéo. En maintenant une fréquence de l'horloge maîtresse plus élevée que celle de l'horloge pixel, on assure un fonctionnement optimal du contrôleur.

8.1.4 Les spécifications actuelles du système par rapport à celles attendues

Si nous reprenons les spécifications fonctionnelles énumérées au CHAPITRE 4, voici celles qui ont été atteintes.

Le contrôleur vidéo numérique comprend :

a. une sortie VGA (RGB, utilisées présentement chez CMC) qui permet aussi de s'adapter dans l'éventualité d'une migration vers un port DVI;
b. une résolution d'affichage allant jusqu'à 1024 x 768 et un taux de rafraichissement de 60 Hz (et théoriquement jusqu'à 75 Hz) en mode 24 bits par couleur;
c. une mémoire vidéo externe au FPGA pour l'entreposage des images, cette mémoire possède une bande passante de 280 Mo/s puisque la fréquence actuelle du système est

de 70 MHz avec 4 octets par cycle. Cette mémoire fonctionne avec un pipeline (mémoire ZBT);

d. une possibilité d'utiliser les modes d'affichage 8 bits avec table de couleurs ou 24 bits;
e. deux bus Wishbone : un pour le processeur host (ou bootloader) et un pour la mémoire vidéo, ce qui permet un interfaçage rapide et une optimisation de la bande passante de la mémoire vidéo;
f. deux banques d'image, ce qui permet de faire un changement rapide de l'image affichée par un changement de la banque d'image sélectionnée sans provoquer de déformation d'image;
g. un module de chargement (bootloader) permettant au minimum de charger les images en mémoire vidéo et de configurer le module d'AVN au démarrage.

La carte de développement répond à toutes les spécifications de départ puisque ces dernières ont été définies en fonction du choix de la plateforme de développement utilisée. Il est donc inutile de répéter la liste de ces spécifications physiques, référez-vous plutôt à la section 4.2 pour les détails.

De la même manière, référez-vous à la section 4.3 pour connaître le guide de syntaxe du code VHDL qui a été utilisé pour programmer le contrôleur vidéo.

8.1.5 Présentation du projet chez CMC Électronique

Une présentation du projet à eu lieu afin de démontrer la fonctionnalité du projet et la capacité de l'ÉTS et CMC Électronique de travailler en partenariat. Il a été question du but du projet, du principe de fonctionnement et des résultats obtenus à ce jour. Les personnes présentes ont été enchantées de ce qu'ils ont vu. Le message est si bien passé que des négociations sont en cours pour démarrer une phase numéro 2 du projet.

8.2 Les problèmes rencontrés et les essais infructueux.

Lors du développement du contrôleur vidéo, plusieurs essais ont été effectués avant d'en arriver à un produit final. Cette section relate des embuches les plus marquantes qui ont été rencontrées.

8.2.1 Chargement de la mémoire par les fichiers *.SVF

Un fichier SVF permet d'utiliser le câble JTAG pour forcer des valeurs sur les broches du FPGA. Nous avons donc essayé, au commencement du projet, de charger la mémoire par cette approche. Le principe était simple, mais le chargement de la mémoire de cette manière pouvait prendre plusieures heures, puisqu'il était impossible d'envoyer uniquement les données utiles. Pour chaque donnée de 32 bits envoyée, il fallait charger en série tous les registres à décalage de la chaine JTAG du FPGA. Avec un port JTAG cadencé à 200 kHz, c'était vraiment trop long. Plusieurs heures de préparation et d'essais ont été perdues avant de s'en rendre compte.

8.2.2 Changement du chip mémoire

Il s'est passé un long moment avant de s'apercevoir que la mémoire ZBT s'adressait par tranche de 36 bits. Il avait été pris comme idée que la mémoire était adressable par tranches de 9 bits, ce qui fait que la lecture se faisait dans le circuit avec cette idée en tête. Personne ne croyait qu'il aurait pu en être autrement. Par contre, lors du développement du code, un problème persistant nous a obligés à revoir nos conceptions de ce que peut être une mémoire. En effet, le problème rencontré était que l'image s'affichait, mais uniquement en partie. Après plusieurs jours de recherche, d'essais et d'erreur, il est devenu évident que le problème était à la base même du fonctionnement de la mémoire. En lisant les spécifications ([10] Cypress, 2004) il est devenu évident que le problème était là. Nous n'utilisions uniquement que le quart de la mémoire en faisant des accès uniquement aux adresses divisibles par

quatre. Nous avons corrigé le tir en faisant nos accès à chaque adresse et l'image est apparue comme par enchantement.

Par contre, cette mauvaise perception de la mémoire n'était pas encore corrigée parfaitement. Étant donné que la composante possédait 23 broches d'adresses reliées au FPGA, il ne nous est jamais venu à l'idée que certaines de ces broches pouvaient être inactives. Ce qui fait qu'avec l'ajout de plusieurs images dans la mémoire est apparue la suite du problème. Les images chargées dans la mémoire se chevauchaient lors de l'affichage. En vérifiant le contenu de la mémoire, on devait conclure que le chargement écrasait les données à mesure. Encore une fois, en approfondissant la lecture des spécifications de la mémoire, on s'est rendu compte que celle-ci ne possédait que 18 bits d'adresse actifs pour un total de 9 Mbits de données adressables. Ceci n'était pas assez pour les besoins du projet. Il a donc été nécessaire de faire une mise à niveau de la composante mémoire. Nous avons commandé la nouvelle pièce et l'avons remplacé sur la plateforme SP305. Nous pouvons maintenant accéder à 9 Mo de données, ce qui est largement suffisant.

8.3 Ce qui peut être ajouté ou amélioré

Cette section relate de ce qui peut encore être fait pour améliorer le système. Il est question des améliorations aux niveaux fonctionnel, opérationnel et physique du contrôleur vidéo.

8.3.1 Implémentation du mode d'économie d'énergie

Le mode d'économie d'énergie activé de manière logiciel n'est pas implémenté puisqu'il n'était pas prioritaire et a été mis de côté. Il est cependant aisé de l'ajouter aux fonctionnalités du système en utilisant un registre du module ctrl_register pour mémoriser l'état du mode d'économie d'énergie et de permettre le transfert de cette information au convertisseur VGA et à la mémoire vidéo pour qu'ils puissent se mettre en mode d'économie. De plus, si le FPGA le permet, il pourrait être intéressant de mettre le contrôleur vidéo en mode d'économie lui aussi.

8.3.2 Simulation post placement et contraintes de timing

Pour améliorer les performances du système, il sera nécessaire d'ajouter des contraintes de timing aux signaux critiques du circuit. De plus, afin de s'assurer du fonctionnement adéquat du système après l'ajout de ces contraintes, il sera nécessaire de refaire toute les simulations. Ceci nous permettra de déceler les détails de synchronisme du système.

8.3.3 Utilisation d'un port DVI plutôt que VGA

Les limites des technologies VGA sont atteintes aujourd'hui et sont dépassées par celles du standard DVI. Cependant, il est nécessaire de procéder à une étude approfondie des besoins de CMC Électronique avant d'entreprendre le développement d'une technologie DVI puisque les performances des technologies VGA répondent amplement aux besoins actuels de la compagnie.

8.3.4 Jumeler les deux scripts PERL

Pour utiliser le projet en production, il deviendra primordial de rendre le processus de création du fichier *.mcs plus automatique. Ceci se traduit par une obligation de jumeler les deux scripts PERL qui permettent, dans un premier temps, de générer le fichier des données d'image et, dans au deuxième temps, de créer le fichier *.mcs. De plus, le nouveau script devrait permettre de convertir rapidement plusieurs fichiers d'image en série sans passer par des méthodes manuelles.

Le script de conversion des fichiers *.BMP en fichier *.txt ne reconnaît pas encore la profondeur des couleurs de manière automatique, il faudrait ajouter cette fonctionnalité au script final.

Il serait aussi possible de réduire légèrement la complexité du module bootloader en réorganisant les bits de données à l'avance pour éviter de le faire à la sortie de la mémoire PROM (se référer à la section 5.2).

Une autre amélioration que le script PERL peut apporter est d'inscrire la table des couleurs directement dans le fichier de configuration du FPGA plutôt que de charger cette table par le bootloader. Ceci permettrait premièrement de réduire le temps de préparation du système à l'initialisation, deuxièmement de réduire le câblage occasionné par le contrôle du chargement dans le module VGA et troisièmement de réduire la complexité du module bootloader.

8.3.5 Configuration du FPGA en mode « Parallel Master »

La configuration du FPGA en mode « Parallel master » nous permettrais de rendre la configuration du FPGA et le chargement de la mémoire vidéo huit fois plus rapide, puisque les données arriverait par groupe de 8 bits au FPGA plutôt que un bit à la fois (voir la section 5.2 pour plus de détails).

8.3.6 Optimisation des modules fifo et dclk_fifo.

Premièrement, l'utilisation d'un module fifo à deux horloges est discutable. Il serait intéressant d'étudier la possibilité d'utiliser un simple fifo dont la profondeur est optimale avec un double tampon pour effectuer le changement de domaine d'horloge en sortie. Il serait peut-être possible de diminuer le nombre de ressources utilisées par le module dclk_fifo.

Deuxièmement, il serait possible de faire disparaitre le module fifo et d'ajouter uniquement deux ou trois registres de plus dans le module pixel_processor. En effet, le module fifo sert uniquement pour assurer que les données provenant de la mémoire ne soient pas perdues dans le cas où le module pixel_processor ne permet pas de prendre de nouvelles données. Étant donné que la mémoire possède un pipeline avec les données décalées de deux cycles d'horloge, il arrive que la consigne d'arrêt de lecture soit lancée et que des données continuent d'être acheminées. Le module fifo permet de palier à cela, mais il serait possible de le remplacer par quelques registres tampons.

8.3.7 Ajout d'un processeur

L'ajout d'un processeur permettrait de rendre le contrôle de l'affichage plus facile et plus complexe, en ce sens qu'il permettrait un prétraitement des images avant l'affichage. En effet, il deviendrait possible de faire rapidement des changements dans la mémoire vidéo afin de rendre les images plus dynamiques. Il serait même possible, par exemple, d'ajouter des algorithmes de compression, de manipulation et de création d'image.

8.4 Conclusion

À la lumière de ce qu'on vient de voir, ce chapitre nous donne toutes les pistes pour bien comprendre quels ont été les défis rencontrés, de quelles manières ils ont étés relevés et en quoi résulte tout le travail accompli. Le CHAPITRE 8 sert aussi de point de départ pour une complexification éventuelle du projet d'AVN.

L'ajout d'un processeur permettra entre autre de faire du traitement d'image ou d'ajouter des formes géométriques aux images. L'optimisation de certaines étapes de mise en marche du système facilitera son utilisation dans les différents projets de CMC Électronique. Aussi, l'ajout de certaines fonctionnalités permettra de rendre le contrôleur vidéo plus versatile et plus performant encore, ce qui laisse envisager un avenir prometteur pour l'évolution de ce dernier.

CONCLUSION

Le système d'affichage vidéo développé dans le cadre de ce mémoire répond aux critères qui ont étés fixés au début du projet. Il est suffisamment fonctionnel et optimisé pour être utile et prouve la faisabilité de son intégration avec les systèmes existants déjà chez CMC Électronique.

Étant donné la possibilité d'atteindre des performances encore meilleures dans le futur, ce circuit pourra être utilisé longtemps par la compagnie en l'adaptant aux différentes applications qui en ont besoin et aussi en l'insérant dans les technologies programmables émergeantes. Il permettra alors à CMC Électronique d'économiser sur l'achat de composantes vidéo spécialisées ainsi que sur toutes les dépenses qu'occasionne le changement de composantes. Entre autre, les dépenses liées au développement d'un nouveau circuit adapté, à la formation sur cette nouvelle composante ainsi qu'à l'attestation de qualité et de sécurité des nouveaux circuits dévelloppés pourront être amoindrie substentiellement. De plus, la prise en charge du dévelloppement d'un contrôleur vidéo par l'entreprise lui permet de prendre possession de l'expertise du développement et du maintien de cette technologie.

Le contrôleur vidéo numérique est entièrement implémenté en langage VHDL, ce qui répond à un critère de première importance pour CMC Électronique puisque cela permet l'intégration du contrôleur dans un FPGA. Il est facilement configurable et adaptable pour répondre aux exigences et aux besoins de chacun des écrans utilisés dans les produits de CMC Électronique. Il permet de remplacer les contrôleurs d'écrans utilisés en ce moment par la compagnie dans leurs systèmes d'avionique. Il possède une architecture qui lui permettra d'évoluer facilement avec les technologies de l'heure et aussi celles qui émergeront dans le futur.

Le présent document explique toutes les étapes franchies pour en arriver au circuit final. On y définit au départ une structure de travail et ensuite les spécifications de départ envisageables. On y décrit ultérieurement la recherche de la plateforme de développement

adéquate pour le travail. Ce document explique aussi en détail le fonctionnement de chaque module et sous-module en relation avec les différentes composantes qui sont attachées au système et aussi en relation avec les différents principes et méthodes utilisés pour l'affichage d'une image numérique sur un écran. L'utilisation des scripts PERL de formatage du fichier de configuration est décrite dans ce mémoire afin de facilité l'insertion d'images dans la mémoire vidéo. Il est à noter que le langage PERL est utilisé puisque qu'il est tout à fait approprié pour la manipulation de fichiers texte (utilisés lors de la manipulation d'images) et que son interpréteur logiciel est intégré à la suite de logiciels de développement de FPGA ISE 8.2i de Xilinx.

L'afficheur vidéo numérique final permet donc d'afficher une image de 24 bits par pixel ou de 8 bits par pixel, avec une résolution de 1024 x 768 avec un taux de rafraichissement de 60 images/seconde. Toute la configuration du FPGA est contenue dans la mémoire PROM, incluant les données d'images. Lors de la mise sous tension du circuit, le FPGA se configure par lui-même et les images sont chargées dans la mémoire vidéo de manière autonome. Les images sont affichées une à la suite de l'autre grâce à l'intervention du module bootloader qui effectue la sélection des images affichées par l'intermédiaire des deux banques d'image. L'implémentation de ces deux banques d'image permet au système de faire des transitions franches entre deux images consécutives, ce qui permettra éventuellement de faire des animations.

Le projet à été développé sur une composante Spartan III afin de valider son utilisation. On remarque que le projet occupe très peu d'espace dans le FPGA, ce qui nous permet de conclure qu'il est inutile de dépenser pour des composantes plus complexes puisque des composantes modestes comme le Spartan III, suffisent emplement. De plus cette composante occupe peu d'espace sur le circuit imprimé et possède amplement d'entrées et de sorties inutilisées, ce qui laisse prévoir un avenir prometteur à l'utilisation de ce genre de composante chez CMC Électronique.

Le Circuit final est dévelopé dans un souci de simplicité. Son utilisation et son implantation dans un système numérique est aisée. Cependant, pour augmenter la flexibilité du système, il

est recommandé de changer le module bootloader et de le remplacer par un processeur. Cela permettra entre autre de faire du traitement d'image et d'ajouter des formes géométriques aux images affichées. La mise en route du système d'affichage est suffisemment automatisée pour l'effectuer en quelques minutes. On recommande par contre de jumeler les deux scripts de formatage PERL du fichier de configuration et de fabriquer un script de commande terminal qui lance la configuration du FPGA. Cela permettra de rendre la configuration du système entièrement automatique et donc l'utilisation du projet plus aisée. Il est aussi conseillé d'ajouter certaine fonctionnalité pour rendre le circuit moins énergivore et plus performant en économie de mémoire vidéo.

ANNEXE I

Le code VHDL du contrôleur vidéo et du bootloader

ANNEXE II

Le code VHDL des bancs d'essai

ANNEXE III

Les scripts de compilation du code VHDL

ANNEXE IV

Les scripts PERL utilisés

ANNEXE V

Le fichier de contrainte top.ucf

BIBLIOGRAPHIE

[1] Altera Corporation. 2005. « Altera Partner Profile: CAST, Inc. ». ln *Altera Partner Profile: CAST, Inc.* En ligne.< http://www.altera.com/products/ip/ampp/cast/cast.html >. Consulté le 16 décembre 2005.

[2] ARM Limited. 1999. « AMBA™ Specification ». En ligne. Rev. 2.0. 230 p. < http://www.gaisler.com/doc/amba.pdf >. Consulté le 14 septembre 2005.

[3] Avnet Electronics Marketing. 2006. « Avnet Electronics Marketing - Xilinx® Spartan™-3 Evaluation Kit ». ln *Xilinx® Spartan™-3 Evaluation Kit*. En ligne. < http://www.em.avnet.com/evk/home/0,1719,RID%253D0%2526CID%253D7816%2526CCD%253DUSA%2526SID%253D4742%2526DID%253DDF2%2526SRT%253D1%2526LID%253D18806%2526PVW%253D%2526BID%253DDF2%2526CTP%253DEVK,00.html >. Consulté le 13 février 2006.

[4] Bourke, Paul. 1998. « BMP Image Format ». ln *BMP files*. En ligne. < http://local.wasp.uwa.edu.au/~pbourke/dataformats/bmp/ >. Consulté le 12 juin 2006.

[5] Cast inc. 2005. « TVOUT_CTRL Video Display Controller Core ». En ligne. 2 p, < http://www.cast-inc.com/cores/tvout_ctrl/cast_tvout_ctrl.pdf >. consulté le 15 décembre 2005

[6] Cast inc. 2005. « TVOUT_CTRL Video Display Controller Core ». ln *TVOUT_CTRL Video Display Controller Core*. En ligne. < http://www.cast-inc.com/cores/tvout_ctrl/index.shtml >. Consulté le 15 décembre 2005.

[7] Cohen, Ben. 2002. « Coding HDL for Reviewability ». ln *2002 MAPLD International Conference*. 49 p.

[8] CommentCaMarche.net. 2007. « Le format BMP ». ln *Le format BMP*. En ligne. < http://www.commentcamarche.net/video/format-bmp.php3 >. Consulté le 10 janvier 2007.

[9] Cypress. 2004. « CY7C1354B CY7C1356B ». En ligne. Rev *C, 29p. < http://download.cypress.com.edgesuite.net/design_resources/datasheets/contents/cy7c1354b.pdf >. Consulté le 22 mars 2006.

[10] Cypress. 2004. « PRELIMINARY CY7C1470V33 CY7C1472V33 CY7C1474V33 ». En ligne. 28 p.< http://download.cypress.com.edgesuite.net/design_resources/datasheets/contents/cy7c1474v33_8.pdf >. Consulté le 2 février 2007.

[11] Daub. 1998. « DAUB ». ln *DaubNET: File Formats Collection: BMP*. En ligne. < http://www.daubnet.com/formats/BMP.html >. Consulté le 12 juin 2006.

[12] Digilent Inc. 2006. « Virtex-II Pro Development System ». ln *Digilent Inc. - Digital Design Engineer's Source*. En ligne. < http://www.digilentinc.com/Products/Detail.cfm?Prod=XUPV2P&Nav1=Products&Nav2=Programmable >. Consulté le 14 février 2006.

[13] Digital Display Working Group. 1999. « Digital Visual Interface DVI ». En ligne. Revision 1.0. 76 p. < http://www.ddwg.org/lib/dvi_10.pdf >, Consulté le 12 octobre 2005.

[14] Engdahl, Tomi. 2005 « Video Timing Details Revealed ». ln *Video Timing*. En ligne. < http://www.tkk.fi/Misc/Electronics/faq/vga2rgb/timings.html >, Consulté le 14 septembre 2005.

[15] Engdahl, Tomi. 2005. « Calculator for video timings v. 1.8 ». ln *Video timings calculator*. En ligne. < http://www.tkk.fi/Misc/Electronics/faq/vga2rgb/calc.html >. Consulté le 14 septembre 2005.

[16] ePanorama.net. 2005. « Video Circuits ». In *ePanorama.net – Links*. En ligne. < http://www.epanorama.net/links/videocircuits.html >. Consulté le 21 septembre 2005.

[17] Expressive System. 2000. « VHDL Source for the following design sheet ». In *Expressive IV - VHDL Code*. En ligne. < http://www.parallelsystems.co.uk /expressive-vhdl.htm >. Consulté le 16 décembre 2005

[18] Gaisler Research. 2006. « GR-XC3S-1500 LEON Development board ». In *GR-XC3S-1500 LEON Development board - Gaisler Research*. En ligne. < http://www.gaisler.com/cms/index.php?option=com_content&task=view&id=187&Itemid=118 >. Consulté le 13 février 2006.

[19] Gaisler, Jiri , Sandi Habinc et Edvin Catovic. 2005. « GRLIB IP Library Users Manual ». En ligne. Version 1.0.3. 50 p. < http://gaisler.com/products/grlib /grlib.pdf >. Consulté le 27 octobre 2005.

[20] Garrault, Philippe et Brian Philofsky. 2005. « HDL Coding Practices to Accelerate Design Performance ». En ligne. < http://www.xilinx.com/support/documentation /white_papers/wp231.pdf >. Consulté le 22 décembre 2005.

[21] Hardi Electronics. 2006. « DVI_1x1 ». In *Haps*. En ligne. < http://www. hardi.com/haps/dvi_1x1.htm >. Consulté le 14 février 2006.

[22] Herveille, Richard. 2001. « Combining WISHBONE interface signals ». En ligne. Révision 0.2. 6 p. < http://www.opencores.org/projects.cgi/web/wishbone/app note_01.pdf >, Consulté le 21 septembre 2005.

[23] Herveille, Richard. 2002. « Specification for the: WISHBONE System-on-Chip (SoC) Interconnection Architecture for Portable IP Cores ». En ligne. Revision B.3. 140 p. < http://www.opencores.org/projects.cgi/web/wishbone/wbspec_b3.pdf >. Consulté le 14 septembre 2005.

[24] Herveille, Richard. 2003, « VGA/LCD Core v2.0 ». En ligne. Révision 1.2. 46 p. < http://www.opencores.org/cvsweb.shtml/vga_lcd/doc/vga_core.pdf >. Consulté le 21 septembre 2005.

[25] HiTech Global. 2005. « Virtex-II Pro RocketIO Optical Module ». In *Optical Module with Virtex 2 Pro RocketIO*. En ligne. < http://www.hitechglobal.com/TED/OpticalModule.htm >. Consulté le 15 décembre 2005

[26] HiTech Global. 2006. « Xilinx Spartan 3 Evaluation board for FPD interface and Image processing ». In *Xilinx Spartan 3 III Board with DVI & FFC*. En ligne. < http://www.hitechglobal.com/ted/tb3s1500img.htm >. Consulté le 14 février 2006.

[27] HiTech Global. 2006. « Xilinx Spartan 3E Display Solution / Image Processing Design Kit ». In *Spartan 3E Display Solution Board*. En ligne. < http://hitech global.com/TED/Spartan3E.htm >. Consulté le 14 février 2006.

[28] HiTech Global. 2006. « Xilinx Spartan 3™ Development Board ». In *Xilinx Spartan 3™ Development Board*. En ligne. < http://hitechglobal.com/Boards/GR-S31500.htm >. Consulté le 14 février 2006

[29] Hwang, Enoch. 2004. « Build a VGA Monitor Controller ». *Circuit Cellar*, Issue 172, November 2004, page 38-43.

[30] IBM Corporation. 1999. « The CoreConnectTM Bus Architecture ». En ligne. 8 p. < http://www-01.ibm.com/chips/techlib/techlib.nsf/techdocs/852569B20050FF77852569910050C0FB/$file/crcon_wp.pdf >. Consulté le 14 septembre 2005.

[31] IBM Corporation. 2005. « CoreConnect Bus Architecture ». In *CoreConnect Bus Architecture - IBM Microelectronics*. En ligne. < http://www-306.ibm.com/chips/techlib/techlib.nsf/productfamilies/CoreConnect_Bus_Architecture >. Consulté le 14 septembre 2005.

[32] IBM Corporation. 2008. « CoreConnect Bus Architecture ». ln *CoreConnect Bus Architecture - IBM Microelectronics*. En ligne. < http://www-01.ibm.com/chips /techlib/techlib.nsf/products/CoreConnect_Bus_Architecture >. Consulté le 26 février 2008.

[33] Intrinsix Corp. 2005, « VGA & SXGA Compatible HDL Cores ». En ligne. 7 p. < http://www.intrinsix.com/intrinsix-ip/vga/Intrinsix_VGA_and_SXGA_ Product_Overview.pdf >. Consulté le 16 décembre 2005.

[34] Intrinsix Corp. 2005. « Intrinsix VGA & SXGA Compatible HDL Cores for FPGA & ASIC ». ln *Intrinsix Corp. >|< VGA & SXGA Cores for ASIC and FPGA*. En ligne. < http://www.intrinsix.com/intrinsix-ip/vga/index.htm >. Consulté le 16 décembre 2005.

[35] Maria, Bolovis. 2005. « ARINC 429 RX ». Rapport de projet de fin d'études soumis comme condition partielle à l'obtention du diplôme de baccalauréat, Montréal, École Polytechniquede Montréal. 183 p.

[36] Marshall, A.D. 2005. « Practical Perl Programming ». ln *Practical Perl Programming*. En ligne. < http://www.cs.cf.ac.uk/Dave/PERL/perl_caller.html >, Consulté le 7 juin 2006.

[37] Microtronix. 2005. « Sendero Evaluation Kit ». ln *Sendero Evaluation Kit*. En ligne. < http://www.microtronix.com/acatalog/Standard_Development _Kits.html >. Consulté le 4 janvier 2006.

[38] Mistral Software Pvt. Ltd. 2004. « VGA-XGA Controlller ». En ligne. 2 p. < http://www.mistralsoftware.com/VGA-XGA_Controller.pdf >. Consulté le 10 novembre 2005.

[39] Opencores.org. 2002. « OpenCores Coding Guidelines ». En ligne. 25 p. < http://www.opencores.org/cvsget.cgi/common/opencores_coding_guidelines .pdf >. Consulté le 22 décembre 2005.

[40] Opencores.org. 2005. « VGA/LCD Controller: Overview ». In *OPENCORES.ORG*. En ligne. < http://www.opencores.org/projects.cgi/web/vga_lcd/overview >. Consulté le 21 septembre 2005.

[41] Perl.org. 2006. « The Perl Directory at Perl.org ». In *The Perl Directory - Perl.org*. En ligne. < http://www.perl.org/ >. Consulté le 7 juin 2006.

[42] Perreault, Martin. 2005. « ELE-748 Architecture des systèmes ordinés et VHDL: Laboratoire : Acquisition de données et visualisation ». Reçu par courriel de la part de M. Renaud Lavoie, chargé de laboratoire de ce cours, à l'ETS. 8 p. < Laboratoire v102.pdf >. Consulté le 5 octobre 2005.

[43] Silica (Avnet). 2006. « SP3 Development Kit ». In *SP3 Development Kit*. En ligne. < http://www.silica.com/en/products/evaluationkits/SP3%20Development%20kit.html >. Consulté le 14 février 2006.

[44] Singh, Deshanand et Aaron Egier. 2004. « A Simple VGA Controller ». In *VGA Controller*. En ligne. < http://www.eecg.utoronto.ca/~singhd/241/vgacon.htm >. Consulté le 21 septembre 2005.

[45] Singh, Deshanand et Aaron Egier. 2004. « A Simple VGA Controller for the Altera UP2 Board ». In *VGA Controller*. En ligne. < http://www.eecg.toronto.edu/~jayar/ece241_05F/vga_new/index.html >. Consulté le 25 Février 2008.

[46] Talarek, Wieslaw, Intel Corporation. 30 septembre 2003. *SERDES (serializer/deserializer) time domain multiplexing/demultiplexing technique*. US Patent 6628679. En ligne. Pattern Storm. < http://www.patentstorm.us/patents/6628679-description.html >. Consulté le 14 septembre 2005.

[47] Usselmann, Rudolf. 2001. « OpenCores SoC Bus Review ». En ligne. Revision 1.0, 12 p. < http://www.opencores.org/projects.cgi/web/wishbone/soc_bus_comparison.pdf >. Consulté le 14 septembre 2005.

[48] XESS Corp. 2004. « VGA Signal Generation with the XS Board ». En ligne. 18 p. < http://www.xess.com/appnotes/vga.pdf >. Consulté le 14 septembre 2005.

[49] XESS Corp. 2005. « XESS homepage announcements ». ln *XESS homepage announcements*. En ligne. < http://www.xess.com >. Consulté le 14 septembre 2005.

[50] Xilinx. 2001. « logiCVC Compact Video Controller ». En ligne. 5 p. < http://www.xilinx.com/products/logicore/alliance/xylon/xylon_logicvc.pdf >. Consulté le 15 décembre 2005.

[51] Xilinx. 2003. « Virtex II Pro Engineering Sample Errata Sheet: XC2VP4 and XC2VP7 ES Devices ». En ligne. Rev 1.6, 1 p. < https://secure.xilinx.com/webreg/clickthrough.do?cid=36705&cancellink=http%3A%2F%2Fwww.xilinx.com%2Fsupport%2Fdocumentation%2Findex.htm >. Consulté le 22 mars 2006.

[52] Xilinx. 2004, « ml401_2_3_schematics ». En ligne. v 1.0, 24 p. < http://www.xilinx.com/support/documentation/boards_and_kits/ml401_2_3_schematics.pdf >. Consulté le 22 mars 2006.

[53] Xilinx. 2004. « ml401_2_3_gerbers ». En ligne. v 8.0.1, 16 p. < http://www.xilinx.com/support/documentation/boards_and_kits/ml401_2_3_gerbers.pdf >, Consulté le 22 mars 2006.

[54] Xilinx. 2004. « Reading User Data from Configuration PROMs ». En ligne. v1.0. 11 p. < http://www.xilinx.com/support/documentation/application_notes/xapp694.pdf >. Consulté le 2 mai 2006.

[55] Xilinx. 2005, « SP305_Gerber », En ligne. Rev B, 16 p. < http://www.xilinx.com/support/documentation/boards_and_kits/SP305_Gerber.pdf >, Consulté le 22 mars 2006.

[56] Xilinx. 2005. « Platform Flash In-System Programmable Configuration PROMs ». En ligne. v2.8. 47 p. < http://www.xilinx.com/support/documentation/data_sheets/ds123.pdf >. Consulté le 27 mars 2006.

[57] Xilinx. 2005. « SCH. SP305 EVAL BOARD 0381198 ». En ligne. v1.0, Rev 01. 26 p. < http://www.xilinx.com/support/documentation/boards_and_kits/SP305_Schematic.pdf >. Consulté le 22 mars 2006.

[58] Xilinx. 2005. « SP305 Spartan-3 Development Platform User Guide ». En ligne. v1.0, 42 p. < http://www.xilinx.com/support/documentation/boards_and_kits/ug216.pdf >. Consulté le 22 mars 2006.

[59] Xilinx. 2005. « Spartan-3 FPGA Family: Complete Data Sheet ». En ligne. v1.7. 204 p. < http://www.xilinx.com/support/documentation/data_sheets/ds099.pdf >. Consulté le 22 mars 2006.

[60] Xilinx. 2005. « Spartan-3 SP305 Development Board ». In *Spartan-3 SP305 Development Board.* En ligne. < http://www.xilinx.com/xlnx/xebiz/designResources/ip_product_details.jsp?key=HW-SP305-US >. Consulté le 2 décembre 2005.

[61] Xilinx. 2006. « AR #19865 - Virtex-4 Configuration - IDCODE, verify, or other JTAG instruction failures on devices in a chain with Virtex-4 ES parts are due to TDO transitions on the rising edge of TCK as it exits SIR or SDR states (IEEE1149 compliance) ». In *AR #19865 - Virtex-4 Configuration - IDCODE, verify, or other JTAG instruction failures on devices in a chain with Virtex-4 ES parts are due to TDO transitions on the rising edge of TCK as it exits SIR or SDR states (IEEE1149 compliance).* En ligne. < http://www.xilinx.com/support/answers/19865.htm >. Consulté le 16 février 2006.

[62] Xilinx. 2006. « AR #20060 - 9.1i EDK - "WARNING:MDT - MDM Peripheral Not Detected on Hardware - XMD does not connect to MicroBlaze via the MDM module on Virtex-4 ». In *AR #20060 - 9.1i EDK - "WARNING:MDT - MDM Peripheral Not Detected on Hardware - XMD does not connect to MicroBlaze via the MDM module on Virtex-4*. En ligne. < http://www.xilinx.com/support/answers/20060.htm >. Consulté le 16 février 2006.

[63] Xilinx. 2006. « AR #20129 - Virtex-4 - LX25's BSCAN_VIRTEX4 RESET and DRCK signals might not function as expected on the ES parts ». In *AR #20129 - Virtex-4 - LX25's BSCAN_VIRTEX4 RESET and DRCK signals might not function as expected on the ES parts*. En ligne. < http://www.xilinx.com/support/answers/20129.htm >. Consulté le 16 février 2006.

[64] Xilinx. 2006. « AR #22462 - Virtex-4 - Why are the FIFO16 flags not working correctly? ». In *AR #22462 - Virtex-4 - Why are the FIFO16 flags not working correctly?*. En ligne. < http://www.xilinx.com/support/answers/22462.htm >. Consulté le 16 février 2006.

[65] Xilinx. 2006. « ML40x Known Issues ». In *Xilinx: Pruducts: Development Boards: Xilinx ML40x Known Issues*. En ligne. < http://www.xilinx.com/products/boards/ml40x/ml40x_known_issues.htm >. Consulté le 16 février 2006.

[66] Xilinx. 2006. « Using Digital Clock Managers (DCMs) in Spartan-3 FPGAs ». En ligne. v1.1 68 p. < http://www.xilinx.com/support/documentation/application_notes/xapp462.pdf >. Consulté le 29 juin 2006.

[67] Xilinx. 2006. « Virtex-4 DCM - DCM does not lock when DFS outputs are used and input clock is outside of DLL output range ». In Virtex-4 DCM - DCM does not lock when DFS outputs are used and input clock is outside of DLL output range. En ligne. < http://www.xilinx.com/support/answers/23624.htm >. Consulté le 16 février 2006.

[68] Xilinx. 2006. « Virtex-4 ES, Errata - All designs must be compiled in the ISE 7.1i Service Pack 1 or later and DCM requirement ». In Virtex-4 ES, Errata - All designs must be compiled in the ISE 7.1i Service Pack 1 or later and DCM requirement. En ligne. < http://www.xilinx.com/support/answers/20529.htm >. Consulté le 16 février 2006.

[69] Xilinx. 2006. « Virtex-4 ML402 SX XtremeDSP Evaluation Platform ». In *Virtex-4 ML402 SX XtremeDSP Evaluation Platform*. En ligne. < http://www.xilinx.com/xlnx/xebiz/designResources/ip_product_details.jsp?sGlobalNavPick=PRODUCTS&sSecondaryNavPick=BOARDS&iLanguageID=1&key=HW-V4-ML402-USA >. Consulté le 14 février 2006.

[70] Xilinx. 2006. « Virtex-4 Video Starter Kit: Ideal hardware platform to evaluate Xilinx FPGAs in a wide range of Video and Imaging applications ». In *Virtex-4 Video Starter Kit*. En ligne. < http://www.xilinx.com/xlnx/xebiz/designResources/ip_product_details.jsp?key=HW-V4SX35-VIDEO-SK-US&sGlobalNavPick=PRODUCTS&sSecondaryNavPick=BOARDS >. Consulté le 14 février 2006.

[71] Xilinx. 2007. « Timing Constraints ». In Timing Constraints. En ligne < http://toolbox.xilinx.com/docsan/xilinx4/data/docs/cgd/types2.html >. Consulté le 31 mai 2007.

[72] Xylon. 2005. « logicBRICKS Designed by XYLON : products ». In *logicBRICKS*. En ligne. < http://www.logicbricks.com/html/products.htm >. Consulté le 15 décembre 2005.