Rapport TER :

Implémentation en temps réel d'un filtre de Sobel sur FPGA

Omar MEGLOUD , M1 SETSIS

Professeur encadrant :
François BERRY

Année universitaire : 2020-2021

Remerciements

Je tiens tout d'abord à remercier Monsieur François Berry enseignant chercheur à l'université Clermont Auvergne et responsable de formation SETSIS, pour m'avoir encouragé à travailler sur ce sujet de Travail Encadré de Recherche, et la mise à disposition du matériel nécessaire pour accomplir mon travail.

Je souhaite aussi adresser mes remerciements les plus sincères au corps professoral de la première année SETSIS, pour la richesse et la qualité de leur enseignement et qui déploient de grands efforts pour assurer à leurs étudiants une formation actualisée.

Finalement, j'adresse mes remerciements à toutes les personnes qui m'ont aidé dans la réalisation de ce Travail Encadré de Recherche.

Table des matières

Liste des figures

Liste des tableaux

Acronymes

ASIC	Application-Specific Integrated Circuit
CPU	Central Processing Unit
DDC	Display Data Channel
DSP	Digital Signal Processor
FIFO	First In First Out
FPGA	Field-Programmable Gate Array
FSM	Finite State Machine
GPIO	General Purpose Input Output
GPU	Graphics Processing Unit
HDL	Hardware Description Language
HPS	Hard Processor System
I2C	Inter-Integrated Circuit
IP	Intellectual Property
PLL	Phase Locked Loop
RAM	Random Access Memory
RTL	Register Transfer Level
SDRAM	Synchronous Random Dynamic Access Memory
SoC	System on Chip
SQRT	Square root
VGA	Video Graphics Array

Introduction

Les applications embarquées font appel à des algorithmes pour traiter les différents types de données et utilisent des algorithmes de plus en plus sophistiqués dont la complexité n'a cessé de croître. Du fait de cet accroissement de la complexité, ces applications et en particulier celles de traitement d'images ou de vision, demandent toujours de plus en plus de puissance de calcul. Cet aspect s'avère encore plus critique lorsque l'implémentation de ces algorithmes nécessite un traitement en temps réel.

Actuellement, les algorithmes de traitement d'images présentant une contrainte temps réel vidéo voient leurs implémentations s'orienter vers des architectures numériques dédiées. Le choix de l'architecture est en fonction de nombreux critères fortement dépendants de l'application. Les solutions adoptées en termes de techniques et technologie, dépendent fortement du contexte et des contraintes de ces systèmes. Un autre facteur important sur le choix d'une solution est bien sûr celui de la capacité du système.

L'implémentation d'algorithme de traitement d'images en temps réel sur des processeurs à exécution série est de moins en moins pratique, car les applications de traitement d'images exigent plusieurs opérations sur chaque pixel, il en résulte une charge extrêmement lourde à gérer pour un seul processeur. Une bonne alternative consiste à utiliser des cartes FPGA.

Le FPGA est le circuit logique programmable le plus couramment utilisé de nos jours, de part ses capacités, sa vitesse et sa grande flexibilité. Ces architectures mixent généralement des composants matériels et logiciels travaillant en concurrence afin de répondre le plus efficacement à la contrainte temporelle imposée. Aussi, les FPGA coûtent généralement moins cher en termes de fabrication et de maintenance que les CPUs et GPUs. Les caractéristiques de reconfiguration d'un FPGA offrent plus de flexibilité que les ASIC qui ne peuvent être reprogrammés après leur fabrication.

Le but de ce TER est d'implémenter un filtre de Sobel sur une carte FPGA en temps réel.

Afin d'illustrer la démarche de travail, nous présentons dans ce qui suit l'organisation de mon rapport :

La première partie : une discussion à propos des différentes cibles embarquée pour le traitement de l'image.

La deuxième partie : une présentation du matériel utilisé dans le projet.

La troisième partie : une étude de l'interface caméra.

La quatrième partie: l'implémentation de l'algorithme.

1. Les cibles embarquées pour le traitement d'images

1.1 FPGA, CPU et GPU

Les trois choix les plus courants pour les plateformes de traitement d'images dans les applications de vision industrielles sont l'unité centrale de traitement (CPU), l'unité de traitement graphique (GPU) et les cartes FPGA. Les avantages et inconvénients de l'implémentation des algorithmes de traitement d'images sur FPGA par rapport aux CPUs et GPUs seront discutés dans cette partie.

Les FPGAs sont généralement considérés comme un dispositif parallèle de bas niveau qui offre une flexibilité permettant de programmer l'implémentation de n'importe quel circuit logique. Ainsi, il est aussi parfois utilisé pour prototyper des conceptions ASIC.

Cependant, la conception FPGA est généralement considérée comme difficile à utiliser par rapport aux autres plateformes, car les systèmes FPGA doivent être développés en utilisant des langages matériels. Les langages les plus courants pour les FPGA sont VHDL et Verilog, appelés langages de description matériels. La principale différence entre ces langages et les langages de programmation traditionnels est que les HDL décrivent le matériel, c'est-à-dire les registres et les fonctions logiques booléennes, tandis que les langages de programmation décrivent les instructions séquentielles sans connaître les détails précis de l'implémentation du matériel. Grâce à leur grande communité de développeurs, les

La structure est un élement important qui influe sur le choix entre FPGA, CPU et GPU. Une architecture adaptée à l'application ciblée peut considérablement améliorer ses performances et réduire le coût du système. Pour le traitement des images, qui peut être considéré comme un traitement de signal 2D, une bande passante mémoire élevée est requise pour les opérations de traitement gourmandes en mémoire. La figure 1 illustre l'architecture d'un processeur et d'un processeur graphique. Le CPU et le GPU ont tous les deux des unités de contrôle, des unités arithmétiques et logiques, une DRAM et une mémoire cache. Mais la différence entre le CPU et le GPU est qu'une grande proportion de la puce GPU est composé de unités arithmétiques et logiques. les FPGAs affichent de bonnes performances sur le parallélisme, mais les besoins en bande passante mémoire son beaucoup plus faibles que lorsqu'ils sont implémentés avec un CPU ou un GPU.

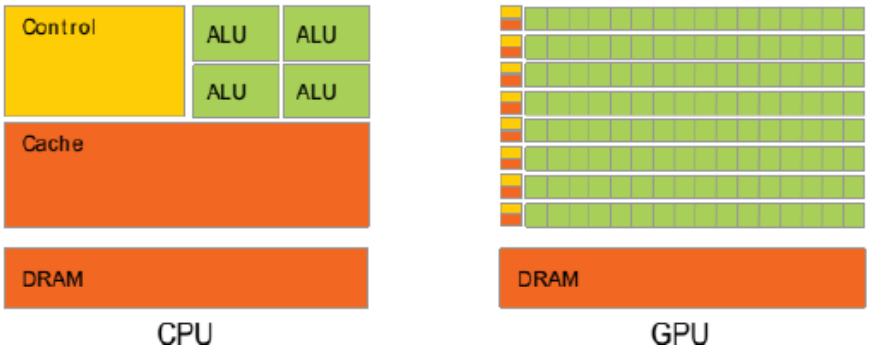

Figure 1: Architecture CPU et GPU

Les différences entre CPU et GPU sont dues à leurs différents objectifs de conception. Les CPUs doivent être très polyvalents pour traiter différents types de données. Cela rend la structure interne du CPU extrêmement compliqué. Alors que les GPUs font face à une grande échelle de données qui est fortement unifié et mutuellement indépendante et fonctionne dans

un environnement pur avec des interruptions externes limitées. Comparé au CPU et GPU, la structure de le FPGA est plus simple. Le choix de la cible dépend de la complexité de la tâche requise. Par exemple si un système de traitement d'images en temps réel nécessite une haute résolution et des calculs complexes, le GPU est plus adapté, mais si le système ne nécessité que des algorithmes de bas niveau basé sur la convolution, le FPGA devient plus approprié.

Par conséquent, on pense que le FPGA a de meilleures performances dans les applications de traitement d'images car ces algorithmes peuvent exploiter un grand degré de parallélisme.

En effet, le traitement parallèle est une méthode permettant de séparer et d'exécuter simultanément des tâches de programme sur plusieurs microprocesseurs, réduisant ainsi le temps de traitement. La capacité d'exécuter les tâches sur des millions de portes logiques est ce que fait le FPGA. L'un des avantages des FPGAs réside dans le fait qu'ils sont beaucoup plus rapides pour certaines applications en raison de leur nature parallèle et leur faible latence. Ils offrent également un potentiel de traitement nettement plus important avec une consommation d'énergie beaucoup plus faible.

1.2 Performance

Il y a divers éléments qui devraient être pris en compte lors de la mesure du fonctionnement d'un système. En ce qui concerne le traitement d'images en temps réel, des facteurs tels que la vitesse et l'efficacité énergétique sont très importants.

La vitesse d'exécution d'un algorithme de traitement d'images implémenter sur une des cibles embarqués citées précedament varie selon chaque application. De façon générale, les FPGAs présentent un parallélisme extrêmement élevé à une fréquence d'horloge inférieure ; en comparaison, le GPU est doté d'une vitesse d'horloge élevée, en plus d'un parallélisme relativement élevé, mais est limité par une mauvaise gestion de la mémoire. Les CPUs au contraire ont une vitesse d'horloge relativement élevée facilitant la gestion de la mémoire mais leur parallélisme est limité. Pour un système de traitement d'images qui contient de nombreux parallélismes inhérents, l'avantage d'un CPU n'est pas évident. Cela permet de comparer les GPUs et les FPGAs. Par exemple, les GPU ont une meilleure mise en oeuvre dans les filtres normalisés de corrélation croisée et bidimensionnels, alors que les FPGA ont été identifiés comme ayant des performances exceptionnelles dans les calculs de filtres adaptés, le partitionnement en k- means et la stéréovision.

Le tableau 1 présente une comparaison de la consommation d'énergie entre un CPU, un GPU et un FPGA qui implémentent un filtre de Canny à différentes résolutions. Les chercheurs utilisent un processeur Intel Core2 Duo E6600, 2,4 GHz ; un GPU GeForce GTX 580, 1,54 GHz et un périphérique FPGA Arria V 5AGXFB3. La consommation d'énergie du FPGA reste presque la même à 1,5 watts, l'effet du changement du système étant négligeable. En attendant, la consommation électrique du processeur et du processeur graphique reste autour de 150 watts et 240 watts. Ainsi, les consommations d'énergie du CPU sont plus de mille fois supérieures à celles du FPGA. Avec les avantages de l'architecture, l'efficacité énergétique du GPU est meilleure que celle du CPU, mais elle reste incomparable avec le FPGA. Cependant, avec des résolutions plus élevées, la consommation d'énergie du FPGA augmente rapidement, soit 98,2 Millijoules à 3936x3936, soit 14,7 fois plus que la performance à 1024x1024. Mais pour le GPU, il augmente d'un facteur de 2,5, la différence de consommation d'énergie entre le FPGA et le GPU est réduite progressivement.

Resolution	CPU		GPU		FPGA	
	Puissance(W)	Energie(J)	Puissance(W)	Energy(J)	Puissance(W)	Energy(J)
512x512	141	2.8	231	0.5	1.5	1.7
1024x1024	147	8.8	244	6.08	1.5	6.7
3936x3936	153	213.1	251	15.0	1.5	98.2

Tableau 1: Comparaison de la consommation d'énergie pour les implémentations du filtre de Canny

2. Matériel utilisé pour le projet

2.1 Carte de développement DE1-SoC Cyclone V

La carte FPGA utilisée pour ce projet est une carte de développement DE1-SoC cyclone V.

Le kit de développement DE1-SoC présente une plate-forme de conception matérielle robuste construite autour du système Altera System-on-Chip (SoC) FPGA, qui combine le dual-core Cortex-A9 avec une logique programmable de pointe pour une flexibilité de conception ultime. Les utilisateurs peuvent désormais tirer parti de la puissance d'une reconfiguration exceptionnelle associée à un processeur hautes performances et basse consommation. Le SoC d'Altera intègre un HPS avec une architecture ARM, composé d'un processeur, de périphériques et interfaces de mémoire associés à la structure FPGA à l'aide d'une bande passante élevée. La carte de développement DE1-SoC est équipée d'une mémoire DDR3 de haute vitesse, de capacités vidéo et audio, de réseau Ethernet et ainsi que d'autres fonctionnalités qui promettent de nombreuses applications.

Voici certaines caractéristiques de la carte développement :

- FPGA : Cyclone V
- ARM 925 MHz Cortex-A9 double coeur
- Mémoire locale maximale de 64 Mo
- Mémoire globale de 1 Go
- Blocs DSP de précision variable...

Pour le développement, le logiciel Quartus II 15.0 (64 bit) est utilisé.

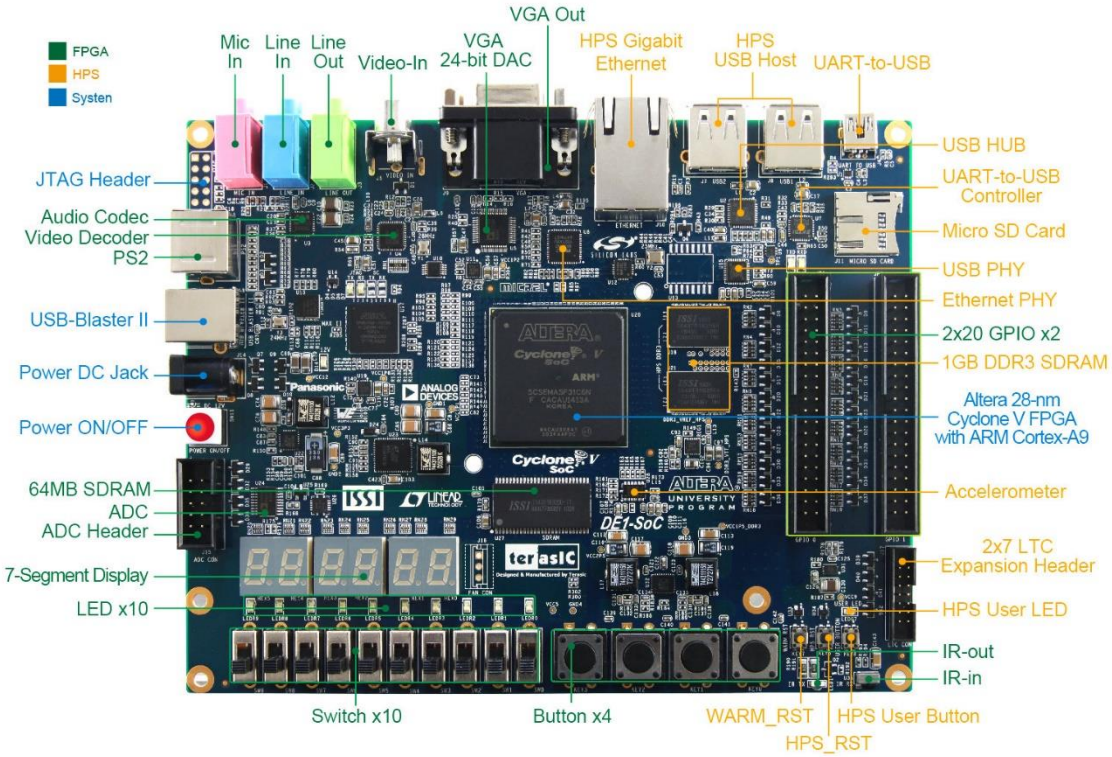

Figure 2: Carte de développement DE1-SoC

2.2 TRDB-D5M CMOS caméra

La caméra Terasic TRDB-D5M est un module de 5 mégapixels basé sur la technologie CMOS et qui fonctionne sur une alimentation de 3.3V. Le module caméra a une résolution de 2592x1944 pixels avec une fréquence d'images maximale de 15 fps. La sortie du module est au format « RGB Bayer Pattern ».

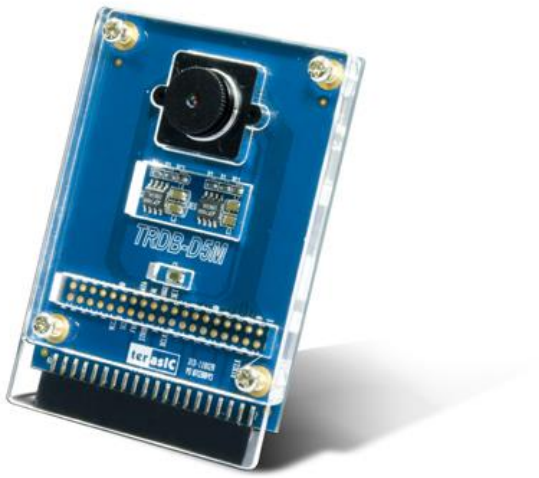

Figure 3: TRDB-D5M CMOS caméra

3. Interface de la caméra TRDB-D5M

Cette section explique les composants qui interfacent la caméra avec la carte, pour envoyer les données des images capturées par le capteur à la carte DE1-SoC et les transformer du format de couleur « Bayer » au format RGB avant de stocker les données dans la SDRAM de la carte DE1-SoC. Le module caméra est connecté au port GPIO de la carte de développement.

La figure 4 ci-dessous représente l'architecture générale du système.

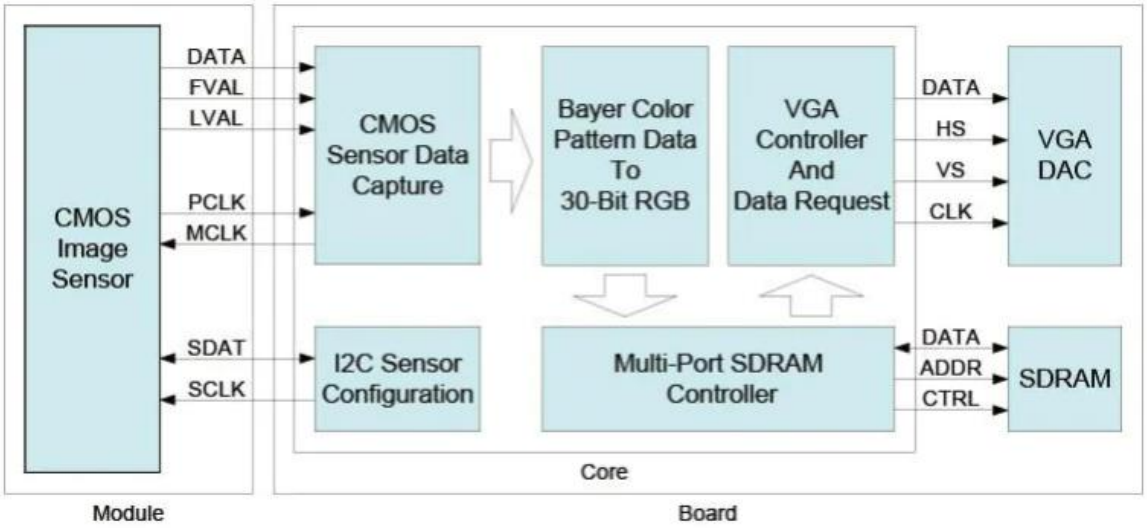

Figure 4: Shéma synoptique de l'interface caméra

Les signaux d'horloges des différents modules de l'architecture sont ajustés et adaptés à l'aide des IP PLL, chacun selon ses exigences de fonctionnement.

3.1 Capture des images

Les images sont d'abord capturées par le capteur CMOS de la caméra et sont transmises au module de capture d'image, où les informations de pixels valides sont extraites en fonction des signaux de contrôle de validité. Les données de sortie du capteur sont divisées en images, qui sont ensuite divisées en lignes. Par défaut, le capteur produit 1944 lignes de 2592 colonnes chacune. Les signaux FRAME_VALID et LINE_VALID indiquent respectivement les limites entre les images et les lignes. Les pixels sont produits dans un format de couleur « Bayer » composé de quatre "couleurs" : Vert1, Vert2, Rouge et Bleu (G1,G2, R, B) : représentant trois couleurs de filtre, ils sont ensuite convertis en format RGB.

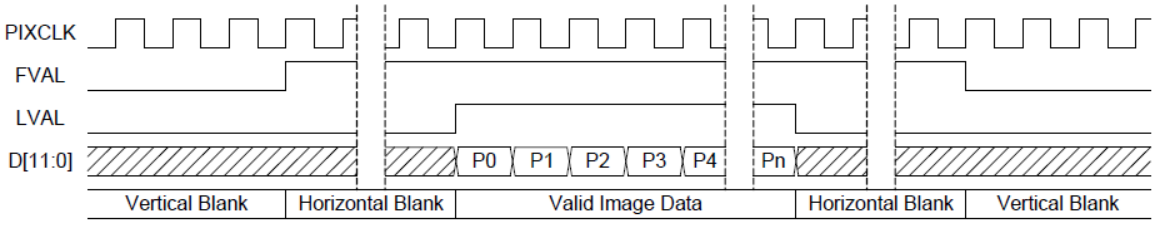

Figure 5: Chronogramme d'acquisition des pixels

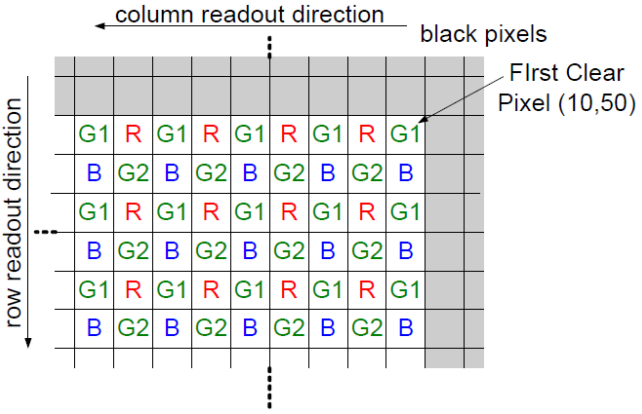

Figure 6: Format de couleur des pixels

3.2 Configuration du module caméra

Le capteur d'image CMOS prend est doté une interface série à deux fils basée sur le protocole I2C. Cette interface série est utilisé pour lire est écrire sur les registres du capteur. Le bus I2C est composé de deux lignes bidirectionnelles, SDA (Serial Data) et SCL (Serial Clock). À travers cette interface on peut configurer par exemple la résolution des images, le nombre d'images par seconde, etc.

3.3 Stockage des images

La carte de développement DE1-SoC est doté d'une mémoire SDRAM (32Mx16) externe de 64 Mo. Cette mémoire est connectée au FPGA via un bus de données de 16 bits (DRAM_DQ), des lignes de contrôle et un bus d'adresse de 13 bits (DRAM_ADDR). L'interface FPGA-SDRAM est affichée dans la figure 7. Les signaux de synchronisation et de contrôle des mémoires sont un peu compliqués, par conséquent, dans ce projet j'ai utilisé un module de contrôleur SDRAM déjà conçu.

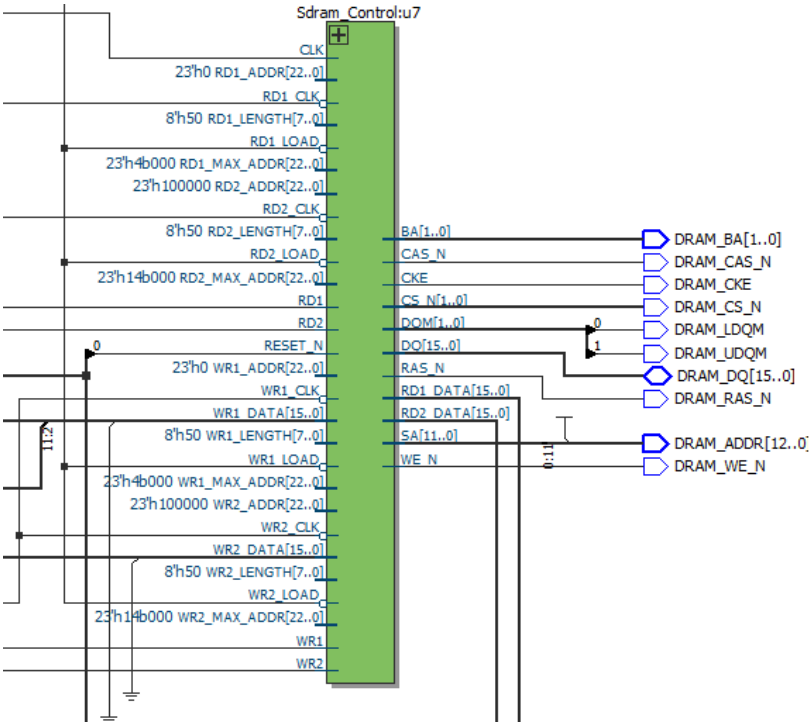

Figure 7: Vue RTL de l'interface FPGA-SDRAM

Dans ce projet spécifique, les données de sorties du module RAW2RGB sont fournies à la fréquence PIXCLK. La SDRAM et le contrôleur SDRAM fonctionnent par défaut à 100 MHz. Pour surmonter les incohérences de fréquence d'horloge, les mémoires FIFO sont utilisés pour la mise en tampon entre les circuits.

Les informations RGB de chaque pixel ont un total de 30 bits (10 bits par couleur). La caméra, avec les configurations par défaut, utilise seulement deux des quatre banques dans la SDRAM. Pour chacune des banques, il utilise deux FIFO - un pour la lecture et un pour l'écriture. L'information RGB d'un pixel est divisée par les deux FIFOS et banques : Les 10 bits de rouge et les 5 bits de vert sont conservés dans une banque, et les 10 bits de bleu et les 5 bits restants du vert sont conservés sur l'autre banque.

Le contrôleur SDRAM génère les signaux de contrôle (CAS_N, RAS_N, WE_N, etc.), les adresses mémoire (BA[1.0], SA[11.0]) et les commandes pour la SDRAM à partir des informations reçues des FIFO de lecture et d'écriture. Les données lues à partir de la SDRAM sont envoyées aux sorties RD1_DATA et RD2_DATA, et les données à écrire dans la SDRAM sont fournies par les entrées WR1_DATA et WR2_DATA. À ce niveau d'abstraction, l'utilisateur peut uniquement se soucier de l'envoi et de la réception de signaux à destination et en provenance des FIFO.

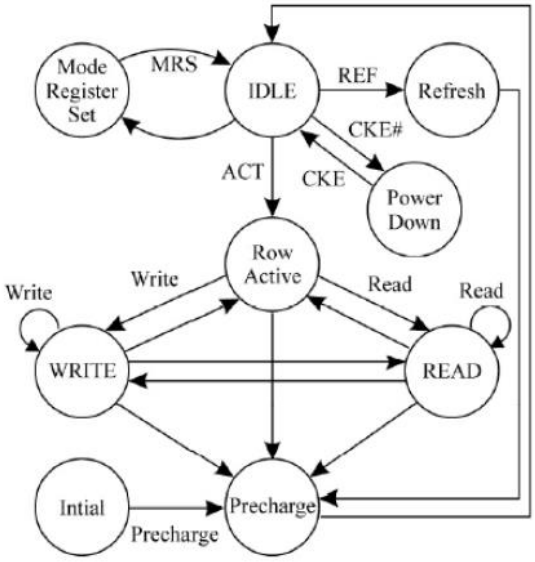

Figure 8: FSM du contrôleur SDRAM

3.4 Interface VGA

VGA est une interface standard introduite par IBM en 1987 pour connecter des ordinateurs à des moniteurs vidéo analogiques.

Le connecteur VGA possède 15 broches utilisées pour transmettre les signaux RGBHV + DDC2:

- Les composantes vidéo analogiques RGB.
- La synchronisation horizantale Hsync.
- La synchronisation verticale Vsync.
- Les signaux numériques DDC2 d'identification des moniteurs.

Les signaux Hsync et Vsync sont responsables de déterminer quand une nouvelle ligne ou une nouvelle trame doit commencé.

Les informations en RGB sorties du module RAW2RGB, seront utilisé ultérieurement comme entrée pour le module contrôleur VGA.

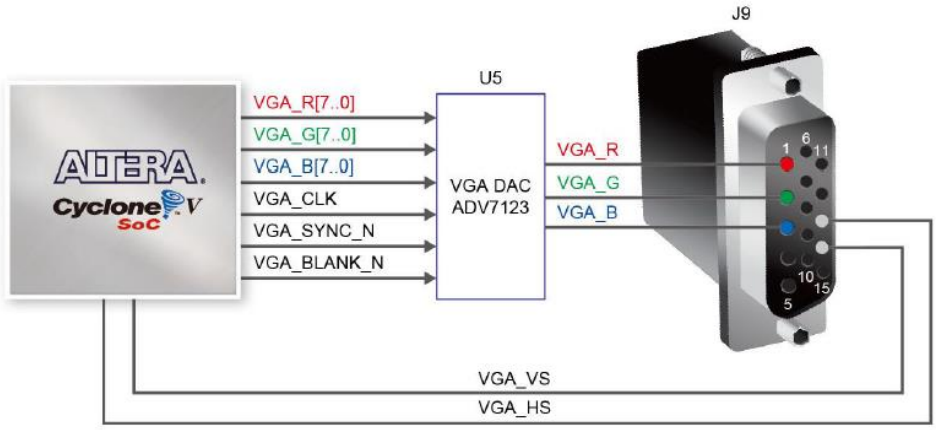

Figure 9: Connection entre FPGA et VGA

4. Implémentation du filtre de Soble dans la carte de développement DE1-SoC

4.1 Filtre de Sobel

L'opérateur Sobel est utilisé pour effectuer une mesure de gradient spatial 2D sur une image afin de mettre en évidence les régions de fréquence spatiale élevée, qui correspondent aux bords. Techniquement, il s'agit d'un opérateur de différenciation discrète, calculant une approximation du gradient de la fonction d'intensité d'image. L'utilisation de cet opérateur Sobel à n'importe quel pixel d'une image produira le vecteur de niveaux de gris correspondant et son vecteur normal.

L'opérateur se compose d'une paire de noyaux à convolution 3x3 comme indiqué dans la figure ci-dessous. Les deux filtres calculent les arêtes verticales (la gauche) et horizontales (la droite). Les filtres sont liés à l'ensemble de l'image d'entrée. Les pixels du milieu sont pondérés plus fortement que les pixels extérieurs, car ils sont plus proches du pixel au centre de la grille et ont donc un impact plus important sur le dégradé de bord pour ce pixel.

-1	0	+1
-2	0	+2
-1	0	+1

Gx

+1	+2	+1
0	0	0
-1	-2	-1

Gy

Figure 10: filtre de Sobel

Le calcul détaillé est présenté ci-dessous :

Gx = [f(x+1,y-1)+2*f(x+1,y)+f(x+1,y+1)]-[f(x-1,y-1)+2*f(x-1,y)+f(x-1,y+1)]

Gy = [f(x-1,y-1) + 2f(x,y-1) + f(x+1,y-1)]-[f(x-1, y+1) + 2*f(x,y+1)+f(x+1,y+1)]

G = sqrt(Gx^2+Gy^2)

4.2 Module de traitement d'image

le module de traitement d'images est responsable de tous les algorithmes de traitement d'images et des calculs. Ce module lit les données de la SDRAM, calcule les pixels et les transfère ensuite dans le contrôleur VGA. Le module de traitement d'images est constitué de plusieurs sous modules. Le sous module de détection des bords calculera le bord à l'aide de l'opérateur Sobel, et il a un tampon de 3 lignes pour s'assurer de la lecture de trois lignes à la fois afin d'améliorer la vitesse du calcul de l'écran entier. En utilisant l'opérateur Sobel à la fois dans la direction verticale et horizontale, la convolution Gx et Gy est calculée par un Multiplicateur-Additionneur, puis additionnée par un additionneur parallèle. Enfin, la racine carrée est effectuée par un module SQRT, et si le résultat est plus grand qu'un certain seuil défini, il serait considéré comme un pixel de bord.

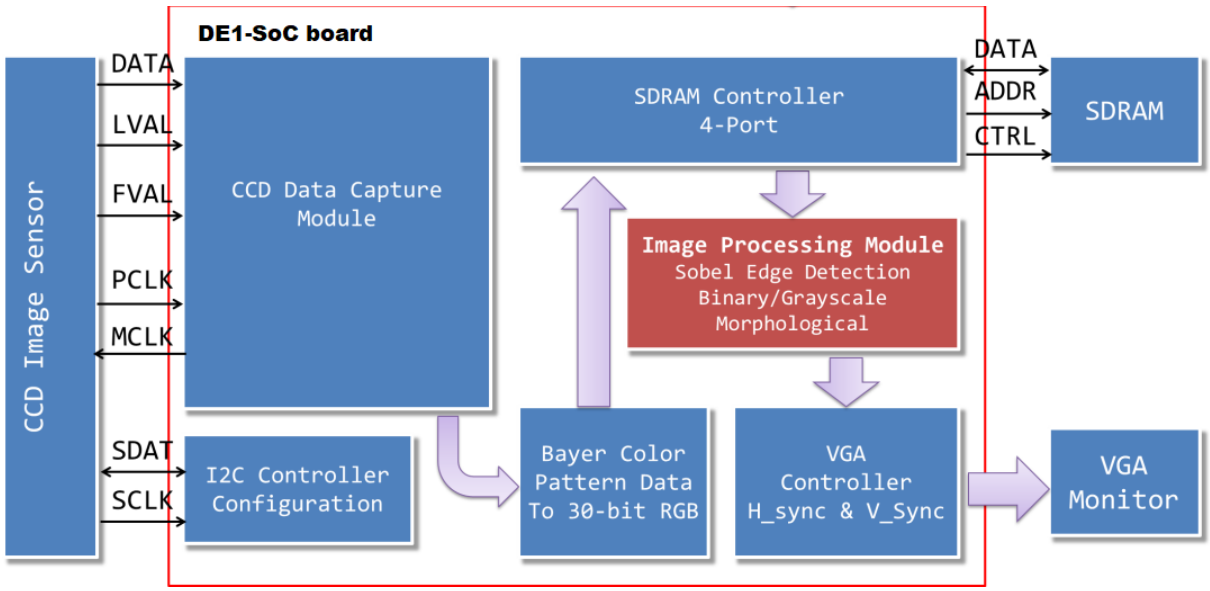

Figure 11: Diagramme d'architecture du système

4.2.1 Mémoire tampon (Buffer)

Un buffer est une zone mémoire virtuelle ou de disque dur utilisée pour stocker temporairement des données, notamment entre deux processus ne fonctionnant pas à la même vitesse.

Le buffer à trois lignes conçu dans notre projet est simplement une série de 3 grands registres construits à l'aide du MegaWizard Plugin Manager. Le tampon est en fait un registre de décalage basé sur la RAM, et le type de blocs de RAM est M4K, qui est optimisé par le compilateur. Les registres sont constitués de 640 pixels par ligne et contiennent les informations sur les pixels pour toute la ligne de l'écran VGA. Chaque pixel a 30 bits d'information, qui se compose de la couleur de rouge, vert et bleu, 10 bits pour chaque couleur. Si le module est activé, alors sur le front de l'horloge notre entrée sera stockée dans le premier pixel de ligne1, avec le reste des pixels décalant vers les autres lignes. Ces informations sur les pixels sont

ensuite utilisées dans le module de niveau supérieur et affichées à l'écran. Nous produisons également une grille de pixels, qui sont plus tard utilisés dans nos modules de détection de bord.

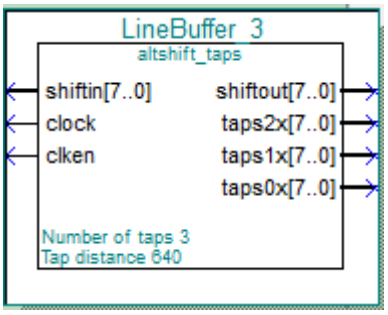

Figure 12: mémoire tampon

4.2.2 Multiplicateur-Additionneur

Le Multiplicateur-Additionneur est également généré par MegaWizard Plugin Manager. Il y a trois multiplicateurs, et un additionneur utilisé dans le module. Les bits de bus des deux entrées sont de 8 bits, tandis que la sortie est de 18 bits. Dans l'algorithme de détection des contours de Sobel, la convolution est effectuée sous la forme d'une série de multiplications et d'addition. Comme indiqué ci-dessous :

$$Gx = [f(x+1, y-1)+2*f(x+1, y)+f(x+1, y+1)]-[f(x-1, y-1)+2*f(x-1, y)+f(x-1, y+1)]$$

$$Gy = [f(x-1, y-1) + 2f(x, y-1) + f(x+1, y-1)]-[f(x-1, y+1) + 2*f(x, y+1)+f(x+1, y+1)]$$

Le module multiplicateur-additionneur est lié aux 3 sorties de la mémoire tampom. Les autres entrées sont les éléments de la structure de matrice 3x3(opérateur de Sobel). Ce module calculera les parties de Gx et Gy, qui seront ensuite utilisées dans les modules suivants.

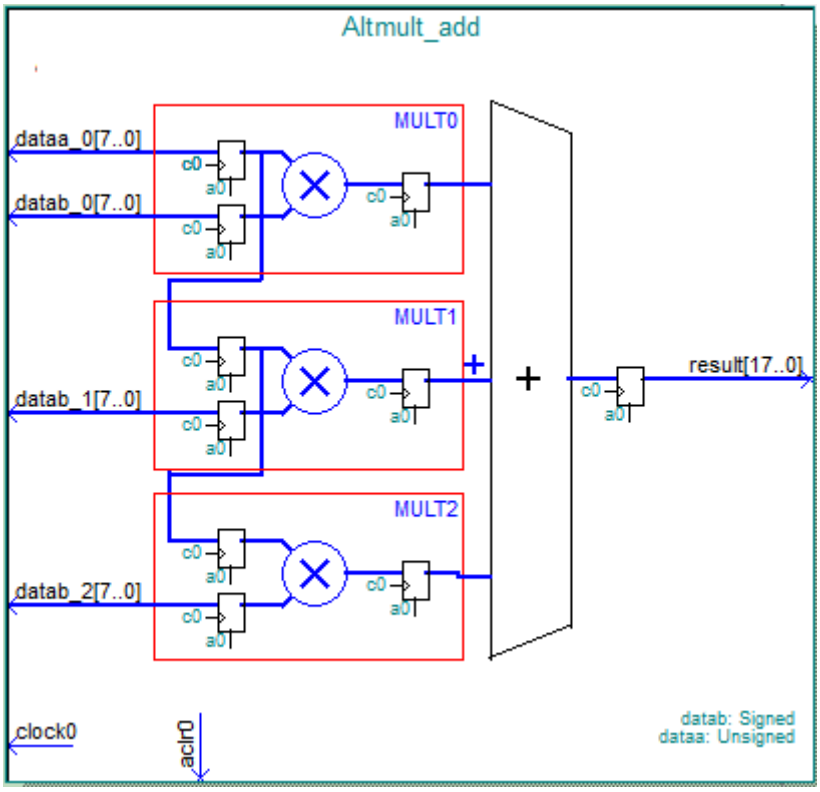

Figure 13:Module Multiplicateur-Additionneur

4.2.3 Module Additionneur parallèle et SQRT (racine carrée)

l'additionneur parallèle est utilisé dans le calcul de l'opérateur Sobel. Le résultat du Multiplicateur-Additionneur sera transmis à l' additionneur parallèle. Après cette étape, les résultats de Gx et Gy seront finalement calculés, puis donnés au module SQRT suivant.

Le module SQRT est généré par MegaWizard Plugin Manager. Il a une entrée 32 bits et une sortie 16 bits.

4.2.4 Génération de niveaux de gris

Pour obtenir une image en niveaux de gris, nous devons transformer les couleurs RGB en niveaux de gris. D'une manière générale, une méthode largement acceptée consiste à utiliser l'équation ci-dessous :

Niveaux de gris = 0,299 * R + 0,587 * G + 0,114 * B

Cependant, une telle équation nécessite certainement un calcul en virgule flottante, qui est non seulement consommatrice de matériel, mais aussi lente et de faible précision. Il est lent même en utilisant la programmation C, sans parler de Verilog. Parfois, on utilise une autre approximation de cette équation :

Niveaux de gris = (R + G + B) / 3

Cependant, cette méthode a besoin de division, ce qui est un peu mieux que les nombre à virgule flottante, mais aussi très consommatrice de ressources..

Remarquant que le poids de R, G et B est différent, et que le poids de G est presque deux fois celui de R, et 5 fois celui de B. C'est-à-dire que l'échelle de gris se situe principalement au niveau de G, mais moins sur R et B. Par conséquent, nous avons décidé d'utiliser le niveau de G pour représenter l'échelle de gris. C'est aussi la meilleure et la plus rapide façon de procéder à une telle transformation sur le matériel.

Après la transformation des données de modèle de couleur Bayer en données RGB, les informations de couleur RGB prennent plus de 30 bits au total. Cependant, le niveau de gris est seulement 10 bits. Par conséquent, nous pourrions économiser la moitié de l'espace sur la SDRAM, ainsi que la moitié de la bande passante. Ici, nous ne pouvons pas enregistrer 2/3 de la mémoire totale, parce que le WR1_DATA de la SDRAM est de 16 bits. Tandis que le niveau de gris n'a que 10 bits, les 6 autres bits sont occupés par zéro. À partir de la SDRAM, nous donnons {Read_DATA1[14:10], Read_DATA2[14:10]}à Grey_G. Les mêmes données sont également fournies à Gray_R et Gray_B.

Conclusion

Le travail sur ce projet est très intéressant et m'a mis au défi du traitement vidéo, du calcul en temps réel et de la conception de systèmes embarqués. Une grande partie est basée sur l'exemple de code fourni par la société Terasic, j'ai ensuite ajouté des modules individuels pour des tâches spécifiques. La plate-forme DE1-SoC s'avère très intéressante pour mettre en oeuvre mes futures idées sur le traitement d'images en temps réel sur FPGA.

Ce travail m'a permis de tester et d'améliorer mes connaissances théoriques et pratiques en programmation HDL ainsi que le traitement d'images sur FPGA.

En perspective, j'envisage d'implémenter un algorithme de reconnaissance faciale sur une cible matérielle et utiliser le HPS pour le traitement des images.

Références

- Terasic Technologies, DE1-SoC User Manual.

- Terasic Technologies, TRDB-D5M Hardware specification, Document Version 0.2.

- CMOS TRDB-D5M Camera and 8MB SDRAM A2V4S40CTP User Manual.

- Tutoriel du langage de description matériel Verilog : https://www.asic-world.com/verilog/index.html

- Dinesh Kumar K, Karunamoorthy, RaniThottungal, "Real Time Monitoring System: Implementation of Face Detection and Recognition Algorithm", International Journal of Innovative Technology and Exploring Engineering (IJITEE), Décembre 2018.

- Saif N. Ismail1, Muataz H. Salih1 and Wahab Y., "DESIGN AND IMPLEMENTATION OF EMBEDDED VISION BASED TRACKING SYSTEM FOR MULTIPLE OBJECTS USING FPGA-SOC", ARPN Journal of Engineering and Applied Sciences, février 2018.

- Lionel Lelong, "Architecture SoC-FPGA pour la mesure temps réel par traitement d'image. Conception d'un système embarqué : imageur CMOS et Circuit Logique Programmable", Université Jean Monnet – Saint Etienne, 2004.

- Article de Brandon Treece : https://www.vision-systems.com/embedded/article/16737656/cpu-or-fpga-for-image-processing-which-is-best

Annexe

Programme principal du projet :

```
module DE1_SoC_CAMERA(
      ///////// CLOCK2 /////////
       input              CLOCK2_50,
      ///////// CLOCK /////////
      input              CLOCK_50,
      ///////// DRAM /////////
      output      [12:0] DRAM_ADDR,
      output      [1:0]  DRAM_BA,
      output             DRAM_CAS_N,
      output             DRAM_CKE,
      output             DRAM_CLK,
      output             DRAM_CS_N,
      inout       [15:0] DRAM_DQ,
      output             DRAM_LDQM,
      output             DRAM_RAS_N,
      output             DRAM_UDQM,
      output             DRAM_WE_N,
      ///////// HEX0 /////////
      output      [6:0]  HEX0,
      ///////// HEX1 /////////
      output      [6:0]  HEX1,
      ///////// HEX2 /////////
      output      [6:0]  HEX2,
      ///////// HEX3 /////////
      output      [6:0]  HEX3,
      ///////// HEX4 /////////
      output      [6:0]  HEX4,
      ///////// HEX5 /////////
      output      [6:0]  HEX5,
      ///////// KEY /////////
      input       [3:0]  KEY,
      ///////// LEDR /////////
      output      [9:0]  LEDR,
      ///////// SW /////////
      input       [9:0]  SW,
      ///////// VGA /////////
      output      [7:0]  VGA_B,
      output             VGA_BLANK_N,
      output             VGA_CLK,
      output      [7:0]  VGA_G,
      output             VGA_HS,
      output      [7:0]  VGA_R,
      output             VGA_SYNC_N,
      output             VGA_VS,
            ////////////// GPIO1, GPIO1 connect to D5M - 5M Pixel Camera //////////
       input           [11:0] D5M_D,
      input                  D5M_FVAL,
      input                  D5M_LVAL,
      input                  D5M_PIXLCLK,
      output              D5M_RESET_N,
      output              D5M_SCLK,
      inout                  D5M_SDATA,
      input                  D5M_STROBE,
      output              D5M_TRIGGER,
      output              D5M_XCLKIN
);
```

```
//=======================================================
//  REG/WIRE declarations
//=======================================================
wire                [15:0]            Read_DATA1;
wire               [15:0]             Read_DATA2;
wire                [11:0]            mCCD_DATA;
wire                                  mCCD_DVAL;
wire                                  mCCD_DVAL_d;
wire         [15:0]                   X_Cont;
wire         [15:0]                   Y_Cont;
wire         [9:0]                    X_ADDR;
wire         [31:0]                   Frame_Cont;
wire                                  DLY_RST_0;
wire                                  DLY_RST_1;
wire                                  DLY_RST_2;
wire                                  DLY_RST_3;
wire                                  DLY_RST_4;
wire                                  Read;
reg              [11:0]               rCCD_DATA;
reg                                   rCCD_LVAL;
reg                                   rCCD_FVAL;
wire         [11:0]                   sCCD_R;
wire         [11:0]                   sCCD_G;
wire         [11:0]                   sCCD_B;
wire                                  sCCD_DVAL;
// Wire RAW2RGB Mirror
wire         [11:0]                   mCCD_R;
wire         [11:0]                   mCCD_G;
wire         [11:0]                   mCCD_B;
wire                                  sdram_ctrl_clk;
wire         [9:0]                    oVGA_R;
wire         [9:0]                    oVGA_G;
wire         [9:0]                    oVGA_B;
// sobel
wire         [9:0]                    DISP_R;
wire         [9:0]                    DISP_G;
wire         [9:0]                    DISP_B;
wire         [9:0]                    Gray_R;
wire         [9:0]                    Gray_G;
wire         [9:0]                    Gray_B;
wire         [9:0]                    mVGA_R;
wire         [9:0]                    mVGA_G;
wire         [9:0]                    mVGA_B;
wire         [9:0]                    Filter_Out;
//power on start
wire                                  auto_start;
//=======================================================
//  Structural coding
//=======================================================
// D5M
assign D5M_TRIGGER  =     1'b1;  // tRIGGER
assign D5M_RESET_N  =     DLY_RST_1;
assign VGA_CTRL_CLK =     VGA_CLK;
assign LEDR         =     Y_Cont;
//  Gray
assign  Gray_R  = mVGA_G;
assign  Gray_G  = mVGA_G;
assign  Gray_B  = mVGA_G;
//  To Display
```

```
assign DISP_R = SW[8] ? mVGA_G  :     //  Niveaux de gris
                SW[7] ? Filter_Out  : //  Sobel
                        mVGA_R  ;     //  Couleur
assign DISP_G = SW[8] ? mVGA_G  :
                SW[7] ? Filter_Out  :
                        mVGA_G;
assign DISP_B = SW[8] ? mVGA_G  :
                SW[7] ? Filter_Out  :
                        mVGA_B;
assign  mVGA_R  = Read_DATA2[9:0];
assign  mVGA_G  = {Read_DATA1[14:10],Read_DATA2[14:10]};
assign  mVGA_B  = Read_DATA1[9:0];
//fetch the high 8 bits
assign  VGA_R = oVGA_R[9:2];
assign  VGA_G = oVGA_G[9:2];
assign  VGA_B = oVGA_B[9:2];
//D5M read
always@(posedge D5M_PIXLCLK)
begin
      rCCD_DATA    <=    D5M_D;
      rCCD_LVAL    <=    D5M_LVAL;
      rCCD_FVAL    <=    D5M_FVAL;
end
assign auto_start = ((KEY[0])&&(DLY_RST_3)&&(!DLY_RST_4))? 1'b1:1'b0;
//Reset module
Reset_Delay               u2    (
                                      .iCLK(CLOCK_50),
                                      .iRST(KEY[0]),
                                      .oRST_0(DLY_RST_0),
                                      .oRST_1(DLY_RST_1),
                                      .oRST_2(DLY_RST_2),
                                      .oRST_3(DLY_RST_3),
                                      .oRST_4(DLY_RST_4)
                                   );
//D5M image capture
CCD_Capture               u3    (
                                      .oDATA(mCCD_DATA),
                                      .oDVAL(mCCD_DVAL),
                                      .oX_Cont(X_Cont),
                                      .oY_Cont(Y_Cont),
                                      .oFrame_Cont(Frame_Cont),
                                      .iDATA(rCCD_DATA),
                                      .iFVAL(rCCD_FVAL),
                                      .iLVAL(rCCD_LVAL),
                                      .iSTART(!KEY[3]|auto_start),
                                      .iEND(!KEY[2]),
                                      .iCLK(~D5M_PIXLCLK),
                                      .iRST(DLY_RST_2)
                                   );
//D5M raw date convert to RGB data
RAW2RGB                           u4    (
                                      .iCLK(D5M_PIXLCLK),
                                      .iRST(DLY_RST_1),
                                      .iDATA(mCCD_DATA),
                                      .iDVAL(mCCD_DVAL),
                                      .oRed(mCCD_R),
                                      .oGreen(mCCD_G),
                                      .oBlue(mCCD_B),
                                      .oDVAL(mCCD_DVAL_d),
```

```
                                        .iX_Cont(X_Cont),
                                        .iY_Cont(Y_Cont)
                                    );
//Frame count display
SEG7_LUT_6              u5      (
                                        .oSEG0(HEX0),.oSEG1(HEX1),
                                        .oSEG2(HEX2),.oSEG3(HEX3),
                                        .oSEG4(HEX4),.oSEG5(HEX5),
                                        .iDIG(Frame_Cont[23:0])
                                    );
sdram_pll               u6      (
                                        .refclk(CLOCK_50),
                                        .rst(1'b0),
                                        .outclk_0(sdram_ctrl_clk),
                                        .outclk_1(DRAM_CLK),
                                        .outclk_2(D5M_XCLKIN),    //25M
                                        .outclk_3(VGA_CLK)       //25M
                                    );
//SDRam Read and Write as Frame Buffer
Sdram_Control          u7  (     //    HOST Side
                                    .RESET_N(KEY[0]),
                                    .CLK(sdram_ctrl_clk),
                                        //    FIFO Write Side 1
                                  .WR1_DATA({1'b0,sCCD_G[11:7],sCCD_B[11:2]}),
                                        .WR1(sCCD_DVAL),
                                        .WR1_ADDR(0),
                                        .WR1_MAX_ADDR(640*480),
                                        .WR1_LENGTH(8'h50),
                                        .WR1_LOAD(!DLY_RST_0),
                                        .WR1_CLK(~D5M_PIXLCLK),
                                        //    FIFO Write Side 2
                                    .WR2_DATA({1'b0,sCCD_G[6:2],sCCD_R[11:2]}),
                                        .WR2(sCCD_DVAL),
                                        .WR2_ADDR(23'h100000),
                                        .WR2_MAX_ADDR(23'h100000+640*480),
                                        .WR2_LENGTH(8'h50),
                                        .WR2_LOAD(!DLY_RST_0),
                                        .WR2_CLK(~D5M_PIXLCLK),
                        //   FIFO Read Side 1
                                        .RD1_DATA(Read_DATA1),
                                        .RD1(Read),
                                        .RD1_ADDR(0),
                                        .RD1_MAX_ADDR(640*480),
                                        .RD1_LENGTH(8'h50),
                                        .RD1_LOAD(!DLY_RST_0),
                                        .RD1_CLK(~VGA_CTRL_CLK),
                                        //    FIFO Read Side 2
                                        .RD2_DATA(Read_DATA2),
                                        .RD2(Read),
                                        .RD2_ADDR(23'h100000),
                                        .RD2_MAX_ADDR(23'h100000+640*480),
                                        .RD2_LENGTH(8'h50),
                                        .RD2_LOAD(!DLY_RST_0),
                                        .RD2_CLK(~VGA_CTRL_CLK),
                                        //    SDRAM Side
                                        .SA(DRAM_ADDR),
                                        .BA(DRAM_BA),
                                        .CS_N(DRAM_CS_N),
                                        .CKE(DRAM_CKE),
```

```
                                        .RAS_N(DRAM_RAS_N),
                                        .CAS_N(DRAM_CAS_N),
                                        .WE_N(DRAM_WE_N),
                                        .DQ(DRAM_DQ),
                                        .DQM({DRAM_UDQM,DRAM_LDQM})
                                      );
//D5M I2C control
I2C_CCD_Config      u8     (      //     Host Side
                                        .iCLK(CLOCK2_50),
                                        .iRST_N(DLY_RST_2),
                                        .iEXPOSURE_ADJ(KEY[1]),
                                        .iEXPOSURE_DEC_p(SW[0]),
                                        .iZOOM_MODE_SW(SW[9]),
                                        //     I2C Side
                                        .I2C_SCLK(D5M_SCLK),
                                        .I2C_SDAT(D5M_SDATA)
                                      );
//VGA DISPLAY
VGA_Controller        u1   (      //     Host Side
                                        .oRequest(Read),
                                        .iRed(DISP_R),
                                        .iGreen(DISP_G),
                                        .iBlue(DISP_B),
                                        //     VGA Side
                                        .oVGA_R(oVGA_R),
                                        .oVGA_G(oVGA_G),
                                        .oVGA_B(oVGA_B),
                                        .oVGA_H_SYNC(VGA_HS),
                                        .oVGA_V_SYNC(VGA_VS),
                                        .oVGA_SYNC(VGA_SYNC_N),
                                        .oVGA_BLANK(VGA_BLANK_N),
                                        //     Control Signal
                                        .iCLK(VGA_CTRL_CLK),
                                        .iRST_N(DLY_RST_2),
                                        .iZOOM_MODE_SW(SW[9])
                                      );
Mirror_Col  u9              (
                                      //  Input Side
                                      .iCCD_R(mCCD_R),
                                      .iCCD_G(mCCD_G),
                                      .iCCD_B(mCCD_B),
                                      .iCCD_DVAL(mCCD_DVAL_d),
                                      .iCCD_PIXCLK(D5M_PIXLCLK),
                                      .iRST_N(DLY_RST_1),
                                      //  Output Side
                                      .oCCD_R(sCCD_R),
                                      .oCCD_G(sCCD_G),
                                      .oCCD_B(sCCD_B),
                                      .oCCD_DVAL(sCCD_DVAL)
);
Sobel        u10 (
                                    .iCLK(VGA_CTRL_CLK),
                                    .iRST_N(DLY_RST_2),
                                    .iDATA(mVGA_G),
                                    .iDVAL(Read),
                                    .iTHRESHOLD({SW[3:1],2'b10}),
                                    .oDATA(Filter_Out)
);
endmodule
```