Réf : AU : 2015-2016

***Université de Sousse***

***Ecole Nationale d’Ingénieurs de Sousse***



***Mémoire de mastère de Recherche en systèmes intelligents et communicants***

Option : Intelligence artificielle

Implémentation d’un algorithme de détection de mouvement sur une carte ZedBoard

Réalisé par :

Sirine Bouslama

# Table des matières

[Liste des figures V](#_Toc471122289)

[Liste des tableaux VII](#_Toc471122290)

[Liste des abréviations VIII](#_Toc471122291)

[Introduction Générale 1](#_Toc471122292)

[Chapitre I Etat de l’art sur la détection de mouvement 4](#_Toc471122293)

[Introduction 5](#_Toc471122294)

[I.1. Approche basée sur la différence inter-images 5](#_Toc471122296)

[III.1.1. Soustraction du fond 6](#_Toc471122300)

[a. Utilisation de l’image de référence 6](#_Toc471122301)

[b. Modélisation de l’arrière-plan : 7](#_Toc471122302)

[b.1. Méthode de la moyenne temporelle 8](#_Toc471122303)

[b.2. Méthode de la médiane temporelle 8](#_Toc471122304)

[b.3. Méthode Min, Max et écart Max 8](#_Toc471122305)

[III.1.2. La différence temporelle 10](#_Toc471122311)

[III.2. Le flot optique 12](#_Toc471122312)

[III.2.1. Introduction sur le flot optique 12](#_Toc471122313)

[III.2.2. Principe du flot optique 14](#_Toc471122314)

[a. Les méthodes différentielles 15](#_Toc471122315)

[a.1. Horn et Shunk 15](#_Toc471122316)

[a.2. Lucas et kanade 16](#_Toc471122317)

[b. Les méthodes basées sur la corrélation 18](#_Toc471122318)

[III.3. Les méthodes hybrides 19](#_Toc471122319)

[Conclusion 21](#_Toc471122320)

[Chapitre II Introduction sur le prototypage d’applications de vision sur FPGA 23](#_Toc471122321)

[Introduction 24](#_Toc471122322)

[I.1. Motivation 24](#_Toc471122323)

[I.2. Architectures dédiées aux implémentations d’algorithmes de vision 26](#_Toc471122324)

[I.2.1. Les DSPs 27](#_Toc471122325)

[I.2.2. Les GPUs 28](#_Toc471122326)

[I.2.3. Les FPGAs 29](#_Toc471122327)

[I.2.4. Les ASICs 31](#_Toc471122328)

[I.3. Les architectures embarquées 32](#_Toc471122329)

[I.4. Zynq 32](#_Toc471122330)

[I.5. Les outils de synthèse haut niveau 34](#_Toc471122331)

[I.5.1. Vivado HLS 34](#_Toc471122332)

[I.5.2. Autres outils de synthèse haut niveau 37](#_Toc471122333)

[Conclusion 38](#_Toc471122334)

[Chapitre III Prototypage rapide d’un système de traitement de vidéo sur la carte ZedBoard 39](#_Toc471122335)

[Introduction 40](#_Toc471122336)

[II.1. La plateforme ZedBoard 40](#_Toc471122339)

[II.2. Logiciels requis 43](#_Toc471122340)

[II.3. Flot de conception à base du Zynq 44](#_Toc471122341)

[III.4. Design de l’architecture d’un système de traitement vidéo 45](#_Toc471122347)

[III.4.1. Les interfaces AXI 47](#_Toc471122348)

[a. AXI-Memory-Mapped 48](#_Toc471122349)

[b. AXI4-Lite 48](#_Toc471122350)

[c. AXI4-Stream 49](#_Toc471122351)

[III.4.2. ADV7511 49](#_Toc471122352)

[III.4.3. AXI-InterConnect 50](#_Toc471122353)

[III.4.4. VDMA 51](#_Toc471122354)

[a. Configuration matérielle des VDMAs 53](#_Toc471122355)

[b. Configuration logicielle des VDMAs 55](#_Toc471122356)

[III.5. Communication avec le matériel et Configuration 55](#_Toc471122357)

[Conclusion 58](#_Toc471122358)

[Chapitre IV Implémentation de la méthode du flot optique sur la ZedBoard 59](#_Toc471122359)

[Introduction 60](#_Toc471122360)

[IV.1. Travaux antérieurs concernant le calcul du flot optique 60](#_Toc471122365)

[IV.2. Accélération Matérielle du calcul du flot optique 61](#_Toc471122366)

[IV.2.1. Accélération matérielle avec les GPUs 62](#_Toc471122367)

[IV.2.2. Accélération matérielle avec les FPGAs 62](#_Toc471122368)

[IV.3. Le flot de travail général de la méthode HS 63](#_Toc471122369)

[IV.4. Calcul des dérivés 65](#_Toc471122370)

[IV.4.1. Principe du calcul des dérivées 65](#_Toc471122371)

[IV.4.2. Architecture de la fenêtre glissante pour le calcul des dérivée 67](#_Toc471122372)

[IV.5. Optimisations du calcul du flot optique 70](#_Toc471122373)

[IV.5.1. Parallélisme dans la méthode de calcul du flot optique 70](#_Toc471122374)

[a. Le parallélisme des tâches 71](#_Toc471122375)

[b. Le parallélisme des données (7) 72](#_Toc471122376)

[c. Le parallélisme du pipeline 72](#_Toc471122377)

[IV.5.2. Techniques d’optimisations 72](#_Toc471122378)

[a. Optimisation des boucles 72](#_Toc471122379)

[b. Optimisation des fonctions OpenCV-HLS 73](#_Toc471122380)

[c. Extension d’itération 73](#_Toc471122381)

[IV.6. Développement matériel et logiciel de l’application 74](#_Toc471122382)

[IV.7. Expérimentation et évaluation 77](#_Toc471122383)

[IV.7.1. L’environnement d’expérimentation 78](#_Toc471122384)

[a. Les images sources 78](#_Toc471122385)

[b. Paramètres d’implémentation 79](#_Toc471122386)

[c. Représentation du flot et références 79](#_Toc471122387)

[IV.7.2. Evaluations 81](#_Toc471122388)

[a. Temps d’exécution 81](#_Toc471122389)

[a.1. Le temps d’exécution et la taille de l’image 82](#_Toc471122390)

[a.2. Le temps d’exécution en fonction du nombre d’itération 83](#_Toc471122391)

[b. Précision 84](#_Toc471122392)

[c. Ressources 85](#_Toc471122393)

[d. Consommation d’énergie 87](#_Toc471122394)

[Conclusion 87](#_Toc471122395)

[Conclusion Et Perspectives 88](#_Toc471122396)

[Bibliographie 91](#_Toc471122397)

[Annexes 95](#_Toc471122398)

# Liste des figures

[Figure ‎III‑1: Détection de mouvement par soustraction de l’image de fond 6](#_Toc471122786)

[Figure ‎III‑2: Exemples de la méthode de soustraction de fond (10) 10](file:///E:\PFE\RapportMaster\RapportMaster_v2.docx#_Toc471122787)

[Figure ‎III‑3: La méthode de la différence temporelle (16) 11](#_Toc471122788)

[Figure ‎III‑4: le champ de mouvement 13](file:///E:\PFE\RapportMaster\RapportMaster_v2.docx#_Toc471122789)

[Figure ‎III‑5: Le flot optique 13](#_Toc471122790)

[Figure ‎I‑6: Détection de mouvement par la méthode de Collins et al. (24). 20](#_Toc471122791)

[Figure ‎III‑7:Résultat de la méthode du flot optique 20](#_Toc471122792)

[Figure ‎III‑8: Résultat de la méthode combinée (25) 21](#_Toc471122793)

[Figure ‎III‑9: Résultat de la méthode de soustraction de l'arrière-plan 21](#_Toc471122794)

[Figure ‎I‑1: Comparaison entre les différentes architectures 27](#_Toc471122795)

[Figure ‎I‑2: Architecture FPGA (5) 30](#_Toc471122796)

[Figure ‎I‑3: Flot de conception sur les FPGAs 30](#_Toc471122797)

[Figure ‎I‑4 : Architecture du Zynq-7000 (2) 33](#_Toc471122798)

[Figure ‎I‑5: Flot de conception sur FPGA avec HLS 35](#_Toc471122799)

[Figure ‎I‑6: Fonctionnalité du testbench 36](#_Toc471122800)

[Figure ‎II‑1: La carte ZedBoard et ses interfaces 42](#_Toc471122801)

[Figure ‎II‑2: Flot de prototypage d'un système de traitement de vidéo sur le Zynq 45](#_Toc471122802)

[Figure ‎II‑3: Design générale d’un système de traitement de vidéo sur le Zynq 47](#_Toc471122803)

[Figure ‎II‑4: Architecture ADV7511 50](#_Toc471122804)

[Figure ‎II‑5: le transfert des données dans le VDMA 51](#_Toc471122805)

[Figure ‎II‑6: Le bloc VDMA 52](#_Toc471122806)

[Figure ‎II‑7 : Architecture haut-niveau du système Linux (8) 56](#_Toc471122807)

[Figure ‎II‑8: Mappage entre l'adresse virtuelle et l'adresse physique (8) 57](#_Toc471122808)

[Figure ‎IV‑1: le flot de travail général de la méthode HS 64](#_Toc471122809)

[Figure ‎IV‑2: calcul du gradient X de sobel 66](#_Toc471122810)

[Figure ‎IV‑3: calcul du gradient Y de sobel 66](#_Toc471122811)

[Figure ‎IV‑4: Ordre de lecture d'un stream source 68](#_Toc471122812)

[Figure ‎IV‑5:Exemple d’Interaction entre les deux interfaces mémoire (5) 69](#_Toc471122813)

[Figure ‎IV‑6: Parallélisme des tâches dans HS 71](#_Toc471122814)

[Figure ‎IV‑7: Optimisation pipelinée 74](file:///E:\PFE\RapportMaster\RapportMaster_v2.docx#_Toc471122815)

[Figure ‎IV‑8: Architecture du système Linux 75](#_Toc471122816)

[Figure ‎IV‑9: Design Complet du système 76](#_Toc471122817)

[Figure ‎IV‑10: Bloc Processing Flow 77](#_Toc471122818)

[Figure ‎IV‑11: Séquence d'image *Yosemite* 78](#_Toc471122819)

[Figure ‎IV‑12: La direction du flot selon la carte de couleur 80](#_Toc471122820)

[Figure ‎IV‑13: Flot réel de la séquence Yosemite 80](#_Toc471122821)

[Figure ‎IV‑14: temps d'exécution (s) 81](#_Toc471122822)

[Figure ‎IV‑15: temps d'exécution (s) du calcul du flot optique HW 82](#_Toc471122823)

[Figure ‎IV‑16: temps d'exécution par pixel 83](#_Toc471122824)

[Figure ‎IV‑17: Temps d'exécution en fonction du nombre d'itération 84](#_Toc471122825)

[Figure ‎IV‑18: Précision du calcul en fonction du nombre d'itération 85](#_Toc471122826)

[Figure ‎IV‑19: Estimation des ressources 86](#_Toc471122827)

[Figure ‎IV‑20: Comparaison entre les ressources utilisées dans la fonction de dérivée d'OpenCV et l'architecture utilisée 86](#_Toc471122828)

[Figure ‎IV‑21:Consommation d’énergie 87](#_Toc471122829)

# Liste des tableaux

[Tableau ‎IV‑1:Relation entre l'interface mémoire et l'utilisation des ressources 77](#_Toc470279294)

[Tableau ‎IV‑2: Procédure de mise à jour du line buffer 79](#_Toc470279295)

[Tableau ‎IV‑3: Procédure de mise à jour du window buffer 79](#_Toc470279296)

[Tableau ‎IV‑4: Paramètres de configuration de HS 89](#_Toc470279297)

[Tableau ‎IV‑5: Comparaison du temps d'exécution entre le SW et HW 92](#_Toc470279298)

# Liste des abréviations

ADAS: Advanced Driver Assistance Systems

AP SOC: All Programmable System on Chip

AXI: Advanced eXtensively Interface

DMA: Direct Memory Access

FHD: Full High Definition

FPGA: Field Programmable Gate Array

FSBL: First Stage Boot Loader

HDMI: High Definition Multimedia Interface

IP: Intellectual Property

OLED: Organic Light Emitting Diode

PL: Programmable Logic

PMOD: Peripheral module

PS: Processing System

SOC: System On chip

SPI: Serial Peripheral Interface

UART: Universal Asynchronous Receiver Transmitter

VDMA: Video Direct Access Memory

# Introduction Générale

Le coût des caméras à haute résolution et des capteurs d’images diminue considérablement. Les écrans hauts résolution deviennent de plus en plus utilisés. La vidéo devrait dépasser 80% du trafic internet d’ici 2020 (1). Tout cela ne fait que créer de nouvelles applications qui impliquent de plus en plus d’intelligence artificielle en se basant sur le traitement d’images/vidéo.

L’image et encore plus la vidéo jouent un rôle trop important dans notre quotidien et elles sont de plus en plus utilisées dans nombreuses disciplines que ce soit dans l'industrie automobile, la robotique, la surveillance visuelle, le diagnostic médical, l’armée, les systèmes de surveillance et drones, l’industrie maritime… Parallèlement, le traitement de l’information s’est lui aussi développé grâce à l’évolution technologique dans la capacité d’intégration des circuits électroniques. Ces deux mondes, ensembles, font parler du traitement d’image et de la vision.

De nos jours, les technologies de la vision intelligente touchent toutes sortes de produits, allons des systèmes les plus sophistiqués jusqu’aux systèmes humbles. En effet, les systèmes sont de plus en plus impressionnants dont la complexité ne cesse de croitre. Une image FHD, par exemple, est définie comme une matrice de 1920 x 1080 pixels et ceci est équivalent à 2 073 600 pixels en totale. Si chaque pixel est de trois dimensions présentant l’espace de couleur ! Avec huit bits pour le rouge, le vert et le bleu. Cela correspond à 24 bits par pixel et donc 49 766 400 bits pour une image HD-RGB. En considérant ces grandes dimensions, n’importe quel traitement impliquant ces pixels exige un nombre important de calcul.

L’évolution technologique dans la capacité d’intégration des circuits électroniques a ouvert de nouveaux horizons pour les applications de vision sur des systèmes embarqués. D’une part des processeurs performants pouvant effectuer plusieurs opérations par seconde et d’autre part des circuits reconfigurables pouvant comporter plusieurs portes logiques permettant ainsi d’éprouver les performances de ces applications en terme de vitesse de traitement.

Dans la plupart des applications de vision, des dizaines d’opérations sont effectuées sur chaque pixel. Si ces opérations sont effectuées, séquentiellement, par un processeur, cela peut conduire à des conséquences néfastes par rapport aux performances. Ce qui distingue les FPGAs des processeurs, c’est le fait que les FPGAs tirent profit du parallélisme matériel.

Ce projet est positionné pour répondre aux nouveaux défis technologiques du domaine de vision qui cherchent à trouver des méthodes de programmation et d’implémentation fiables, faciles et rapides sur les FPGAs, des algorithmes prévisionnelles de vision. Ce projet consiste, en fait, à implémenter un algorithme de détection de mouvement sur le circuit Zynq de la carte ZedBoard basée sur les FPGAs et profiter de l’accélération matérielle disponible.

Ce rapport aborde les problématiques suivantes :

* Quelles sont les méthodes existantes dans la littérature qui permettent de détecter le mouvement ?
* Comment les développeurs ayant une expérience limitée avec les FPGAs peuvent-ils développer rapidement une application de vision ?
* Quelle est la productivité de Vivado-HLS ? Quels changements doivent être apportés pour que le programme puisse être synthétisé afin de l’implémenter sur le matériel ?
* Comment les performances et la consommation d’énergie de l’algorithme d’implémentation de l’algorithme de détection de mouvement implémenté sur l’FPGA sont-elles évaluées par rapport à son implémentation sur le processeur ARM ?

# Etat de l’art sur la détection de mouvement

## Introduction

La détection de mouvement est l’un des domaines de vision qui joue un rôle considérable dans l’analyse des séquences d’images de par le nombre considérable de ses applications. Parmi ses applications, nous pouvons citer la vidéo-surveillances, la reconnaissance des activités humaines et l’interaction homme machine. L’analyse du mouvement nous permet de prendre la décision par rapport à la correspondance ou non d’une partie de l’image à un objet mobile. La diversité des recherches et la variété des méthodes proposées sont liées à la difficulté et la variété des applications. Dans ce sujet, la littérature est énormément abondante. C’est pour cela, nous ne détaillerons pas toutes les méthodes existantes. Nous donnons juste un aperçu général sur les méthodes les plus utilisées.

Les méthodes de détection de mouvement peuvent être classées en trois ensembles. Nous citons les approches basées sur la différence inter-images, les approches basées sur le calcul du flot optique et les approches hybrides basées sur le couplage des approches précédentes.



## Approche basée sur la différence inter-images

L’approche basée sur la différence inter-images permet de segmenter chaque objet en mouvement dans une scène quelconque. Elle est généralement utilisée pour le cas d’une caméra fixe. C’est l’approche la plus simple et la plus facile à mettre en œuvre. Elle repose sur la soustraction entre deux ou plusieurs images successives. Les pixels dont leur intensité est non nulle sont considérés ceux appartenant à un objet en mouvement.

* 1. Nous citons dans ce contexte, la méthode de soustraction de fond et la méthode de la différence temporelle.

### Soustraction du fond

Pour détecter les objets mobiles, plusieurs travaux procèdent par une opération de segmentation qui consiste à faire la soustraction entre l’image courante définie comme I(x, y, t) et une image de référence définie comme R(x, y, t).

L’image de référence doit représenter la scène neutre où aucun objet mobile n’existe. Un résultat de la méthode de soustraction de fond est représenté sur la Figure ‎I‑1.

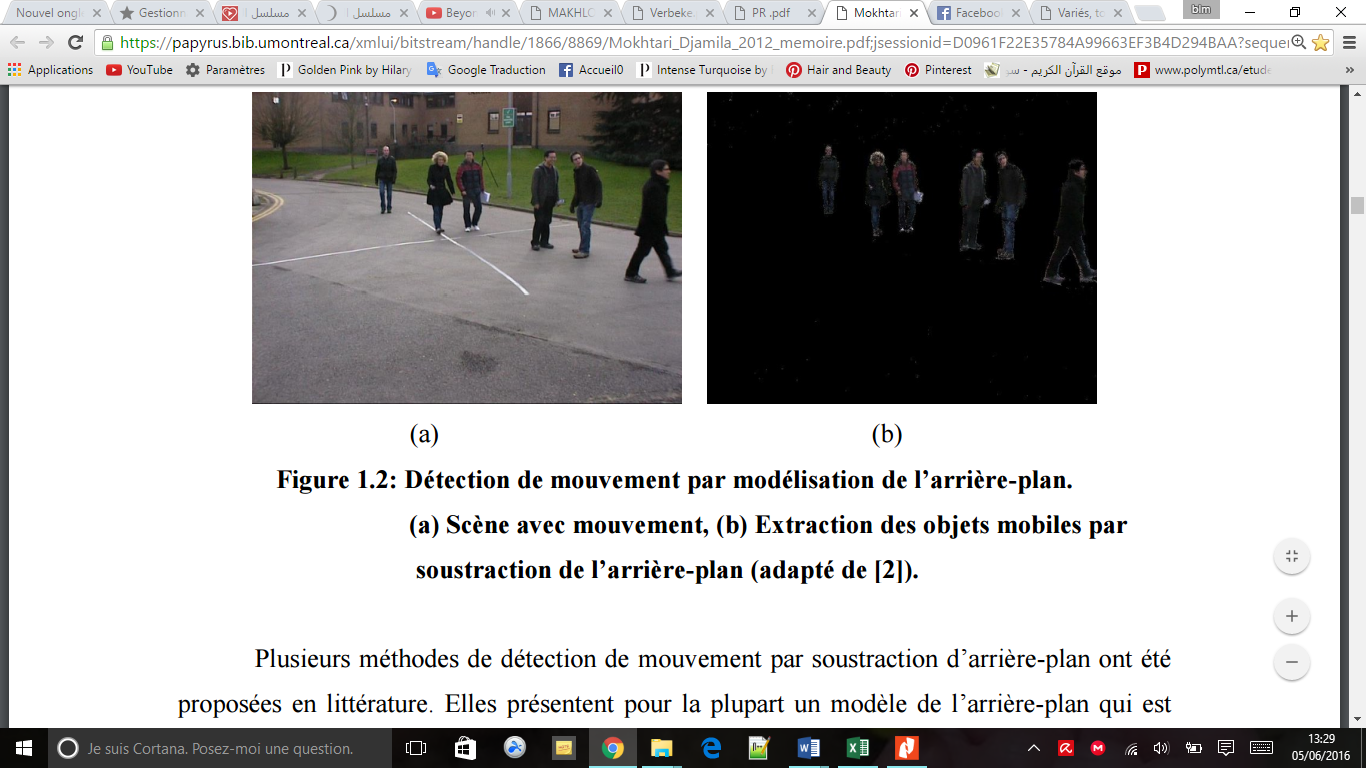


Figure ‑: Détection de mouvement par soustraction de l’image de fond

Cette méthode de soustraction de fond est particulièrement populaire pour la détection de mouvement. La segmentation du mouvement, dans ce cas, n’est réalisable qu’à partir d’un modèle d’arrière-plan de la scène. Les modèles de construction de l’arrière-plan de la scène permettent de construire l’image de référence.

#### Utilisation de l’image de référence

L’image de référence est une image qui contient les éléments statique d’une scène. Elle représente l’environnement dans lequel les objets évoluent. Chaque image de la séquence vidéo est comparée avec l’image de l’arrière-plan afin de détecter le mouvement.

Les pixels de l’image ayant une différence significative par rapport au modèle de l’arrière-plan sont considérés comme appartenant à un objet en mouvement.

Nous pouvons modéliser cette méthode comme suit :

Si | I(x, y, t) - R(x, y, t) | > β alors le pixel de I est en mouvement (‎I‑1)

Où β représente un seuil déterminé empiriquement

En partant du fait que nous devons exploiter un modèle pour modéliser l’arrière-plan, plusieurs méthodes ont apparues. La plus basique consistait à prendre l’image de l’arrière-plan comme référence. La mise à jour du modèle s’effectue avec un filtre linéaire comme l’indique l’équation ci-dessous.

*(‎I‑2**)*

*Où : I(x,y,t) représente l’intensité du pixel (x,y) à l’instant t de l’image courante R(x,y,t) représente l’intensité du pixel (x,y) à l’instant t de l’image de référence et α représente un paramètre compris entre 0 et 1*

Pour le cas de α = 1, nous aurons le cas exclusif de différence entre deux images consécutifs et pour le cas de α = 0, le modèle de l’arrière-plan ne change pas et ne se met pas à jour.

Bien que la méthode de soustraction de fond soit simple, le choix manuel de la valeur du seuil la rend moins robuste. En plus cette méthode est extrêmement sensible aux changements de l’intensité observée dans les scènes dynamiques et nécessite l’utilisation d’une caméra fixe. Pour remédier aux fluctuations des intensités et tenir compte des conditions d’éclairage et les variations temporelles de l’image, nous devons régulièrement mettre à jour l’image de fond. Le modèle le plus simple pour cela, si, surtout, nous ne disposons pas de l’image de l’arrière-plan, est sa reconstruction. Néanmoins, la mise à jour du modèle ne s’effectue pas de la même manière pour toutes les méthodes existantes. Certaines méthodes seront exploitées dans la partie suivante de ce document.

#### Modélisation de l’arrière-plan :

Pour la modélisation de l’arrière-plan, il existe plusieurs méthodes en littérature. Nous pouvons citer :

* la méthode de la moyenne temporelle
* la méthode de la médiane temporelle
* la méthode Min, Max et écart Max

Ces différentes modélisations présentent le modèle de l’arrière-plan mis à jour puisque l’image de fond sera reconstruite à fur et à mesure de l’acquisition de nouvelles images.

##### Méthode de la moyenne temporelle

Cette méthode consiste à construire l’arrière-plan en calculant l’image moyenne à partir de la séquence temporelle des images. L’image moyenne sera calculée à partir d’un nombre M des images acquises :

() = (‎I‑3)

*Où : I(x,y,t) représente l’intensité du pixel (x,y) à l’instant t de l’image courante R(x,y,t) représente l’intensité du pixel (x,y) à l’instant t de l’image de référence*

La mise à jour, quant à elle, se fait continuellement lorsque nous recevons une image ; donc cette image courante contribuera à la construction du modèle comme suit :

( (‎I‑4)

##### Méthode de la médiane temporelle

Cette méthode consiste à construire l’arrière-plan en calculant l’image médiane à partir de la séquence temporelle des images. Les recherches ont prouvé que, pour la construction de l’arrière-plan, la prise en compte de la valeur médiane est beaucoup plus robuste que la valeur moyenne. En effet, Cucchiara et *al*. (2) ont proposé un algorithme pour la construction de l’image de fond en prenant la valeur médiane de la couleur du pixel au cours d’une série d’images. Cet algorithme pourrait gérer certaines incohérences dues aux changements d’éclairage, etc…

##### Méthode Min, Max et écart Max

Cette méthode consiste à construire l’arrière-plan en se basant sur la valeur minimale et maximale que peut prendre un pixel dans une séquence d’image. Il s’agit de la méthode de Haritaoglu et *al.* (3) . Ils ont commencé par calculer la médiane des images pendant les 20 à 40 secondes des premières images de la scène. A un instant t, chaque pixel X se caractérise donc par certains paramètres ; sa valeur minimale, sa valeur maximale et sa valeur d’écart maximale qui est calculé entre deux images successives, sur les N images successives.

Un pixel X donné est considéré comme un pixel appartenant à un objet en mouvement s’il satisfait une des conditions suivantes :

| X- min(X) | <β m ou | X- max(X) | > β m (‎I‑5)

*Où β représente un seuil déterminé manuellement et m la valeur de la médiane*

*Min(X) : valeur minimale du pixel X, Max(X) : valeur maximale du pixel X*

##### Autres méthodes

Plusieurs chercheurs ont montré plus d’intérêt aux différents modèles pour la construction de l’arrière-plan. Ces méthodes sont sensibles aux différentes dégradations dues aux changements de la météo, luminosité… Pour remédier à ces contraintes, des modèles se basant sur la distribution statique du pixel ont été apparues. Wren et *al.* (4) ont représenté chaque pixel par une densité de probabilité gaussienne définie par la couleur moyenne du pixel et la covariance qui dépend de la couleur. Ensuite, ils utilisent une distance en log de vraisemblance pour comparer l’image courante à l’image de l’arrière-plan. A chaque nouvelle image, la moyenne et la covariance sont mises à jour en utilisant la même équation I.2.

Quelques chercheurs ont résolu le problème de la modélisation de l’arrière-plan différemment en soumettant d’autres méthodes. Des chercheurs font la modélisation en utilisant les vecteurs propres en appliquant l’analyse en composante principale sur les images de l’arrière-plan (5) . Une autre méthode a été proposée par Barnich et *al.* (6) et elle consiste à attribuer aux pixels de l’image un nombre N d’échantillons extrais des images précédente. Un pixel en mouvement est détecte si le nombre d’échantillon du modèle qui se trouvent à l’intérieure d’un cercle centré sur le pixel est supérieur à un nombre défini.

Nous présentons dans la Figure ‎III.2 quelques exemples de la méthode de soustraction de fond issus de l’article de Benezeth et *al.* (7).

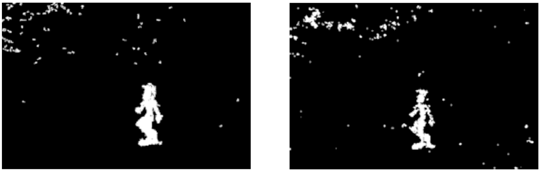
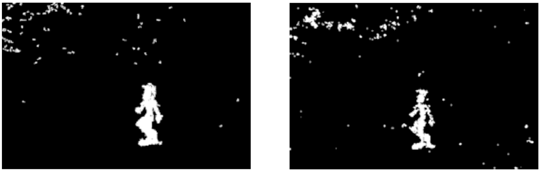
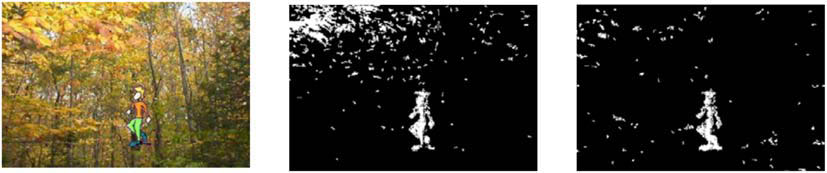
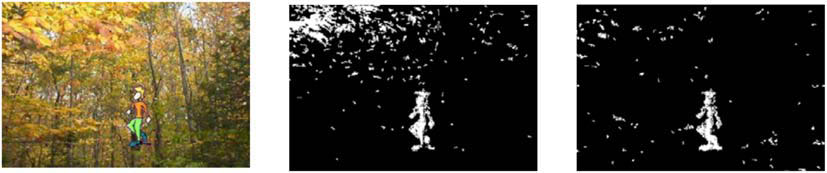
Figure ‎I‑2: Exemples de la méthode de soustraction de fond (7)

1. *Image originale, (b) méthode basique, (c) MiniMax, (d) Médiane temporelle*

b

c

a



d



### La différence temporelle

La méthode de la différence temporelle utilise la différence entre deux ou plusieurs images consécutives pour extraire les régions en mouvement. Cette méthode est très adaptée pour les environnements dynamiques mais elle est généralement peu utilisée lorsqu’il s’agit de l’extraction des pixels pertinents et peut même générer des trous à l’intérieur des objets en mouvement. Nous pouvons citer comme exemple de la méthode de Lipton et *al.* (8) qui détectent les mouvements en temps réel en utilisant la différence temporelle. Après avoir appliqué la différence absolue entre deux trames successives, ils aboutissent à un seuillage pour déterminer ce qui a changé. Cette méthode ne détecte pas l’objet en entier comme montre la Figure ‎I‑3.

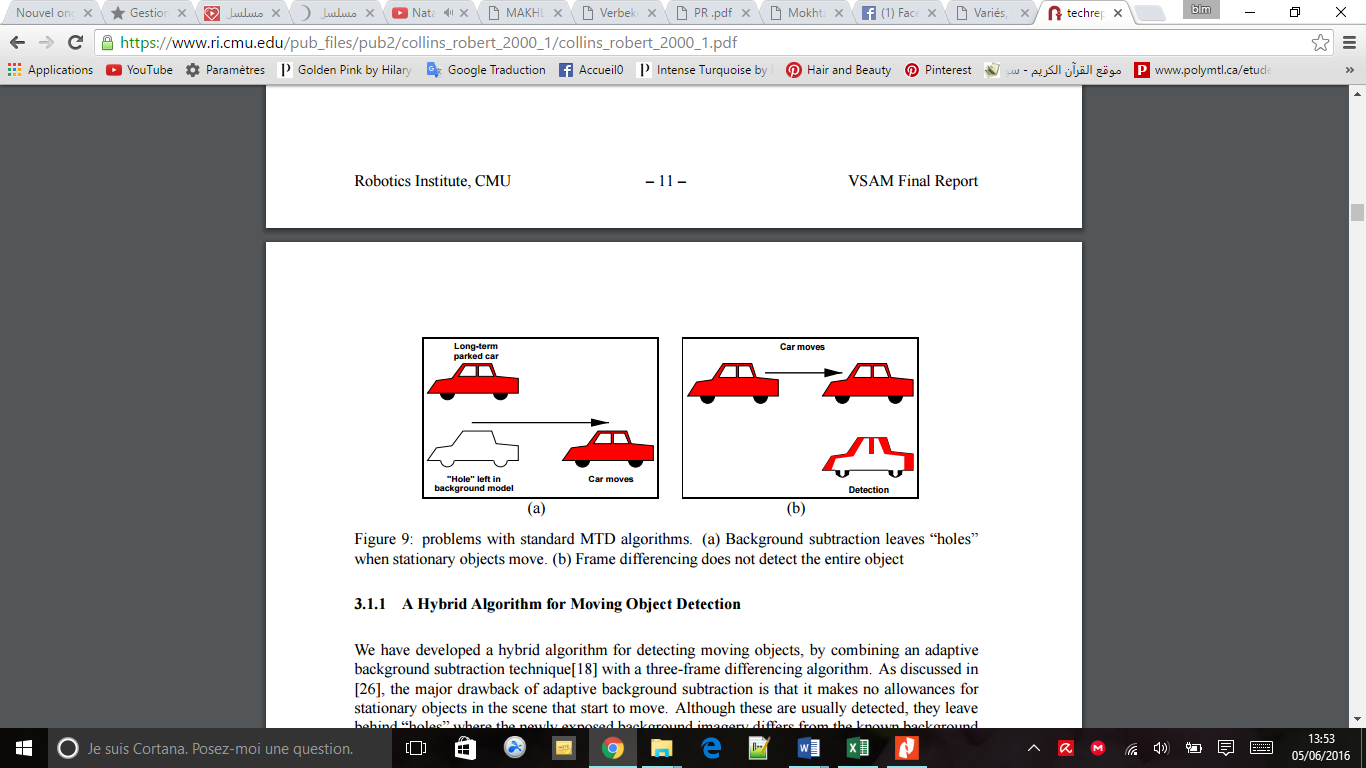


Figure ‑: La méthode de la différence temporelle (8)

Une version plus récente et plus améliorée consiste à utiliser la différence entre trois trames successives au lieu de la différence entre deux trames. Nous pouvons prendre l’exemple de WEI et *al.* (9) qui ont développé un algorithme de détection basé sur la différence temporelle accumulatives. Après avoir calculé la différence absolue entre les deux premières trames successives (N°1 et N°2), ils appliquent le filtre de Gauss pour diminuer le bruit et puis binariser l’image résultante. Le filtre de gauss passe-bas est construit comme suit :

(‎I‑6)

Où

Ainsi, l’image Id1 est obtenue

*Id1 =* (‎I‑7)

L’étape suivante consiste à faire la même procédure pour les deux trames suivantes (N°3 et N°4) de la séquence d’images et l’image binaire Idk sera calculé à partir de l’image N° (2k-1) et N° (2k).

*Idiff1 = k* (‎I‑8)

Cette image est par la suite transformée en une image binaire comme indiqué sur l’équation I.7 et elle contient tous les pixels des objets en mouvement dans les trames (depuis la première trame jusqu’à la trame N° (2k) de la séquence d’images. Une autre image binaire Idiff2 est calculée à partir des étapes précédentes à partir de l’image N° (2k) jusqu’à l’image N° (4k-1). L’étape finale permet, grâce à des opérations arithmétiques sur les trames binaire d’extraire le mouvement. La région présentant les objets en mouvement est calculée en appliquant le produit entre Idiff1 et Idiff2.

Ils ont réussi, suite à cette méthode, de détecter les objets en mouvement d’une façon plus précise tout en éliminant les trous qui existaient dans les méthodes précédentes et détecter ainsi l’objet en entier.

## Le flot optique

### Introduction sur le flot optique

Le concept du flot optique a été introduit par le psychologue américain James J.Gibson lors son étude sur le système visuel humain. Par définition, le flot optique est le motif du mouvement apparent des objets, des surfaces et des contours dans une scène visuelle, causé par un mouvement relatif entre un observateur (œil ou caméra) et la scène (10).

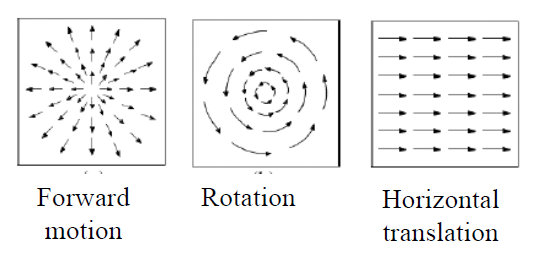
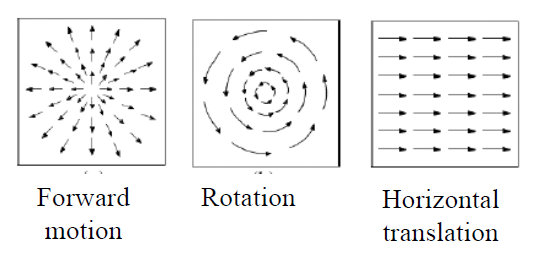
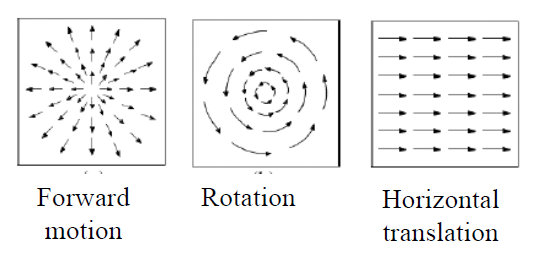
En vision par ordinateur, quand nous parlons des méthodes du flot optique, nous étudions réellement le champ du flot optique. Afin de comprendre c’est quoi le champ du flot optique, nous devons comprendre un nouveau terme : le champ de mouvement.

Le champ de mouvement décrit le mouvement tridimensionnel dans le monde réel, y compris la vitesse tridimensionnelle en termes d’amplitude et de direction. Cependant, lorsque nous observons le monde à partir d’une caméra, ce que nous voyons est une séquence d’images en deux dimensions. Le monde réel tridimensionnel a été projeté dans un plan bidimensionnel. Donc, afin de décrire le mouvement des objets dans les images, le champ du flot optique est utilisé comme un substitut du champ de mouvement.

Le champ du flot optique est un champ de vitesse bidimensionnel présenté dans une image comme la projection du champ de mouvement tridimensionnel. Une fois le flot optique est calculé, nous pouvons connaitre le mouvement des objets dans l’image, bien que cela ne soit pas exactement le champ du mouvement réel, mais ça reste toujours utile. C’est pourquoi la méthode du flot optique est considérée l’une des solutions classiques pour estimer le mouvement. Les figures suivantes montrent un exemple du champ de mouvement et un exemple du flot optique.

A : Mouvement en avant

B : Rotation



C : Mouvement horizontale

Figure ‎I‑4: le champ de mouvement

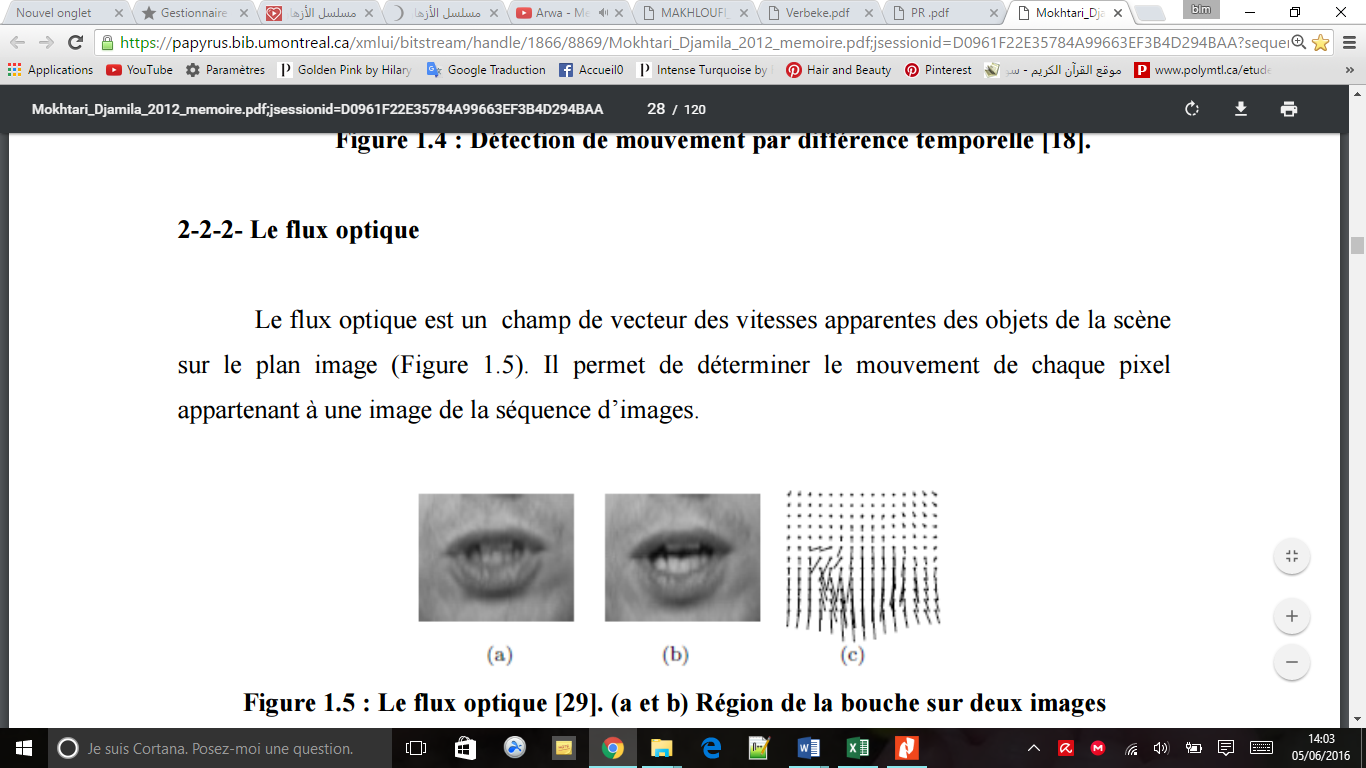


Figure ‑: Le flot optique

Finalement, le but du flot optique consiste à estimer le vecteur vitesse de chaque pixel, défini comme flot optique dense, en utilisant l’information spatio-temporelle fournie par une séquence vidéo.

Depuis 1980, le terme de flot optique est utilisé dans plusieurs applications de traitement d’images : reconstruction 3D d’une scène, détection de mouvement, compression des vidéos… La méthode d’estimation du flot optique, dite de bas niveau, a été initialement proposée par HORN et SHUNCK (11) . Plusieurs contributions, proposant divers méthodes, ont suivi ce premier travail. Quelle que soit la méthode choisie, elle ne pourra pas être temps réel si nous requérions une grande précision et ceci est dû au fait que le calcul du flot reste une tâche coûteuse en temps de calcul.

Les méthodes du flot optique peuvent être utilisées indépendamment des objets en mouvement et même si nous utilisons une caméra en mouvement. Cependant, la plupart de ces méthodes sont complexes et très sensibles au bruit et ne peuvent pas être appliquées à des flots vidéo en temps réel sans un matériel spécialisé.

### Principe du flot optique

Le calcul du flot est basé sur l’intensité constante, c'est-à-dire que l’intensité observée se conserve entre deux images successives. Cette manière de faire suppose donc, que le mouvement est faible et instantané d’une trame à une autre et que le changement entre deux trames successives n’est causé que par le mouvement.

Selon cette hypothèse d’intensité constante (12), l’intensité d’un pixel X à un instant t s’écrit comme suit :

I (X, t) = I(X + ΔX, t +Δt) (‑)

*Où : ΔX représente le déplacement qu’a effectué un pixel X dans un intervalle de temps Δt*

L’équation (I.6) est développée en série de Taylor pour obtenir le vecteur vitesse.

∇ + It = 0 (‑)

*Avec : ∇I=(,) et It = les dérivées partielle d’ordre 1 de l’intensité respectivement par rapport à (x, y) et t*

*T*

Cette équation est appelé équation de **contrainte de mouvement**. Elle peut bien évidemment être écrite comme suit :

(‎I‑8)

*Où Ix =, Iy= Vx=représentent les vitesses de déplacement suivant l’axe x et l’axe y*

C’est une équation d’une droite dans l’espace 2D. Elle permet d’exprimer le vecteur vitesse dans la direction du gradient spatial, il s’agit de la vitesse normale. Le flot normal est ainsi en fonction du gradient spatio-temporel de l’image.

= (III‑9)

Mais, cette équation reste insuffisante pour déterminer le flot complet puisque le problème est mal posé ; nous avons une seule équation linéaire qui possède deux inconnues. Cela veut dire que nous aurons besoin d’une contrainte en plus pour déterminer le flot optique. Dans ce contexte, plusieurs méthodes de calcul ont été développées. En littérature, les auteurs ont classifié les méthodes en cinq catégories :

* Méthodes différentielles
* Méthodes fréquentielles ou énergétiques
* Méthodes basées sur la corrélation
* Méthodes «*Multiple Motion* »
* Méthodes de raffinement temporelles

Dans cet état de l’art, nous allons seulement nous intéressé aux méthodes les plus utilisées dans la littérature. Il s’agit des méthodes différentielles et celle basées sur la corrélation. Des états de l’art sur les autres méthodes peuvent être trouvés dans (12).

#### Les méthodes différentielles

Les méthodes différentielles estiment le flot optique en se basant sur l’information spatio-temporelle. Elles introduisent une nouvelle contrainte sur la contrainte de mouvement.

Selon cette contrainte, la méthode différentielle peut être considérée comme méthode globale ou locales. En fait, dans la méthode globale, la contrainte s’applique sur toute l’image. Par contre, dans la méthode locale, la contrainte ne s’applique que sur un bloc de pixel de l’image.

##### Horn et Shunk

La méthode de Horn et Shunk est la première méthode de calcul proposée pour le calcul du flot optique. Ils ont tenté de minimiser sur le domaine entier de l’image l’amplitude du gradient du flot optique. Cette contrainte est appelé **une contrainte de lissage sur le gradient** dite aussi de **régulation**. Cette contrainte est appliquée sur toute l’image. Cette méthode est donc considérée comme une approche globale.

Le flot optique est calculé en minimisant la quantité suivante :

(‎I‑10)

Dans cette équation, il existe un terme de donnée qui représente la contrainte de mouvement (I.8) et un terme de lissage. Intuitivement, minimiser cette équation revient d’une part à minimiser la contrainte de mouvement ainsi que la variation spatiale du champ de flot optique.

Cette approche nous permet d’avoir un flot dense (un résultat pour chaque pixel). En plus, ce résultat est lisse puisque les mouvements principaux sont identifiés et non pas ce qui varient localement. Pour remédier à ça, il existe une méthode de minimisation itérative qui consiste à recalculer de nouveau le flot selon le dernier champ de vitesse calculé et ainsi avoir un résultat meilleur.

Le calcul du flot optique pour cette approche revient à l’application d’une méthode d’optimisation qui minimise cette intégrale double. La solution de cette équation se résout par exemple par la méthode de Gauss-Seidel. Le résultat est donc donné de manière itérative comme suit :

(‎I‑11)

*Où : [,] moyenne des voisins du pixel à traiter*

*: Coefficient de régularisation*

Le coefficient de régularisation doit être réglé empiriquement et selon l’application. De même, le nombre d’itération de la méthode de résolution se fixe aléatoirement.

##### Lucas et kanade

A l’inverse de la méthode de Horn et Shunk, la méthode de Lucas et kanade est une méthode locale. Elle consiste à prendre des hypothèses sur un domaine plus petit pour différencier le flot optique (une petite variation dans un endroit de l’image influence sur tout le champ du flot optique). Ils minimisent un critère sur un domaine plus petit et obtiennent un flot optique sur ce petit domaine (13).

La nouvelle contrainte ajoutée pour ce type d’approche est une contrainte appliquée sur le voisinage d’un pixel bien particulier. En fait, la vitesse est supposée constante sur un voisinage Ω. Le flot optique est alors calculé en minimisant la contrainte de mouvement localement sur un voisinage Ω. Ce qui permet d’obtenir un système de plusieurs équations à deux inconnues. Le flot optique est calculé en minimisant la quantité suivante :

*(‎I‑12**)*

*Où W est la fenêtre locale qui peut être interprétée comme une pondération de la méthode des moindres carrés*

Cette méthode locale est très intéressante puisqu’elle est parallélisable (le calcul sur une fenêtre particulière est indépendant des autres). Les résultats sont moins sensibles aux bruits et les variations locales sont prises en compte.

La solution la plus simple pour cette équation consiste à faire une estimation au sens des moindres carrés. Autrement dit, La résolution se fait de la façon suivante :

Soient A= , b=, alors V== -1 (‎I‑13)

*Où : n représente le nombre des voisins du pixel dont nous voulons calculer la vitesse*

Ce qui est équivalent à :

G= , µ = , alors V==G-1 µ (‎I‑14)

L’estimateur des moindres carrées est très sensible au bruit et aux erreurs de mesure. En effet, la matrice soit mal conditionnée, autrement dit avec un déterminant quasi nul, donc non inversible. Cela provient du fait que le pixel à traité se trouve dans une région où l’intensité des pixels ne varient pas sur le domaine, d’où un gradient nul. C’est ce qui peut conduire à une estimation aberrante.

Pour remédier à ce genre de problème, une technique de régulation est utilisée. La nouvelle équation dérivée de l’équation *(‎I‑12)*. La nouvelle fonction s’écrit comme suit.

(‎I‑15)

L’estimateur des moindres carrés est toujours linéaire et s’exprime de la manière suivante :

= -1 (‎I‑16)

*Où est un terme réglable représentant la régularité de la solution de Richard (14).*

#### Les méthodes basées sur la corrélation

Les méthodes basées sur la corrélation déterminent le flot optique en utilisant la correspondance des modèles. Cette méthode se base sur la recherche d’un déplacement d’un bloc de taille précise dans l’image entre deux images successives.

La mise en correspondance entre les blocs peut être mesurée avec des mesures telles que :

* La somme des différences absolues « *SAD : Sum of Absolute Differences*»
* La somme des différences quadratiques « *SSD : Sum of Squared Differences*»
* Le coefficient de corrélation

SAD= (‎I‑17)

SSD= (‎I‑18)

Corrélation = (‎I‑19)

Afin de chercher la distance de correspondance minimale, il faut soit utiliser l’équation de la somme des différences absolues ou la somme des différences quadratiques. Alors que le coefficient de corrélation est utilisé pour trouver le bloc assurant la corrélation maximale, ou encore le bloc qui correspond le mieux au bloc de départ.

Pour chercher le bloc d’intérêt, il faut bien évidemment parcourir toute l’image. Cette recherche est extrêmement exhaustive. Elle donne néanmoins de bons résultats mais son coût calculatoire est très élevé. Il existe des méthodes dans la littérature qui permettent de remédier à ce problème en spécifiant un déplacement maximale des blocs, à partir duquel, la correspondance est non calculée. Néanmoins, cette méthode est très peu utilisée et ne va pas figurer dans cet état de l’art.

### Les méthodes hybrides

Les méthodes hybrides pour détecter un mouvement sont une fusion des méthodes qui se basent sur la modélisation de l’image de fond et des méthodes qui ne cherchent pas à modéliser l’arrière-plan.

Collins et *al.* (15) présentent une méthode de détection de mouvement pour pouvoir l’appliquer pour un système de vidéo surveillance qui se repose sur la combinaison de la méthode de la différence temporelle en utilisant trois images et la méthode de soustraction de l’image de fond. La méthode de la différence temporelle utilisant trois images leur a permis de détecter les régions en mouvement tandis que la méthode de soustraction de l’image de fond leur a permis d’extraire tout le mouvement en comblant les trous.

La méthode différentielle entre trois images considère qu’un pixel est en mouvement si son intensité change de manière significative entre l'image courante et la dernière image, et l'image courante et l'image dernière-suivante Le pixel est en mouvement si :

(‎I‑20)

L’image de fond est initialisée comme étant la première image de la séquence et elle est mise à jour comme suit :

= (‎I‑21)

Le seuil est initialisé par une valeur quelconque non nulle puis elle est mise à jour comme suit :

(‎I‑22)

Cette méthode hybride qui combine deux méthodes de détection de mouvement donne de bons résultats comme indiqué sur la Figure ‎I‑6.

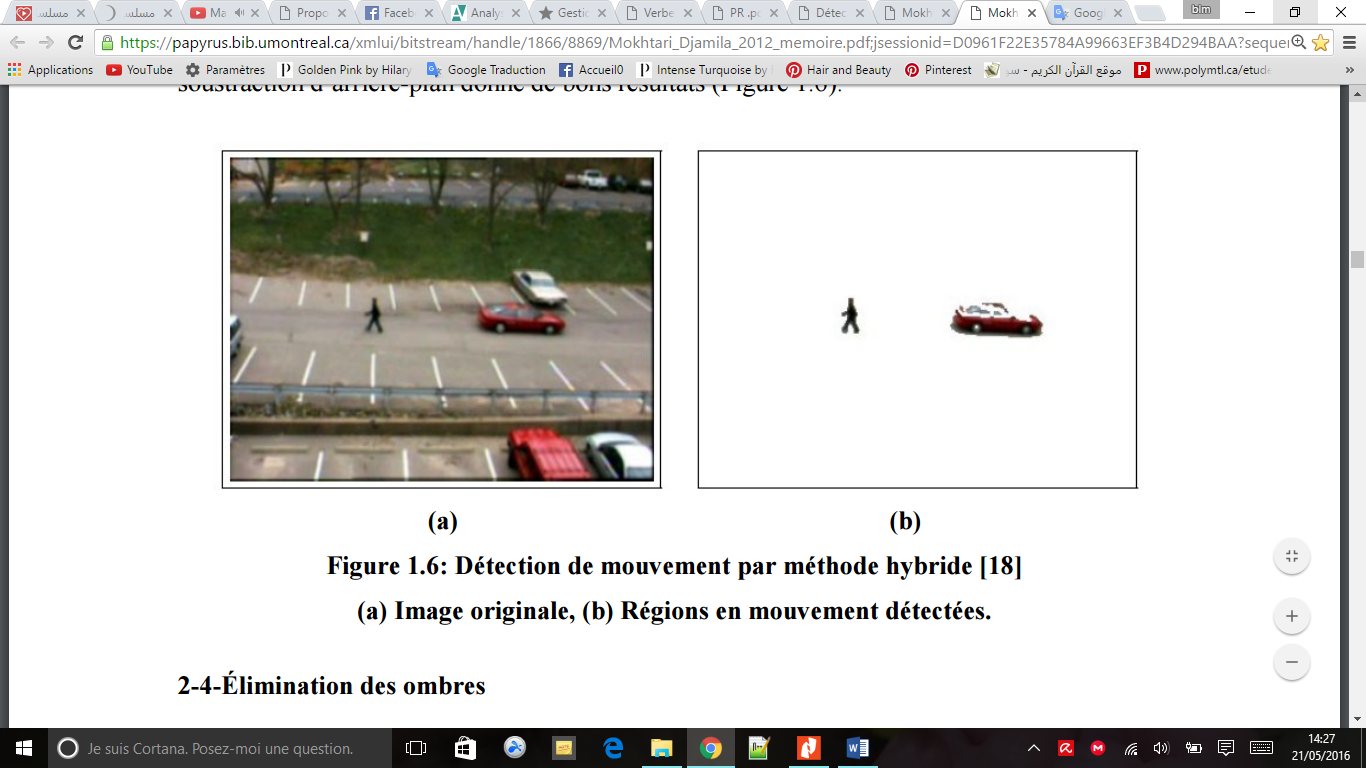
‎I.2.2. a.1

Figure ‑: Détection de mouvement par la méthode de Collins et al. (15).

Une autre méthode hybride a été présentée par Kui Liu et *al.* (16). Leur méthode se base sur la combinaison de la méthode du flot optique et la méthode d’analyse en composante principale. Ils appliquent l’approche de l’analyse de données pour analyser le flot optique et l’ACP peut efficacement détecter les objets en mouvement et éliminer le bruit. C’est une méthode qui peut efficacement être utilisée que ce soit pour un arrière-plan statique ou dynamique.

Deux images sont construites en utilisant les pixels des vecteurs du flot optique. Le flot optique est calculé par la méthode de Horn & Shunk discutée dans la section ‎I.2.2. a.1. Ensuite, une analyse de données est appliquée sur ces deux images pour déterminer le mouvement. Un masque de taille nxn glisse à travers les images u et v. En un point (i, j), une matrice de donnée X bidimensionnelle peut être construite en contenant tous les vecteurs 2D couvert par le masque. La matrice de covariance peut être calculée ainsi :

Après la décomposition de la matrice de covariance en valeur propre, deux valeurs propres (λ1, λ2) sont assignés au pixel central du masque. La détection de mouvement est accomplie en analysant ou en faisant le seuillage des valeurs propres. Puisque λ1 est la composante majeure du flot et λ2 est la composante mineure du flot, ils ont jugé qu’il est plus efficace de considérer les deux valeurs propres que les valeurs des images originales u et v.

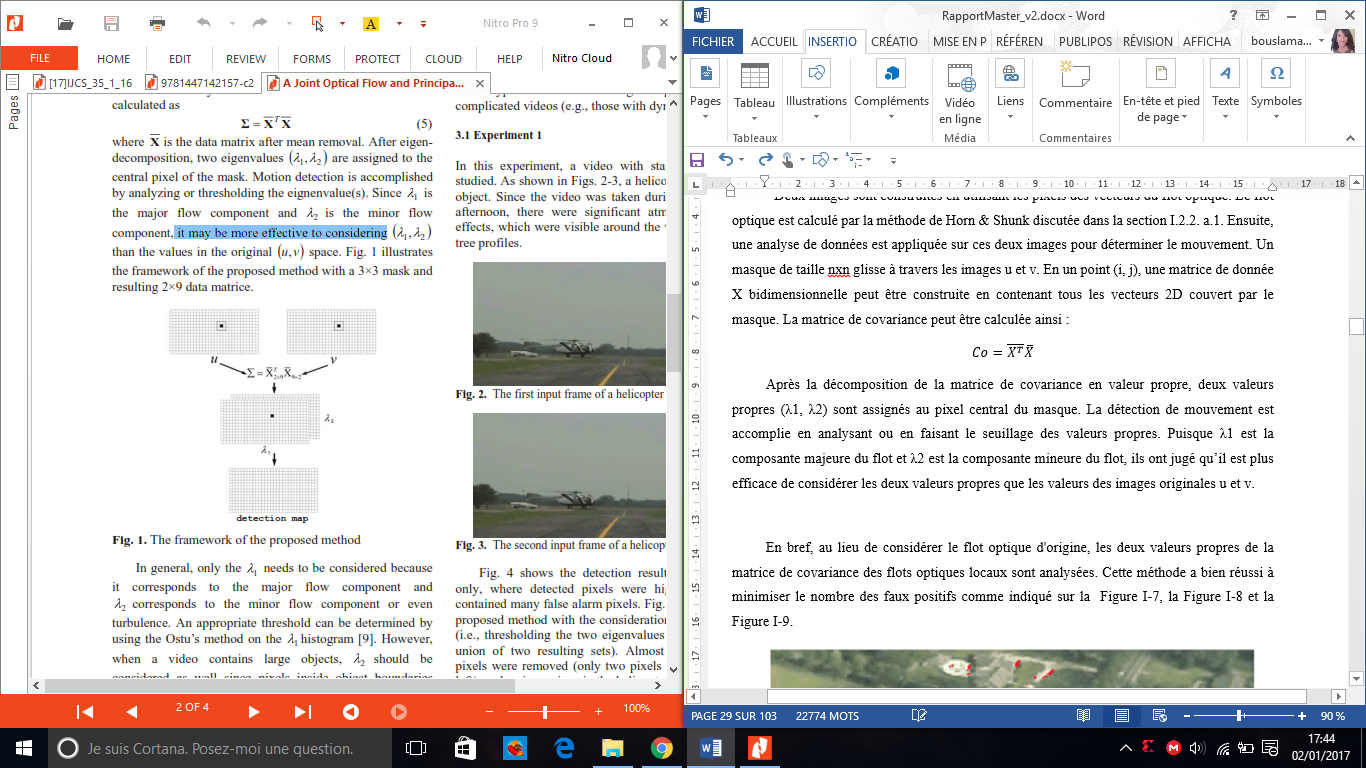


Figure ‑: Structure de la méthode (16)

En bref, au lieu de considérer le flot optique d'origine, les deux valeurs propres de la matrice de covariance des flots optiques locaux sont analysées et seulement λ1 sera considéré puisqu’elle correspond à la composante majeure du flot. Cette méthode a été testée sur une séquence d’images où le mouvement n’est causé que par l’hélicoptère.



Figure ‎I‑8: La séquence d'image d’un hélicoptère en mouvement

Cette méthode a bien réussi à minimiser le nombre des faux positifs comme indiqué sur la Figure ‎I‑9, la Figure ‎I‑10 et la Figure ‎I‑11. Les pixels détectés sont marqués avec la couleur rouge. La méthode du flot optique représenté sur la Figure ‎I‑9 contient beaucoup de fausses alarmes. Le résultat de la méthode est représenté sur la Figure ‎I‑10. Ici, tous les faux positifs sauf deux sont éliminés et la partie majeure de l’hélicoptère en mouvement est détectée.



Figure ‑:Résultat de la méthode du flot optique



Figure ‑: Résultat de la méthode combinée (16)



Figure ‑: Résultat de la méthode de soustraction de l'arrière-plan

## Conclusion

La détection de mouvement joue un rôle trop important dans les applications de vision temps réel. Les chercheurs ont développé de nombreuses méthodes pour traiter les différentes situations. Ces algorithmes peuvent être divisés en trois méthodes : différence inter-images, le flot optique et les méthodes hybrides. En particulier, le calcul du flot optique est utilisé dans nombreuses applications. On lui trouve l’analyse de mouvement de fluides en physique expérimentale, la compression de séquence d’images vidéo par compensation de mouvement, la détection d’obstacles dans le domaine automobile ou encore son utilisation pour des phases de traitement d’images de plus haut niveau comme la reconstruction de scènes tridimensionnelles. Dans le dernier chapitre, nous allons découvrir une implémentation optimisée d’une méthode pour le calcul du flot optique sur la carte ZedBoard. Cette méthode est très intensif en terme de temps de calcul et nécessite afin qu’elle peut être temps réel une architecture performante tel que les FPGAs. C’est dans ce contexte que nous allons découvrir dans le chapitre suivant une introduction sur le prototypage des applications de vision sur les FPGAs.

# Introduction sur le prototypage d’applications de vision sur FPGA

## Introduction

Les applications concernant les domaines de vision se retrouvent dans de nombreux marchés. Ces applications sont de plus en plus impressionnantes et leur complexité ne cesse pas de croitre. Le défi des nouveaux systèmes englobant une large gamme d’applications de vision nécessitent un traitement des images à haut-performance. Dans ce contexte, nous présentons dans la première section de ce chapitre les motivations liées à l’implémentation de ces applications nécessitant un calcul temps-réel sur les FPGAs. Nous présenterons par la suite les différentes autres architectures dédiées aux implémentations des applications prévisionnelles de vision tout en expliquant les différences majeures entre eux. Le circuit FPGA le plus populaire de nos jours dans l’industrie est le circuit Zynq (17). Nous allons pour cela se focaliser sur la description de ce circuit dans la section suivante de ce chapitre. Ainsi quelques outils de synthèse haut niveau qui permettent le prototypage rapide des différentes applications de vision sont représentés dans la cinquième section de ce document. Nous allons détailler en particulier l’outil de synthèse haut niveau faisant partie de l’environnement de développement du circuit Zynq.

## Motivation

La vision intelligente est devenue de plus en plus populaire dans une grande variété de domaines dans l’industrie. Certaines applications d’algorithmes de vision incluent la détection de visage qui peut être trouvé dans les appareils-photo modernes ou les nouveaux smartphones, la reconnaissance des plaques d’immatriculation utilisé par la loi, la détection de la voie pour la conduite autonome et le suivi des objets utilisé dans les drones commerciaux.

Bien qu'il ne soit pas nécessaire d'avoir une capacité de traitement vidéo en temps réel pour certaines applications, d’autres telles que la détection de voies et la détection d'objets pour les systèmes de conduite autonome nécessiterait un flot d'entrée provenant de caméras à traiter en temps pour pouvoir communiquer des signaux à l'unité de commande et de direction afin de répondre correctement à la tâche conçue.

Les FPGAs sont des plateformes hautement souhaitées pour le traitement des vidéos en temps réel en vue de leur capacité d'extraire des calculs fortement parallélisés et de leur efficacité énergétique. En outre, les FPGAs à notre porté aujourd’hui contiennent des processeurs embarqués, des communications internes ultrarapides et d’autres fonctionnalités. Ils sont plus denses et plus rapides que jamais ce qui augmente la capacité de réaliser de systèmes complexes et performants. Le potentiel de reconfigurabilité joue aussi un rôle trop important puisque l’évolution des standards peut être prise en compte.

Cependant, le développement matériel consomme normalement plus de temps et de connaissances humaines qu'un développement logiciel semblable consommerait. Pour un développement basé sur des FPGAs, le développeur doit obligatoirement avoir de bonne connaissance du circuit logique numérique ainsi que certains langages de description matérielle ‘’ HDL*: Hardware Description Langage* ‘’ tels que Verilog et VHDL pour construire et configurer des circuits au niveau registre ‘’ RTL : *Register Transfer Level*’’ dans un FPGA.

Ces dernières années, le développement sur les FPGAs a été orienté vers des niveaux d'abstraction plus élevés. Cette évolution contribue non seulement à améliorer la productivité, mais abaisse également la barrière pour plus de concepteurs logiciels à avoir un accès aux attrayantes plateformes basées sur les FPGAs. En plus, il existe un large choix d'outils disponibles sur le marché qui peuvent être utilisés pour la synthèse de haut niveau. Conventionnellement, les concepteurs logiciels préfèrent utiliser des langages de haut niveau tels que C / C ++ pour le développement d'algorithmes. Le nouveau outil de Xilinx, Vivado HLS, est l'un des outils capable de synthétiser un code C / C ++ et le transformer en un code au niveau RTL pour une implémentation matérielle.

La plupart des outils de synthèse haut niveau ne pouvaient pas traduire directement un algorithme écrit avec un langage de haut niveau en une implémentation au niveau RTL, et les utilisateurs devaient restructurer leurs algorithmes afin de les rendre synthétisables et adaptées à une architecture matérielle spécifique. Par conséquent, il devient important de s'adapter à l'outil de synthèse haut niveau et de découvrir les approches permettant d'atteindre une conception efficace avec des performances élevées et une faible utilisation des ressources.

## Architectures dédiées aux implémentations d’algorithmes de vision

Le plus grand défi d’une plateforme de vision, consiste à traiter les données vidéo en temps réel puisque ce n’est pas toujours possible de gérer un tel système en utilisant un ordinateur, étant donné la limitation en termes de ressources.

Cependant, le portage d’un algorithme de vision dans une plateforme spécifique présente un défi puisque ce type de système doit couvrir certaines autres exigences liées à la flexibilité, le coût, la performance, l’ergonomie et la consommation d’énergie.

* Le coût détermine finalement la viabilité d’une solution. Il est parmi les critères majeurs car nous prenons en compte les coûts humains relatifs aux différentes phases de prototypage et les coûts matériels.
* L’ergonomie définit la facilité liée à l’utilisation d’un système
* La flexibilité est liée à l’adaptation de la solution proposée aux impératifs liés à l’évolution des algorithmes et des standards
* La consommation est l’un des critères primordiaux dans les systèmes autonome car il quantifie l’énergie dissipée.

C’est dans ce contexte que nous introduisons les principales architectures permettant l’implémentation des algorithmes de vision. Nous pouvons citer : les microprocesseurs, les ASICs, les DSPs et les FPGAs.

L’amélioration de la performance d’un microprocesseur a été largement évoluée par l’intégration d’un nombre de plus en plus croissant des transistors sur une seule puce. La loi de Moore prévoit que le nombre de transistors sera doublé tous les deux ans, ce qui est avérée valable et faisable depuis les dernières années jusqu'aujourd’hui. Cependant, à moins que les scientifiques puissent trouver de meilleurs matériaux que le silicium ou faire progresser la technologie de fabrication, le rythme croissant deviendra plus lent puisque la densité des transistors et leurs consommations d'énergie continueront de pousser à la limite. De plus, l'un des plus grands défis qui empêche un seul cœur CPU d'améliorer ses performances est la puissance : la consommation d’énergie et la température. Cette puissance limite la possibilité pour qu’un processeur puisse augmenter sa fréquence afin d’améliorer ses performances puisque le fait d’augmenter la fréquence pourrait provoquer une consommation excessive d’énergie et ainsi la surchauffe du composant. La consommation d’énergie d’un processeur est devenue si cruciale puisque les processeurs ordinaires maintiennent, depuis plusieurs années, toujours une fréquence d’horloge de presque 3GHz. Au lieu d’augmenter la fréquence d’horloge ou encore la densité des transistors, l’architecture des ordinateurs doit passer le cap et se focaliser sur les architectures de calcul hétérogènes, les DSPs ‘’*Digital Signal Processor*’’ et les GPUs ‘’ *Graphics Processing Unit*’’. En outre, les logiques programmables telles que les FPGAs et les ASICs valent la peine d’être investigués pour devenir l’architecture informatique du futur vu leur haute performance et leur efficacité énergétique.

La Figure ‎I.1 montre les avantages et les compromis de chaque architecture mentionnée. Alors que les circuits logiques programmables tels que les FPGAs et les ASICs sont en mesure de fournir la plus grande performance et la plus grande efficacité énergétique, les développements sur ces plateformes peuvent être coûteux. En revanche, comme les développeurs logiciels peuvent étendre les développements logiciels aux DSPs et aux GPUs, le coût de développement est moins important. En tant que compromis, les DSPs et les GPUs ne peuvent pas fournir le même niveau de performance et d'efficacité énergétique que ce qui peut être réalisé dans une conception logique personnalisée.

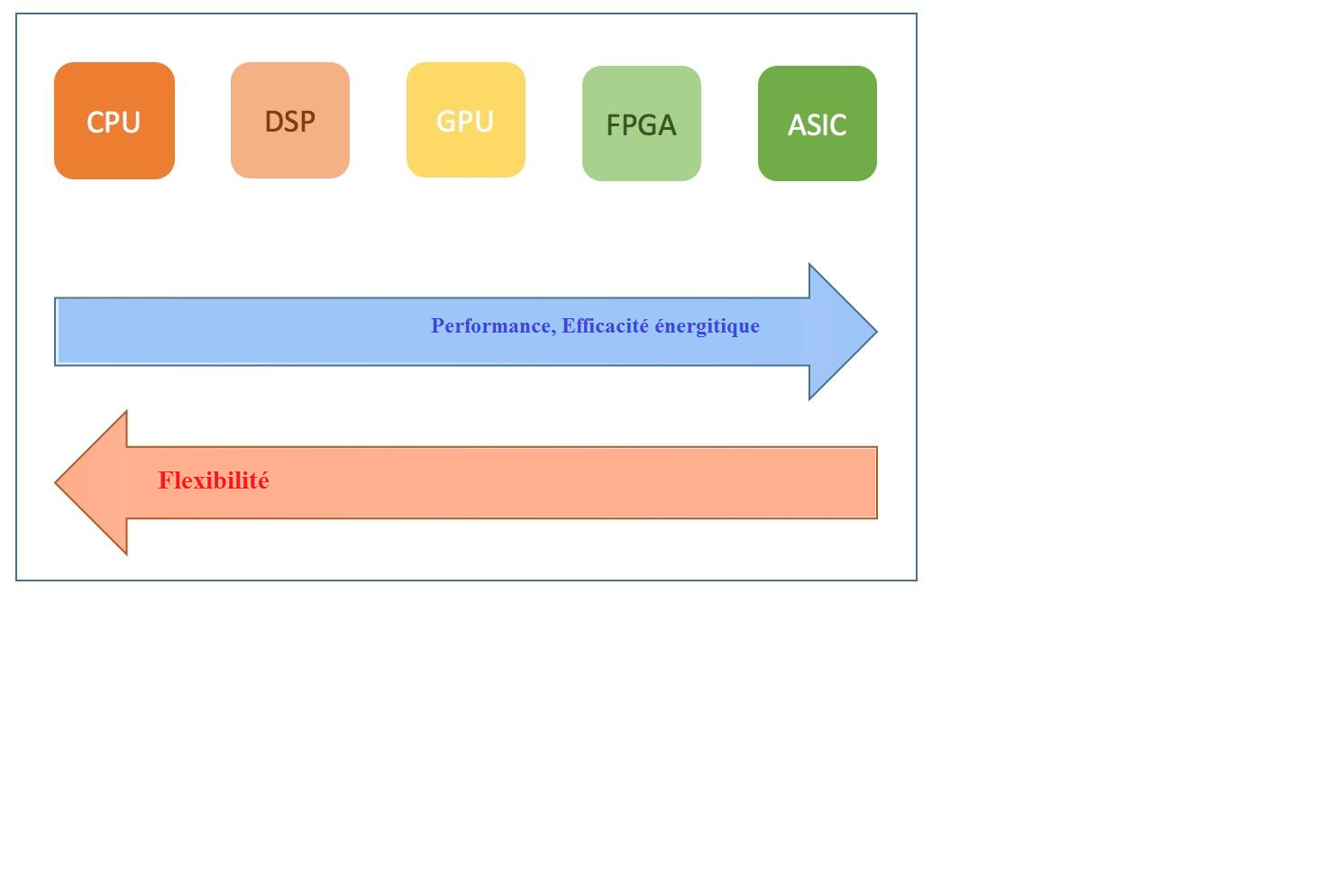


Figure ‑: Comparaison entre les différentes architectures

### Les DSPs

Le DSP *‘’Digital Signal Processor’’* est un type de microprocesseur personnalisé capable d'accélérer des applications impliquant une grande quantité de traitement des signaux numériques tels que le traitement audio et vidéo. Comme un DSP dispose d'une architecture spécialisée et optimisée pour le traitement du signal numérique, les appareils dotés de DSP tels que les tablettes et les smartphones peuvent exécuter des processus en utilisant moins d'énergie dans certaines catégories par rapport à ce qu'un microprocesseur polyvalent peut offrir. Les DSPs sont couramment observés dans les SoC actuels. Avec les nouveaux DSPs, les appareils mobiles peuvent offrir des performances exceptionnelles de caméra et d'audio tout en économisant l'énergie et ainsi améliorer les performances de la batterie. Bien que les DSPs requièrent une certaine connaissance matérielle, les développeurs logiciels feront moins d’effort pour apprendre la programmation sur les DSPs par rapport à d’autres plateformes telle que les FPGAs, ce qui réduit la TTM ‘’*Time To Market*’’.

Les DSPs ont été la première alternative dans les applications de vision. Ils sont basés sur l’architecture VLIW *‘’Very Large Instruction Word’’* et sont conçus pour accélérer le calcul des opérations arithmétiques. En plus, ils offrent des capacités de traitement en parallèle ce qui augmente considérablement la puissance de calcul. Il existe dans la littérature plusieurs applications de vision où l’implémentation est réalisée sur les DSPs comme pour le cas d’une application d’assistance à la conduite qui consiste à détecter la voie pour les voitures et l’avertissement de collision (18) et encore Zhang et *al.* ont implémenté une application d’assistance à la conduite sur les DSPs (19).

### Les GPUs

Le processeur graphique GPU ‘’*Graphical Processing Unit* ‘’ est parmi les architectures le plus populaire de ceux ayant une structure hautement parallèle. Il est très utilisé dans le cas d’applications avec des calculs intensives et ceci est dû à ses performances élevées et sa structure parallèle.

En tant qu’architecture mono / multi-cœurs, un processeur GPU moderne a typiquement beaucoup plus de cœurs qu’un CPU multi-cœur, ce qui permet l’exécution simultané de plusieurs calculs. Un processeur GPU typique est fortement parallélisable et repose sur le pipelining massif des opérations afin d’atteindre de hauts performances. Dans une tel architecture typique, il peut y avoir de centaines de multiprocesseurs et chaque multiprocesseur peut comprendre un ensemble de processus ‘’*thread’’*. La plupart d’entre eux utilisent l’architecture SIMD ‘*’Single Input Multiple Data*’’ et chaque processeur thread exécute la même opération pour différentes données. En outre, les GPUs ont une mémoire indépendante de la mémoire hôte et ils ont une bande passante mémoire extrêmement importante. Les multiprocesseurs sont *multithread* et ils sont capables de switcher rapidement entre la récupération des données et leurs exécutions. Par conséquent, les GPUs peuvent généralement offrir un rendement massif alors qu’il y ait une latence élevée pendant l’accès à la mémoire entre la mémoire GPU et les unités de traitement.

### Les FPGAs

Les FPGA sont de plus en plus populaires en tant qu’une architecture configurable conçue pour des applications qui requièrent des calculs intensives. Les FPGAs ont généralement le potentiel de fournir une amélioration remarquable des performances par rapport aux autres architectures quand il s’agit des calculs conventionnels tout en consommant constamment moins d'énergie. Les FPGA ont été largement adoptés dans diverses applications telles que le calcul complexes pour la science, le traitement du signal numérique à haute vitesse, le traitement réseau à faible latence et le traitement vidéo en temps réel. La capacité de reconfiguration des FPGA d'aujourd'hui permet aux utilisateurs de modéliser de façon flexible une application spécifique au matériel.

Bien que les performances d'une implémentation sur les FPGAs puissent être plus réduite qu'un ASIC ‘’ *Application Specific Integrated Circuit*’’ et que les FPGAs consomment plus de surface qu'un ASIC, les FPGAs sont nettement moins chers que les ASICs, beaucoup plus faciles à fabriquer et à utiliser et hautement personnalisable.

Un circuit FPGA est composé de milliers de blocs logiques programmables et d'interconnexions afin que les ingénieurs matériel puissent personnaliser leurs fonctions logiques spécialisées pour leurs applications. Comme le montre la Figure ‎I.2, les blocs logiques à l'intérieur d'un FPGA sont composés de bloc logique basique, des blocs de mémoire, des blocs de traitement de signaux numériques (DSP) tels que des blocs multiplicateurs, des FF *‘’Flip Flops*’’, LUT *‘’Look-Up Tables’’* et beaucoup d’autres. Les programmeurs peuvent contrôler les connexions entre les différents blocs logiques et périphériques d'E/S en configurant les interconnexions entourées afin de créer des circuits numériques complexes.

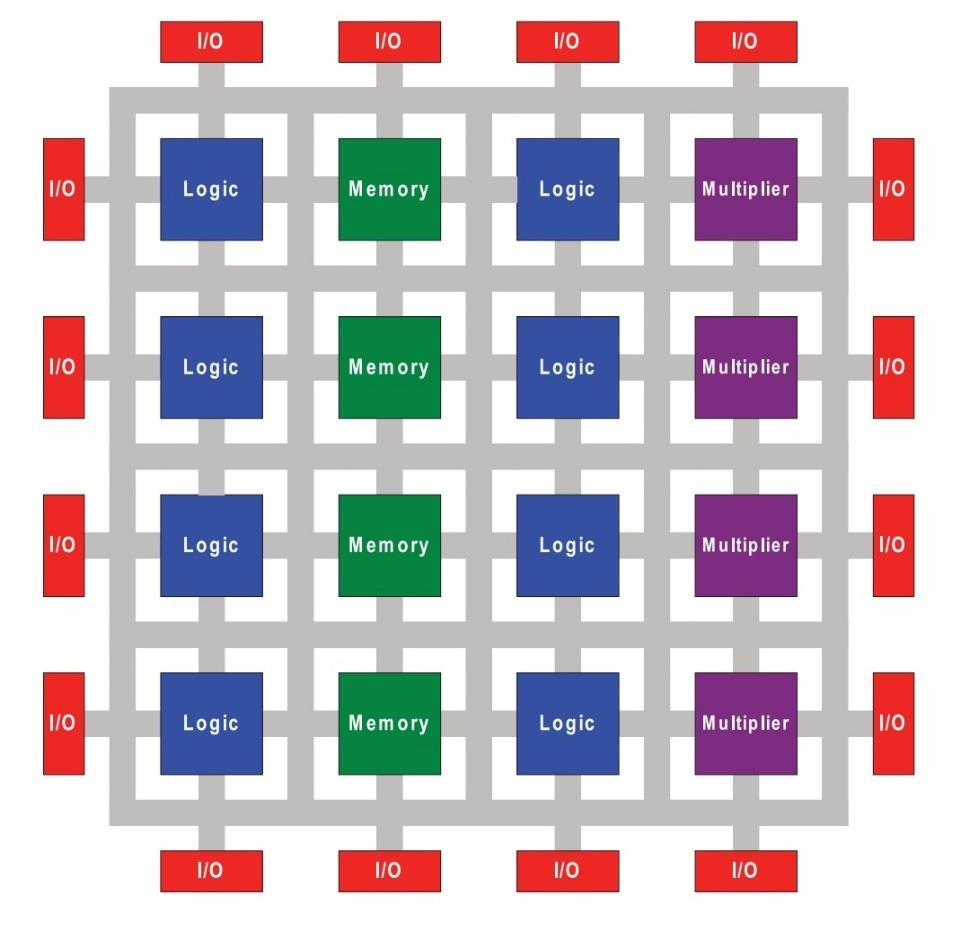


Figure ‑: Architecture FPGA (20)

Un flot de conception typique des FPGAs implique la conception de l’architecture du design, le codage HDL, la simulation comportementale, la synthèse, l’implémentation, la vérification fonctionnelle, le timing et l'analyse de puissance comme le montre la Figure ‎I.3.

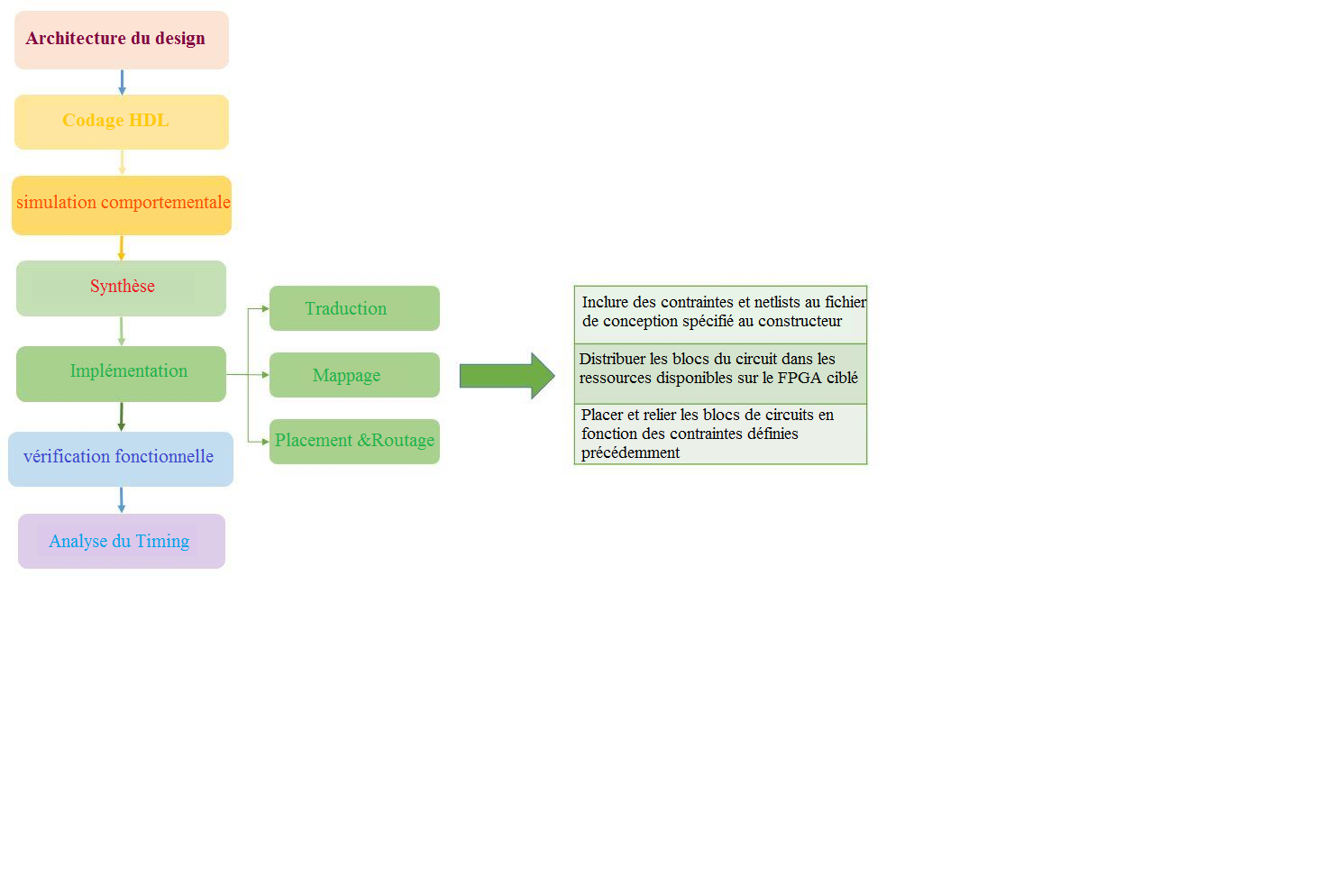


Figure ‑: Flot de conception sur les FPGAs

Des outils de deux grands fournisseurs de circuit FPGAs, Altera et Xilinx, sont hautement automatisés et ils ont rendu le développement sur les FPGAs plus que jamais efficace. Bien que certaines étapes tels que la synthèse, l’implémentation, le timing et l'analyse de puissance puissent être pris en charge automatiquement à travers ces outils, il est cruciale aux développeurs de concevoir l’architecture et de créer des algorithmes qui peuvent réaliser les fonctionnalités souhaiter et de respecter les contraintes imposées et le timing. Bien que les FPGAs puissent offrir une meilleure flexibilité que les ASICs, l’implémentation au niveau RTL prend beaucoup plus de temps puisque cela exige plus de compétence et de connaissance sur l’architecture du circuit.

L’une des solutions les plus compétitives de nos jours concernant les systèmes à base d’FPGA est le circuit **Zynq** de la famille Xilinx (17). C’est le circuit utilisé pendant ce projet.

### Les ASICs

Les ASICs ‘’*Application-Specific Integrated Circuits*’’ sont notamment des circuits intégrés personnalisés conçu pour une application précise. Un ASIC peut être utilisé pour certaines applications qui exigent des performances élevées et doivent consommer moins d’énergie. En plus, l’un des plus grands avantages de choisir un ASIC pour une application qui nécessite beaucoup d’espace est le coût. En fait, vu que les ASICs sont entièrement personnalisables, les développeurs peuvent économiser les ressources dont ils n’ont pas besoin et ceci peut réduire considérablement le coût.

Il y a désormais un nombre d’inconvénients quand il s’agit de les utiliser. Contrairement aux FPGAs et aux microprocesseurs, la conception ASIC ne peut pas être reconfigurable une fois le processus de fabrication est effectué. Cela diminue l’intérêt de leur utilisation pour des applications nécessitant généralement des mises à jour. En plus de la faible flexibilité d'une conception ASIC, le flot de conception est également plus complexe car les processus de routage, de placement et d'analyse de timing ne peuvent pas être effectués automatiquement par un logiciel.

Bien que les ASICs puissent offrir des performances encore meilleures et une efficacité énergétique plus élevée que les FPGAs, les ASICs peuvent être très inefficaces en termes de coûts de développement et l’indicateur TTM est excédent. Le temps de fabrication d’un design dans un circuit ASIC et sa validation peuvent prendre plusieurs mois à plusieurs années (20). L’indicateur TTM est l'une des raisons les plus cruciales qu'une entreprise doit prendra en compte pour déterminer si c'est une bonne option de prototypage ou non. En outre, il faut beaucoup de ressources techniques et humaines pour une conception ASIC, et ces facteurs vont probablement augmenter rapidement le coût du développement ASIC.

## Les architectures embarquées

On appelle système embarqué tout système combinant le matériel et le logiciel, conçu pour des fonctions spécifiques, mais n’est pas un ordinateur d’usage général. Il est généralement caractérisé par des contraintes plus au moins sévères liées à la consommation d’énergie, la taille, les performances, l’encombrement mémoire et les contraintes temps réel …

Les architectures embarqués sont diverses. Leur variété dépend essentiellement de leur puissance de calcul, la taille, le coût, la complexité de leur programmation et la consommation énergétique. La plateforme jugée efficace est celle qui fournit une grande puissance de calcul et un faible coût, qui possède une petite taille et qui est facilement programmable. Afin d’aboutir à ces performances, plusieurs architectures ont été créées. Cependant, l’un des problèmes majeurs des systèmes embarqués sont les surcharges système. Ils provoquent l’augmentation de la latence des données, une gestion des interruptions retardée, et un débit de données plus faible. Le traitement parallèle, que les FPGAs peuvent accomplir, est efficace pour améliorer les performances des systèmes critiques. Par conséquent, une solution traditionnelle est la construction des systèmes hybrides, combinant un ou plusieurs microprocesseurs avec un FPGA. Ceci permettra l’amélioration des performances. Dans ce contexte, le Zynq-7000 AP SOC est apparu. Cette nouvelle technologie combine une matrice de FPGA avec un ou plusieurs processeurs. Dans notre projet, nous allons utiliser une carte qui se base sur ce circuit.

## Zynq

Ce FPGA est considéré comme le premier SoC ‘’*System On Chip*’’ commercial combinant un microprocesseur avec un circuit programmable. Le circuit Zynq-7000, dont son architecture est représentée sur la Figure ‎I.4, est composé de deux processeurs ARM Cortex-A9 double cœur et une partie logique identique aux FPGAs. Pour ce circuit, on appelle PS ‘’*Processing System’’* la partie processeur et ses périphériques tels que : UART, SPI, DMA, Timers, etc. et PL ‘*’Programmable Logic’’* la logique programmable (représenté en couleur jaune).

Les processeurs ARM Cortex-A9 double cœur sont le cœur du PS. Ils peuvent fonctionner à une fréquence d’horloge de 1GHz. Mais, le PS ne contient pas que les processeurs ARM, il comprend aussi un ensemble de ressource pour le traitement tel que : l’unité de traitement multimédia, l’unité de traitement à virgule flottante ‘’*FPU*’’, la mémoire cache de niveau 1 et 2, une mémoire sur puce OCM ‘’*On Chip Memory* ‘’, des contrôleurs de mémoire pour DDR2 et DDR3 SDRAM… En outre, les utilisateurs peuvent interagir avec le contrôleur d’interruption et divers E/S telles que SPI, I2C, CAN et UART via le logiciel. Il inclut aussi une large gamme de fonction d’interfaçage spécifiques tel que les interfaces standards AXI ‘’*Advanced eXtensible Interface*’’ pour la communication rapide entre le PS et le PL et les interfaces MOIs ‘*’Multiplexed Input/Output’’* pour la communication avec les périphérique externes. Selon Xilinx, la partie programmable est de type Artix-7 et a une capacité de 13,300 cellules logiques, 220 DSP48E1 et 280 blocks de mémoires RAM.

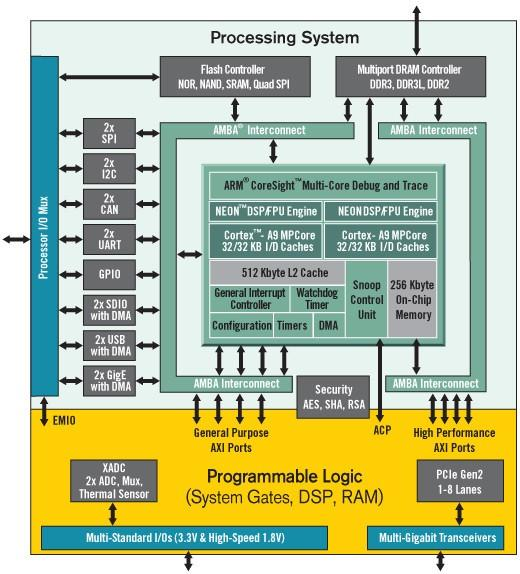


Figure ‑ : Architecture du Zynq-7000 (17)

Xilinx offre également des circuits FPGA plus substantiels dans la famille Zynq tels que le Zynq-7030 qui est équipé d'un Kintex-7 qui peut fournir des utilisations plus polyvalents ainsi que des performances plus élevées.

## Les outils de synthèse haut niveau

Traditionnellement, les développeurs logiciels devaient consacrer beaucoup de temps à apprendre la programmation sur les FPGAs quand il s’agissait d’utiliser des langages de description matérielle comme VHDL et Verilog. En effet, la modélisation sur un matériel est très différente de la conception sur un logiciel et nécessite de bonnes connaissances sur le matériel utilisé. L’écriture d’un code informatique d’une façon séquentielle en utilisant un langage tel que C/C++ correspond parfaitement à la façon dont un cerveau humain réfléchit. En outre, développer une application sur un circuit FPGA pour une fonction donnée requiert plus d’effort que le développement de la même fonction via le logiciel puisque les étapes de synthèse, de simulation et de vérification exigent beaucoup de temps. Visiblement, il y a une barrière pour les développeurs logiciels de se lancer sur le développement matériel. Heureusement, il existe maintenant une variété d’outils de synthèse haut niveau qui permettront non seulement aux développeurs logiciels avec des compétences limitées sur le matériel de concevoir un design matériel, mais aussi aux développeurs matériels en accélérant le processus de développement.

Les outils de synthèse haut-niveau sont capables de traduire des langages de programmation de haut niveau tels que C/C++ en un code RTL dans le but de programmer un circuit FPGA spécifique. Les outils de synthèse haut niveau ne sont pas de nouvelles idées, les outils de première génération ont été étudié depuis les années 1970 et ils n’ont été adopté qu’à partir des années 1990 (21).

### Vivado HLS

Vivado HLS est un outil de conception qui permet de traduire un programme en un composant matériel en automatisant la conversion des algorithmes C/C+ + ou system C en un code RTL. C’est une méthode de conception potentiellement rapide, permettant une description d’un plus haut niveau d’abstraction pour des sous-systèmes. Ceci est particulièrement pertinent dans les systèmes électroniques d’aujourd’hui où le TTM ‘’ *Time To Market*’’ est devenu un facteur de plus en plus important. En plus, C/C++ est, pour les développeurs logiciels, l’un des langages de programmation le plus efficace pour écrire leurs algorithmes et même pour les développeurs matériels ayant une expérience avec l’écriture HDL, l’outil de synthèse haut niveau peut leur fournir une manière pour améliorer les performances et économiser des ressources.

Comme le montre la Figure ‎I.5, l’étape de codage HDL et l’étape de la simulation comportementale dans le flot de conception FPGA classique peut être remplacée par le flot de conception de Vivado HLS. Les utilisateurs peuvent donc utiliser Vivado HLS pour créer des implémentations au niveau RTL et ceci en partant d’un code C/C++ ou system C des restructurations appropriées de concevoir un accélérateur ‘’IP’’ afin de l’intégrer dans un design. Ainsi, Vivado HLS est devenu de plus en plus populaire pour la modélisation d'algorithmes de traitement de signal numérique qui comprend des algorithmes de traitement d'image et de vision.

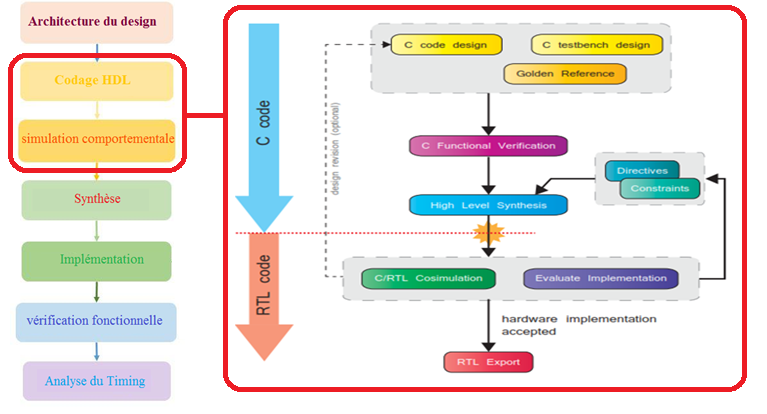


Figure ‑: Flot de conception sur FPGA avec HLS

Néanmoins, le flot de conception complet comporte d’autres étapes intermédiaires incluant des éléments de vérification en particulier. En effet, Vivado HLS permet aux développeurs de vérifier leur conception avec la C / RTL co-simulation afin de corriger le fonctionnement. En plus, les développeurs auront la possibilité d'améliorer encore le timing et les performances en insérant des directives d'optimisation pour le code ainsi que d'exploiter plus de parallélismes dans l'algorithme.

Ainsi, le flot de conception de Vivado HLS comporte :

* Les fichiers Sources de HLS
* La synthèse haut-niveau
* La C/RTL co-simulation
* Evaluation de l’implémentation
* Les fichiers sources de HLS :

La première étape consiste à écrire les fichiers sources en C/C++ ou SystemC. Les fonctions développées doivent être testées pour vérifier leur bon fonctionnement. C’est pourquoi un testbench est requis. Donc, le testbench doit comporter une référence appelée ‘’*golden reference*‘’ pour pouvoir la comparer avec les résultats produits par la fonction prévue pour la synthèse comme montre la Figure ‎I.6: Fonctionnalité du testbench. La référence peut être sous la forme d’une image préparée à l’avance ou générée directement depuis le testbench.

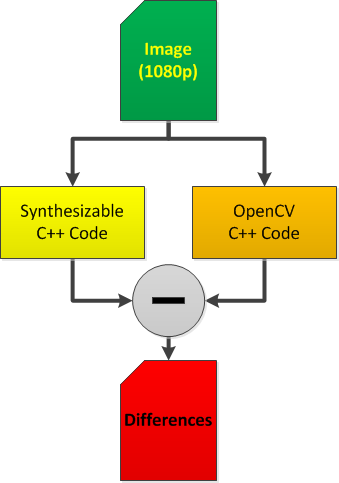


Figure ‑: Fonctionnalité du testbench

* La Synthèse Haut-niveau :

L’étape suivante consiste à traiter le code écrit en C/C++ ou SystemC, ainsi que des directives et des contraintes fournies par l’utilisateur pour créer une description RTL du circuit. Une directive c’est en fait une instruction donnée qui indique comment les fonctions devraient être traduites dans le matériel. Une fois cette étape terminée, un ensemble de fichiers sont produits, notamment les fichiers de conception au niveau RTL, des scripts et beaucoup d’autres fichiers.

* C/RTL Co-simulation :

Une fois le modèle RTL équivalent produit, il peut être vérifié par rapport au code C/ C++ via l’outil C/RTL co-simulation. Cette étape réutilise le testbench pour déterminer les entrées pour la version RTL générée par HLS et vérifie si les sorties produites sont équivalentes aux sortie attendues.

* Evaluation de l’implémentation :

Cette étape est très importante parce qu’il est nécessaire d’évaluer le code RTL en terme de performance et d’implémentation. Par exemple, il faut vérifier le nombre de ressources requises dans le PL, la latence du design, la fréquence maximale de l’horloge supporté et si le timing est respecté.

* Exporter le RTL :

Une fois la conception du design validée, il sera intégré dans un système plus complet. Ceci peut être réalisé directement en utilisant les fichiers RTL créé par HLS (code VHDL ou Verilog). Mais, il est plus pratique d’utiliser les outils de Vivado HLS pour faire le packaging de l’accélérateur. Le packaging réalisé par Vivado HLS veut dire que l’accélérateur pourra être facilement utilisé dans d’autres outils de Xilinx, par exemple l’intégrateur d’IP.

### Autres **outils** de synthèse haut niveau

Ils existent d’autres outils récents de synthèse haut niveau tels que : Bluespec, Catapult C Synthesis, HDL Coder. De nombreux leaders de l'industrie tels que : Ericsson, Nokia, Toshiba et Fujitsu se sont basés sur l’outil Catapult C Synthesis, qui a été produit par Cadence, pour générer des algorithmes dans diverses applications (21). Ce dernier supporte non seulement la génération de code HDL à partir de C ++, mais aussi de SystemC, ce qui offre la possibilité de contrôler le timing pour certaines parties du design.

De plus, Bluespec SystemVerilog est un langage de description matériel innovant qui peut réduire considérablement la complexité de la modélisation d'une conception matérielle et fournit un niveau d’abstraction plus élevé, autant pour la modélisation comportementale que structurale, par rapport à la description matériel basique comme Verilog et VHDL.

Il existe aussi MATLAB qui est, depuis longtemps, un outil puissant quand il s’agit de traitement d'image. Diffusé par la même société, HDL Coder est un outil capable de générer des codes VHDL et Verilog à partir des fonctions prédéfinies par MATLAB afin que les développeurs puissent intégrer le code généré à un IP et l’utiliser dans une conception FPGA. L'émergence des outils avancés de synthèse haut niveau apporte de l’aide aux développeurs logiciels ayant des connaissances limitées en HDL mais qui veulent encore accélérer leurs algorithmes sur les FPGAs tout en traduisant leur code logiciel en un code matériel avec un minimum d’effort.

## Conclusion

Ce chapitre était consacré à l’évaluation des systèmes dédiés aux traitements d’images et à l’exposition du circuit Zynq qui est l’un des circuits les plus performants et le plus rapide pour le prototypage d’une application de vision sur les FPGAs. Nous avons aussi présenté les différents outils de synthèse haut niveau qui permettent aux non-spécialistes en développement sur les FPGAs de développer facilement leur application en se basant sur un langage haut niveau comme C/C++ et pouvoir rapidement la convertir en un langage matériel afin de l’intégrer au sein du circuit FPGA. Le chapitre suivant présentera le flot de conception complet du prototypage d’une application de vision sur la carte ZedBoard basée sur le circuit Zynq. Nous allons mettre l’accent sur l’architecture matérielle, prototypée sur le circuit Zynq de la carte, susceptible de traiter un flot vidéo provenant d’une caméra ainsi qu’aux différentes configurations matérielles et logicielles nécessaires pour le traitement des images.

# Prototypage rapide d’un système de traitement de vidéo sur la carte ZedBoard

## Introduction

Implémenter une application de vision sur un circuit basé sur la logique programmable tel que le circuit FPGA-Zynq représente un challenge pour les développeurs ayant déjà utilisé des circuits tels que les ASICs où un langage de description matériel comme VHDL et Verilog. A travers ce chapitre, nous allons présenter un flot de conception rapide permettant de prototyper une application de vision sur le circuit Zynq. Nous allons, en fait, suivre le procédé de traitement complet depuis l’acquisition des données depuis une caméra jusqu’à l’affichage des résultats en passant par les étapes de traitement, de communication et de stockage. L’environnement de développement Xilinx est notre principal outil pour atteindre notre objectif qui consiste à implémenter un algorithme de détection de mouvement sur la plateforme embarquée ZedBoard. Dans la première section de ce chapitre, nous allons présenter cette carte de développement. Ensuite, nous donnerons un petit aperçu sur l’environnement de développement Xilinx. Dans la troisième section de ce chapitre, nous introduisons le flot de conception à base du circuit Zynq et dans la quatrième section la réalisation du design d’un système de traitement de vidéo. Nous expliquerons notamment les différents composants utilisés pour la réalisation d’un tel design. Ainsi, les différentes configurations matérielles et logicielles, le procédé de communication et de stockage des données sont exploitées dans les sections qui suivent.



## La plateforme ZedBoard

La carte de développement utilisée dans ce projet est la carte ZedBoard (*Zynq Evaluation & Development Board*) conçue par Digilent et Avnet. Cette carte utilise le circuit FPGA Zynq commercialisé par Xilinx dont nous avons présenté son architecture dans la section‎ I.4.

Au niveau connectique, la Zedboard est assez complète et permet d’interfacer de nombreux éléments comme indiqué dans la Figure ‎II.1 .Ses caractéristiques sont résumées ci-dessous.

|  |  |
| --- | --- |
|  | Caractéristiques |
| Processeur: Zynq-7000 SoPC XC7Z020 | * Jusqu’à 667MHz. * Le moteur de traitement NEON de type SIMD ‘’Single Input Multiple Data’’ pour accélérer les calculs DSP. * ARM Double cœur Cortex-A9 |
| Mémoires | * 512 Mo de mémoire DDR3. * 256Mo Quad Flash SPI. * Une carte mémoire de 4GB au minimum. |
| Communication | * Programmation USB-JTAG. * Ethernet 10/100/1000 Mbps. * USB OTG 2.0 et le pont USB-UART. |
| Horloge | * Horloge de 33.33333MHz pour le PS. * Oscillateur de 100MHz pour PL |
| Affichage | * Sortie HDMI supportant 1080p60 avec une résolution couleur de 16 bits. * Sortie VGA avec une résolution couleur de 12 bits. * Affichage OLED 128x32. |
| Audio | * Audi Line-in. * Line-out. * Casque. * Microphone. |
| Composants IHM | * 9 LEDs. * 7 boutons poussoirs. * 8 interrupteurs. |
| Configuration et débogage | * Interface USB-JTAG bord. * Connecteur Xilinx Platform Câble JTAG. * Connecteur FMC * USB OTG 2.0. |

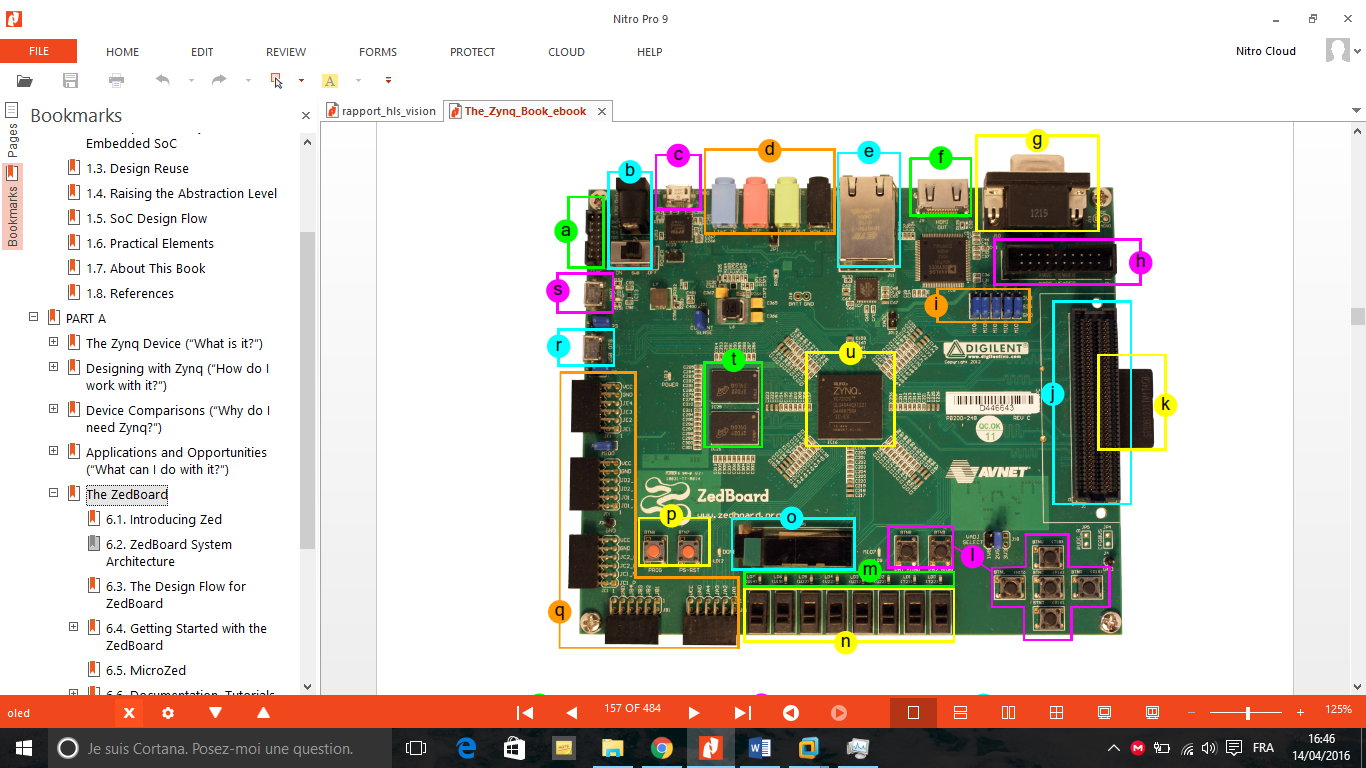




Figure ‑: La carte ZedBoard et ses interfaces

## Logiciels requis

La suite d’outils Vivado conçue par Xilinx est l’environnement idéal pour tout dispositif à base du circuit Zynq, tel que la ZedBoard. Vivado comporte les outils de développement, conception et d’analyse pour toute plateforme à base de ce circuit. Il s’agit de Vivado HLS, Vivado IDE et SDK et. Vivado HLS a été introduit dans la section ‎I.5.1 alors que SDK et Vivado IDE sont introduits par la suite :

* **Vivado IDE**

C’est un environnement de développement désigné pour la création de la partie matérielle ; i.e. le processeur, les mémoires, les accélérateurs, les interfaces externes et les liaisons de bus. Il interagit aussi avec d’autres outils de Vivado et permet d’inclure facilement des IPs personnalisés ce qui augmente les possibilités de réutilisation du design. Il permet en plus de vérifier le design, de faire la synthèse, le placement et le routage et de générer le Bitstream.

* **SDK *‘’Software Development Kit’’***

C’est l’environnement dans lequel les applications logicielles peuvent être créées, compilées et débogués. C’est un outil basé sur la plateforme Eclipse. Il supporte les drivers de tous les IPs Xilinx et inclut la chaine de compilation GNU (compilateur GCC, déboguer GDB…).

La spécification de la partie matérielle importée à partir de Vivado IDE peut être utilisée pour configurer le parie logique du circuit. Ceci peut être réalisé via le logiciel SDK en écrivant un programme C/C++ pour contrôler le comportement du matériel. Mais dans notre cas, ce n’est pas via ce logiciel que nous allons configurer la partie matérielle.

Afin de communiquer avec le matériel, la première étape consiste à décider quel système d’exploitation va être utilisé puisqu’il permet de faire le lien entre le matériel et l’application logicielle. Cela peut être un véritable système d’exploitation tel que Linux et Android, un système d’exploitation embarqué ou un système d’exploitation temps réel (RTOS). Par conséquent, nous allons présenter dans la section suivante le système d’exploitation Linux embarqué, l’OS qui est utilisé dans le cadre de ce projet.

## Flot de conception à base du Zynq

Dans cette section, nous allons présenter un flot de conception rapide pour le prototypage d’un système de traitement de vidéo, qui permet aux développeurs avec une expérience limité avec le développement au niveau matériel de développer un système de traitement de vidéo sur un FPGA.

Un flot de conception rapide concernant le prototypage d’un système de traitement de vidéo est représenté sur la Figure ‎II.1. Nous allons seulement se concentrer sur trois étapes cruciales du flot.

* Une architecture du système de traitement de vidéo doit être conçue sur le SoC basé sur un FPGA. Le système devrait permettre l’intégration des IPs générés à partir des outils de synthèse haut niveau afin d’aboutir à des performances de traitement en temps réel.
* La conception du design doit être flexible de manière à ce que les accélérateurs conçus par les développeurs en utilisant les outils de synthèse haut niveau puissent rapidement être reconfigurés. Dans notre projet, les algorithmes de traitement de vidéo seront écrits en langage C++ et synthétisés en RTL via Vivado HLS. Les IPs générés pour le traitement de vidéo peuvent être intégrés au niveau du système.
* La communication au niveau du système permet aux développeurs d’utiliser le logiciel ‘’software’’ pour initialiser et configurer des modules développés dans le matériel ‘’hardware’’ du système. En plus, les développeurs pourront même interagir avec la mémoire DDR pour la transmission des données entre le matériel et le logiciel.

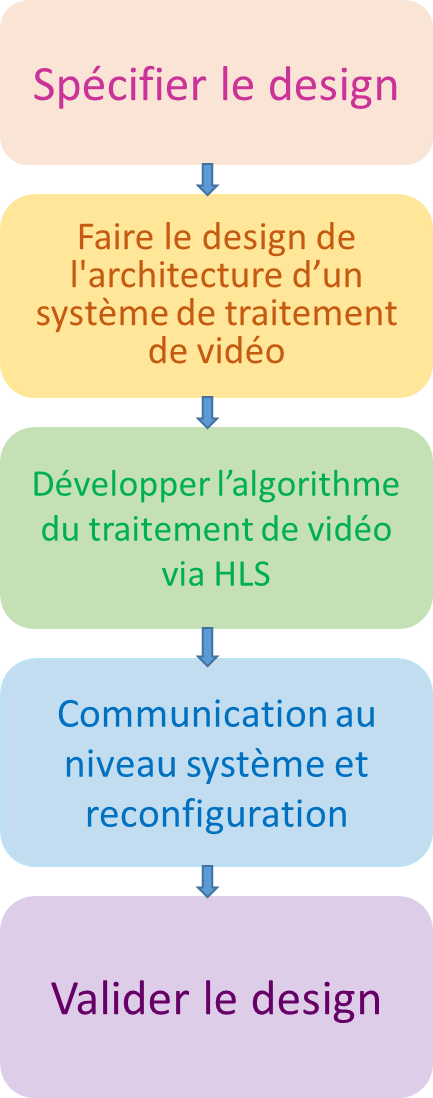


Figure ‑: Flot de prototypage d'un système de traitement de vidéo sur le Zynq

Le développement du système de traitement vidéo en temps réel proposé est divisé en deux parties:

* Conception de l'architecture du système de traitement vidéo
* conception des algorithmes de traitement vidéo.

Au cours de ce chapitre, nous allons traiter les principaux composants contribuant au design du système de traitement vidéo sur la plateforme Zynq, en étudiant les différents IPs ‘*’Intellectual Property’’* et interfaces utilisés et dans le dernier chapitre de ce rapport, nous traiterons la méthode de détection de mouvement à implémenter et les différentes optimisations utilisé pour le développement de l’IP sous Vivado HLS.



## Design de l’architecture d’un système de traitement vidéo

Le système a été conçu pour traiter la vidéo avec le moins temps de latence possible. Pour cette raison, nous avons décidé de concevoir un système où tous les modules utilisés seront intégrés dans la partie logique du circuit plutôt que sur la partie processeurs. Ceci est dû au fait que la partie logique est capable d’effectuer le traitement en parallèle et aboutir à des applications temps réel. Néanmoins, il existe une latence associée au mouvement de la vidéo via la mémoire et c’est ce qu’il faut prendre en considération à travers les processeurs de telle sorte que ces transactions doivent être optimisées.

Avant de concevoir un accélérateur de traitement de vidéo, il est essentiel de développer un design pour le traitement de vidéo sur la plateforme Zedboard. Pour cela, il nous faut un module pour capturer le flot vidéo. Pour ce faire, il existe plusieurs accessoires fournis par Digilent que nous pouvons intégrer dans notre plateforme, notamment le module E/S FMC-HDMI qui peut transmettre directement dans la partie logique du circuit Zynq un flot vidéo FHD et recevoir en retour la vidéo extraite de la sortie HDMI. Mais, ce module n’était pas à notre disposition pendant la réalisation du projet.

Dans notre cas, le flot vidéo sera capturé à partir d’une webcam classique et affiché via une interface HDMI. La caméra sera interfacée avec la carte Zedboard via un port USB et l’interface HDMI sera construite en utilisant le design de référence fourni par Analog Devices ADV7511 (22) intégré dans la partie logique du circuit Zynq. Lorsque la vidéo provient de la caméra, les images seront stockées dans la mémoire. Ensuite, la vidéo va être traitée afin d’ajouter la fonction de vision souhaitée. Cette fonction sera développée via le logiciel de synthèse haut niveau Vivado HLS. Finalement, la vidéo doit être transféré vers le processeur afin qu’elle soit affiché au développeur.

Le design globale du système est divisé en plusieurs modules, comme le montre la Figure ‎II.2, cela inclut principalement deux blocs matériel implémentés dans la partie logique du circuit Zynq, le bloc du traitement de vidéo ‘’Video Processing’’ et le bloc d’affichage ‘’ Video Display’’, ainsi que le contrôleur de mémoire DDR qui utilise le ou les processeurs ARM du circuit Zynq.

Un noyau linux est employé sur les processeurs ARM. Il s’agit du *kernel* 4.0 intégrant les pilotes des composants Xilinx et Analog Devices. En termes de distribution logicielle, une version OpenSus a été déployée. Elle nous permet d’intégrer tous les périphériques standards (clavier, souris, écran, caméra…) et permettre ainsi l’utilisation des interfaces HDMI, USB et Ethernet. En plus, elle est compatible avec la majorité des bibliothèques de traitement d’images et de vision dont OpenCV. La procédure du boot de la carte Zedboard peut être trouvée dans le livre du circuit Zynq (23).

La webcam connectée via le port USB permet l’acquisition d’un flot vidéo continu avec différents résolution. Ce couple caméra-SoC permet ainsi de prototyper un système de traitement de vidéo rapide et à faible coût.

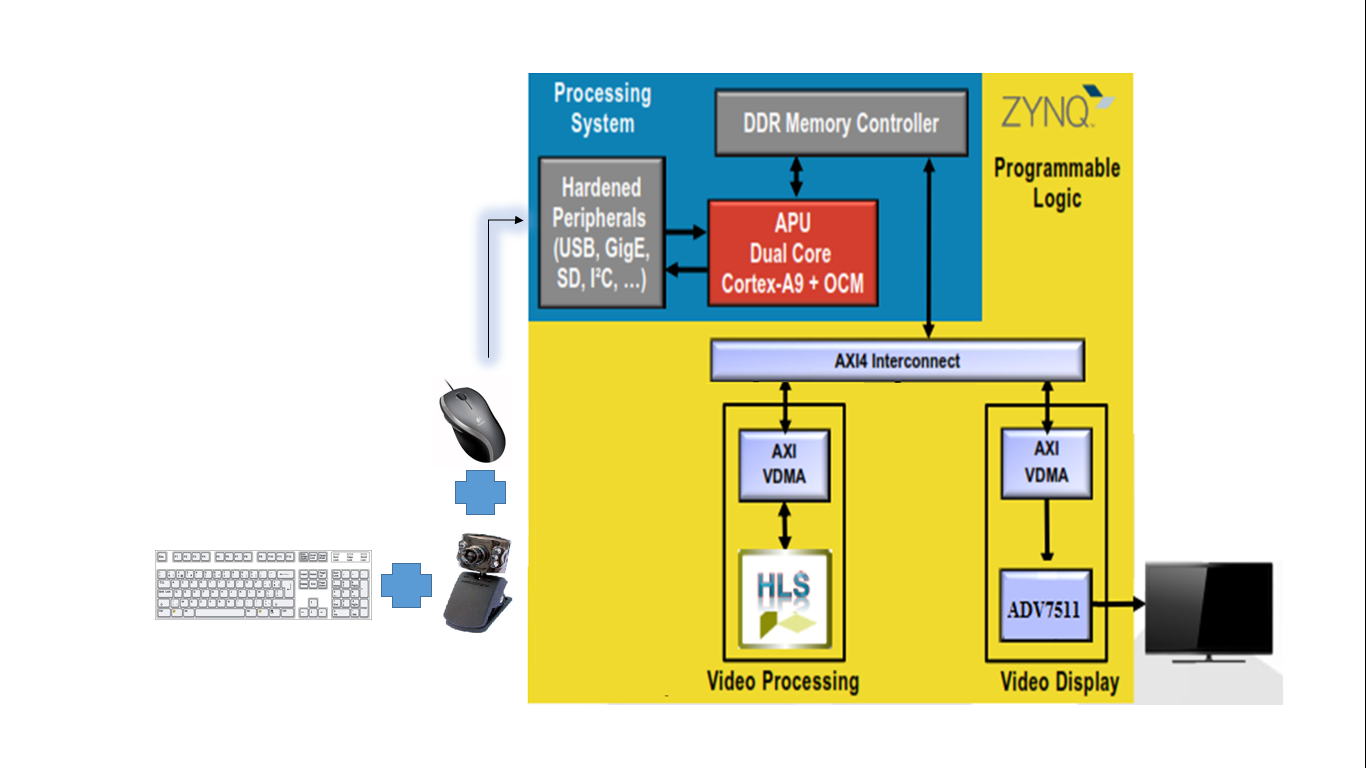


Figure ‑: Design générale d’un système de traitement de vidéo sur le Zynq

En utilisant les processeurs ARM du circuit Zynq, les utilisateurs peuvent accéder au contrôleur de mémoire DDR et configurer tous les modules utilisés dans la partie logique du circuit avec une interface AXI4-Lite. En effet, la communication entre les processeurs el ses composants utilisateurs s’effectue via un bus AMBA et des liaisons normalisées de type AXI.

### Les interfaces AXI

Les performances du circuit Zynq sont remarquables vu le couplage FPGA-Processeur. Mais ces performances ne résident pas dans les propriétés de ses parties constitutives, le PS et le PL, mais dans la possibilité de les utiliser en tandem afin de former des systèmes complets. Le facteur clé à cet égard est l’ensemble des interconnexions et des interfaces AXI formant le pont entre le PS et le PL.

Le protocole de bus AXI est utilisé pour les transactions de données entre le processeur et la partie programmable du circuit Zynq. L'interface AXI permet à différents blocs de communiquer entre eux avec le même standard, AMBA. La norme AMBA régit la connexion entre les blocs fonctionnels, et elle permet aux ingénieurs de prototyper efficacement les conceptions sur un SoC. Il est donc nécessaire de comprendre les protocoles concernant l'interface AXI4 car un nombre trop important d'IP fournis par Xilinx supportent l'interface AXI4. L'interface AXI4 la plus couramment utilisée comprend AXI4-Memory-Mapped, AXI4-Lite et AXI4-Stream. Afin de créer avec succès un système de traitement vidéo sur le périphérique Zynq, il est essentiel de comprendre la différence entre ces trois interfaces, et comment les utiliser correctement.

#### AXI-Memory-Mapped

AXI4-Memory-Mapped est l'interface la plus sophistiquée des trois. Elle est destinée à la transaction maitre-esclave avec la mémoire. Dans notre projet, ils sont utilisés dans la conception pour le transfert de données entre la vidéo du côté de la logique programmable (PL) et la mémoire principale du côté PS. Cette interface contient cinq canaux entre l'esclave AXI et le maître AXI, qui sont le canal de réponse en écriture, le canal de données d'écriture, le canal d'adresse d'écriture, le canal d'adresse de lecture et le canal de réponse de lecture (17). Pour émettre une transaction de lecture / écriture par le maître AXI, ce dernier doit envoyer une adresse de lecture / écriture avec les données à l'esclave AXI.

#### AXI4-Lite

AXI4-Lite est un sous-standard de l’interface AXI4-Memory-Mapped. Il est destiné à la configuration des périphériques mappés en mémoire. Les périphériques dans ce contexte sont tout IP ou matériel contrôlés par un processeur tel qu'un Microblaze ou le système PS. L’interface AXI4-Lite se comporte de la même manière que l'interface AXI4-Memory-Mapped, sauf qu'elle ne prend pas en charge le transfert par un burst de données de 256 mots. Par conséquent, l'interface AXI4-Lite peut être utilisée par le PS pour configurer les paramètres matériels et accéder aux différents registres de contrôle des périphériques. L'interface AXI4- Lite fournit les principaux paramètres matériels à l’application logicielle pour pouvoir piloter le matériel.

#### AXI4-Stream

L’interface AXI4-stream (parfois abrégé AXIS) est sensiblement différent des deux bus AXI précédents car il n'est pas mappé en mémoire. AXIS est un bus maître à esclave standard pour contrôler le flot de données périodiques entre les modules. Il supporte le transfert non limités des données par burst. Il n’existe pas de mécanisme d’adresse. C’est le bus qui convient aux transferts direct des données d’une source à une destination.

### ADV7511

L’ADV7511 est un transmetteur multimédia haute définition ‘’HDMI’’ d’une fréquence de 225 MHz fourni par Analog Devices. Cette conception de référence fournit l'interface audio-vidéo entre le FPGA et l’ADV7511 à bord.

La conception de référence présentée sur la Figure ‎II.3 se compose de deux modules pcores indépendants. La partie vidéo se compose d'une interface VDMA ‘’*Video Direct Memory Access’’* de Xilinx et de l'interface vidéo ADV7511. L'interface ADV7511 utilise une interface de 16 bits 422YCbCr avec des signaux de synchronisation distincts (h-sync,V-synch, data-enable …). Le VDMA fournit les trames reçu de la mémoire DDR à ce composant. Sur l’écran d’affichage, Zynq configure la transmission HDMI via le bus IIC et transmet le streaming vidéo via l’IP contrôleur HDMI (HDMI core).

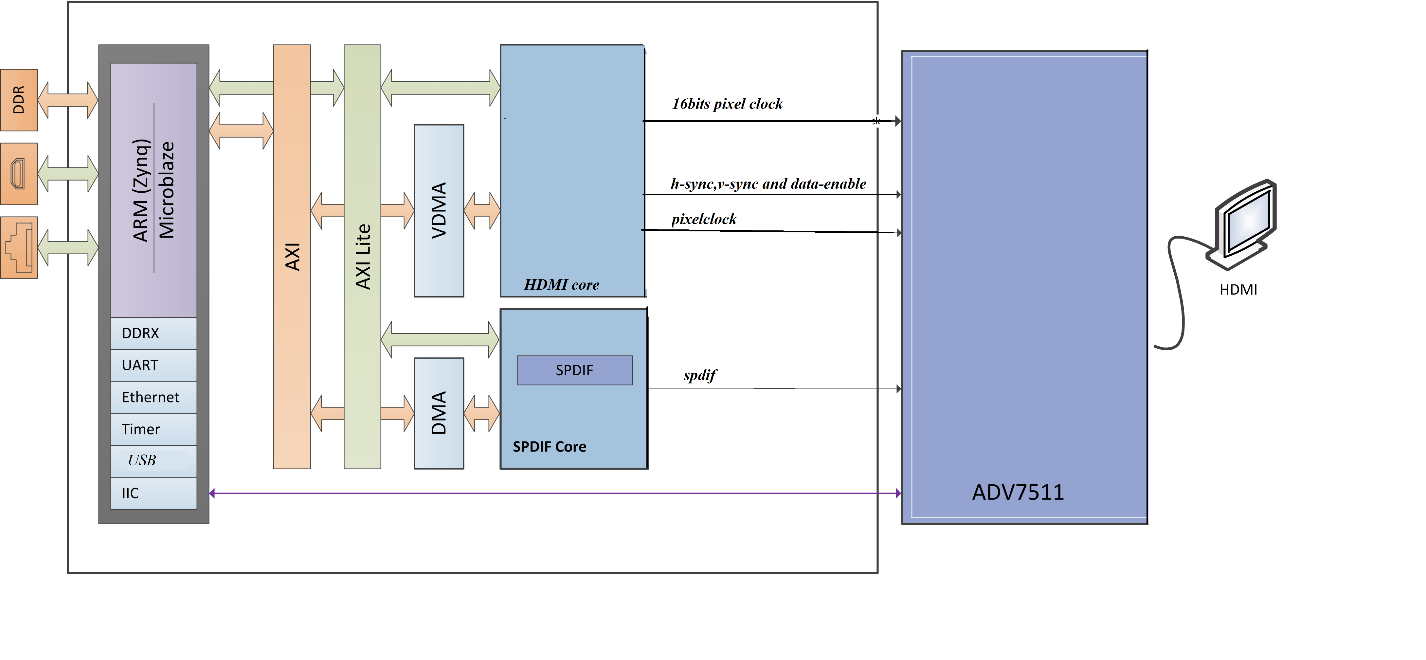


Figure ‑: Architecture ADV7511

### AXI-InterConnect

Un AXI4-interconnect est utilisée pour relier plusieurs maîtres et esclaves dont leur type d’interface est AXI4. Il permet aux esclaves d'hériter le protocole de données provenant des maîtres. De plus, un AXI4-interconnect peut être utilisée pour mapper les données en streaming vers une interface AXI4-memory-mapped afin d’accéder aux périphériques mappés en mémoire. Une interconnexion AXI4 contient une table de décodage d'adresse qui peut être servie aux transactions de lecture et d'écriture entre maîtres et esclaves. Une autre grande fonctionnalité de l'interconnexion AXI4 est la flexibilité d'utiliser différentes horloge. Cette fonctionnalité est activée par l'intégration d'une FIFO asynchrone pour l'adaptation de différentes fréquences d'horloge.

### VDMA

AXI VDMA est une version personnalisée de la famille AXI DMA. AXI VDMA est responsable de la transmission des données des trames entre la partie FPGA et l’interface mémoire DDR3. Il existe plusieurs raisons qui font de l'AXI VDMA une interface mémoire efficace pour une conception du système de traitement vidéo à haut débit. Tout d'abord, le VDMA contient un bloc de transfert qui permet de convertir les types d’interface d’AXI4-Memory-Mapped à AXI\_Stream et ainsi pouvoir communiquer avec les données stockées dans la mémoire DDR. Le flot vidéo sortant de l’IP de traitement d'image est à l'interface de type AXI4-Stream. Cependant, pour stocker l'image dans une mémoire partagée telle que la DDR3 à laquelle les processeurs ARM peuvent également accéder, il est essentiel de transférer l'interface de type AXI4-Stream vers l'interface de type AXI4-Memory-Mapped. De même, lorsque les données doivent être converties de nouveau au type de données AXI4-Stream pour les périphériques de sortie vidéo tels que le HDMI. La Figure ‎II.3 illustre le mécanisme de transfert de données implémentées dans le module VDMA qui peut gérer la conversion entre les interfaces AXI4-Stream et AXI4-Memory-Mapped. Le contrôleur DMA-MM2S ‘’ *Memory to stream’’* et le contrôleur S2MM ‘’*Stream to Memory*’’ sont utilisés pour émettre des signaux de commande afin de configurer le transfert DMA, tandis que l'adresse et le nombre d'octets des données transférées sont enregistrés respectivement dans le registre d'adresses et le registre de donnée.

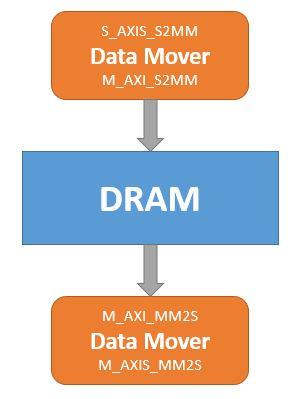


Figure ‑: le transfert des données dans le VDMA

En se basant sur la conception de ce bloc représentée sur la Figure ‎II.5, le port S\_AXIS\_S2MM du *‘’Data-Mover’’* lit les données de l’interface AXI4-Stream, tandis que le port M\_AXI\_S2MM écrit les données converties vers AXI4-Memory-Mapped dans la mémoire DDR externe. D'autre part, le port M\_AXI\_MM2S *‘’Data-Mover’’* lit les données AXI4-Memory-Mapped depuis la DDR et les écrit tout en faisant la conversion en AXI4-Stream sur un périphérique de sortie tel que HDMI via le port M\_AXIS\_MM2S.

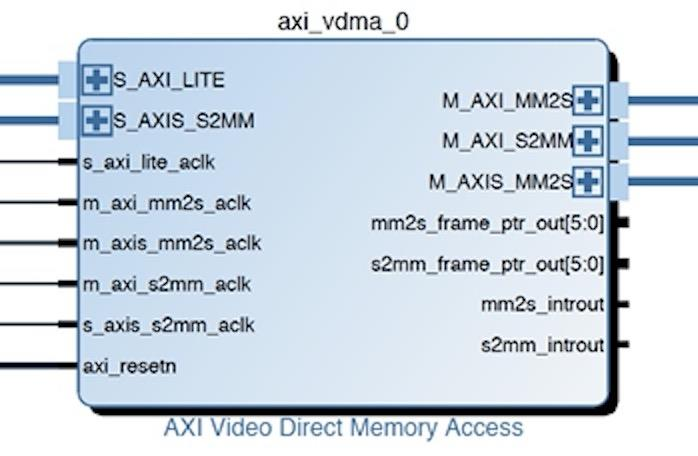


Figure ‑: Le bloc VDMA

En plus, un design de frame buffer peut être réalisé en utilisant l’IP AXI VDMA. Les frames buffers sont instanciés dans la mémoire externe DDR3 et donc ils peuvent être accédés via l’IP VDMA. En outre, un mode Genlock peut être activé pour synchroniser les opérations de lecture et d'écriture dans les frames buffers. En général, au moins deux frames buffers doivent être alloués lors de l'utilisation du mode Genlock. Lorsque le mode Genlock est activé, les trames peuvent être automatiquement ignorées ou répétées si une écriture et une lecture potentielle sur la même trame et en même temps sont détectées. Cela peut être extrêmement pratique lorsque les trames vidéo entrantes ne proviennent pas du même domaine d'horloge. En outre, lors de l'utilisation de VDMA avec une interface mémoire à bande passante élevée, le module VDMA doit être connecté aux ports hauts performance AXI-HP du circuit Zynq.

#### Configuration matérielle des VDMAs

Les configurations matérielles pour le VDMA ne sont pas aussi intuitives qu'elles paraissent, mais une fois comprises, elles peuvent être facilement exploitées pour effectuer de nombreuses opérations DMA utiles. Ces paramètres sont saisis via l'interface de l’IP dans l'environnement de développement Vivado IDE. La plupart d'entre eux ont un effet sur quel opération peut être réalisé par ce module après sa synthèse puisque la désactivation de certaines fonctions aboutira à la suppression de la logique requise depuis l’unité HDL du DMA. Le module VDMA doit également être correctement connecté aux différents IPs transmettant les frames et au PS afin d'accéder à la mémoire principale.

Activer ou désactiver les canaux d’écriture ou de lecture de cet IP est l’un des premiers réglages à effectuer. Par défaut, l'unité VDMA possède à la fois un canal de lecture et d'écriture, ce qui est logique dans le contexte de frame buffer. Les données stockées dans la mémoire principale sont lues à la demande. Dans notre application où la vidéo est acquise par la CPU à partir d’un emplacement précis dans la mémoire principale, le VDMA peut être utilisé avec seulement un canal de lecture afin de copier les données et vice versa. Il existe également une option à l'intérieur de l’unité de configuration de l’IP pour sélectionner le nombre de frames buffers. Il existe aussi des paramètres qui permettent de définir la taille du burst, la taille des données du stream et la profondeur du buffer.

Il existe aussi des buffers FIFO intégrés à l'unité VDMA, car l'unité VDMA doit transférer en paquets ses transactions vers la mémoire principale. L'unité VDMA doit accumuler au moins un paquet de données afin de démarrer une transaction d'écriture. Les paramètres doivent indiquer la quantité de données contenues dans chaque transaction. Des tailles de burst plus importantes et une taille du bus de données plus grandes sont plus efficaces et doivent être utilisé pour le cas des systèmes de traitement de vidéo hauts performances. Une plus grande taille pour le FIFO est également souhaitée dans ce cas d’application puisque la latence de la communication avec la mémoire peut rapidement dépasser le temps dans lequel le FIFO peut être vidé par les différents processus de traitement de la vidéo.

Le mode ‘’Fsync’’ a différents effets sur les canaux de lecture et d'écriture. Sur le canal d'écriture, fsync est extrêmement important et n'est pas défini par défaut. Un signal fsync indique la séparation entre les images de la vidéo. Le VDMA a besoin d'une source fsync pour séparer les trames de la vidéo entrante dans leurs frames buffers appropriés. Sans cela, les trames commenceront à un point quelconque et ne seront pas correctement alignées dans les frames buffers. Un début arbitraire de la trame sera sélectionné en partant de l'hypothèse que les premières données reçues par la VDMA proviennent de la première ligne de l’image et du premier pixel.

Genlock est un paramètre très essentiel qui détermine comment le VDMA va défiler entre les frames. Le mode « *Dynamic master and Dynamic slave* » sont utilisés ensemble si nous souhaitons que le VDMA exécute intérieurement un fonctionnement triplé c'est-à-dire utiliser un frame buffers de taille trois. Le terme ‘’ *Dynamic’’* fait référence au fait que les canaux de lecture et d'écriture de la même unité VDMA utilisent un bus interne pour communiquer entre eux sur ce qu'il faut faire. La Lecture et l’écriture peuvent prendre l'un ou l'autre rôle (maître ou esclave), selon la préférence. Le maître sera le propriétaire de tous les frames buffers, sauf un, et l'esclave sera informé du frame buffer restant à utiliser.

En ce qui concerne le mode « *Dynamic-Master/Dynamic-Slave* pour le mode écriture/lecture respectivement », le maître peut être utilisé dans une situation où seul un canal de lecture ou d'écriture est utilisé, ou lorsque deux unités VDMA séparées sont utilisées chacune avec un canal. En mode esclave, le VDMA sera programmé avec une liste possible d'adresses de frame buffers et sélectionnera un buffer comme indexé par une interface matérielle.

L'unité VDMA doit également être connectée à la partie PS. Dans Vivado IDE, le PS est représenté comme un module IP «*Zynq7 Processing System*». Ce module doit être toujours utilisé dans l'environnement de développement Vivado, Il permet d’automatiser le câblage des bus et les IPs dans les conceptions complexes. En bref, cet IP est l'interface utilisée pour indiquer à la chaîne d'outils de mise en œuvre comment connecter les fonctions du système PS. Elle inclut aussi de nombreux paramètres pouvant affecter la PL et les ressources externes du PS (horloge, interruption…)

Les interfaces M-AXI4 ‘’*Master AXI4*’’ du VDMA doivent être connectées aux ports hauts performances (HP). En raison de la différence dans la version de protocole, l’IP AXI-InterConnect est nécessaire entre ces deux modules. Le bus AXI4-Lite doit être connecté depuis les unités VDMA aux ports AXI-GP ‘*’General Purpose’’* du module Zynq7. Encore une fois, en raison des différences de version de protocole, cela doit passer par une interconnexion de mémoire AXI.

#### Configuration logicielle des VDMAs

La plupart des paramètres du VDMA sont configurés au moment de l'exécution de l’application logicielle sur le processeur ARM. Le code source utilisé pour la configuration logicielle a été adapté d'un projet qui consistait à implémenter l’algorithme de sobel (24), et en raison des accords de licence et des permissions de publication du code, ce code va être inclut dans ce rapport. Les paramètres configurés sont importants et peuvent être exploités pour réaliser des opérations de transfert entre les VDMAs et la mémoire.

Les paramètres suivants sont configurés via le logiciel, dont certains peuvent remplacer les options définies dans l'outil de configuration de l’IP via Vivado IDE.

* Début/fin des VDMAs
* L'adresse de base dans la mémoire principale de chacun des buffers de trame (indépendante pour chaque canal)
* Taille/Résolution des frames d’E/S
* Nombre de frames buffers
* La taille horizontale et verticale du buffer
* Activer ou désactiver les interruptions

Toutes les fonctions utilisées pour ces configurations ont été inspirées des fonctions générées par l’outil de développement logiciel de l’environnement Xilinx via un package de support de carte ‘*’ BSP : Board Support Package’’.* Les BSPs vont générer une API ‘’Application programming Interface’’ de fonctions pour tous les modules matériel inclus dans un fichier de description matériel (HDF). Un HDF est automatiquement exporté de Vivado lors de la création d'un environnement SDK associé à un projet matériel. Ce fichier contient des informations sur les IPs de Xilinx utilisées dans un projet, comment elles sont connectées aux processeurs, quelle est l'adresse de base de leurs registres et quelles sont leurs paramètres personnalisés des IPs.

## Communication avec le matériel et Configuration

Le design global se compose de l’application logicielle dans la partie PS où s'exécute la partition de linux et tous les accélérateurs utilisés dans la conception du design dans la partie logique PL. Notre système est notamment basé sur l'OS OpenSus de linux comme expliqué dans la section ‎I.2. Une architecture de haut niveau de l’OS Linux est présentée sur la Figure ‎II.6. Afin d’avoir accès à la couche matérielle *‘’Hardware Devices’’* de cette architecture, les développeurs doivent intervenir pour écrire manuellement une application logicielle ‘’*USER application*’’ qui s'exécute dans le PS. Cette application sert essentiellement à configurer les composants principaux du design, ceci dit les VDMAs et l'accélérateur de traitement de vidéo généré par HLS.

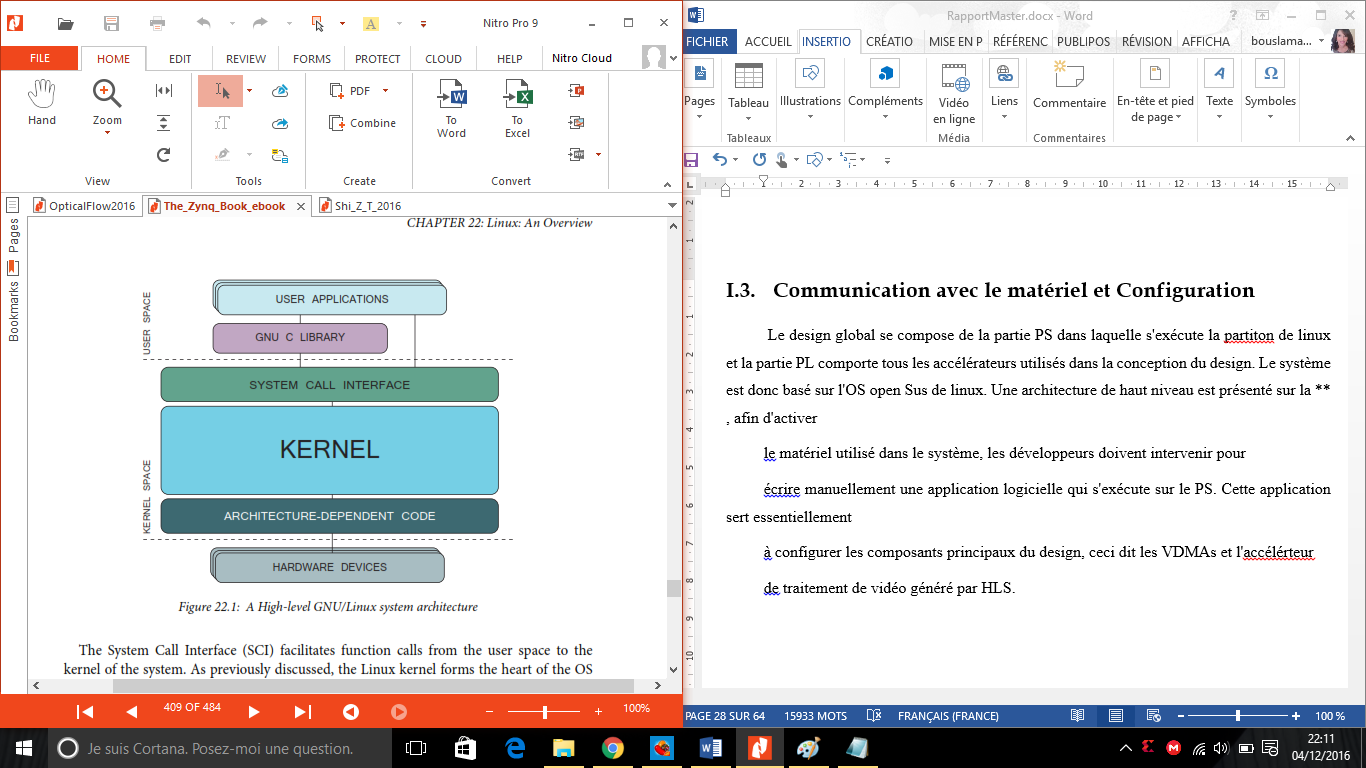


Figure ‑ : Architecture haut-niveau du système Linux (23)

Dans l’architecture logicielle de linux, le noyau linux est jugé le composant principal de l’OS. Il est le programme qui fait l’interfaçage avec le matériel. En effet, il est le cœur du système car il fournit aux applications logicielles une interface pour communiquer avec le matériel. Les applications et les programmes système, ainsi que la bibliothèque GNU C fonctionnent sur la partie top du noyau dans l'espace utilisateur. Les composants matériels physiques résident à l'extrémité opposée de l’architecture et, en tant que tel, ils ne seront accessibles par l’espace utilisateur qu’à travers le noyau. L’interface d’appels système (SCI) facilite les appels depuis l’espace utilisateur jusqu’au noyau. Comme nous l’avons déjà dit, le noyau Linux forme le cœur du système d'exploitation et fournit un ensemble d'outils avec lesquels l'espace utilisateur peut interagir avec le matériel.

L’appel système SC abrégé en ‘’*SysCall’’* est l’ensemble des services fournis par le noyau Linux et utilisés par les programmes s’exécutant dans l’espace utilisateur. En programmation C, l’appel système utilise souvent des fonctions définies dans ‘’libc’’ qui fournissent un wrapper pour de nombreux appels système. Chaque système possède un nombre de fonctions d’appel système définie dans la bibliothèque <syscall.h> ou <unistd.h>. La fonction d’appel système définie pour interagir avec les composants matériels est **mmap** ‘*’memory map’’*. Cette fonction d’appel système établit une projection en mémoire des composants matériels de notre design.

Les applications de l’espace utilisateur s’exécutent dans une mémoire, dans laquelle toutes les adresses manipulées sont virtuelles. Ces adresses sont mappées à des adresses physiques par l’unité de gestion de la mémoire du processeur MMU ‘*’Memory Management Unit’’*.

Pour avoir accès à n’importe quel périphérique, notamment le VDMA et l’IP de traitement de vidéo, nous utilisons une adresse virtuelle valide après l’avoir mappée à l’adresse physique pour pouvoir ainsi accéder à ce périphérique et le configurer. Les périphériques faisant partie du PL vont donc être mappés à une adresse virtuelle puisque leur configuration est établie via cette adresse virtuelle. Ceci est accompli via la fonction mmap. L’utilisation de cette fonction permet le mappage d’un espace mémoire traitant les adresses physiques à l’espace mémoire traitant les adresses virtuelles.

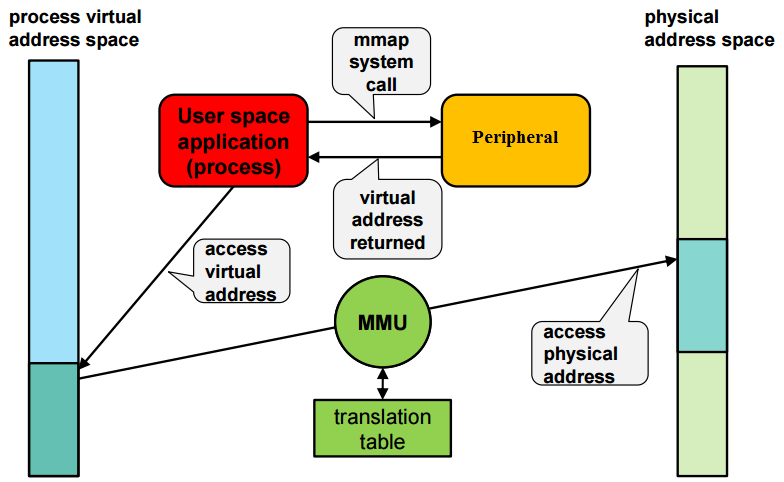


Figure ‑: Mappage entre l'adresse virtuelle et l'adresse physique (8)

Ensuite, nous pouvons contrôler ce composant matériel à partir de la lecture et l’écriture dans ses registres et ainsi pouvoir le configurer selon notre besoin. Chaque IP possède ses propres registres. Donc, la configuration diffère d’un IP à un autre. Donc les utilisateurs doivent également définir l'adresse de début de toutes les trames transmises ainsi que la dimension vidéo afin d'avoir une transmission de données efficace entre les données de flot vidéo et VDMA. Enfin, le filtre de traitement vidéo doit être configuré en fonction des informations de registre fournies dans le dossier du driver généré par Vivado HLS.

## Conclusion

Nous avons proposé au cours de ce chapitre un flot de conception rapide pour les applications de vision. Ce flot réduit la quantité d’effort et le temps nécessaire pour développer un système de traitement de vidéo sur les FPGAs. Nous avons ainsi découvert les différents composants introduits généralement dans une application de vision sur le circuit Zynq. Nous avons notamment pu voir les configurations matérielles et logicielles qui doivent être réalisé pour réaliser le prototypage rapide et efficace de notre application sur une plateforme embarquée et autonome. Ainsi, l’utilisation d’un système d’exploitation de type Linux associé à une distribution OpenSus sur cette carte nous a permis la gestion simplifié des interfaces de communication. Avec l’intégration de fonctionnalités générées à partir de Vivado HLS, le design proposé sera semblable à une caméra intelligente puisque notre système est composé d’une caméra et d’une carte de développement intégrant le circuit Zynq.

La fonctionnalité que nous allons générer depuis Vivado HLS, dans notre projet, est la détection de mouvement. Cette approche était, et l’est toujours, l’objet d’un gros nombre de recherche. Dans le chapitre suivant, nous allons découvrir une implémentation optimisée d’une méthode pour le calcul du flot optique sur la carte ZedBoard.

# Implémentation de la méthode du flot optique sur la ZedBoard

## Introduction

Le flot optique est un champ de déplacement visuel permettant de visionner les variations dans une image animée. Les algorithmes de calcul du flot optique hauts performance sont très intenses en termes de calcul. La vitesse de calcul des algorithmes étant faible cause beaucoup de difficulté dans des applications temps réel. A travers ce chapitre, nous proposons l’implémentation optimisée du modèle classique Horn & Shunk du calcul du flot optique sur le circuit FPGA-Zynq.



## Travaux antérieurs concernant le calcul du flot optique

Le flot optique est une mesure quantitative du mouvement visuel entre deux images consécutives. Ceci peut être obtenu en établissant la correspondance entre les pixels dans deux trames. Comme déjà-vu dans l’état de l’art, Horn and Shunk ‘’HS’’ (11) ont proposé deux hypothèses très utilisées pour calculer la correspondance entre les pixels.

**Hypothèse 1 (contrainte de luminosité constante) :** la luminosité d’un pixel ne change pas entre les images.

**Hypothèse 2 (contrainte de lissage globale) :** Le vecteur du flot optique varie légèrement entre les images.

En se basant sur ces deux hypothèses, les algorithmes du flot optique peuvent être formulés comme un problème de minimisation d'énergie, dans lequel la fonction d'énergie comprend un terme de données et un terme de lissage. Pour mieux décrire le lissage du flot optique, Lucas et Kanade ‘’LK’’ (13) ont proposé une autre hypothèse, qui est la suivante.

**Hypothèse 3 (Contrainte locale de flot optique):** les vecteurs du flot optique dans un petit voisinage sont plus ou moins constants.

Les hypothèses 1 et 3 sont utilisées par Lucas et Kanade pour établir la fonction de minimisation.

Bruhn et *al.* (25)ont combiné les trois hypothèses proposées par HS et LK et ont proposé le modèle combiné local-global ‘*’CLG :* *Combined-Local-Global*’’ pour une meilleure robustesse du calcul du flot optique dans des conditions bruitées. Brox et *al.* (26)introduisent une autre hypothèse comme complément de la contrainte de la luminosité constante.

**Hypothèse 4 (contrainte du gradient constant) :** le gradient de l’intensité de l’image reste constant sur les images malgré le déplacement des pixels.

En combinant les hypothèses 1,2 et 4, Brox et *al*. ont proposé le modèle ‘’*CBG : Combined-Brightness-Gradient*’’ (26). Étant donné que la contrainte de gradient constant est très sensible aux changements de luminosité qui apparaissent souvent dans les scènes naturelles, la méthode CBG convient très bien aux scènes naturelles où de légers changements de luminosité peuvent avoir lieu et en particulier pour les scénarios où la caméra n’est pas fixe.

Bien que de nombreuses techniques différentes ont été proposé dans le domaine du calcul du flot optique, d'un point de vue général, ces techniques peuvent être classées en trois modèles: le modèle HS, le modèle CLG et le modèle CBG.

Dans ce rapport, nous allons représenter le flot de travail général de la méthode la plus utilisé dans la littérature qui est la méthode HS et proposer comment implémenter et optimiser cette méthode sur un FPGA-SoC.

## Accélération Matérielle du calcul du flot optique

Accélérer le calcul du flot optique est essentiel pour les applications temps réel, que ce soit par l'optimisation logicielle ou par l'accélération matérielle. Il existe plusieurs approches basées sur l’accélération matérielle pour le calcul du flot optique, notamment les encodeurs MPEG. La norme MPEG utilise le déplacement des pixels entre l’image courante et une image de référence pour compresser l’information. Des processeurs spécialisés basés sur les ASICs ont aussi été développés et permettent de calculer le champ de mouvement des images en se basant sur la méthode corrélative utilisant la SAD ; ces processeurs sont appelés des MEPs ‘*’Motion Estimation Processor*’’. D’autres approches sont fondées sur l’implémentation sur les GPUs ou les FPGAs.

### Accélération matérielle avec les GPUs

Mizukami et *al.* (27) ont réalisé le modèle de HS sur une GeForce GTX 8800. L'utilisation de cette approche nécessite 0.443 secondes pour calculer le flot optique d'une image avec une résolution de 316 × 252. Gwosdek et *al*. (28) ont mis en œuvre un meilleur flot optique sur une GeForce GTX 285 qui coûte 0,98 seconde par image pour une image avec une résolution de 640 × 480. Sundaram et *al.* (29) ont réalisé une méthode de flot optique capable de traiter de grands déplacements sur une GeForce GTX 480 et le temps de traitement est de 1,84 secondes.

D'autres implémentations de la méthode du flot optique sur le GPU sont introduites dans Zimmer et al. (30) et Xu et al. (31). Le taux d'accélération de l’implémentation basée sur le GPU est important. Cependant, la consommation d'énergie est très élevée. La consommation élevée limite ses applications, en particulier dans les systèmes embarqués.

### Accélération matérielle avec les FPGAs

Nous avons déjà mentionné dans le premier chapitre que les FPGAs supportant différents types de parallélisme devraient recevoir plus d’attention, en particulier dans le cas des systèmes embarqués avec des contraintes liées à la consommation d’énergie. Arribas et al. (32) ont implémenté l’algorithme HS sur un Altera EP20K100 ; pour les images de résolution 50\*50, la vitesse est de 19 images par secondes (fps) pour 3 itérations. Martin et al. (33) ont mis en œuvre le même algorithme sur un Altera APEX20K ; la vitesse est de 60 fps pour des images de résolution 256\*256 mais le nombre d’itération n’a pas été mentionné. Gultekin et al. (34) ont accéléré l’algorithme HS sur un Altera Cyclone et ont atteint 257 fps pour des images de résolution 256\*256. Kunz et al. (35) ont proposé une architecture pour diviser le bloc d’itération en quatre phases pipelinées afin de réduire les calculs nécessaire pour les futures itérations. Sur une Altera Stratix, cette implémentation est capable d’obtenir jusqu’à 30 fps pour les images de résolution 640\*512. Komorkiewicz et al. (36) ont présenté une implémentation matérielle efficace de la méthode de HS. Dans leur implémentation, toutes les itérations sont extraites et pipelinées dans le FPGA afin d'accélérer le calcul. Ils ont pu traiter 60 fps d'une image de 1920\*1080 sur le circuit Xilinx Virtex 7. L'inconvénient est qu'il nécessite des ressources FPGA considérables et limite le nombre d'itérations.

Les avantages des FPGAs sont la flexibilité de l’architecture, la vitesse du traitement et la consommation basse d’énergie. Néanmoins, la mise en œuvre des algorithmes sur les FPGAs purs prend du temps vu les difficultés de débogage. Par conséquent, à notre connaissance, seul l’algorithme basique du calcul du flot optique (modèle HS) avec une faible précision ont été mis en œuvre sur la plateforme ZedBoard.

## Le flot de travail général de la méthode HS

Dans cette section, nous allons discuter le flot de travail général de la méthode HS. En se basant sur l’étude théorique faite la section ‎III.2.2. a.1 et Comme le montre la Figure ‎IV.1, l’algorithme du calcul du flot optique consiste généralement à procéder aux différentes étapes suivantes :

* Le lissage
* Le calcul des dérivées
* La création des tenseurs
* Le calcul des vitesses

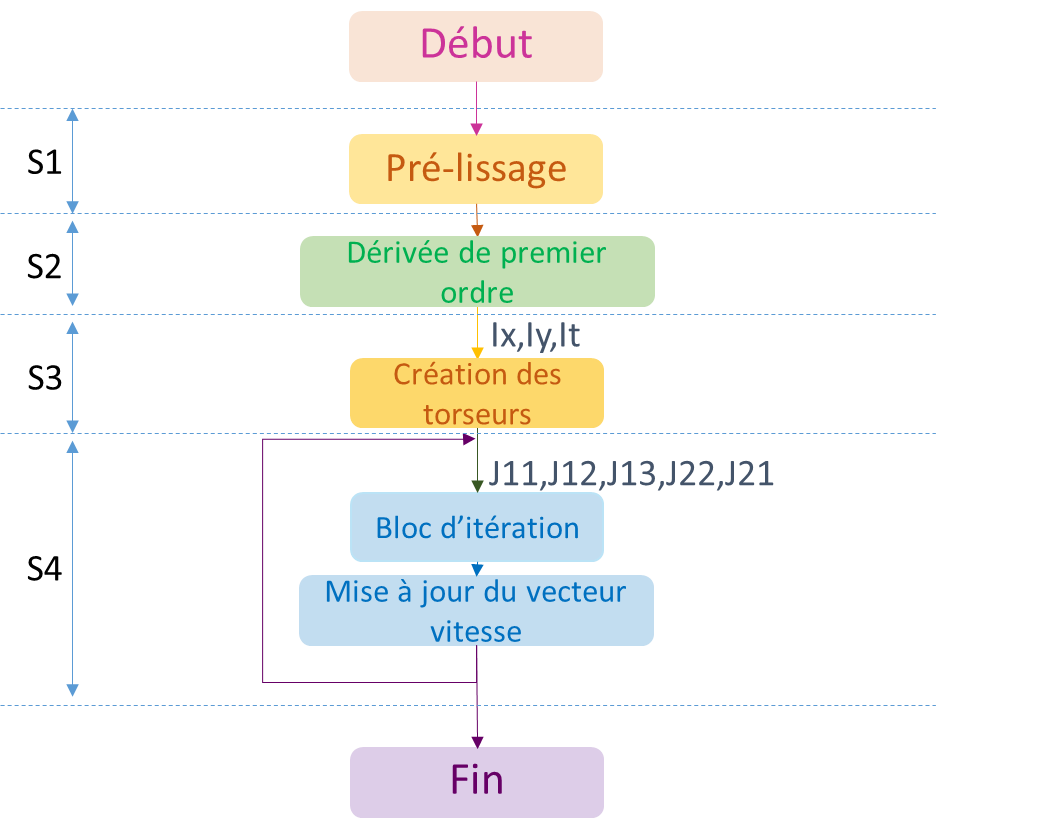


Figure ‑: le flot de travail général de la méthode HS

* **Le lissage (S1) :** Cette étape réduit l'influence du bruit et des valeurs aberrantes en lissant les séquences d'image d'entrée, généralement par convolution d'images avec des filtres de lissage.
* **Le calcul des dérivées (S2) :** Cette étape calcule les dérivées spatio-temporelles. Le calcul des dérivés du premier ordre, nous obtenons les dérivées spatiales (Ex, Ey) et la dérivé temporelle Et des deux images lissées précédemment.
* **La création des tenseurs (S3) :** Cette étape génère 5 tenseurs de mouvement (J11, J12, J13, J21, J22). la méthode de construction des tenseurs de mouvement pour la méthode de HS est la suivante :

**J11= Ix\*Ix**

**J12=Ix\*Iy**

**J13=Ix\*It**

**J22=Iy\*Iy**

**J23=Iy\*It**

* **Le Bloc d’itération (S4):** Cette étape calcule les résultats du flot à partir des itérations. Pour la méthode de HS, les flot u et v doivent être mis à jour. Le bloc d’itération est donné par le système d’équation découlé de l'équation (III. 11)

*Où : i représente le nombre du pixel de la i éme ligne*

*j représente le nombre du pixel de la j éme colonne*

*J représente le tenseur de mouvement*

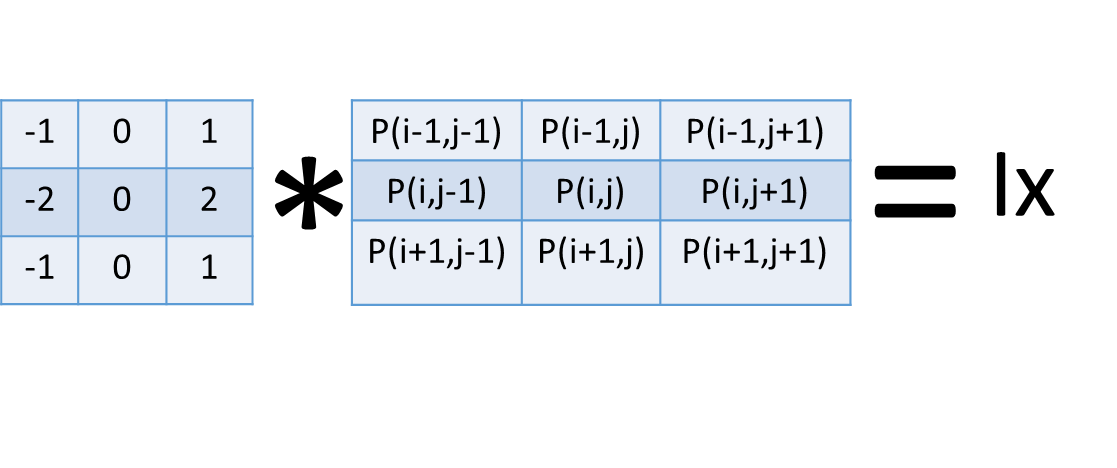
*u et v représentent le flot horizontale et verticale du mouvement*

## Calcul des dérivés

La méthode de HS comme celle de LK utilise, en quelque sorte, dans l’équation du flot optique ci-dessus les dérivées temporelle et spatiale de l’image, qu’il faut estimer numériquement. De nombreux masques de convolutions sont à possible pour le calcul de ces dérivées.

### Principe du calcul des dérivées

Dans cette section, un algorithme de calcul des dérivées utilisant un masque 3x3 est présenté. Les dérivées verticales et horizontales sont calculées à partir des masques de sobel. Le filtre Sobel a été largement utilisé dans nombreuse applications de traitement d'images, et il constitue la base pour des algorithmes plus avancés dans le domaine de la vision par ordinateur. Les filtres Sobel sont principalement utilisés pour les détections de contours. Les filtres Sobel utilisent la convolution pour approximer les dérivées d'un pixel de l’image. L'idée principale d'utiliser un filtre Sobel est d'utiliser les deux maques de Sobel 3 x 3 dans la direction horizontale (Figure ‎IV.2) et dans la direction verticale (Figure ‎IV.3) pour le calcul approximatif des gradients.



=

Ix

Figure ‑: calcul du gradient X de sobel

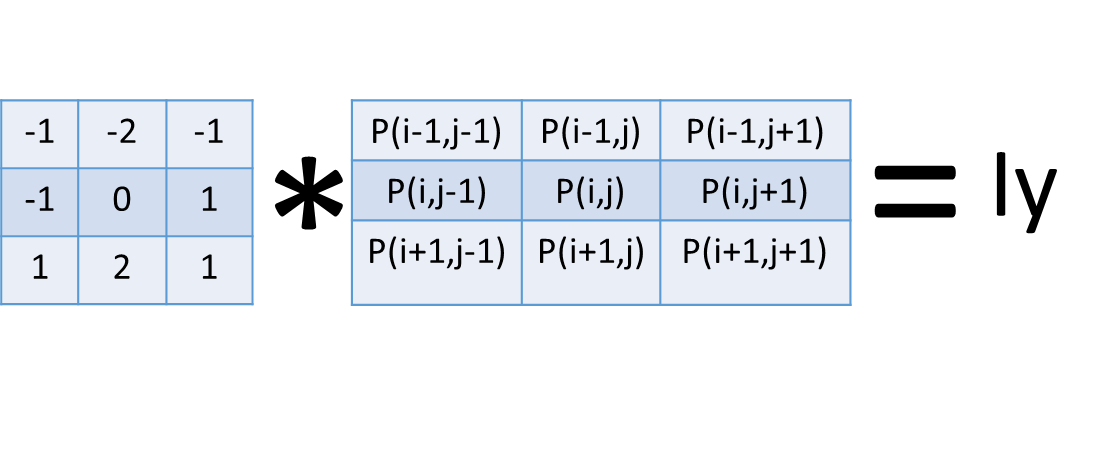


Figure ‑: calcul du gradient Y de sobel

Pour le calcul du flot optique, nous avons besoin de deux images successives depuis la vidéo. Les dérivés spatiaux sont toujours appliqués sur la première image de la séquence, l’image courante. Tan disque la dérivée spatiale est calculée en exploitant les deux images considérées. En effet, nous avons appliqué le produit de convolution sur l’image courante et l’image précédente en utilisant respectivement les masques de taille 2x2: M1= et M2=.

Ces masques sont ceux utilisés pour calculer le flot optique  pendant ce projet; mais il existe plusieurs autres alternatives de calcul des dérivées spatio-temporelles, notamment la méthode originale de Horn & Shunk. Pour le calcul des dérivées spatio-temporelles, ils utilisent les équations suivantes :

Ix=\*im1+\*im2

Iy=\*im1+\*im2

It=\*im1 +\*im2

### Architecture de la fenêtre glissante pour le calcul des dérivée

Le but d’une architecture mémoire de fenêtre glissante est de pouvoir l’utiliser pour une variété d’algorithmes de traitement de vidéo tels que le lissage, la détection de contour … Cette architecture contribue également à l’accès rapide à la mémoire et à l’utilisation de moins de ressources. Elle implique l’utilisation d’un buffer de ligne ‘’ *line buffer* ‘’et un buffer fenêtre ‘*’window buffer* ‘’. L’un des avantages majeures de l’utilisation du *line buffer* et *window buffer* est le fait qu’ils n’occupent qu’une petite quantité de mémoire et qu’ils peuvent être synthétisés seulement en BRAM et FF ‘’*Flips Flops’’* sur FPGA comme montre le Tableau ‎IV‑1. Ces structures sont notoirement connues comme les portions de mémoire dont leur accès est le plus rapide sur les FPGAs. L'accès permanent aux BRAM et aux FF au lieu des DRAM peut entraîner un accès de plus faible latence à la mémoire, ce qui contribuera positivement aux performances de traitement vidéo en temps réel.

|  |  |
| --- | --- |
| Interface Mémoire | Ressource |
| Line Buffer | BRAM |
| Window Buffer | FF |

Tableau ‎IV‑1:Relation entre l'interface mémoire et l'utilisation des ressources

Lors de l'utilisation d'un *line buffer* dans la conception de l'algorithme, un nombre fixe de lignes d'une image est copié dans un ce *line buffer*. Ce dernier se met à jour lorsque les données de flot entrent, tandis que le *window buffer* est mis à jour en fonction de l'état du *line buffer*. Plus précisément, le *line buffer* lit dans les données de flot pixel par pixel comme le montre la Figure ‎IV.4.

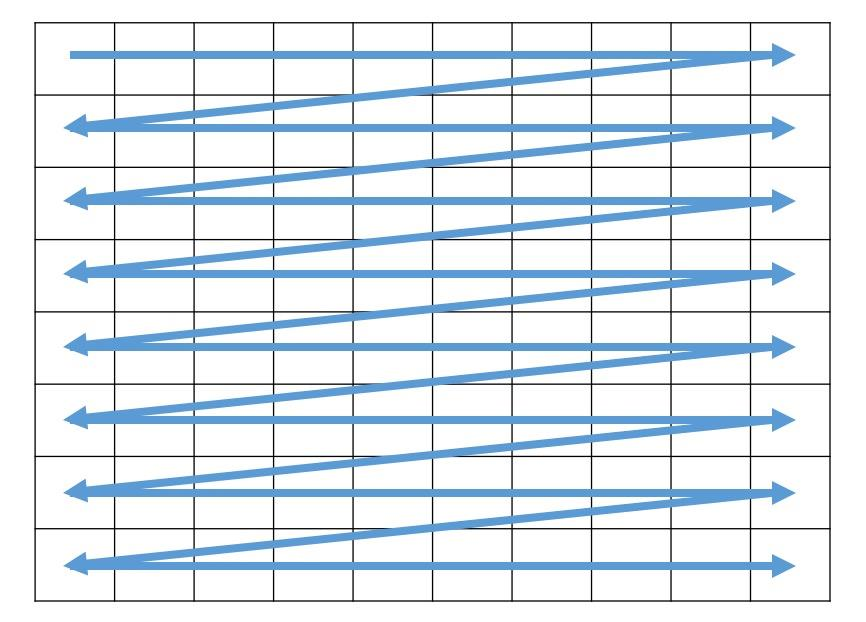


Figure ‑: Ordre de lecture d'un stream source

Le *window buffer* joue le rôle d’une fenêtre glissante qui mémorise les données requises pour le calcul courant. Avant de créer des algorithmes de traitement vidéo, il est important de comprendre les interactions entre le *line buffer* et le *window buffer*. Avec des conceptions appropriées pour l'interaction entre ces deux interfaces, les masques d'utilisateur peuvent accéder aux données stockées dans le *window buffer* à chaque cycle d'horloge pour des calculs spécifiques.

La section suivante présente une approche pour traiter le line buffer et les opérations du *window buffer* pour les modèles d'algorithme de traitement vidéo de type de stream de sorte qu'il peut apporter une gestion de mémoire efficace et un accès de plus faible latence à la mémoire. Les données stockées dans le *window buffer* sont normalement utilisées comme données pour le calcul. Par exemple, dans le cas des filtres Sobel, les données contenues dans le *window buffer* sont utilisées pour calculer le produit de convolution en se basant sur les coefficients des masques de sobel. La Figure ‎IV.5 montre un exemple d'interaction entre le *line buffer* et le *window buffer*.

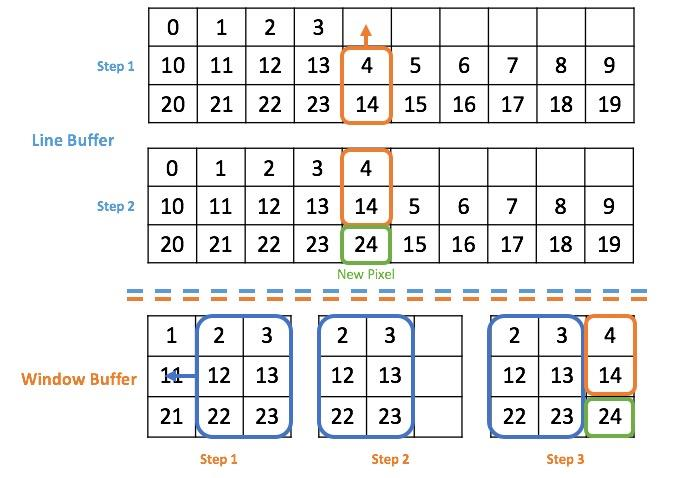


Figure ‑:Exemple d’Interaction entre les deux interfaces mémoire (20)

Conformément à l'exemple illustré à la Figure ‎IV.5, les procédures de mise à jour du *line buffer* et le *window buffer* sont indiquées dans le Tableau ‎IV‑2 et le Tableau ‎IV‑3 . La procédure de mise à jour du *line buffer* est indiquée sur le Tableau ‎IV‑2.

|  |  |
| --- | --- |
| L1 | Déclarer un *line buffer* de taille MxN, où M est la taille du masque de convolution et N le nombre de colonne de l’image. |
| L2 | Déplacer la colonne i vers le haut. |
| L3 | Insérer la nouvelle valeur de pixel au bas de la colonne i. |
| L4 | Entrer l'opération du *window buffer* W1. |
| L5 | Commencer les itérations jusqu’à la colonne suivante i + 1, répéter à partir de l'étape L1. |

Tableau ‎IV‑2: Procédure de mise à jour du line buffer

La procédure de mise à jour du *window buffer* est indiquée sur le Tableau ‎IV‑3.

|  |  |
| --- | --- |
| W1 | Déclarer un *window buffer* de taille MxN, où M est la taille du masque de convolution |
| W2 | Déplacez la fenêtre glissante vers la gauche pour faire en sorte que nous avons une colonne vide sur la droite. |
| W3 | Récupérez les M-1 pixels du haut de la colonne i dans le line buffer et insérez-les sur le haut de la colonne la plus à droite dans le window buffer. |
| W4 | Récupérez le nouveau pixel du bas de la colonne i dans le line buffer et insérez-le au bas de la colonne la plus à droite dans le window buffer. |
| W5 | Effectuer des calculs de filtrage. |
| W6 | Finir le procédé de calcul et commencer l’opération du line buffer à L5. |

Tableau ‎IV‑3: Procédure de mise à jour du window buffer

Ce type d’architecture conçu par Zhun (20)est devenu un composant essentiel dans les applications de vision temps réel. Avec une utilisation appropriée des BRAM et des FF dans sa conception plutôt que les DRAM, il est très probable que l'utilisation du line buffer et du window buffer peut aider à améliorer le débit et à réduire la latence du calcul de l'algorithme.

En outre, les pixels sur la bordure de la trame d'entrée doivent être correctement traités afin d'empêcher un accès violent mémoire. Étant donné que la convolution dépend des huit pixels voisins, aucun résultat ne peut être calculé avec un pixel de bordure manquant. Par conséquent, cette conception ignore la convolution des pixels de bordure et affecte automatiquement une valeur nulle à ces pixels.

## Optimisations du calcul du flot optique

Bien que les outils de synthèse haut niveau tels que Vivado HLS soient en mesure d'optimiser le code C++ pour les plateformes FPGA dans une certaine mesure, nous devons encore comprendre l'algorithme et l'architecture de la plateforme pour une optimisation supplémentaire. Cette section prend HS comme un exemple pour analyser les techniques d'optimisation. Tout d'abord, nous analysons les parallélismes qui existent dans le calcul du flot optique. Ensuite, nous proposons différentes approches étape par étape pour restructurer et optimiser HS. Ces approches sont basées sur les parallélismes intrinsèques de l'algorithme et l'architecture matérielle.

### Parallélisme dans la méthode de calcul du flot optique

Le parallélisme dans la méthode de calcul du flot optique inclut le parallélisme des tâches, le parallélisme des données et le parallélisme du pipeline qui représente la base de la restructuration d’un code C/C++ sur les FPGAs.

#### Le parallélisme des tâches

La Figure ‎IV.6 montre le flot de travail du modèle de HS. Visuellement, nous pouvons comprendre que la dépendance des données existe dans les opérations entre les étapes adjacentes. Le résultat d’une étape est utilisé comme entré pour l’étape suivante. Ainsi, les opérations lors des différentes étapes doivent être calculées l’une après l’autre.

Nous pouvons aussi remarquer que plusieurs opérations appartenant à la même étape peuvent être calculées indépendamment. Etant-donné que nous avons besoin de deux trames d’images pour calculer le flot optique, comme indiqué sur la Figure ‎IV.6, le lissage des deux images (img1 et img2) peut être traité simultanément (sig1 et sig2 respectivement). De plus, le calcul des premières dérivées Ix et Iy est réalisé à partir de l’image lissées sig1. It va être calculé à partir des deux images lissées sig1 et sig2. Toutes les dérivées peuvent être calculées indépendamment et simultanément. Finalement, les tenseurs (J11, J12, J13, J21, J22) Peuvent également être calculés simultanément selon la figure. Ainsi, il est facile d'accélérer les opérations à chaque étape en les considérants comme des tâches parallèles. Le parallélisme du bloc d'itération sera discuté dans la section ‎I.5.2.

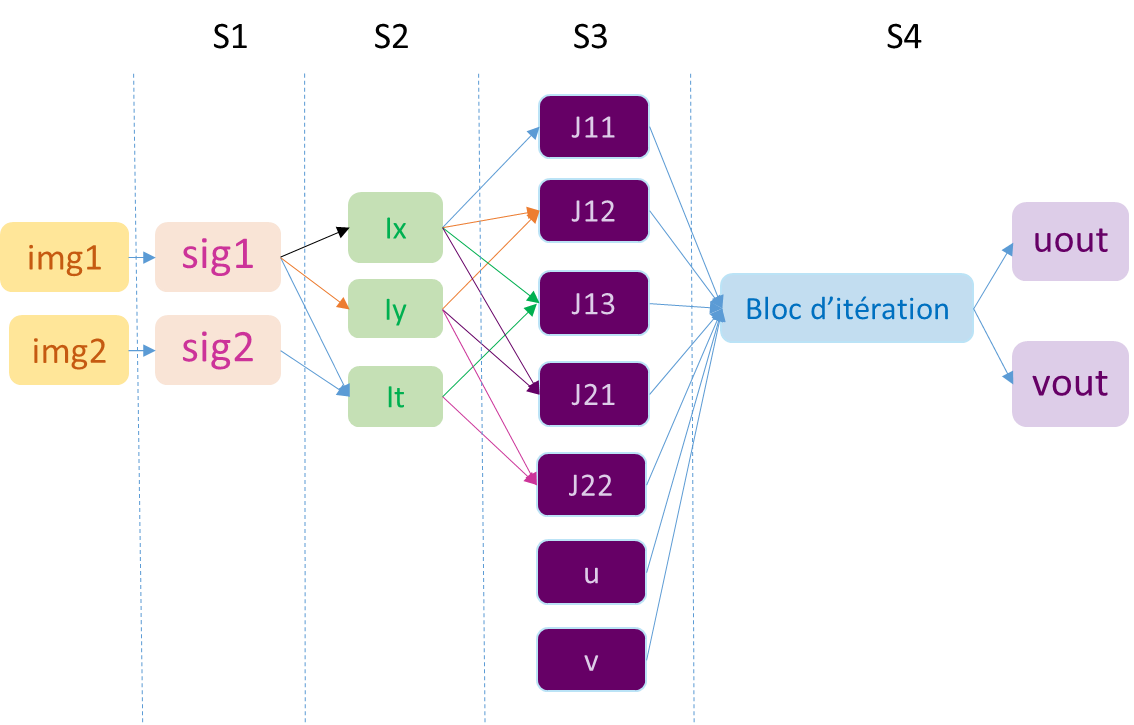


Figure ‑: Parallélisme des tâches dans HS

#### Le parallélisme des données (22)

En général, les opérations de calcul du pré-lissage et du calcul des dérivés sont calculées par l’application d’un produit de convolution qui contient un type typique de parallélisme des données. En outre, pour chaque construction de tenseur de mouvement, de multiples paires de données peuvent être calculées par la même instruction. C'est aussi un type de parallélisme des données qui peut être efficacement mis en œuvre sur un matériel massivement parallèle.

#### Le parallélisme du pipeline

Bien que la dépendance des données existe à différentes étapes du flot, comme mentionné précédemment, et les opérations à différentes étapes doivent être calculées les unes après les autres, il n'est pas nécessaire d'attendre que tous les résultats des étapes précédentes soient disponibles pour démarrer les étapes suivantes. Pour chaque étape, seuls les pixels des lignes et des colonnes de début doivent être disponibles pour démarrer l'opération. Ainsi, lorsqu’un matériel approprié comme le circuit Zynq est disponible, toutes les étapes de HS peuvent fonctionner simultanément en pipeline.

### Techniques d’optimisations

Dans cette section, des techniques d'optimisation du calcul du flot optique basées sur le circuit ZYNQ sont introduites. Tout d'abord, les modules C possédant des calculs intenses sont migrés vers FPGA pour l'accélération. Ensuite, Vivado HLS offre la commodité d'optimiser la conception matérielle avec l'insertion de directives dans les implémentations de haut niveau.

##### Optimisation des boucles

Cette section se focalise sur les optimisations des boucles afin d’obtenir des calculs parallèles. Comme technique d’optimisation, les boucles internes ‘’*Inner-Loop*’’peuvent être complètement déroulées s’il n'y a pas une dépendance de données entre chaque calcul. En utilisant cette technique l'outil HLS allouera plus de ressources pour les opérations afin d’atteindre le parallélisme. En revanche, les boucles de niveau supérieur ‘’*Outer-Loop*’’ peuvent être optimisées en utilisant la directive de pipeline. Contrairement au déroulage de boucle, le pipeline induira le partage de ressources des opérations à travers chaque itération de la boucle de niveau supérieur. Ceci induira à la consommation de plus de ressources mais améliore le temps de latence

##### Optimisation des fonctions OpenCV-HLS

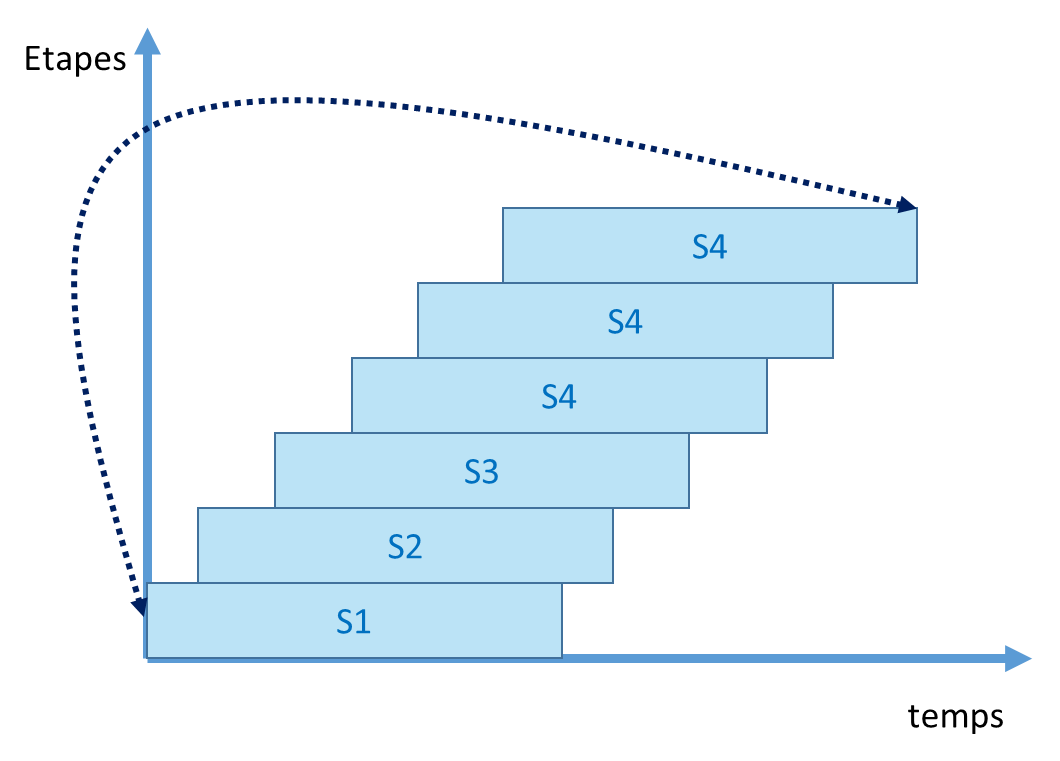
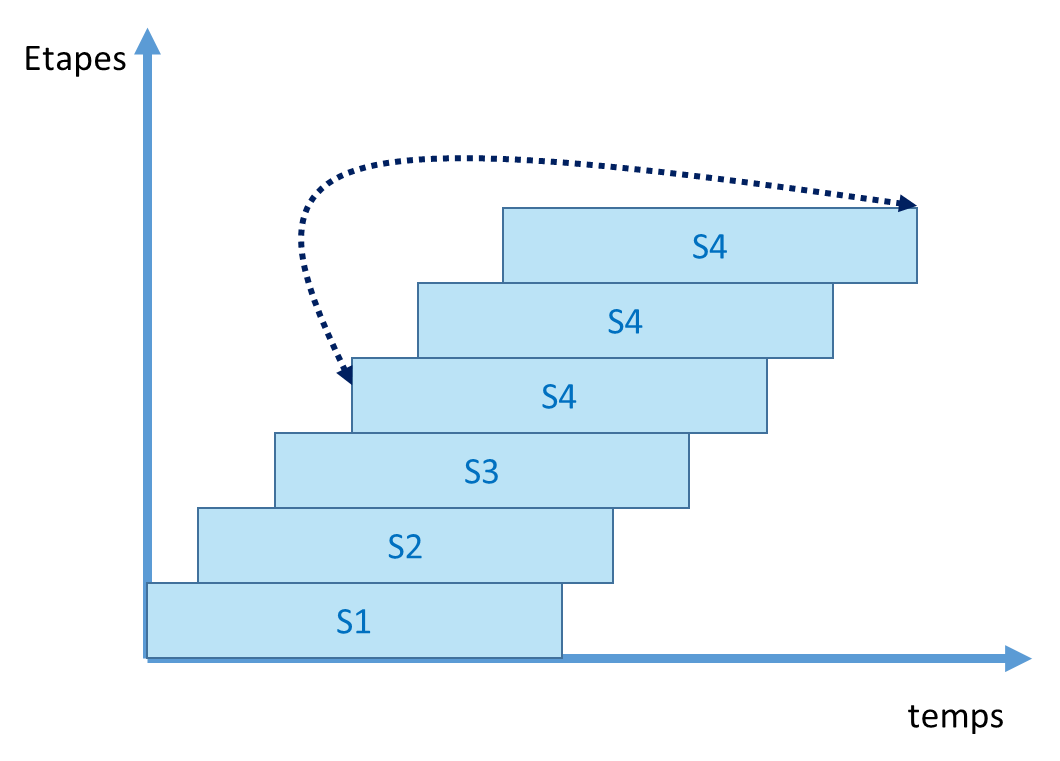
Pour accélérer le traitement, les fonctions OpenCV de HLS ne doivent en aucun cas être traitées séquentiellement. Les appels de fonction séquentiels dans une fonction de niveau supérieur peuvent être pipelinés automatiquement en utilisant la directive **#HLS pragma dataflow**. Elle peut potentiellement réduire la latence entre l'exécution des fonctions et améliorer le débit total. Par défaut, chaque fonction sera représentée sous forme d’un bloc distinct des autres fonctions au niveau du RTL et l’ensemble des fonctions seront disposées d’une manière hiérarchique. Le pragma *‘’* dataflow*‘’* prends les fonctions séquentielles et en crée une architecture pour un traitement en parallèle.

##### Extension d’itération

Comme le montre la Figure ‎IV.7, cinq tenseurs de mouvement sont génères suite à l’étape S3. Si seulement l’étape S4 est prise en compte comme le corps d’itération exécutée sur le PL alors ces tenseurs doivent être sauvegardés à l’avance dans la mémoire DDR. Pour chaque itération, cinq torseurs de mouvement et deux torseurs de flot, par exemple 640\*480 pixels chacun, doivent être extraits de la mémoire, ce qui prend une grande partie du temps d'exécution total. Pour réduire l'exigence de bande passante de la mémoire, les étapes S1, S2 et S3 sont également prises en compte dans le bloc d'itération, comme représenté sur la Figure ‎IV.7 .b . Dans ce cas, pour chaque itération, les images initiales des flot, l’image courante et précédente et les tenseurs du flot u et v doivent être extraites de la mémoire.

Figure ‑: Optimisation pipelinée

Loop



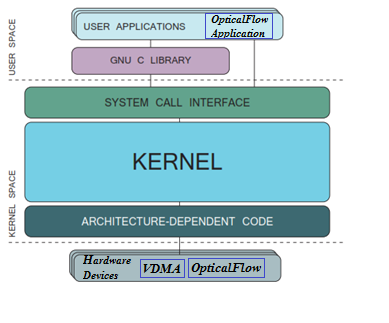
A : optimization par déroulage

B : Optimisation par extension



## Développement matériel et logiciel de l’application

L’application du flot optique va être implémentée sur la plateforme ZedBoard, une plateforme d’évaluation basée sur le circuit Zynq, qui est présentée dans la section ‎I.4. Le système est basé sur une partition de linux. Une architecture logicielle de haut niveau, concernant notre projet, est présentée sur la Figure ‎IV.8. En général, le développement du système comprend la conception logicielle et la conception matérielle. Une explication détaillée sur le développement du design d’un système de traitement de vidéo est donnée dans la section ‎II.4. L’application logicielle s’exécute dans la partie PS du Zynq et le matériel, y inclut l’accélérateur développé pour le calcul du flot optique, est principalement implémenté dans la partie PL à l’exception de certaines configurations définies sur le PS.



2

1

Figure ‑: Architecture du système Linux

Le développement logiciel consiste à créer l’application dans l’espace utilisateur (1). La conception matérielle est basée sur les accélérateurs dans la partie FPGA (2). Comme mentionnée dans le chapitre II, à part l’IP du flot optique qui représente l’IP de base de notre application de traitement de vidéo, l’IP VDMA est aussi utilisé pour gérer les données des différentes trames.

La Figure ‎IV.9 présente la conception détaillée de la partie PL. Dans notre système, les données des trames transmises entre les différents IPs utilisent les interfaces AXI4-Stream. Ces derniers sont représentés sur la Figure ‎IV.10 avec la couleur Rose. Les connexions de control utilisent l’interface AXI4-Lite et sont présentés avec la couleur verte et les connections avec la couleur oranger représentent les interfaces de données de type **AXI4** qui permettent la connexion à la mémoire DDR3 à travers les ports HP. En effet, ce sont les ports hauts performance les mieux adaptés aux applications de traitement de vidéo car ils assurent un accès rapide à la mémoire DDR. Les quatre ports HP qui existent incluent des buffers FIFO pour gérer le transfert des images à lire ou à écrire dans la mémoire. Ils fournissent un transfert de donnés à une grande vitesse avec une faible latence entre le PL et la mémoire du Zynq.

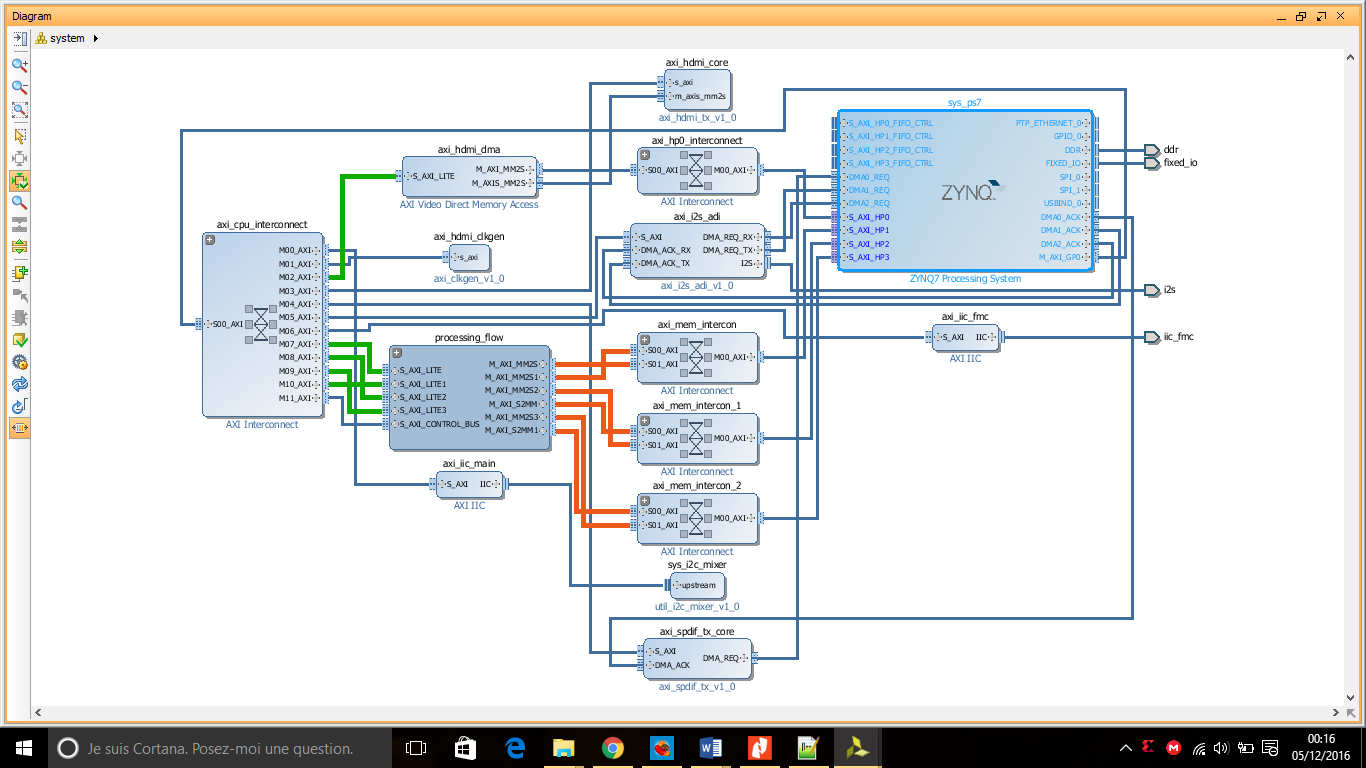


Figure ‑: Design Complet du système

Comme exemple d’implémentation, la Figure ‎IV.10 illustre la conception matérielle du bloc ‘’processing Flow’’. Notre accélérateur est doté de quatre entrées et deux sorties. Les quatre entrées *(Input\_current, Input\_previous, Input\_Velx, Input\_Vely)* comme le montre le flot générale de l’algorithme de HS présenté sur la Figure ‎IV.1 sont les deux images sur lesquels l’algorithme est appliqué ainsi que les deux torseurs d’initialisation du flot. Les sorties du bloc *(Output\_Velx, Output\_Vely)* représentent les deux torseurs du flot mis à jour.

L’accélérateur du calcul du flot optique est, bien-évidemment, interfacé avec quatre blocs VDMAs. Parmi eux, deux sont configurés en mode lecture et deux en mode lecture / écriture. Les entrés (*Input\_current, Input\_previous)* sont connectées avec deux VDMAs configurés en mode lecture puisque dans ce cas nous avons besoin de récupérer les deux trames correspondante depuis la mémoire DDR pour les traiter. Par contre, les deux entrées (*Input\_Velx, Input\_Vely)* sont connectées avec deux VDMAs configurés en mode lecture / écriture puisqu’il faut non seulement lire les torseurs du flot d’entrés mais aussi écrire les flot mis à jour dans la mémoire et ainsi pouvoir les afficher.

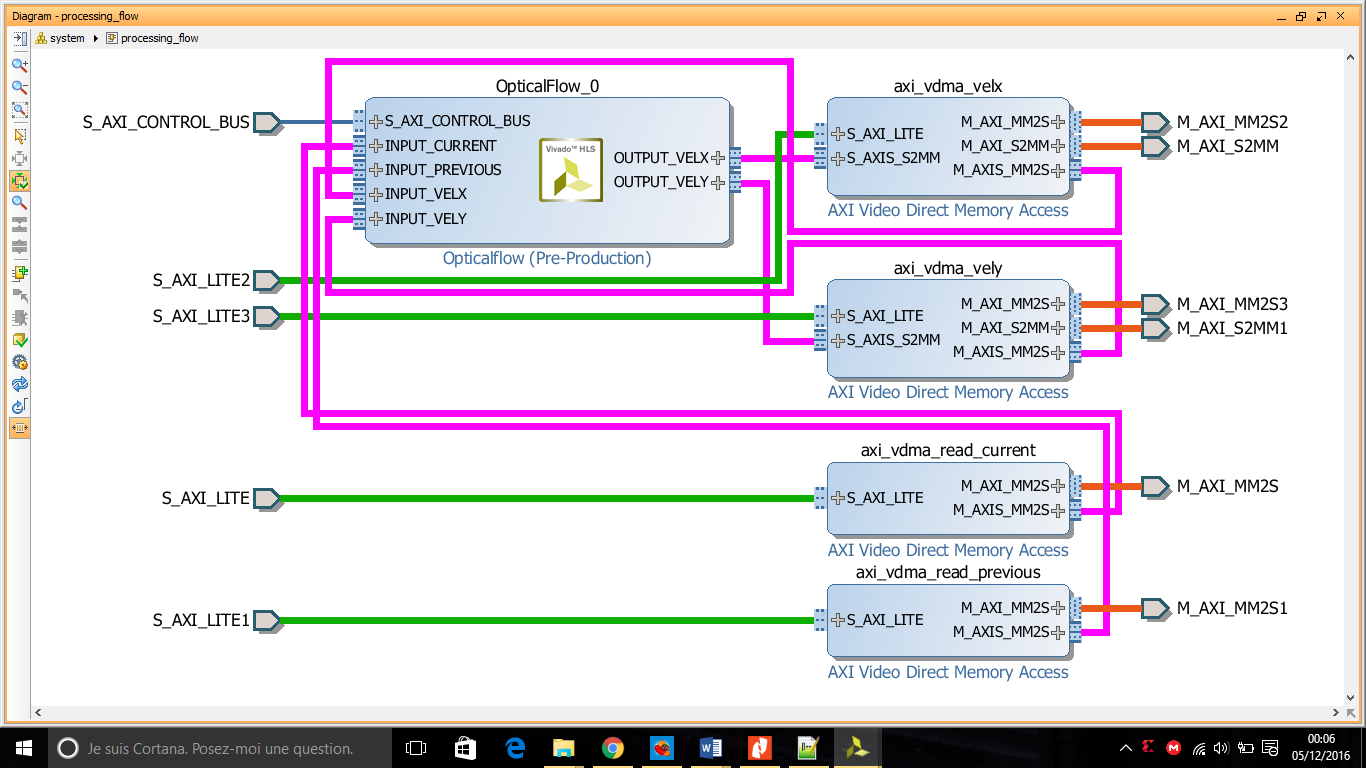


Figure ‑: Bloc Processing Flow

## Expérimentation et évaluation

Dans cette section, nous allons évaluer les optimisations et l’effet de l’accélération et analyser les effets des différents facteurs. En premier lieu, Nous commençons par établir l’environnement d’expérimentation. Ensuite, nous implémentons le modèle HS du flot optique par logiciel pure sur un CPU général afin d’obtenir les données basique précis et le temps d’exécution. En second lieu, nous implémentons la méthode avec les différentes techniques d’optimisation discutée dans la section ‎I.5 et analyser l’effet de l’accélération. En troisième lieu, nous analysons les effets de différents paramètres tels que la précision, le nombre d’itération, le temps d’exécution et la taille des images avec le modèle HS. En outre, nous discutons les ressources matérielles utilisées et la consommation d’énergie.

### L’environnement d’expérimentation

L’environnement d’implémentation est basé sur les images sources, les paramètres de l’algorithme, la représentation du flot.

#### Les images sources

En général, dans un système de traitement de vidéo, les images sources ne sont que les images provenant de la caméra et transférés vers la part logique (PL) de notre plateforme pour qu’ils soient traitées. Cependant, pour comparer et évaluer équitablement notre algorithme, une séquence d'images normalisées est utilisée comme référence. Dans ce projet, nous sélectionnons les images de test de la base de données Middlebury (37). Les images sont stockées sous forme de fichiers sur une carte SD. Ils ont été lus par la partie PS et transférés à la partie PL. Dans notre expérimentation et comme le montre la Figure ‎IV‑11, les images 1 et 2 extraites depuis la séquence d’images Yosemite du cette base de donnée avec une résolution de 312\*252 pixels sont utilisées.

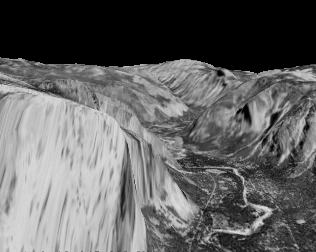


Image2

Image1

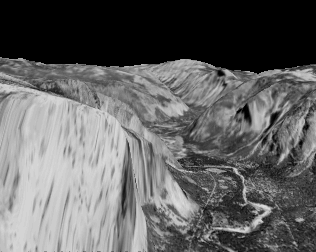


Figure ‑: Séquence d'image Yosemite

Depuis la figure, nous pouvons constater que les scènes sont assez complexes. Cette séquence est la plus utilisée car elle a l’avantage d’être complexe suite à ses mouvements composés et ne sont pas de simples rotations ou translations.

#### Paramètres d’implémentation

L'expérience évalue également l’algorithme HS. Les configurations de ses paramètres pour l'estimation du flot optique sont représentées dans le Tableau ‎IV‑4 . Sauf indication contraire, tous les résultats d’évaluations de notre algorithme sont en rapport avec les paramètres suivants.

|  |  |
| --- | --- |
| Paramètres | Valeur |
| Nombre de trames  Taille du filtre pour le lissage  Taille du masque pour les dérivées  Poids de lissage α  Nombre d’itération | 2  3  3  1.5  100 |

Tableau ‎IV‑4: Paramètres de configuration de HS

#### Représentation du flot et références

La représentation visuelle du flot optique peut se limiter sur les images en niveau de gris ; u et v. Mais la solution la plus intéressante pour la représentation du flot est l’utilisation de la carte de couleurs, qui permet à partir du flot horizontale u et verticale v, d’estimer la direction du flot.

La Figure ‎IV‑12 montre la direction du flot selon la carte de couleur. Le tracé du champ de vitesse est tiré de la littérature, il permet à partir de la couleur comprendre la direction du flot.

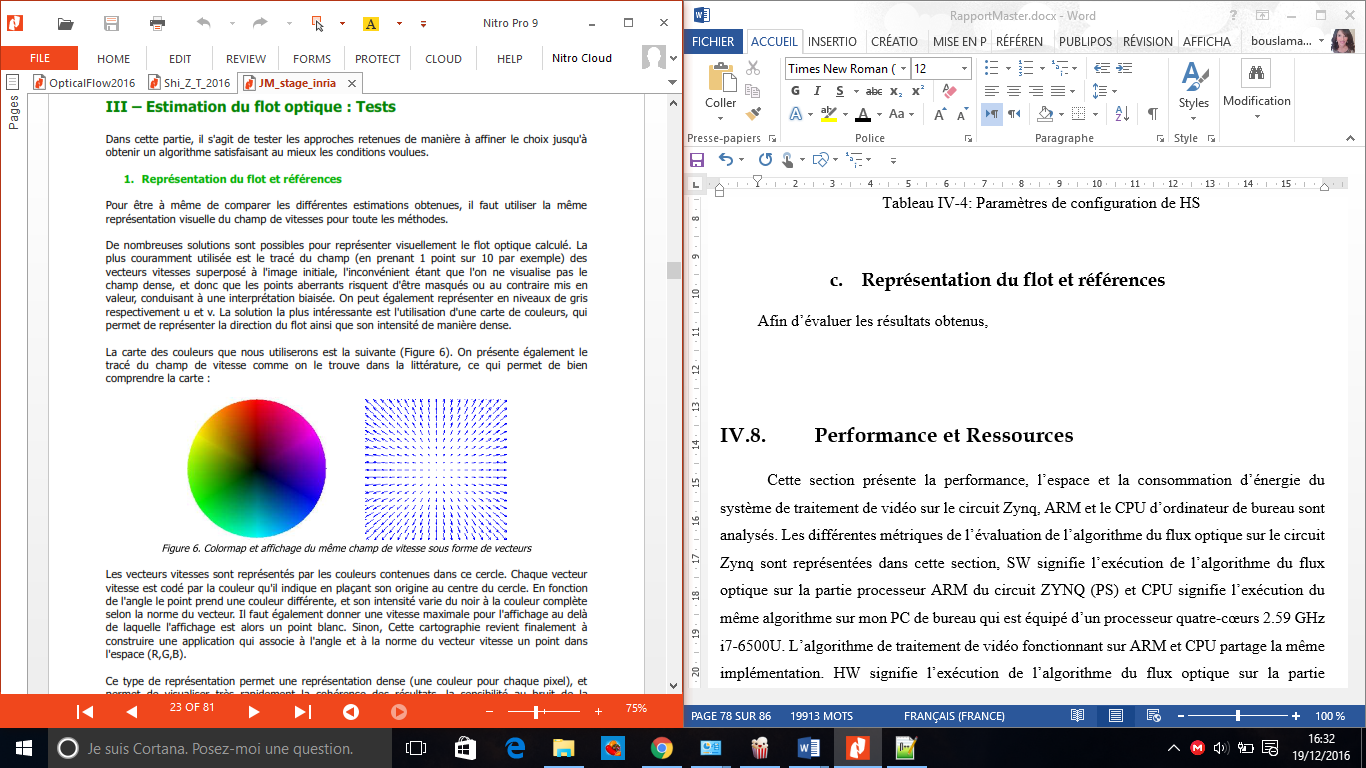


Figure ‑: La direction du flot selon la carte de couleur

Le flot final est représenté par les couleurs que contient le cercle. A partir de l’origine du centre du cercle, la couleur est déterminée par le codage de chaque vecteur vitesse. En fait, la couleur est définie en fonction de l’angle et suivant la norme du vecteur, l’intensité de la couleur varie. Une valeur de vitesse maximale doit être fixée au-delà de laquelle la couleur blanche est affichée. Finalement, cette cartographie nous permet à partir d’un angle et de la norme du vecteur vitesse de construire le flot finale en tant qu’une image de type RGB. Le flot finale est alors un flot dense (chaque pixel possède une couleur) et suite à ce flot, nous pouvons évaluer la précision des résultats.

La séquence des images Yosemite sur laquelle nous avons effectué les tests dispose bien évidemment d’une image référence du flot. Cette image de référence représente le flot réel extrait de la base de données Middlebury et il est représenté sur la Figure ‎IV‑13.

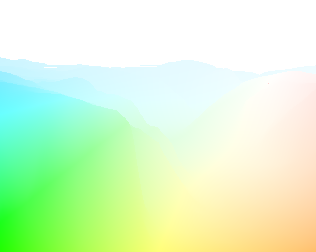


Figure ‑: Flot réel de la séquence Yosemite

### Evaluations

Cette section évalue la performance (temps d’exécution et précision), l’espace et la consommation d’énergie du système de traitement de vidéo sur le circuit Zynq, le processeur ARM et la CPU d’un ordinateur de bureau. Les différentes métriques de l’évaluation de l’algorithme du flot optique sur le circuit Zynq sont représentées dans cette section, SW signifie l’exécution de l’algorithme du flot optique sur la partie processeur ARM du circuit ZYNQ (PS) et CPU signifie l’exécution du même algorithme sur mon PC de bureau qui est équipé d’un processeur i7-6500Uquatre-cœurs d’une fréquence de 2.59 GHz. L’algorithme de traitement de vidéo fonctionnant sur ARM et CPU partage la même implémentation. HW signifie l’exécution de l’algorithme du flot optique sur la partie processeur ARM du circuit ZYNQ (PS) et la partie FPGA (PL).

#### Temps d’exécution

Le temps d’exécution de la méthode HS pour une séquence d’images de taille 640\*480 pixels est affiché sur la Figure ‎IV‑14. L’implémentation sur le processeur embarqué ARM est largement plus important (environ 27.4%) que le matériel HW. Notamment, le bloc d'itération (S4) dépense la plus grande partie du temps d'exécution quel que soit l’implémentation ; sur le logiciel ou le matériel. D’ailleurs, pour le cas du processeur ARM, le bloc d’itération pour cent itérations dépense 10.51 s et pour le cas du CPU, il dépense 5.14 s.

Figure ‑: temps d'exécution (s)

La Figure ‎IV‑15 montre le temps d’exécution de la méthode de HS, où ‘’W/O memcopy’’ signifie le temps nécessaire pour le calcul du flot optique sur FPGA sans tenir compte du temps de l’accès à la mémoire. Donc, si nous ne considérerons pas le temps des accès à la mémoire pour lire et écrire les images, le calcul du flot optique (les étapes S1 jusqu’à S4) sur FPGA est accéléré d’environ 93.58% par rapport à son implémentation sur le processeur ARM.

Figure ‑: temps d'exécution (s) du calcul du flot optique HW

##### Le temps d’exécution et la taille de l’image

Le Tableau ‎IV‑5 montre le temps d’exécution et l’accélération apportée du calcul du flot optique pour différentes tailles d’images. Plus la taille de l’image augmente, plus le temps d’exécution augmente et plus l’accélération est importante.

Tableau ‎IV‑5: Comparaison du temps d'exécution entre le SW et HW

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Image | Taille | SW(s) | HW(s) | HW W/O memCopy(s) | accélération |
| Yosemite | 316x252 | 4.739 | 3.38 | 0.37//7.3ms | 14.36//649 |
| Teddy | 420x360 | 9.60 | 6.41 | 0.48 | 20 |
| Army | 584x388 | 14.33 | 10.13 | 0.65 | 22.05 |
| GroveSun | 640x480 | 19.81 | 12.99 | 0.82 | 24.15 |

La Figure ‎IV‑16 illustre le temps d’exécution par pixel entre le SW et le HW. En ce qui concerne le SW, le temps d’exécution augmente avec la taille de l’image puisque plus de données intermédiaires causent plus de temps de calcul. En revanche, pour le HW, le temps d’exécution par pixel diminue lorsque la taille de l’image augmente en raison de l’efficacité du pipeline. Comme le montre la Figure ‎IV‑16, le temps d’exécution par pixel du SW augmente de 59.51 µs à 63.49 µs alors que celui du HW diminue de 4.14 µs à 3.17 µs. Ce résultat indique que l’accélération matérielle sur un FPGA devrait être consultée pour le traitement en temps réel et en particulier pour les images de taille importante.

Figure ‑: temps d'exécution par pixel

##### Le temps d’exécution en fonction du nombre d’itération

Le temps d’exécution de l’algorithme de calcul du flot optique si l’implémentation est réalisée sur le circuit Zynq est représenté sur la Figure ‎IV‑15. Comme indiqué précédemment, le temps d’exécution de l’algorithme en prenant en compte le temps des accès à la mémoire pour lire et écrire les données est très important. Comparer le temps d’exécution pris par le HW et le SW dans ce cas de figure n’est pas très impressionnant. Mais, l’accélération est quand même apportée. Avec l’augmentation du nombre d’itérations, que ce soit pour le matériel ou le logiciel, le temps d’exécution augmente. Par exemple, pour 100 itération, le logiciel consomme 4.739s et le matériel 3.38s ce qui apporte une accélération de x1.40.

Figure ‑: Temps d'exécution en fonction du nombre d'itération

#### Précision

Afin d’évaluer la précision de notre algorithme, il existe dans la littérature (38) une méthode qui permet de faire cela. Il s’agit du calcul de l’erreur angulaire moyenne AAE ‘’*Average Angular Error*’’. Plus cette valeur est petite plus la précision est meilleur. Cette dernière est définie comme suit :

AAE=

*Où : () le flot réel et () le flot calculé*

*Rows et cols sont respectivement le nombre de ligne et de colonne de l’image*

Comme indiqué sur la Figure ‎IV‑16, la précision s'améliore avec le nombre d'itérations; Cependant, ceci correspond à plus de temps d'exécution. Ce résultat signifie que le modèle HS nécessite beaucoup plus d'itérations pour obtenir plus précision et ceci diminue ses avantages en termes de vitesse d'exécution. Comme mentionné précédemment, la plupart des travaux liés à l'accélération du calcul du flot optique dense sur des plateformes FPGA ont choisi HS comme modèle représentatif. Il est le modèle le plus simple, mais il est moins précis parce qu’un nombre minimal d'itérations a été utilisé dans ces travaux pour démontrer la rapidité d’exécution.

Figure ‑: Précision du calcul en fonction du nombre d'itération

En générale, et comme le montre cette figure, le logiciel est toujours plus précis que le matériel et ceci est dû à l’architecture de la fenêtre glissante utilisée dans notre conception qui diffère de la fonction de filtrage2D de la bibliothèque OpenCV du logiciel HLS. D’ailleurs, parmi les différences majeures entre ces deux méthodes c’est l’affection des pixels de bord à la valeur zéro ce qui n’est pas le cas pour la fonction OpenCV.

#### Ressources

L'estimation des ressources est une métrique cruciale pour déterminer l’espace utilisé dans une conception FPGA. En règle générale, une utilisation minimale de ressources induit à un coût plus faible de la conception. Par conséquent, une meilleure conception devrait entraîner une utilisation minimale des ressources.

La Figure ‎IV‑19 montre le rapport de synthèse concernant l’utilisation des ressources de l’accélérateur de calcul du flot optique généré à partir de Vivado HLS et le rapport de l’implémentation du système de traitement complet généré via Vivado IDE en considérant les VDMAs et les autres IPs de notre design. Comme indiqué sur la figure, tous les types des ressources sont quantifiés. Le calcul du flot optique consomme beaucoup de ressources vu la complexité de calcul. En plus, dans notre implémentation, nous avons utilisé des valeurs 32-bit flottantes pour conserver la précision des calculs. Ceci requiert bien évidemment plus de ressources et plus de temps de latence.

Figure ‑: Estimation des ressources

Nous avons utilisé une architecture de la fenêtre glissante pour le calcul de dérivée. Cette architecture consomme beaucoup moins de ressources que la fonction du filtrage 2D de la bibliothèque OpenCV de Vivado HLS. En effet, cette architecture et comme le montre la Figure ‎IV‑20 réduit le nombre de FF ‘’Flip Flops ‘’ jusqu’à moins 53% et économise le nombre d’utilisation des LUTs ‘’Look Up Tables’’ jusqu’à 75% quand nous la comparant avec la fonction déjà prédéfinie.

Figure ‑: Comparaison entre les ressources utilisées dans la fonction de dérivée d'OpenCV et l'architecture utilisée

#### Consommation d’énergie

L'efficacité énergétique est une autre métrique très importante pour déterminer si la plateforme convient à une application donnée. Quand il y a des contraintes de puissance pour ces applications, les ingénieurs devraient accorder plus d’attention à la consommation d'énergie pour s'assurer de l’efficacité de leur conception. La Figure ‎IV‑21 présente la consommation d’énergie du système de calcul du flot optique comme évaluée à partir de VIVADO IDE.

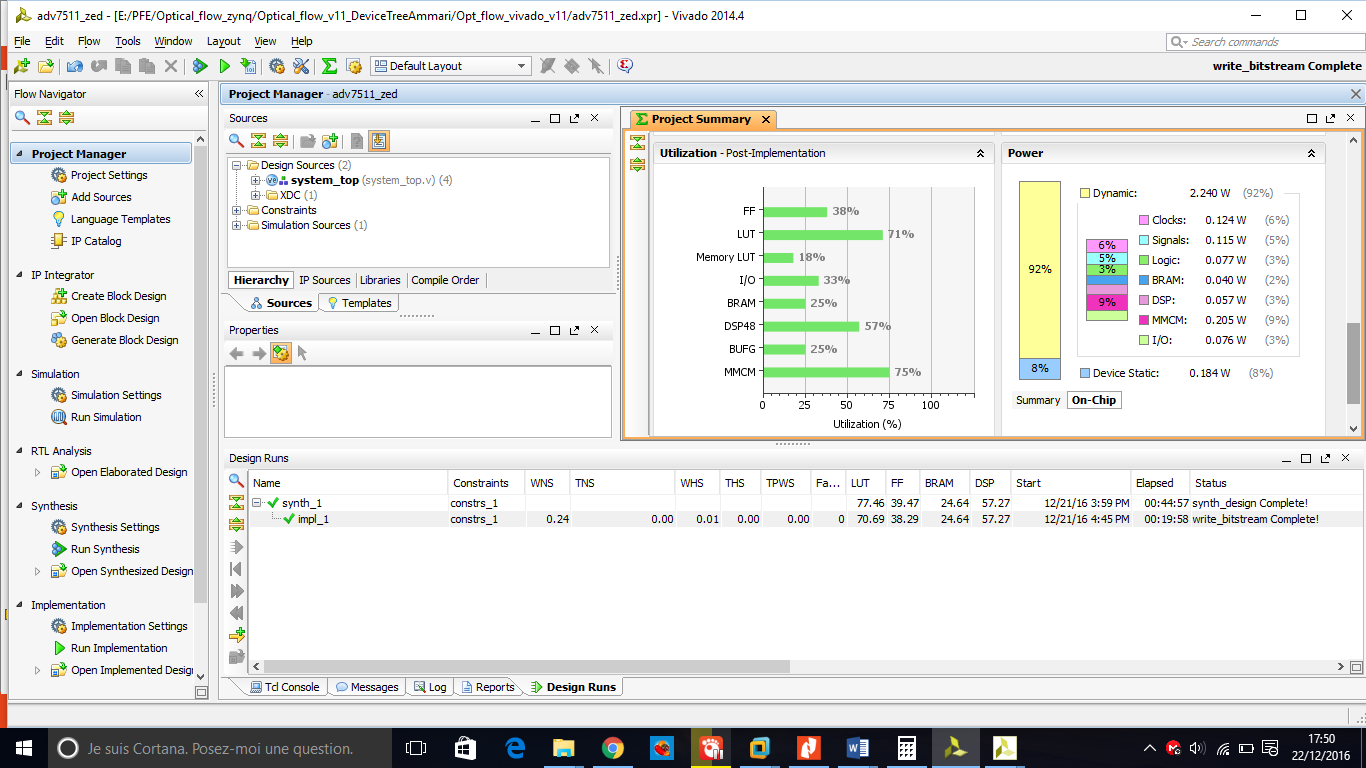


Figure ‑:Consommation d’énergie

En totale, notre système consomme 2.422W. Néanmoins, le processeur i7-6500U a une puissance TDP ‘’*Thermal Design Power*’’ de 15W. Donc, le Zynq consomme significativement moins de puissance que le processeur i7 (6.19 moins de puissance).

## Conclusion

Dans ce chapitre, nous avons proposé un flot de travail général pour le calcul du flot optique pour les trois modèles présents dans la littérature. La méthode d’implémentation optimisée représentait le modèle classique du calcul du flot optique HS sur le circuit de Xilinx FPGA-Zynq en exploitant les parallélismes inhérents de l’algorithme et du circuit FPGA-Zynq. Le temps d’exécution diminue à 0.82s avec une consommation d’énergie de 2.422W. Ces résultats sont bien meilleurs que l’implémentation sur le processeur Intel i7 qui coute 6.05 s et 15 W.

# Conclusion Et Perspectives

Le travail que nous venons de présenter s’inscrit dans le contexte des systèmes embarqués dédiés aux traitements d’images. Nous avons pu valider un flot de conception rapide pour une implémentation matérielle d’un algorithme de détection de mouvement sur les FPGAs.

Dans ce projet, nous avons utilisé le langage C au lieu de HDL pour implémenter l’algorithme de vision sur le circuit Zynq basé sur les FPGAs. Tous les accélérateurs conçus pour fonctionner sur les FPGAs sont développés via l’outil de Xilinx Vivado-HLS.

Suite au passage à un niveau d’abstraction plus élevé lors de la conception d’une application de vision pour une implémentation matérielle sur les FPGAs, la capacité de développer de tels systèmes sur les FPGAs et profiter des accélérations matérielles a considérablement progressé.

Notre implémentation nous a permis de mettre en œuvre deux progressions majeures. Tout d’abord, le processus de développement d’applications sur les FPGAs a été simplifié de manière considérable. Les outils HLS facilitent la tâche aux ingénieurs ayant une expérience limitée avec la programmation bas-niveau car ils sont plus efficace et produisent des codes plus robustes. Ensuite, la vitesse de calcul du flot optique a été considérablement améliorée. Nous avons obtenu une amélioration de 25 fois par rapport à la vitesse de calcul de l'approche logicielle pure tout en conservant une faible consommation d'énergie.

Plusieurs applications de vision peuvent tirer profit des avantages de l’accélérateur matériel et améliorer les performances temps réel des applications de vision grâce à cette nouvelle plateforme. Les résultats démontrent que les SOCs commerciaux basés sur les FPGAs et couplés à des outils de développement ont le potentiel de fournir une plate-forme haute performance et efficaces pour les applications de vison embarquées. L'optimisation proposée constitue une solution très utile pour la mise en œuvre du flot optique sur des plateformes basées sur FPGA. Dans notre projet, les performances temps réel ne sont pas satisfaisantes. Ceci est dû au temps nécessaire pour accéder à la mémoire afin de lire et écrire l’image.

Notre conception n’est donc pas totalement optimisée. Le délai pour qu’une copie de l’image depuis les VDMAs se transmet vers la mémoire, et vice-versa, est dû principalement à la latence de la mémoire DDR.

Ainsi, les perspectives futures sont d’une part l’utilisation d’un port qui fournit un accès directe à la mémoire sur puce. Il s’agit du port ACP ‘’Accelerator Coherency Port’’ dont le débit est supérieur aux ports HP. En effet, ACP fournit un chemin de plus faible latence à la mémoire.

D’autre part, nous pouvons tenter de diviser le programme sous Linux et exécuter chaque partie par un cœur du processeur ARM. C’est-à-dire que nous pouvons exploiter l’utilisation en parallèle des deux cœurs du processeur ARM.

# Bibliographie

1. **Embedded vision solutins . *Xilinx.* [En ligne] [Citation : 24 11 2016.] https://www.xilinx.com/applications/megatrends/video-vision.html.**

**2. Cucchiara, R., Grana, C., Piccardi, M., and Prati. *Detecting objects, shadows and ghosts in video streams by exploiting color and motion.* s.l. : In Image Analysis and Processing, 2001. Proceedings. 11th International, 2001.**

**3. Haritaoglu, D. Harwood, L.S. Davis. *W4: real-time surveillance of people and their activities, IEEE Trans. on Pattern Analysis and Machine Intelligence.* 2000.**

**4. Wren, C., Azarbayejani, A., Darrell, T., and Pentland. *Pfinder : real-time tracking of the human body.* s.l. : Pattern Analysis and Machine Intelligence, 1997.**

**5. Oliver, N., Rosario, B., and Pentland, A. *A bayesian computer vision system for modeling human interactions.* s.l. : Pattern Analysis and Machine Intelligence, IEEE Transactions, 2000.**

**6. Barnich, O. and Van Droogenbroeck, M. *A universal background subtraction algorithm for video sequences.* s.l. : Image Processing,IEEE Transactions, 2011.**

**7. Benezeth, Y., Jodoin, P.-M., Emile, B., Laurent, H., and Rosenberger. *Review and evaluation of commonly-implemented background subtraction.* s.l. : Pattern Recognition, 2008. ICPR 2008. 19th International Conference, 2008.**

**8. A.J. Lipton, H. Fujiyoshi, R. S. Patil. *Moving target classification and tracking from real-time video.* s.l. : Proc. of IEEE Workshop on Applications of Computer Vision, 1998.**

**9. WEI Shuigen, CHEN Zhen, LI Ming, ZHUO Liang. *An Improved Method of Motion Detection Based on Temporal Difference.* CHINA : International IEEE Workshop on Intelligent Systems and Applications, 2009.**

**10. A., Dziri. *Suivi visuel d'objets dans un réseau de caméras intelligentes embarquées.* Clermont-Ferrand : s.n., 2015.**

**11. Schunck, B. Horn et B. *Determining optical fow.* s.l. : Artificial Intelligence, 1981.**

**12. Fleet, D. and Weiss, Y. *Optical flow estimation.* s.l. : Handbook of Mathematical Models in Computer Vision, 2006.**

**13. S.Baker, I.Mathews. *Lucas-kanade.* Carnegie Mellon University ,tech : A unifying Framework, 2002.**

**14. Richard, A. *raitement statistique du signal.* s.l. : cours de 3ème année ENSEM, 2008.**

**15. al., R.T. Collins et. *A system for video surveillance and monitoring.* Carnegie Mellon University : VSAM, 2000.**

**16. Kui Liu, He Yang, Ben Ma, Qian Du. *A Joint optical Flow and Principal Component Analysis Approach for motion detection.* Mississippi State University : Department of Electrical and computer engineering , 2010.**

**17. Xilinx. *Zynq-7000 All Programmable SoC.* [En ligne] 2014. [Citation : 25 novembre 2016.] http://www.xilinx.com/products/silicon-devices/soc/zynq-7000.html.**

**18. H.-Y. Lin, L.-Q. Chen, Y.-H. Lin, and M.-S. Yu. *Lane departure and front collision warning using a single camera.* s.l. : IEEE International Symposium on Intelligent Signal Processing and Communications Systems, 2012.**

**19. B. Zhang, V. Appia, I. Pekkucuksen, A. U. Batur, P. Shastry, S. Liu, S. Sivasankaran, K. Chitnis, Y. Liu. *A surround view camera solution for embedded systems.* s.l. : IEEE Conference on Computer Vision and Pattern Recognition Workshops, 2014.**

**20. S., Zhun. *Rapid Prototyping of an FPGA-Based Video Processing System.* Blacksburg, Virginia : faculty of the Virginia Polytechnic Institute and State University, 2016.**

**21. G. Smith, G. Martin. *High-level synthesis: Past, present, and future. Design.* 2009. IEEE.**

**22. Devices, Analog. ADV7511 Transmetteur HDMI basse consommation avec canal de retour audio. [En ligne] [Citation : 26 novembre 2016.] Analog Devices '' ADV7511 Transmetteur HDMI basse consommation avec canal de retour audio [en ligne] valable '' http://www.analog.com/media/fr/technical-documentation/user-guides/ADV7511\_Hardware\_Users\_Guide.pdf '' [Visité Le 13-juin-2016]..**

**23. Crockett H., Elliot A., Enderwitz A., Stewart W. *The Zynq Book, Embedded Processing with the ARM Cortex-A9 on the Xilinx Zynq-7000 All Programmable SoC.* Glasgow, Scotland, UK : Department of Electronic and Electrical Engineering University of Strathclyde, 2014.**

**24. Fernando, Shakith. Sobel Filter Application on the Xilinx Zynq Zedboard. [En ligne] 21 Décembre 2012. [Citation : 13 juin 2016.]**

**25. Bruhn A., Weickert J. *Towards ultimate motion estimation: Combining highest accuracy.* s.l. : Proceedings of the 10th IEEE International Conference on Computer Vision, 2005. Vol. 1.**

**26. T. Brox, A. Bruhn, N. Papenberg et J. Weickert. *High Accuracy Optical.* s.l. : Proc. European Conf. Computer Vision, 2004. Vol. 4.**

**27. Mizukami Y., Tadamura K. *Optical ﬂow computation on compute uniﬁed device architecture.* s.l. : Proceedings of the 14th International Conference on Image Analysis and Processing, 2007.**

**28. Gwosdek P., Bruhn A. et Weickert J. *Variational optic ﬂow on the sony PlayStation3 .* s.l. : Real-Time Image Process, 2010.**

**29. Sundaram N., Brox T. et Keutzer K. *Dense point trajectories by GPU-accelerated large displacement optical ﬂow. .* s.l. : Computer Vision–ECCV 2010, 2010.**

**30. Zimmer H., Bruhn A. et Weickert J. *Optic ﬂow in harmony .* s.l. : J. Comput. Vis., 2011.**

**31. Xu L., Jia J. et Matsushita Y. *Motion detail preserving optical ﬂow estimation.* s.l. : IEEE, 2012.**

**32. Arribas P., Monasterio F. et Macia H. *FPGA implementation of Santos-Victor optical ﬂow algorithm for real-time image processing: An useful attempt.* s.l. : International Society for Optics and Photonics, 2003.**

**33. Martin L., Zulaga A., Cuadrad C., Lazaro J. et Bidarte U. *Hardware implementation of optical ﬂow constraint equation using FPGAs.* s.l. : Comput. Vis. Image , 2005.**

**34. S., Gultekin G.K et Afsar. *An FPGA based high performance optical ﬂow hardware design for computer vision applications.* s.l. : Microprocess. Microsyst, 2013.**

**35. kunz M., Ostrowski A., Zipf P. *An FPGA-optimized architecture of Horn and Schunck optical ﬂow algorithm for real-time applications.* s.l. : In Proceedings of the 2014 24th International Conference on Field Programmable Logic and Applications IEEE, 1–4., 2014.**

**36. Komorkiewicz M., Kryjak T., and Gorgon M. *Efﬁcient hardware implementation of the Horn-Schunck algorithm for high-resolution real-time dense optical ﬂow sensor.* s.l. : Sensors , 2014.**

**37. Middlebury. Datasets of Middlebury. [En ligne] 2015. [Citation : 23 novembre 2016.] http://vision.middlebury.edu/ﬂow/data/.**

**38. Baker S., Scharstein D., Lewis J.P., Roth S., Black J.M et Szeliski R. *A database and evaluation methodology for optical ﬂow.* s.l. : Computer Vision 1992, 2011.**

**39. K.P. Karmann, A. Brandt. *Moving object recognition using an adaptive background memory, in V Cappellini, Time-varying Image Processing and Moving Object Recognition.* Amsterdam, The Netherlands : s.n., 1990.**

**40. Kilger, M. *A shadow handler in a video-based real-time traffic monitoring system.* s.l. : IEEE Workshop on Applications of Computer Vision, 1992.**

**41. McFarlane, N. and Schofield. *Segmentation and tracking of piglets in images.* s.l. : Machine Vision and Applications, 1995.**

**42. Verbeke N., Nicole V. *Détection de movement cohérents dans une sequence vidéo.* s.l. : 4th International Conference : Sciences et Electronic Teechnologies of Information ans Telecomunications, 2007.**

# Annexes