Titre: Title:	Implementation of a Bacterium Tracking System on FPGA
Auteur: Author:	Vahid Talei
Date:	2010
Type:	Mémoire ou thèse / Dissertation or Thesis
Référence: Citation:	Talei, V. (2010). Implementation of a Bacterium Tracking System on FPGA [Mémoire de maîtrise, École Polytechnique de Montréal]. PolyPublie. https://publications.polymtl.ca/502/

UNIVERSITÉ DE MONTRÉAL

IMPLEMENTATION OF A BACTERIUM TRACKING SYSTEM ON FPGA

VAHID TALEI

DÉPARTEMENT DE GÉNIE INFORMATIQUE ET GÉNIE LOGICIEL

ÉCOLE POLYTECHNIQUE DE MONTRÉAL

MÉMOIRE PRÉSENTÉ EN VUE DE L'OBTENTION

DU DIPLÔME DE MAÎTRISE RECHERCHE

(GÉNIE INFORMATIQUE)

DÉCEMBRE 2010

UNIVERSITÉ DE MONTRÉAL

ÉCOLE POLYTECHNIQUE DE MONTRÉAL

Ce mémoire intitulé:

IMPLEMENTATION OF A BACTERIUM TRACKING SYSTEM ON FPGA

présenté par: TALEI Vahid

en vue de l'obtention du diplôme de : Maîtrise ès sciences appliquées

a été dûment accepté par le jury d'examen constitué de :

Mme NICOLESCU Gabriela, Doct., présidente

M. LANGLOIS J. M. Pierre, Ph.D., membre et directeur de recherche

M. MARTEL Sylvain, Ph.D., membre

To my parents and my brothers…

ACKNOWLEDGEMENT

I am thankful for the great support and assistance of my research director Dr. Pierre Langlois without his help this project would not be possible. I would like to thank him for his valuable insights and his patience during my time as a master's candidate.

I thank the CMC Microsystems for the starter kit which they provided to implement the project. I specially thank Ms. Susan Xu and Mr. Robert Mallard.

I would like to thank Professor Sylvain Martel who supported me by giving access to the equipments of the Laboratoire de Nanorobotique. Actually, the atmosphere of the laboratory had a positive effect on my research. I thank all the laboratory researchers and staffs, especially Mahmood Mohammadi and Charles Tremblay.

Thanks to Mr. Abbas Nemr who guided me through the synthesis of the design. I thank all my professors and the staffs of the computer engineering and software engineering department of the École Polytechnique de Montréal.

I express my deep gratitude toward my parents and my brothers without their emotional and financial support this project would have not been possible.

ABSTRACT

Tracking in computer vision is monitoring the movement of an object in a sequence of images. This object tracking may be performed for different reasons such as security systems, animation production, etc. Hundreds of publications in the form of books, articles, and journal papers indicate the importance of tracking as the research domain in universities and research centers. Tracking consists of the answer to two principal problems, the motion problem and the matching problem. Two solutions for the motion problem are adjacent regions method and the Kalman filter. Many solutions also exist for matching problem such as window tracking, detection of the moving object by its specifications like edges, corners and contour, detection of the target by simple shapes like circles, squares, rectangles and triangles, detection by 3-dimensional shapes like cylinders and finally matching with complex shapes like human, vehicle, etc. The implemented tracking algorithm in this project applies the adjacent regions for motion detection and window tracking for matching. Three different algorithms are simulated by Matlab software and one of them is implemented on the target hardware.

The subject of tracking in this project is Magnetotactic Bacterium (MTB). MTB is applied in medial and biomedical fields. As an example, they can carry necessary materials into the patient's vascular system, where only nanoparticles are able to travel.

The hardware of this project is the Xilinx ML402 video starter kit, camera and microscope. This hardware system consists of the ML402 main board and the VIODC daughter board. The VIODC contains a Virtex-II FPGA which holds the drivers of different video input and output interfaces. The ML402 has a Virtex-4 FPGA and developers can implement their designs on this FPGA chip. The camera captures the video frames and the VIODC receives the frames from camera and sends them to ML402. ML402 executes the tracking algorithm on the received frames and sends the appropriate output to VIODC to be displayed. The basic module of the tracking system which is called "vid_tracking" processor core (pcore) is available with the Simulink library of the Matlab software. It can also be simulated either by real-time video or by simulated video signals. Vid_tracking consists of EDK processor and System Generator blocks. System Generator translates the pcore to physical hardware. Through the translation process, the EDK processor of Simulink library is implemented as a MicroBlaze processor which is the standard processor architecture of Virtex-4 FPGAs. The design is then exported into the XPS platform. The system

developer can modify the design using the VHDL hardware description language. VHDL is very efficient and highly flexible to implement hardware systems with different abstraction levels.

We applied three DSP48 modules from the Virtex-4 library in order to implement the multiplication operation in a more efficient manner. This library contains some already designed components for different Xilinx FPGAs. These components are very fast and easy to use. In order to implement the tracking algorithm, we profit from three synchronization signals which are pixel enable, horizontal synchronization and vertical synchronization. Pixel enable indicates when a new pixel is detected by the VIODC. The horizontal synchronization signal indicates the start of a line and the vertical synchronization shows the start of a frame. The tracking algorithm counts the pixels and lines to find a window of about 20-pixel by 20-pixel in the middle of the frame. It looks for a bacterium by comparing the pixel's intensity. When it finds a bacterium, it looks for it in the 20-pixel by 20- pixel regions around the previous window in the next frame.

The Matlab simulation of MTB tracking is 100% robust since the bacterium was not missed in any of the frames during the tracking and the algorithm is able to follow the trajectory of the bacterium during all the simulation time. The simulation indicates 100% precision in 2-dimensional movements of the bacterium, since all the pixels which are recognized by the human eye as the bacterium pixels are also detected as the bacterium pixels by the simulator algorithm.

The designed hardware detects whether a bacterium is in camera field of view or not. The bacterium moves at an average speed between 180 to 240 micrometers per second. A speed of 300 μm/s has also been recorded for this type of bacterium. Our tracking system could detect the moving bacterium at either of these speeds.

RÉSUMÉ

Dans le domaine de la vision par ordinateur, le suivi est la surveillance du mouvement d'un objet dans une séquence d'images. Ce suivi d'objet peut être accompli dans différentes applications comme les systèmes de sûreté, la production de dessins animés, etc. Des centaines de publications sous forme de livres, articles et documents de journaux indiquent l'importance du suivi dans la recherche académique et dans les centres de recherche. Le suivi contient la réponse à deux problèmes principaux, le problème du mouvement et le problème de la correspondance. Deux solutions pour le problème du mouvement sont la méthode des régions contiguës et le filtre de Kalman. D'autres solutions existent également pour le problème de correspondance, par exemple le suivi par fenêtre, la détection d'objet mobile par des spécifications comme des arrêtes, des coins et le contour, la détection d'objet par des formes simples comme des cercles, des carrés et des rectangles, la détection par des formes en 3-dimensions comme des cylindres, et finalement la correspondance par des formes complexes comme l'humain, le véhicule, etc. L'algorithme de suivi dans ce projet applique la méthode des régions contiguës pour le problème du mouvement et la méthode de suivi par fenêtre pour le problème de correspondance.

Le sujet du suivi de ce projet est la bactérie magnétotactique (MTB). Les MTB sont utilisées dans des applications médicales. Par exemple, elles peuvent amener les médicaments nécessaires dans le système vasculaire du patient, où seulement des nanoparticules sont capables de se rendre.

Le matériel utilisé dans ce projet est la planchette de développement pour les applications de vidéo ML402 de la compagnie Xilinx, une caméra et un microscope. Ce système matériel contient le circuit principal ML402 et la planchette VIODC. La VIODC contient un FPGA Virtex-II qui comprend des pilotes pour différentes interfaces de l'entrée et de la sortie vidéo. La ML402 est dotée d'un FPGA Virtex-4 sur laquelle les développeurs peuvent implémenter leurs designs. La VIODC reçoit les trames vidéo capturées par la caméra et les envoie vers la ML402. Cette dernière exécute l'algorithme de suivi sur les trames reçues et envoie la sortie appropriée vers la VIODC pour être affichée. Le composant principal du système de suivi qui s'appelle le processeur cœur (pcore) "vid_tracking" est disponible dans la librairie Simulink du logiciel Matlab. Ce composant peut être simulé soit par vidéo en temps-réel soit par des signaux simulés de vidéo. Le vid_tracking contient le processeur EDK et des blocs de générateur de système. Le bloc générateur de système traduit le pcore en du matériel physique. Pendant le processus de la

traduction, le processeur EDK de la librairie Simulink est implémenté comme un processeur MicroBlaze qui est l'architecture de processeur standard des FPGA Virtex-4. Ensuite, le design est exporté à la plateforme XPS. Le développeur de système peut modifier le design en utilisant le langage de description matériel VHDL. VHDL est très efficace et extrêmement flexible pour implémenter des systèmes matériels avec des niveaux d'abstraction différents.

On a appliqué trois modules de DSP48 de la librairie Virtex-4 pour implémenter la multiplication d'une manière plus efficace. Cette librairie contient des composants déjà conçus pour différents FPGAs de Xilinx. Ces composants sont très rapides et faciles à utiliser. Pour implémenter l'algorithme de suivi, on profite de trois signaux de synchronisation : -pixel actif, synchronisation horizontale et synchronisation verticale. Le signal pixel actif indique la détection d'un nouveau pixel par la VIODC. Le signal de la synchronisation horizontale indique le début d'une ligne et le signal de la synchronisation verticale montre le début d'une trame. L'algorithme de suivi compte le nombre des pixels et le nombre de lignes pour trouver une fenêtre de 20-pixel par 20-pixel au milieu de la trame. Il cherche une bactérie par comparaison des intensités de pixels. Quand il trouve une bactérie, il la recherche dans les régions de la taille 20-pixel par 20-pixel dans les fenêtres qui suivent au voisinage de la fenêtre précédente.

La simulation Matlab de suivi de MTB est 100% robuste car la bactérie n'a été perdue dans aucune trame pendant le suivi de bactérie. L'algorithme est capable de suivre la trajectoire de la bactérie pendant toute la durée de la simulation. La simulation atteint 100% de précision dans le cas de mouvement à 2-dimensions de la bactérie en comparant les résultats obtenus par l'algorithme à ceux obtenus par un observateur humain.

Le matériel conçu détecte si une bactérie est dans le champ de vue de la caméra ou non. La bactérie se déplace avec une vitesse moyenne de 180 à 240 micromètres par seconde. Une vitesse de 300 µm/s a été aussi enregistrée pour ce type de bactérie. Notre système de suivi peut détecter la bactérie qui se déplace à n'importe quelle vitesse dans la gamme spécifiée ci-dessus.

CONDENSÉ EN FRANÇAIS

Dans le domaine de la vision par ordinateur, le suivi est la surveillance du mouvement d'un objet dans une séquence d'images. Le suivi d'objet peut être exécuté dans différentes applications comme les systèmes de sûreté, la production de dessins aminés, etc. Des centaines de publication dans la forme de livres, articles et journaux indiquent l'importance du suivi comme le domaine de recherche dans les universités et les centres de recherche. Le suivi contient la réponse de deux problèmes principaux, le problème du mouvement et le problème de la correspondance. La solution au problème du mouvement définit les régions dans l'image où l'objet en mouvement est plus probable à exister. Ces régions sont les candidats sur lesquelles les techniques de la correspondance sont appliquées. Les solutions au problème de la correspondance appliquent différents algorithmes à chercher l'objet en mouvement entre la région ou les régions qui sont sélectionnées pendant le processus de la solution au problème du mouvement. Deux méthodes principales à résoudre le problème du mouvement sont des régions contiguës et le filtre de Kalman. Dans la méthode des régions contiguës, la possibilité que l'objet soit trouvé au voisinage de son endroit précédent est plus élevée parce que l'objet en mouvement ne peut pas déplacer loin dans deux trames capturées consécutivement. L'autre solution, le filtre de Kalman, est une méthode qui prévoit l'état suivant du système. Cette solution est plutôt utilisée dans les systèmes de contrôle qui prévoit l'état suivant.

Les solutions au problème de la correspondance sont diverses. Certains algorithmes appliquent une ou plus de techniques à trouver l'objet en mouvement entre les objets candidats dans une image. La méthode du suivi par fenêtre est une solution au problème de la correspondance. Dans cette méthode, la région de la recherche est comparée aux régions sélectionnées dans la solution du mouvement pour trouver la région la plus similaire. Donc, les solutions au problème du mouvement sont utilisées à déterminer les régions avec qui la région de recherche doit être comparée. En plus de la méthode du suivi par fenêtre, il y a d'autres méthodes qui utilisent les caractéristiques de l'image à reconnaitre l'objet en mouvement. Ces caractéristiques peuvent être des spécifications d'objet comme des arrêts, lignes, coins et contour. Certaines méthodes reconnaissent l'objet de cible par des formes primitives en 2-dimensions comme des cercles, rectangles et triangles. Les algorithmes plus compliqués cherchent l'objet par des formes en 3-dimensions comme cylindres. Finalement, des algorithmes très compliqués cherchent la forme entière comme l'humain, véhicule, etc.

Le sujet du suivi dans ce projet est la bactérie. La bacterie magnétoctactique (MTB) est un type de bactérie qui est sensible au champ magnétique à la cause de l'organe magnétique de son corps. Elle change sa direction selon le champ magnétique. La MTB a une forme ronde. Elle a une taille de 1 – 3 µm de diamètre avec une vélocité du mouvement de 40 à 80 µm/s. Ce type de bactérie est utilisé dans les traitements médicaux. La bactérie est située sous le microscope et elle est suivie par le système du suivi qui est présenté dans ce projet.

Les anciens travails appliquent différentes approches pour implémenter l'algorithme du suivi. La plupart d'entre eux appliquent les régions contiguës comme la solution pour le problème du mouvement. Il y a des algorithmes qui sont implémentés en deux dimensions. Les algorithmes plus forts et plus compliqués sont exécutés en 3-dimensions. L'illumination du microscope est aussi importante. Dans une sorte de l'illumination, on a les bactéries blanches sur un arrière-plan noir. Dans une autre sorte, les bactéries sont noires et l'arrière-plan est blanc. La taille et la forme des bactéries sont variables dans différents cas. Ce dernier dépend de la configuration du microscope.

La méthode de solution au problème du mouvement qui est utilisée dans le système du suivi de ce projet est la méthode des régions contiguës et la solution de la correspondance est la méthode du suivi par fenêtre. L'algorithme du suivi est développé et simulé avec le logiciel Matlab et les résultats sont discutés plus tard.

Dans les systèmes du suivi, on a un objet qui doit être suivi. Cet objet est appelé la région de l'intérêt qui doit être suivi automatiquement. Le but des systèmes du suivi est de ne pas perdre la trajectoire de la région de l'intérêt. C'est très important que la région de l'intérêt doit être suivie en temps-réel dans ce projet. Contrairement à certains travaux qui effectuent le suivi sur les séquences de vidéo qui sont déjà enregistrées, le sujet du suivi est un objet vivant dans ce projet.

Ce projet surveille la bactérie et continue à suivre sa trajectoire. L'ancien travail sur ce sujet a perdu la trajectoire de la bactérie. Pour résoudre ce problème, il faut augmenter la fréquence de capturer les images par la caméra. Quand on augmente la fréquence des images, on a besoin d'un système du suivi rapide. Ce système doit être assez rapide à exécuter l'algorithme du suivi sur toutes les images entrantes. Par conséquence, le système du suivi doit contenir un processeur assez efficace pour traiter l'algorithme du suivi sur les images.

L'ancien travail a été implémenté sur un ordinateur personnel avec un processeur générique. Le processeur n'était pas assez rapide à traiter toutes les trames de la séquence de vidéo entrantes. Donc, certaines trames sont laissées sans être traitées. Par conséquence, un processeur spécialisé comme un FPGA qui est plus rapide et plus efficace est considéré à être approprié pour ce projet.

La planchette de développement pour les applications vidéo (VSK) ML402 est le matériel sur lequel l'algorithme du suivi est implémenté. Cette planchette contient la planchette principale ML402 et la planchette de l'entrée et de la sortie de vidéo VIODC. Ce dernier contient différentes interfaces de l'entrée et de la sortie vidéo comme VGA, DVI, SDI, Composite Video, et S-Video. VGA est un standard de l'entrée et la sortie analogique. DVI est une interface de vidéo numérique. SDI supporte l'entrée et la sortie de la vidéo numérique de série. S-vidéo et Composite Vidéo supportent les standards NTSC, PAL, et d'autres formats de la définition standard (SD). La VIODC contient un FPGA Virtex-II de Xilinx et différents DPSs et contrôleurs qui comprennent les pilotes pour les interfaces vidéo. La VIODC a aussi le port d'entrée pour la caméra. La caméra qui est utilisée dans ce projet capture les images de 752 × 480 pixels à la fréquence de 60 Hz. La VIODC reçoit les trames de vidéo de la caméra et les envoie vers ML402 à travers de bus VIOBUS. Le VIOBUS est un bus de la largeur 64 bits. Le connecteur XGI est l'interface pour connecter les planchettes ML402 et VIODC. Le VIODC constitue de deux bus chacun de la largeur 27 bits. Un bus porte le signal vidéo de la ML402 à la VIODC et l'autre bus porte le signal vidéo de la VIODC à la ML402. La ML402 contient un FPGA Virtex-4 de Xilinx qui peut être programmé à implémenter des algorithmes du traitement vidéo. Effectivement, la ML402 reçoit des signaux vidéo de la VIODC à travers du VIOBUS, exécute l'algorithme du traitement vidéo et envoi les résultats vers la VIODC à être envoyés à la sortie. La ML402 a d'autres interfaces comme le LCD, port série RS-232, Ethernet 10/100/1000, deux ports PS/2, contrôleur USB et DDR SDRAM. La ML402 VSK fournit la simulation du système du traitement de vidéo pour les concepteurs. Simulation est nécessaire pour étudier la fonctionnalité du système. On peut obtenir une bonne mesure de la fiabilité du système. On peut aussi analyser le comportement du système dans différentes circonstances et s'assurer si une erreur a lieu pendant la simulation. Il y a deux méthodes que les concepteurs peuvent utiliser pour simuler les systèmes du traitement de vidéo. Dans une méthode le concepteur peut simuler tout le système par le logiciel. Dans une deuxième méthode le concepteur peut utiliser le matériel et une séquence de la vidéo réelle et effectuer une simulation en temps-réel. Donc, le VSK et les

logiciels accompagnants permettent le concepteur à développer et tester les modules de traitement vidéo en matériel réel et le flux vidéo en temps-réel.

La simulation par le logiciel impose certaines limitations. Le contrôle du flux vidéo en temps-réel n'est pas facile; en plus, le stockage d'une grande quantité de la donnée vidéo à traiter a besoin d'une énorme quantité de la mémoire et une domination sur la gestion de mémoire.

La simulation par le matériel est plus rapide que la simulation par le logiciel. Dans cette méthode, la VIODC et ses périphériques fournissent le flux vidéo pour la ML402. La ML402 exécute le traitement de vidéo en temps-réel sur le flux vidéo. La ML402 est équipée par un FPGA Virtex-4 comme le processeur principal qui supporte l'architecture du processeur Microblaze, la mémoire DDR SDRAM pour enregistrer la donnée vidéo, le moyen de la communication comme le port série RS-232, et l'environnement de développement logiciel comme Simulink. Le concepteur du système peut développer le composant vidéo par le Simulink et générer le composant par le générateur du système. Quand la simulation est démarrée par le Simulink, le composant est intégré dans le FPGA de la ML402. Un flux vidéo soit temps-réel soit non-temps-réel est fournit par la VIODC pour être traité par ce composant. Finalement, le résultat est renvoyé au Simulink pour être affiché sur l'écran.

Dans ce projet on génère une séquence de vidéo par le Simulink. Cette séquence de vidéo contient les trames de 10 lignes. Chaque ligne de cette trame contient 100 pixels. L'horloge du système fonctionne à une fréquence de 100 MHz. La fréquence du signal pixel actif est 26.6 MHz. On conçoit le système du suivi et effectue une simulation par logiciel dans ce projet. Les résultats sont affichés sur l'écran par les outils du Simulink. Donc, on peut étudier le comportement du système et vérifier sa fonctionnalité.

Le processeur est le composant le plus important dans chaque système embarqué du traitement de vidéo. Le processeur standard de la ML402 est le MicroBlaze. Le processeur configurable de MicroBlaze est configuré par la trousse du développement embarqué (EDK) et le générateur système (System Generator) pour Virtex-4. En plus, le générateur système génère des interfaces pour lire et écrire les données et réduit beaucoup de temps et d'effort de la conception. Il y a deux méthodes pour développer le système de microprocesseur de MicroBlaze; soit le système qui est développé par le générateur système est exporté à l'EDK ou un design qui est conçu par l'EDK est importé dans le générateur système. La première méthode est utilisée dans ce projet.

Le pcore est développé par le logiciel Matlab avec l'aide de la libraire Simulink et il est généré par le générateur système. Ensuite, le pcore est exporté à la plateforme Xilinx Platform Studio (XPS). En plus du processeur EDK qui est nécessaire pour tous les pcores, certaines moyennes de la communication sont nécessaires à transférer des données entre le processeur EDK et de périphérique. Les mémoires partagées dans la forme de FIFO, le registre et la mémoire RAM sont en charge de la communication.

Pour développer le VSK, il y a certains designs de référence disponibles sur le site web de Xilinx. Ces designs de référence facilitent la conception du système désiré pour les développeurs. Le design de référence VSK_pcore est appliqué à développer ce projet. Le VSK_pcore est modifiable avec le logiciel Matlab. Ce design contient le pcore "vid_tracking". Ce pcore exécute l'algorithme du suivi. Ce pcore contient les composants ci-dessous : un processeur EDK qui sera traduit à un processeur MicroBlaze après la génération du pcore, les ports de l'entée et de la sortie pour le signal vidéo, les modules unpack et repack pour séparer et joindre les éléments de signal vidéo, et le module du générateur système qui générera le pcore.

Afin d'implémenter l'algorithme du suivi, il va falloir lire les pixels de chaque trame de vidéo. Il faudrait aussi ajouter un registre au design et produire les interfaces nécessaires à communiquer avec le processeur MicroBlaze. Ensuite, on génère le pcore et l'exporte à l'EDK. On ouvre le pcore dans la XPS et le connecte au pcore vio_if qui est l'interface de la VIODC. Les fonctions nécessaires pour lire les données du processeur et les écrire sont générées automatiquement. On écrit tout simplement le code désiré dans le corps des fonctions pour envoyer la valeur de pixels au PC à travers du port série RS-232. Ces valeurs peuvent être lues avec une application de la communication série comme HyperTerminal. Le matériel du système est compilé et synthétisé et les fichiers de format bit sont mis à jour pour être capable a télécharger le design sur la VSK. On utilise la carte de mémoire de flash SystemAce pour copier le design sur la VSK et ce dernier lit les configurations des FPGAs de cette mémoire externe. Dès que la VSK est mis en marche, la configuration du FPGA de la ML402 et de la VIODC est chargée et les deux circuits commencent à fonctionner. Le résultat du design est un flux de données de l'intensité des pixels. On a besoin de savoir les coordonnées des pixels de bactérie. Ce design ne fournit pas d'information des coordonnées du pixel. En conséquence, on doit chercher un moyen qui fournit plus de détails sur les valeurs des pixels et les coordonnées des pixels. Autrement dit, on a besoin de plus d'information sur la ligne de pixel et son décalage dont la valeur est affichée sur l'écran.

Le générateur système produit la description matérielle des pcores quand il traduit des designs conçus par Simulink au matériel. Cette description de matériel est au langage VHDL. VHDL est un langage de la description matériel. Il fournit très haute flexibilité de la conception des systèmes matériels dans différents niveaux de l'abstraction. La XPS fournit des moyens pour modifier le pcore au langage VHDL. VHDL profite des techniques pour travailler avec des signaux. Pendant le développement du code du pcore en VHDL, on a d'abord développé le code dans la plateforme Modelsim. Modelsim est un logiciel pour compiler et déboguer le code VHDL. Ensuite, on a copié le code dans la XPS pour compiler et synthétiser le code. Pendant la compilation du code dans la Modelsim et la XPS, on a remarqué qu'il y a certaines instructions qui ne sont pas synthétisables par la XPS. Donc, il faut être conscient à utiliser des structures synthétisables pendant le développement de code en VHDL. Par exemple, on convertit l'image en couleur à l'image grise dans notre algorithme. Cette conversion réduit le calcul de l'algorithme. Parce que toutes les instructions de l'algorithme doivent être effectuées sur trois canaux de rouge, vert et bleu dan une image en couleur, tant que l'algorithme est effectué sur un seul canal gris dans une image grise. Par conséquence, le coût de calcul est réduit par presque trois fois dans une image grise. Pour convertir une image en couleur à une image grise, tous les pixels doivent être changés. Ce dernier est faisable par la formule de la conversion du pixel en couleur au pixel gris. La formule de la conversion du pixel en couleur au pixel gris contient la multiplication et la division. La division n'est pas synthétisable par la XPS. En conséquence, on doit appliquer des composants de la librairie de Virtex-4. Ces composants sont déjà conçus et ajoutés à la librairie de Virtex-4. Le concepteur doit simplement appeler la librairie dans le code VHDL et relier les ports de l'entrée et de la sortie aux signaux. Ces composants aident notre code à devenir synthétisable. De plus, ces composants sont conçus dans un moyen efficace et leur délai est très petit. Autrement dit, ils fonctionnent très rapide. Ils aussi aident à réduire le coût matériel. Car ces composants ont également les architectures efficaces. On applique trois modules de DSP48 à exécuter trois multiplications de la formule de conversion de RGB à gris.

Les données de vidéo sont dans la forme d'un signal de 33 bits. Les premiers 10 bits sont alloués au canal rouge, les deuxièmes 10 bit sont pour le canal vert et les troisièmes 10 bits pour le canal bleu. Trois bits du pixel actif, la synchronisation horizontale et la synchronisation verticale sont utilisés pour synchroniser les pixels avec la trame vidéo. Le pixel actif change au zéro quand un nouveau pixel est détecté par la VIODC. Le signal de la synchronisation

horizontale change au zéro quand une nouvelle ligne a lieu dans la trame. De la même façon, la synchronisation verticale change au zéro quand une nouvelle trame est détectée par la VIODC. Par conséquence, les changements des signaux sont importants dans notre code VHDL. Pour trouver la position de la bactérie, on a besoin de compter les changements de '1' à '0' des signaux pixel actif, synchronisation verticale et synchronisation horizontale. On applique deux bascules pour implémenter chaque changement. La première bascule garde le statut de signal au moment dernier et la deuxième bascule garde le statut de signal au moment actuel. On compare le statut de signal dans les deux moments consécutifs et on vérifie si un changement de signal a eu lieu.

Pour implémenter l'algorithme du suivi, on compte des pixels et des lignes pour trouver une fenêtre 20-pixel à 20-pixel au milieu de la trame. Dès qu'on trouve une bactérie dans la fenêtre, on continue à suivre la bactérie. On garde les coordonnées de la position de bactérie dans un bloc mémoire RAM. Le module RAM est aussi disponible dans la librairie de Virtex-4. Les coordonnées sont lues de la RAM et l'algorithme cherche la bactérie dans les régions contiguës de la fenêtre dans la trame suivante. Si une valeur de l'intensité plus que 100 est détectée, le pixel appartient à la bactérie. Car la bactérie a une forme ronde avec des pixels blancs sur un arrière plan noir, elle est facilement détectée par application d'un seuil sur l'image.

Après la compilation et la synthèse de l'algorithme du suivi, on génère les fichiers bit pour télécharger sur les puces FPGA. Il y a deux méthodes pour télécharger les fichiers bit. On peut utiliser soit le câble JTAG ou la mémoire Flash pour télécharger les fichiers. Pour télécharger le fichier avec le câble JTAG, on a besoin du logiciel iMPACT. Si le câble est connecté à la ML402 et le VSK est allumé, iMPACT détecte tous les FPGAs sur les planchettes automatiquement. Ensuite, on peut choisir quel fichier doit être téléchargé sur quel FPGA. L'autre méthode est télécharger les fichiers de la configuration sur le VSK en utilisant la mémoire Flash. On doit générer un fichier "ace" pour télécharger sur les FPGAs dans cette méthode. Le programme "ml40x_bit2ace" génère le fichier ace des fichiers bit. Ce fichier ace configurera les FPGAs.

On a besoin d'un lecteur de carte pour télécharger le fichier ace sur la Flash. On connecte le lecteur de carte à l'ordinateur et copie le fichier sur la mémoire Flash. Huit différentes configurations peuvent être téléchargées sur la carte de mémoire Flash. Chaque configuration a son adresse unique. Quand l'utilisateur allume le système et indique l'adresse de la configuration favorable, la configuration pertinente est téléchargée sur les FPGAs de la ML402 et la VIODC.

La simulation Matlab du suivi de la MTB est 100% robuste car la bactérie n'a été perdue dans aucune trame pendant le suivi de la bactérie et l'algorithme est capable à suivre la trajectoire de la bactérie pendant toute la simulation. La simulation indique 100% de la précision dans le cas du mouvement à 2-dimensions de la bactérie, car des pixels qui sont reconnus par un observateur humain et ceux qui sont détectés par l'algorithme de simulation se ressemblent.

Ce système du suivi consomme moins de 50% de ports logiques du FPGA. Donc, notre algorithme est assez efficace à propos de matériel disponible. La consommation de bascules est 19% (5872 du total 30720) et la consommation des unités logiques est 31% (9570 du total 30720).

Ce projet est bien développé pour les concepteurs qui décident à implémenter leur algorithme du suivi. Le langage VHDL fournit la facilité et la flexibilité aux concepteurs à implémenter des algorithmes du suivi plus compliqués pas seulement pour suivre une bactérie, mais pour suivre toutes sortes d'objet en mouvement.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ACRONYMS AND ABBREVIATIONS

ASIC	Application Specific Integrated Circuit
BF	Bright Field
BTR	Branch Target Register
CR	Chlamydomonas Reinhardtii
CLB	Configurable Logic Block
DCM	Digital Clock Manager
DF	Dark Field
DIC	Differential Interface Contrast
DVI	Digital Video Interface
DXCL	Data cache over CacheLink
EAR	Exception Address Register
EDK	Embedded Development Kit
ESR	Exception Status Register
FIFO	First-In First-Out
FPGA	Field Programmable Gate Array
FPU	Floating Point Unit
FSL	Fast Simplex Link
FSR	Floating point Status Register
HAMSM	Hierarchical Adaptive Merge Split Mesh
HASM	Hierarchical Adaptive Structured Mesh
HD	High Definition
HDL	Hardware Description Language
IC	Integrated Circuit

IP	Intellectual Property
ISE	Integrated Software Environment
IXCL	Instruction cache over CacheLink
KF	Kalman Filter
LMB	Local Memory Bus
LUT	Look Up Table
MGT	Multi-Gigabit Transceiver
MSR	Machine Status Register
MTB	Magnetotactic Bacterium
NCC	Normalized Cross Correlation
OPB	On-chip Peripheral Bus
PC	Personal Computer
PC	Program Counter
PVR	Program Version Register
RAM	Random Access Memory
RISC	Reduced Instruction Set Computer
ROI	Region Of Interest
ROM	Read-Only Memory
SAD	Sum of Absolute Differences
SD	Standard Definition
SSD	Sum of Squared Differences
STA	Static Timing Analysis
TOI	Target Of Interest
VIODC	Video Input Output Daughter Card

VSK	Video Starter Kit
XPS	Xilinx Platform Studio
XST	Xilinx Synthesis Technology

CHAPTER 1 INTRODUCTION

In the computer vision domain, tracking means monitoring the motion of objects in a given sequence of images. This monitoring usually follows a higher goal rather than solely discovering the trajectory of an object. Further processing of tracking results makes it an interesting research domain. Many researchers in video processing have identified the tracking problem as a challenging subject. They have considered a variety of applications including different conditions and have proposed different solutions to these problems. Different conditions may include occlusion, low intensity or high intensity scenes, and various weather conditions. Some of the considered applications include:

- Motion capture: This application tracks all the motions of a person to make a record of them. We can then process this record to track a person in a crowded scene, or to make cartoon characters according to this person's motions [1].
- Recognition from motion: We may study different characteristics of objects by tracking them. We obtain the object trajectory, or we may determine its identity [2].
- Surveillance: Means monitoring the activities of definite objects in a scene and give a warning when a problem or a dangerous case is going to take place. Surveillance is used for security purposes by detecting the objects left in public places and also detecting suspicious objects [3], [4].

In tracking problems, we have a target object, also called region of interest, which should be followed automatically. Designing a tracking system involves finding solutions to two main problems: *Segmentation* and *Registration* (or *Tracking*) [5]. These two parts should be designed as efficiently as possible.

Segmentation defines the parts of the image which are of special importance. These important and meaningful parts can be intended colors, special shapes, or interesting regions. In tracking algorithms, the answer to segmentation determines the moving objects in a scene. These moving objects are candidates for target objects which are detected in the next step and defined by the registration solution.

The solution to the registration part identifies the target object in the next frame among the objects which were segmented. Two consecutive images are compared according to metrics

which are specified in the algorithm. The moving object in the current image which has the most correspondence with the moving object in the previous image will be extracted as target. The metrics of correspondence can be color, shape, and location.

Many challenges take place during the study and development of tracking algorithms. One of the most popular subjects is occlusion. In occlusion problems, two or more items overlap each other so that recognizing them is difficult. Therefore, the algorithm developer has to look for solutions to assign the scene regions to the appropriate object. A solution based on neural network models was developed to recognize the visible and invisible occluding objects in [6]. Occlusion in cell tracking algorithms appears in the form of touching spots. There are many publications which address the separation of the touching spots according to methods such as sharpen concavity, strong gray-contrast between two spots, etc [7].

Another challenge is reducing the shadow of the objects on the scene. Since the shadow moves with all the displacements of the object, it is very difficult to recognize the moving object from its shadow. Some techniques such as background subtraction are not very useful in recognizing the shadow; therefore, some effort is also allocated to suggest solutions for reducing the effects of shadows on efficient tracking of moving objects [8].

1.1 Problem statement

As for any other project, the design constraints which are imposed by clients, tools, technology and market should be respected. Thus, we must consider the whole limitations while providing a design which is comparable to other systems regarding the performance, cost, design time, reliability, reusability, etc.

As can be imagined, the problem in tracking is that the moving object must not be missed. The most important problem of this specific case in which we try to track a bacterium cell is real-time tracking. Contrary to some tracking systems which process recorded video sequences, the subject of tracking in this project is an alive moving object. This moving object should not be lost in any single frame during the tracking process. Therefore, the tracking system must be fast and efficient enough not to lose the bacterium trajectory. Previous works in the Nanorobotics Laboratory resulted in inadequate processing speed; therefore, the tracking system missed some video frames [9]. As a result, it missed the trajectory of the bacterium. Two solutions for this problem are

possible. First, we should reduce the frequency of entering frames. When the tracking system cannot keep pace with the speed of the moving object, we should reduce the number of entering frames. Therefore, the tracking system has enough time to process all frames. The problem is that if the interval between two consecutive frames increases, it will be difficult to distinguish the new location of the bacterium. As we will see in the next chapter, the basis of tracking is to compare the location of the bacterium in two consecutive frames in a limited area of the frame which is called the search window. If the distance between two locations of bacterium is so far that we will not be able to correspond the new location of the bacterium with its previous location, the bacterium will not be distinguishable between other bacteria. Therefore, the target may be not correctly recognized among the surrounding bacteria and the bacterium trajectory is lost in the second frame.

The second solution to the problem is to increase the system processing speed to keep pace with the bacterium motion. This solution is preferred for this project and we have applied this solution for our implementations. The tracking system which was designed in [9] is based on a general-purpose processor. In other words, a personal computer (PC) is in charge of tracking the bacteria. The systems based on general-purpose processors are less complicated to design, but slower than systems based on single-purpose processors. Furthermore, the design cost of an embedded system (with a single-purpose processor) will be much more than a general-purpose processor system. In this project, we decided to design a system which is based on a single-purpose processor. Field Programmable Gate Arrays (FPGAs) can support the implementation of single-purpose processors in which both the hardware and software can be designed, on the contrary of microcontrollers, in which the hardware is fixed and the system developer can just develop a software program to determine the system functionality. FPGAs profit from high frequency system clocks. Some efficient and well designed standard processors (such as PowerPC, MicroBlaze, and Nios processors) are designed for them. Many of the instructions of these processors are executed in one or two clock cycles. As a result, they are very fast.

In addition to fast tracking speed, our tracking system should not be expensive to be implemented. Considering the low prices of single-purpose processors these days, the price of the embedded system is not as expensive as its price in previous decades.

As this project may be further applied and developed by other researchers in the Nanorobotics Laboratory, our design should be reusable. Thus, our tracking system should at least have the following specifications: fast enough, reliable, not expensive, and reusable.

1.2 Tracking bacteria

The subject of tracking in this project is a bacterium. Bacteria are unicellular microorganisms. The organelle member of the bacteria is named flagella, and its rotary activity provides propulsion to move them. The bacteria move in a pattern which consists of periodic runs and stops (tumbles) such that after running, they stop for a moment and then continue their movement in a random direction. Different stimuli like chemical, osmotic, and aerobic affect the frequency of tumble. Among different bacteria types, the ones used in this project are Magnetospirillum gryphiswaldense magnetotactic bacteria (MTB). These bacteria are appropriate to carry out computer-based controlled micromanipulations of micro-objects [10]. This MTB has a length of about 1-3 μm in diameter with a swimming speed between 40 and 80 μm/s [10]. Swimming speeds as fast as 300μm/s have also been recorded for MTB of type MC-1 cultivated in the NanoRobotics Laboratory of Computer and Software Engineering of École Polytechnique de Montréal [10], [11], [12].

In addition to chemotaxis and aerotaxis factors, the motion behavior of MTB is more influenced by magnetotaxis. A chain of magnetosomes inside MTB bodies causes the MTB to be categorized in magnetotaxis type. Therefore, directional control of MTB is achieved by inducing a magnetic field on this chain of magnetosomes, so that the magnetosomes cause the bacteria to move according to the direction of the magnetic field. Although a magnetic field with a magnitude 0.5 Gauss affects the direction of MTB, a higher magnetic field of 3 Gauss gives highly predictive responses.

Micromanipulation of MTB is feasible by reversing the orientation of the external magnetic field. Generally, by changing the angle between the inducing magnetic field and swimming microchannels, the MTB reorient in a corresponding direction. The microfabrication of microchannels is explained in [12].

According to their behavior in a magnetic field, MTB can be classified as polar or axial. The MC-1 bacterium used in the Microsystems in [12] is a North-seeking polar MTB.

MTB are also capable to push microbeads or nanoarticles. Some techniques are proposed in [10] and [11] to attach micro/nanobeads to MTB. MTB are able to push 3 μm beads at an average velocity of 7.5 μm/s under a magnetic field of at least 0.5 Gauss [10] which means a thrust of 0.5 pN. However, a maximum thrust of above 4 pN per MTB has been recorded by the NanoRobotics group [10], [11], [12].

1.3 Thesis outline

This thesis consists of 5 chapters which are described briefly as follows. Chapter 1 reviews basic tracking concepts. It continues with the general problem in all embedded systems, it then explains the problem on which we have focused in the thesis. We finally give a brief introduction about the MTB bacteria.

Chapter 2 surveys some previous works about tracking. Some tracking algorithms will be introduced; furthermore, we will concentrate on cell tracking algorithms. Then, the results of simulation of these tracking algorithms on a recorded video sequence will be shown. The simulations are performed in Matlab programming language and the results will be discussed.

Chapter 3 begins with the architecture of the video starter kit which is used in this project. The main board and the daughter board of the video starter kit are described thoroughly. Then, the hardware of the project is implemented by using System Generator in Matlab software. We show how to design a processing core and how to export it in Xilinx Platform Studio in order to link it to a MicroBlaze soft processor. We develop the necessary programs which are executed by MicroBlaze processor. These programs are written in the C language. The design is finally downloaded on the hardware and the result is depicted.

In chapter 4, we explain how we can improve the design proposed in the previous chapter. We will explain the suitable peripheral and the modifications that must be made to it. Our new approach to implement the tracking algorithm has many challenges. We will introduce a new processing core which is later modified in the VHDL programming language. The most important challenge, synthesizability, is explained. We have to apply some components from the Xilinx Virtex-4 library in order to make the design synthesizable. Therefore, the Virtex-4 library is introduced and its application is explained.

Chapter 5 discusses the results. We will see how our proposed solution provides flexibility for video processing algorithm developers. We explain the limitations of the work. The future works which can be considered as development of the system are indicated. Some applications of this project which may be useful in industry are also discussed.

CHAPTER 2 LITERATURE REVIEW

Thousands of publications in the form of books, scientific articles, and journal papers are available focusing on the video processing subject. Many conferences and professional reunions are held every year discussing different aspects, problems, and solutions of video processing. Amongst them, tracking has always been an interesting subject which attracts students, professors and researchers. The majority of video processing works consider tracking as the main subject of discussion. The tracking algorithms are interesting to researchers due to their various applications, some of which was explained in previous chapter.

Trucco and Plakas [13] review different algorithms of video tracking and focus their article on subsea tracking which is a special case of video tracking. They concisely introduce some techniques which are applied and published in many articles. This article can be a thorough reference for the researchers who study different methods of tracking. Trucco and Plakas divide the tracking problem in two general parts which are the *motion problem* and the *matching problem*.

2.1 Motion problem

In the motion problem, the algorithm predicts some regions of the next frame where the target object may be found with a higher probability than other regions in the frame. Two solutions to define these regions will be proposed. The first solution is adjacent regions and the second one is Kalman filter. The adjacent regions method is easier to implement and needs fewer calculations than Kalman filter, while Kalman filter is more precise and can be used not only in tracking systems but in many control systems in order to predict the next state of the system.

2.1.1 Adjacent regions method

The adjacent regions method defines that the regions in which the moving object is located should not be very far from the location of the target object in previous frame, since the object does not have much displacement in the new frame compared to the previous one. Thus, the solution to this problem defines a search area around the target in the next frame. The size of the search area is variable, however it can be chosen a bit larger than the size of the target.

2.1.2 Kalman filter method

A better and more reliable solution than the adjacent regions method is the Kalman filter method. Kalman filter is an optimal recursive data processing algorithm which is used to estimate the state of a dynamic system.

Any physical system like an aircraft, a chemical process, or the national economy can be defined by a mathematical model. This model represents the behavior of that system and is defined by functions that accept the inputs and produce the pertinent outputs. The measurement devices are introduced to observe the output signals of the system.

All systems cannot be implemented by deterministic models and control theories. The three reasons that such models and theories are not sufficient for analysis and design of all systems are as follows.

First, no mathematical system model is perfect. For example, to show a curve that a vehicle performs while bending, we do not need all points in order to model the curve. Information about the behavior of the system, the inputs and the outputs removes the need to know all the details of the system to model it. Thus, some critical points and system responses are enough to model each system. When we study the parameters in modeling the systems, we find that the parameters are not necessarily precisely used in mathematical models. For example, when we want to suppose that something travels at the speed of light, there may be differences between the real speed of that object and the speed of light. Therefore, we make approximations to the speeds near the speed of light. As a result, there is often much uncertainty in system mathematical models.

Second, there are some factors other than inputs to the system that cause disturbances. For example, the wind affects the exact angular orientation that the pilot has commanded to the aircraft. These disturbances can neither be controlled nor modeled deterministically.

Third, the sensors have inherent imprecision and the exact system state and input cannot be known.

The Kalman filter is an algorithm which responds to the problem of modeling the systems which contain some extent of uncertainty, the systems which are posed to noise-corrupted data due to imprecise sensors [14]. This is an optimal, high performance algorithm which is also used in video tracking applications [15]. In a short definition, the Kalman filter is an optimal recursive

data processing algorithm. This algorithm has a high performance and is optimal for all criteria that we evaluate its performance. It considers all inputs and all factors that can affect the functionality and the status of the system in order to define the next state and the outputs of the system. We know the filters as a set of wires and electronic components. This previous description of a filter which is a black box of electrical circuits is changed to new definition of a filter. That is, a computer program which processes the data.

Figure 2.1 illustrates a typical application of the Kalman filter. In this system, the inputs are applied to the system, and the outputs are measured by the measurement devices. This information is all we know about this system to describe the functionality of the system and to define the state of the system. Sometimes, the measuring devices are not precise enough to measure the exact values of output or the noise produced by different external factor does not let the sensors provide the real values of the system output. In such cases we need a filter to provide us with the optimal estimation of the system state. The results of applying the Kalman filter contain fewer errors in comparison with the results of applying other filters for the same application. Actually, the role of the Kalman filter is to give an estimate of the system state according to the values observed on measuring devices and the previous knowledge about the system in a manner that fewer errors in the desired variables are estimated.

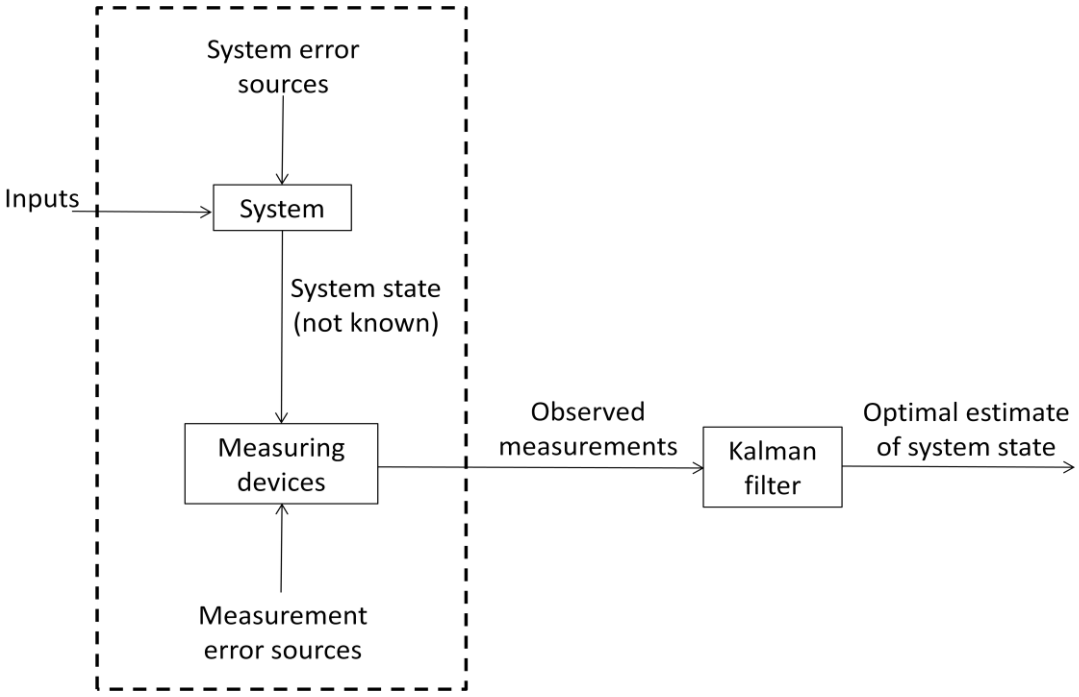


Figure 2.1 – Typical Kalman filter application (Maybeck, Peter S., *Stochastic Models, Estimation, and Control*, Vol. 1)

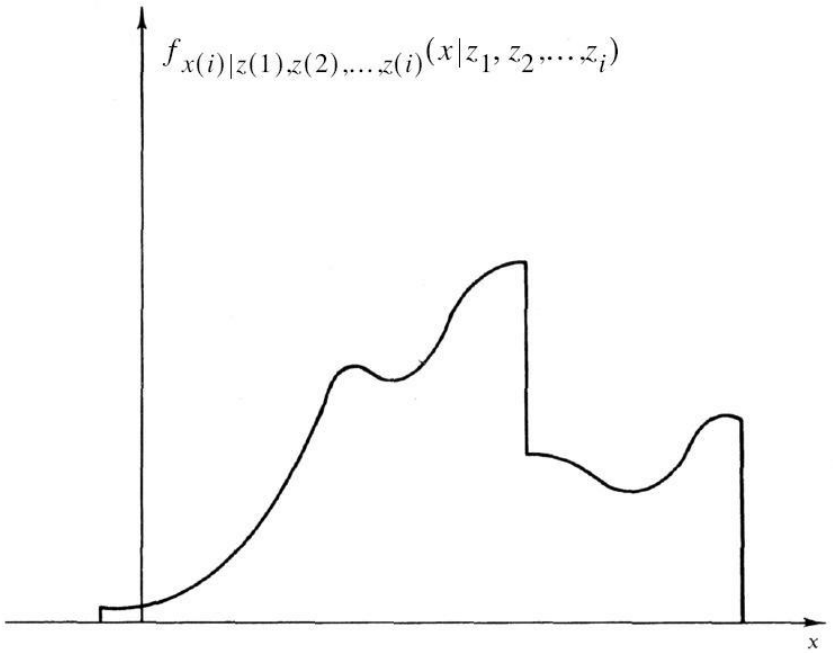


Figure 2.2 – Conditional probability density (Maybeck, Peter S., *Stochastic Models, Estimation, and Control*, Vol. 1)

A Kalman filter propagates the conditional probability density of quantities conditioned on the knowledge of the data measured from measuring devices. As an example, the conditional probability density of the quantity x at time instant $i\big(x(i)\big)$ is shown in Figure 2.2. This plot is conditioned on the knowledge that the value of the measurement $z(1)$ at time instant 1 is z_1. Respectively, we have the values z_2 to z_i for instants 2 to i, according to the function $f_{x(i)|z(1),z(2),\ldots,z(i)}(x|z_1, z_2, \ldots, z_i)$.

The plot indicates the amount of certainty of the knowledge about the measured value x. The most certainty about the value of x causes the plot to have a narrower shape. On the contrary, if we are not certain about the value of x, the plot will be wider and the weight of the plot will spread on a wider surface of the plot. If a system can be described by a linear model and the system and measurements noises are white and Gaussian, the Kalman filter applies the above conditional probability density propagation for it. For this system we will have the best estimate of the value x and the Kalman filter is the optimal filter. Thus, we can define the mean, mode, and median of the plot as three criteria for that system. The mean is the center of the probability density plot. The mode is the value x which has the highest value in the probability density plot. The median is the value x that has equal probability weight on both left and right sides.

Now we consider why a system should be linear and the measurement noises should be white and Gaussian to be modeled by a Kalman filter.

Linear systems have some characteristics which are more suitable to be modeled. First, they are powerful enough so that the nonlinear systems are not necessary; then, a nonlinear system should first be changed to a linear one which imposes some kind of noise, perturbation and error to the system. Second, linear systems are easier to work with and easier to be manipulated. The third reason is that linear systems are more complete and practical than nonlinear systems.

White noise is a noise whose values do not have any relation or dependency at different time instants. It means that knowledge of the noise value in a definite time does not help to predict the noise value in another time. White noise has equal power at all frequencies and this means that white noise has infinite power. This causes a contradiction about the power in white noise since this kind of noise cannot really exist. A system is normally driven by a noise which has power at the frequencies above the system bandpass and constant power for the frequencies within the system bandpass. The white noise is also spread a constant power level across all the frequencies both inside and outside of the system bandpass. Therefore, the white noise is identical to the real noise for all the frequencies of the system. So, we can replace the real noise of the system with a white noise.

Gaussianness deals with the amplitude of the noise and its probability density has the shape of a normal bell-shaped curve. Since the noises are produced from small sources, and considering the fact that the sum of a number of independent variables can be described by a Gaussian probability density, we can describe the noises with a Gaussian probability density. The first and second order statistics determine a Gaussian density while other densities need a large number of orders of statistics to define their shape. The Gaussian density is useful when the order statistics are higher than the first and second order statistics (mean and variance or standard deviation). Therefore, the Kalman filter which covers the first and the second order statistics contains all the data about the conditional probability density. In other words, the use of Kalman filter guaranties the inclusion of all information of the conditional probability density.

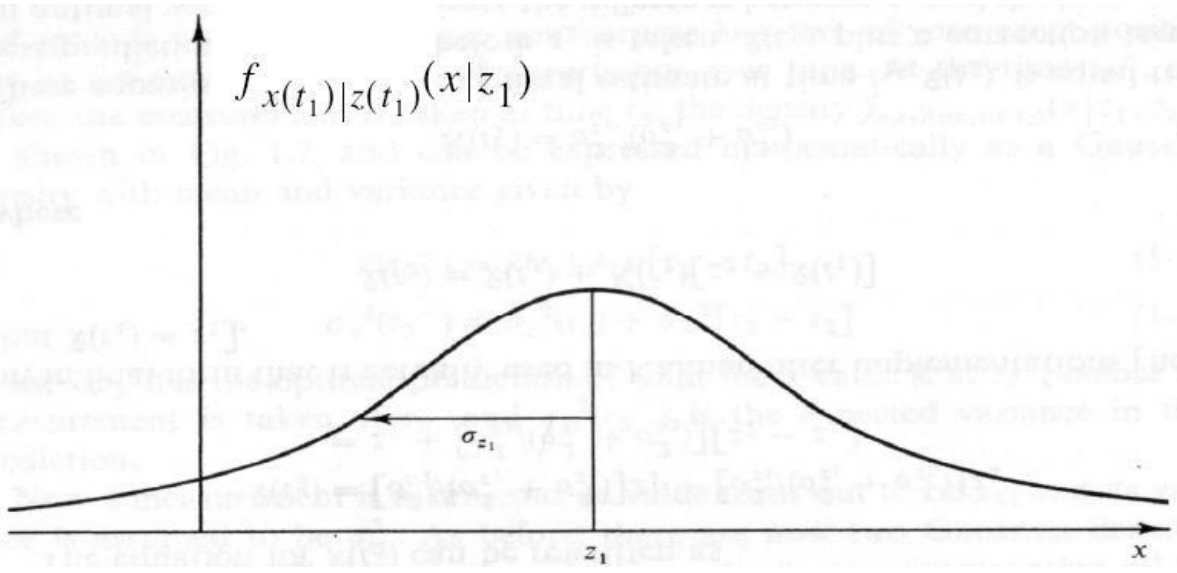


Figure 2.3 – Conditional density of position based on measured value z_1 (Maybeck, Peter S., *Stochastic Models, Estimation, and Control*, Vol. 1)

The three conditions which should be respected for a system to be modeled by Kalman filter are necessary to develop a system; while, if we only want to design a good descriptive model, it is not necessary to limit the design to linear models with white or Gaussian noise.

An example of the Kalman filter makes its design clearer. Any data read from a measuring device can be as an example for this subject, like estimation of the position of an object. Consider you are lost in the sea during the night and you decide to find your location with the aid of a star sighting. We suppose a one-dimensional location for simplicity. At time t_1 you estimate to be at the location z_1. Obviously, every person has errors while measuring and your measurement is uncertain; then, your precision is such that the standard deviation (σ value) of your measurement is σ_{z_1} and the variance or second order statistic is $\sigma_{z_1}^2$. Therefore, you can plot the conditional probability of $x(t_1)$ conditioned on your measurement z_1. The value $x(t_1)$ is your position at time t_1. Figure 2.3 shows the plot of the probability of being in any location based on the measurement you took. This plot is a function of the location x and is shown as $f_{x(t_1)|z(t_1)}(x|z_1)$. As mentioned earlier, the more precise the measurement and the more certain you are about your estimated position, the narrower the peak of the plot because of the smaller variance; thus, a large amount of weight of the plot will be concentrated in a narrow band of x.

Then, based on the above conditional probability density, the best estimate of the position is

$$\hat{x}(t_1) = z_1 \qquad (2-1)$$

According to the definition of the mode, the $\hat{x}$ is the mode (peak) of the above plot. The $\hat{x}$ is also the median of the plot (Median is the value with ½ of the probability weight to each side). It is also the mean (center of mass) of the plot. The variance of the error in the estimate is

$$\sigma_x^2(t_1) = \sigma_{z_1}^2 \qquad (2-2)$$

Now suppose that an estimator which is more precise than you estimates your position right after you at time $t_2 \cong t_1$. The measured position is z_2 with variance σ_{z_2}. As shown in Figure 2.4, since this estimator is more precise, its conditional density of the position has a narrower peak due to smaller variance.

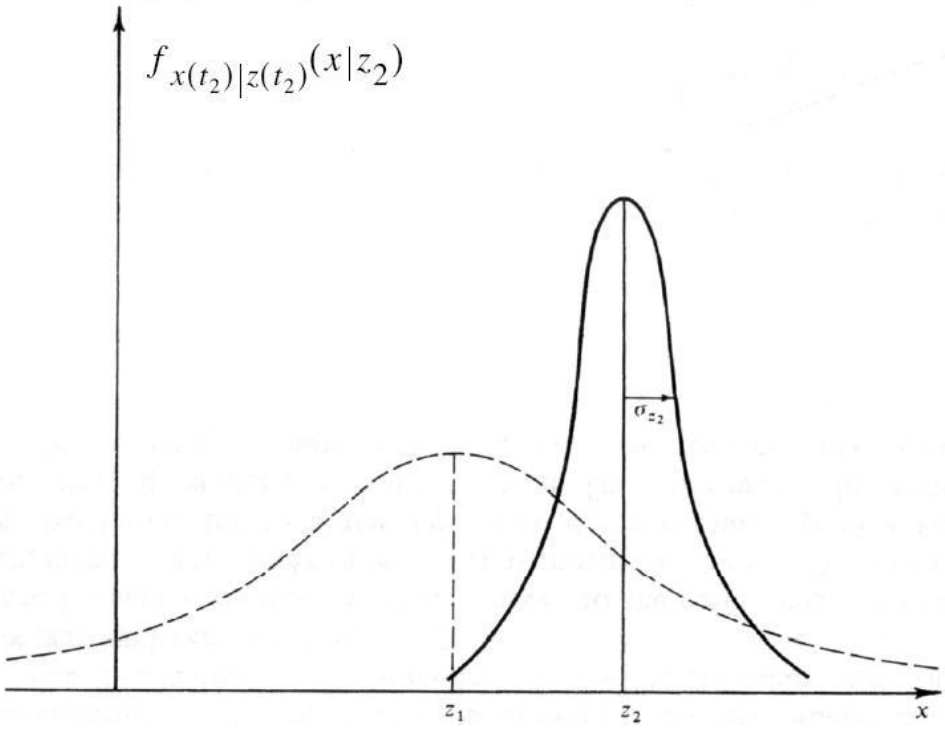


Figure 2.4 – Conditional density of position based on measured value z_2 (Maybeck, Peter S., *Stochastic Models, Estimation, and Control*, Vol. 1)

Now we have two measurements for your current position according to two different measuring devices. In order to decide which one of these estimations are used as the conditional density of your position, the Gaussian density gives the best assumption with mean μ and variance σ^2 as the following formulae:

$$\mu = \left[\sigma_{z_2}^2/(\sigma_{z_1}^2 + \sigma_{z_2}^2)\right]z_1 + \left[\sigma_{z_1}^2/(\sigma_{z_1}^2 + \sigma_{z_2}^2)\right]z_2 \qquad (2-3)$$

$$1/\sigma^2 = (1/\sigma_{z_1}^2) + (1/\sigma_{z_2}^2) \quad (2-4)$$

In (2 – 4), σ is less than either σ_{z_2} or σ_{z_2} which means the uncertainty of the position is less than the two previous measurements. To illustrate equations (2 – 3) and (2 – 4), consider if the precision of the measurements were equal ($\sigma_{z_1} = \sigma_{z_2}$), the optimal estimation of the position would be the average of the two measurements $\mu = \left(\frac{1}{2}\right) z_1 + \left(\frac{1}{2}\right) z_2$.

Then, with the above density, the best estimate of the position is:

$$\hat{x}(t_2) = \mu \quad (2-5)$$

$$\hat{x}(t_2) = \left[\sigma_{z_2}^2/\left(\sigma_{z_1}^2 + \sigma_{z_2}^2\right)\right] z_1 + \left[\sigma_{z_1}^2/\left(\sigma_{z_1}^2 + \sigma_{z_2}^2\right)\right] z_2$$

$$= z_1 + \left[\sigma_{z_1}^2/\left(\sigma_{z_1}^2 + \sigma_{z_2}^2\right)\right][z_2 - z_1] \quad (2-6)$$

The final form of the Kalman filter is obtained by replacing z_1 with $\hat{x}(t_1)$ and the correction term of an optimal weighting value $\sigma_{z_1}^2/\left(\sigma_{z_1}^2 + \sigma_{z_2}^2\right)$ with $K(t_2)$:

$$\hat{x}(t_2) = \hat{x}(t_1) + K(t_2)[z_2 - \hat{x}(t_1)] \quad (2-7)$$

$$K(t_2) = \sigma_{z_1}^2/(\sigma_{z_1}^2 + \sigma_{z_2}^2) \quad (2-8)$$

According to the (2 – 7), the optimal estimate of the position at time t_2, $\hat{x}(t_2)$, is equal to the best prediction of the position at previous time, $\hat{x}(t_1)$, plus the correction term $K(t_2)$ times the difference between z_2 and $\hat{x}(t_1)$. This algorithm predicts the value of the variables at the next state of the system based on the information of the current state of the system. Then, the correction value corrects the prediction which was made. Therefore, this algorithm is called a "predictor-corrector".

As an application example of Kalman filter, A. Ali and S. M. Mirza describe an application of the Kalman filter in [16]. This tracking algorithm looks for the target of interest (TOI) in a limited region of the video frame which is called the region of interest (ROI). Then, the previous position of the object is replaced with the current position in the current frame in the next step of the algorithm, the template-updating stage. The ROI comes from the assumption that the location of an object is not far from its location in the previous frame. Therefore, the ROI is in the surroundings and in a place near of the object in the previous frame. The algorithm first starts with the correlation method. In this method, if the correlation is greater than a threshold value,

the template is detected, the object replaces its previous position, and the template is updated with the new target. The authors did not mention the threshold value or further details about the correlation method.

There are some cases like the change in shape or orientation of the target in which the correlation value decreases significantly under the threshold value. The template cannot obviously be updated in this case. Therefore, the correlation does not work and another technique is needed for tracking. To solve this problem, the Kalman filter is used. The Kalman filter estimates the next state of the system according to the measurements which are taken from the template updating stage of the system. When the predicted location is much different from the measurement and the correlation is less than the threshold value, the Kalman prediction is considered as the next state of the object.

In some cases, neither the Kalman prediction nor the correlation give the correct position of the object. In situations like a cluttered scene, the measurements provided by the detector are noisy and the Kalman predictor is unable to give a correct estimation of the location of the object. In this case, the correlation value may be larger than the threshold, but this value is unreliable and another solution should be considered for the tracking.

The authors modify the algorithm by applying the mean shift algorithm [16]. Mean shift is an algorithm which tracks the object based on the probability distribution of the color of the ROI in an iterative procedure. This algorithm calculates the location offset from the previous position toward the region with the highest similarity in the probability distribution of the ROI color.

Then, if the difference between the estimated location of the target by Kalman prediction and the correlation is greater than a fixed value, the algorithm uses the mean shift algorithm to locate the next position of the object.

2.2 Matching problem

As mentioned in section 2.1, the motion problem introduces some regions where the target object may be found. Now, in the matching problem, the algorithm finds the target in the next frame between the candidate regions which were identified in the previous frame.

In this section some algorithms which focus on the matching problem are proposed. There are hundreds of publications about algorithms which concern matching problem. These algorithms

are categorized according to the complexity of the target. In other words, some algorithms search the moving object in predicted windows by a simple threshold. More complex algorithms look for simple specifications of moving object, for example its corners [17], edges [18] or lines [19]. Some algorithms consider the moving object by its parts as some two-dimensional shapes [20]. They look for shapes such as circles, rectangles and triangles in the moving object. Some algorithms as well try to find three-dimensional shapes in the scene [21]. They look for shapes such as cylinders. Finally, we may define complex forms and search these forms in the scene. For instance, an algorithm searches the objects which look like a chair, a car or a human in the next frame [22]. The methods based on learning the shape and dynamics of non-rigid targets are more complicated. We will get familiar with some basics of the first matching problem solution in the next section. However window tracking is a simple solution for the matching problem, it is widely used not only in tracking problems, but also in other fields of computer vision.

2.2.1 Window tracking

In addition to tracking the moving object, window tracking techniques are applied in other domains of computer vision like stereo reconstruction and optical flow. These techniques include Sum of Squared Differences (SSD), Sum of Absolute Differences (SAD), and Normalized Cross Correlation (NCC).

N. Mekuz et al. [23] describe the SSD method as a window tracking algorithm which does the matching based on similarity between images. The smaller the value of SSD, the most similar are two images. In this approach, a chosen window should be slid on the search image and the SSD is calculated between the image piece specified by this window and the target window in each step. If the SSD is less than a specific threshold, the similarity is acceptable and a match is detected. We suppose the low and high limits of the intensity of the pixels are I_L and I_H , respectively. The algorithm looks for the target object, T, of size $m \times n$ in a search image, S, of size $M \times N$. The SSD between two images P and Q is

$$SSD(P,Q) = \sum_i \sum_j (P_{i,j} - Q_{i,j})^2 \qquad (2-9)$$

As depicted in Figure 2.5, the window moves slowly on the search image, S; when the highest similarity between the target window, T, and this chosen window is detected, a match is found. The exact difference of pixels between the window and the target window depends on the search

image but the maximum difference between the pixels at each step of shifting the window on the search image can be computed. This value depends on the target window and is independent of the search image.

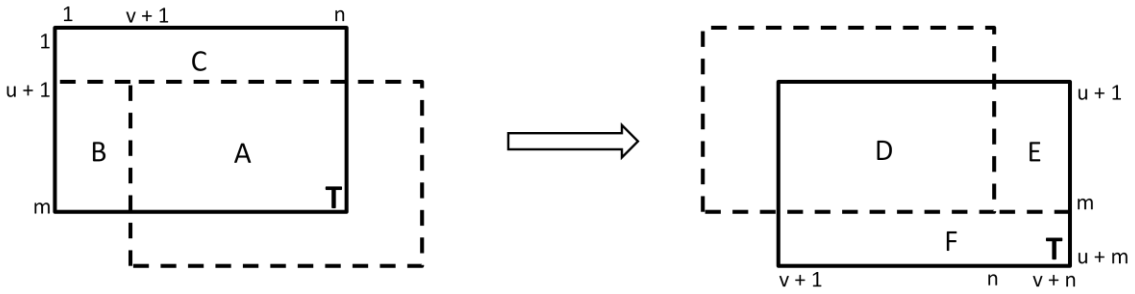


Figure 2.5 – A subdivision of the two images to be compared, to facilitate the analysis

During the displacement of the window from $(1,1)$ to $(u+1, v+1)$, the change in SSD score is defined by $\Delta_{u,v} = d_{1,1} - d_{u+1,v+1}$. The SSD in $(1,1)$ and $(u+1, v+1)$ is

$$d_{1,1} = A + B + C \tag{2-10}$$

$$d_{u,v} = D + E + F \tag{2-11}$$

Regions A and D are the common regions between the moving window and the search image. Before we start comparing the moving window and the image, the upper region which is not common is called C and the left region which is not common is called B. The number of pixels which belong to the C region (uncommon pixels) is u and the number of pixels which only belong to B region is v. While shifting the position of the moving window, as soon as the moving window exceeds the bounds of the search region, the right region is called E and the lower region is called F. Again, u and v are the number of pixels which respectively belong to E and F and do not belong to D.

The supremum of $\Delta_{u,v}$ is the preferred value to be compared to the threshold:

$$\sup(\Delta_{u,v}) = \sup(d_{1,1}) - \inf(d_{u,v}) = \sup(A - D) + \sup(B) + \sup(C) - \inf(E) - \inf(F) \tag{2-12}$$

The abbreviations sup and inf in (2 – 12) indicate the supremum and the infimum values of the variables, respectively. The supremum of some values is the least upper value among them. In the same way, infimum is the greatest lower value among some values.

Since I_L and I_H are the low and high intensity values, $T_{i,j}$ and $S_{i,j}$ carry the values between these two bounds. Thus, $I_L \leq T_{i,j} \leq I_H$ and $I_L \leq S_{i,j} \leq I_H$. Therefore,

$$\inf(E) = \inf(F) = 0.$$

$$\sup(B) = \sum_{i=u+1}^{m} \sum_{j=1}^{v} \max \left\{ \left(I_H - T_{i,j}\right)^2, \left(I_L - T_{i,j}\right)^2 \right\} \quad (2-13)$$

$$\sup(C) = \sum_{i=1}^{u} \sum_{j=1}^{n} \max \left\{ \left(I_H - T_{i,j}\right)^2, \left(I_L - T_{i,j}\right)^2 \right\} \quad (2-14)$$

$$\sup(A - D) = \sum_{\substack{i=u+1 \ldots m \\ j=v+1 \ldots n \\ T_{i,j} \geq T_{i-u,j-v}}} \left(T_{i,j} - T_{i-u,j-v}\right)\left(T_{i,j} + T_{i-u,j-v} - 2I_L\right)$$

$$+ \sum_{\substack{i=u+1 \ldots m \\ j=v+1 \ldots n \\ T_{i,j} \leq T_{i-u,j-v}}} \left(T_{i,j} - T_{i-u,j-v}\right)\left(T_{i,j} + T_{i-u,j-v} - 2I_H\right) \quad (2-15)$$

Since the value of $\sup(\Delta_{u,v})$ is independent from the search image, it can be obtained offline for any target image. Then, this value can be compared to any search window and if it is less than a specific threshold, a match between two image subdivisions occurs.

The complexity of this algorithm for a target size of $m \times n$ in a search image size of $M \times N$, is $O((M-m)(N-n)mn)$.

As mentioned earlier, there are many methods for solving the matching problem. Other methods are planar rigid shapes, solid rigid objects, tracking deformable contours, and visual learning. Since the MTB bacteria which are applied in this project have simple round shapes and they look like white moving objects on a black color background, the solutions which were covered in previous sections are adequate to develop robust and efficient tracking algorithms. These solutions are very simple to be developed. Besides, they consume few hardware resources and require few lines of software code for implementation.

2.3 Cell tracking

The previous sections introduced two problems an algorithm developer encounters while tracking moving objects. We studied techniques which are useful in solving these problems. In this section, we study in more detail the tracking of moving objects which are more related to subject of tracking in this project, that is cell tracking. Cell tracking is widely used in medical and

biomedical applications. The tracking target can be any kind of microorganisms such as bacterium or any microparticle or nanoparticle. We generally address these various kinds of particles as cells, since they are uni-cell bodies. The cells usually have simple round shapes. Depending on the imaging technique, they may be either black spots on a white background or white objects on a black background. Therefore, cell tracking applies specific approaches. We introduce some of them in this chapter. The simulations explained in chapter 5 are based on these algorithms.

Wang et al. [24] introduce a tracking system to track a freely swimming microorganism. This system tracks a *Chlamydomonas reinhardtii* (CR) cell in two dimensions and keeps the cell in the center of the microscope field of view. The position, velocity, and orientation of the cell are determined for at least 300 seconds under two imaging modes of bright-field (BF) imaging and phase contrast imaging. Phase contrast imaging is applied for examining the cells or bacteria which are very difficult to be seen in an ordinary microscope. Phase contrast is a kind of light that enhances contrasts of transparent and colorless objects.

The field of view indicates the region that can be seen through the microscope. We mentioned earlier that the tracking algorithm functions by studying the object position in two consecutive frames. Therefore, if the tracking object leaves the field of view in a certain frame, the algorithm will be unable to continue following the trajectory of the object. Then, if the object returns into the field of view of the microscope, it cannot be distinguished as the tracking object.

On the other hand, during the calculation of the location of the object, the pixels of a frame are indexed by two coordinates, for example x and y. The coordinates x and y are allocated the positive values starting from either zero or one. The pixels belonging to the tracking object are also indexed with x and y coordinates. So, if the object leaves the field of view of the microscope, the coordinate values will hold invalid values. These values may be greater than the size of the frame or may be negative. These invalid values produce errors in the tracking algorithm. Thus, the field of view and its size is very important in tracking algorithm.

In order to start the system, the largest cell in the first image is tracked or a cell is manually selected to be tracked. The vision system sends recent positions of the cell to a computer. The computer calculates the new position of the stage of the microscope according to the algorithm.

Then, it moves the stage in two dimensions by activating the XY servo motors. A DSP controller is in charge of activating the servo motors.

The hardware used in this experiment consists of a CMOS digital camera (Basler 602fc), an upright optical microscope (COIC XSZ-HS3), and an image capturing card (Matrox Meteor-II/1394). The DSP motion controller (TMS320F2812) is able to make movements at a resolution of 0.25 μm. A microfluidic chamber is designed to limit the cell to move in two dimensions. A cavity which is 30 μm in depth and 1 cm in diameter is created in a Poly slide. This slide is covered by a glass slide at the bottom and a cover slip on top.

Wang et al. have used a simple threshold method to identify the cells. In the BF case, since the cells are black and the background is white in the video, there is a large difference between objects and background intensity. Therefore, the system can easily distinguish between objects and the background by applying a threshold. Wang et al. have not mentioned the value of the threshold in their article. In the phase contrast case, the cell is surrounded with bright spots. Because a single threshold does not detect moving objects in the image, the authors use a multi threshold approach; however, they do not explain how they applied this technique.

In order to track the cell in the BF case, the algorithm considers a small window around the target cell. The window should be centered on the cell's centroid. The size of the window is made as small as possible while the target should remain in the window area in the next frame. In the next frame, the tracking system compares the threshold with the pixels of the search window to find the new coordinates of the cell. The authors have mentioned neither an estimation of cell size in pixels in each image, nor the window size they considered in this algorithm.

The computer calculates the new location (X, Y) of the cell for the controller according to equation (2 – 16).

$$X = p\left(x - \frac{m_x}{2}\right) + X_S \qquad Y = -p\left(y - \frac{m_y}{2}\right) + Y_S \ , \qquad (2-16)$$

where p is the length of each pixel in micrometers and m_x and m_y are the number of horizontal and vertical pixels of the image, respectively. (x, y) is the target centroid and (X_s, Y_s) is the displacement of the XY stage. This is shown in figure 2.6.

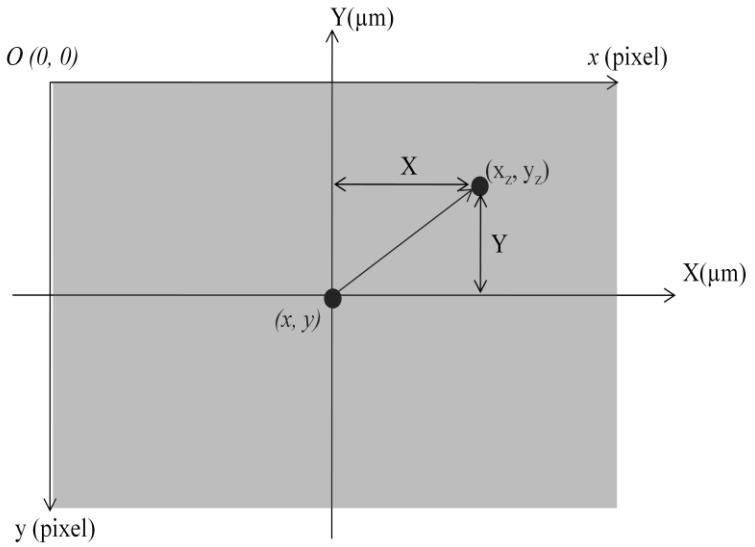


Figure 2.6 – Example to illustrate the calculation of the location of microscope stage

The units of the size of each pixel (p) and the displacement of XY stage (X_s, Y_s) should be the same and this unit will define the unit of the new location of the cell (X, Y).

The computer repeatedly calculates the new cell position captured by the vision system and compares it with previous coordinates. Then, it sends the difference value to DSP controller to move the XY stage in the opposite direction.

The system also calculates the velocity and orientation of the moving target in each frame according to the equations (2 – 17). These values are recorded in the computer and are not used in the tracking algorithm.

$$V_C = \frac{\sqrt{(x_n - x_{n-1})^2 + (y_n - y_{n-1})^2}}{t} \qquad \theta = \arctan \frac{y_n - y_{n-1}}{x_n - x_{n-1}} \qquad (2-17)$$

In (2 – 17), (x_n, y_n) and (x_{n-1}, y_{n-1}) are the coordinates of a cell in two consecutive frames and t is the time interval between these two frames.

This system is able to track the CR cell for more than 300 seconds at a magnification of 40×. The CR cells are spherical in shape and 10 μm in diameter with a velocity of about 150 μm/s. The system can track the target up to a velocity of 900 μm/s, but the cell would be lost at a speed of 1000 μm/s.

Oku et al. [25] describe the tracking of a cell at magnifications of 5× and 20×. For detailed observation of the target and high resolution images, the tracking system needs a vision system

which captures images at a high frequency. Therefore, an I-CPV system which is a high-speed vision system was mounted on the microscope. A personal computer executes the tracking algorithm and sends the relevant instructions to an XY automated stage to keep the microorganism in the microscope field of view.

The I-CPV high speed vision system captures eight-bit gray-scale images with 128 × 128 pixels at 1 KHz frame rate. The LAL00-X70 is the XY automated stage which was used in the project. This stage is able to move at the resolution of 1 µm. BX50WI (OPLYMPUS) was used as the optical microscope. This microscope supports different imaging methods like dark-field (DF) illumination and differential interference contrast (DIC). It also supports two ports for CCD cameras and CPV cameras. A personal computer with a Celeron 1.8 GHz processor and ART-Linux real-time operating system executes the tracking algorithm.

In order to limit the movements of cell, a 15 mm × 15 mm × 150 µm chamber was designed. Therefore, the cell is not able to move along the optical axis of the microscope and its movement is limited to two dimensions.

Oku et al. propose tracking methods based on different imaging methods: bright-field (BF) illumination, dark-field (DF) illumination, and differential interference contrast (DIC). DIC, like phase contrast, is another method of optical microscopy illumination. This technique is used to enhance the contrast in transparent objects. DIC produces an image with black to white objects on a gray background.

In the first step, the area of objects should be detected. Since in the first and second methods there is a high contrast between target and background - black cells on a white background in BF and white cells on a black background in DF - the cells can be identified with a single threshold. The thresholds which the authors have applied are not mentioned in the article.

In the DIC method, again a single threshold is used to distinguish a cell from the background. In this method the background is gray and the parts of the objects whose spatial phase distribution are not constant become white or black. Thus, if the absolute difference between the pixel's intensity and background is more than a given threshold, that pixel belongs to the object.

The second step determines which object is the target. The binarized version of the previous image captured by the vision system is used as a mask. This mask should then be dilated. The dilation helps the foreground pixels to be enlarged. Therefore, the size of the areas of foreground pixels grows while the holes within those regions become smaller. The AND operation between this dilated version and the binarized version of the current image gives a new image. Since a cell has a small movement in two consecutive frames, the new image contains only the target cell. Indeed, the target should overlap in two consecutive images. Therefore, this method works when the target moves very little in two consecutive frames and the frame rate is high.

The authors have also developed an algorithm to detect collision and division, but they have not described their method in this article.

The system is able to track the target for at least 60 seconds by a high image resolution when the velocity of the cell is less than 1000 μm/s, equivalent to 50 diameters/s. The velocity of 50 diameters/s corresponds to 324 km/h on the human scale for a human body of 1.8 meters in length. The system can also track the cell at a speed of 25000 μm/s which corresponds to 125 diameters/s; however, the center of the cell is near the boundary of the field. At a velocity of 40000 μm/s the system is unable to track the cell.

Taboada et al. [26] propose a tracking system which is based on finding the center of mass of the bacteria. This system first detects all the bacteria in the frame. Then, it constructs the bacteria trajectories by following their center of mass in consecutive images. The tracking system also detects overlapping cells and it presents a method to distinguish the tracking object between two overlapping cells.

An SIT video camera (MTI) captures the images through a Nikon E-600 microscope in a dark field method. A video recorder records several minutes of swimming cells through a Pinnacle Micro DC30 Pro video capture card. Then, the tracking algorithm is implemented on a selected sequence. The sequence contains 60 frames at a rate of 30 frames per second within 2 seconds. The tracking program is executed on a PC with a Pentium IV processor and Windows operating system.

First, since the color in the tracking is not important, the images are converted to black and white. A flatten filter is applied to even out background variations. This filter reduces noise by applying low pass filters. A Gaussian filter is also applied to reduce noise and to enhance the image. As a

result of this procedure, all images consist of white objects corresponding to the cell bodies of bacteria on the dark background. The recognition of white objects from black background is obtained by applying a simple threshold whose value is not indicated in the article. This procedure is applied on the images to detect the bacteria. In this system, the bacterium size is also considered; therefore, if a detected object is larger or smaller than average size of bacterium, it is automatically ignored.

To find the bacteria trajectories, the current frame is binarized to result in white cells on black background and the center of mass of each bacterium is calculated and recorded as the current location of each cell. The next image is also binarized. Then, these two consecutive binary images are averaged to give a new image. As a result, the pixels whose value is 255 belong to cell pixels in both images while the pixels whose value is 127 belong to non-overlapped pixels of the cells. Then, only overlapped bacteria are segmented by applying a threshold which is greater than 127. Then, a logical AND operation between this image and the binary image in the time t + 1 is applied. Therefore, only the white pixels in both of these images are detected. The center of mass in this image is recorded as the new position of the bacterium at time t + 1. By keeping the center of mass, the system can keep the coordinates (x, y) of the center of mass of the cell which constitute the trajectory of the bacterium.

The tracking algorithm proposed in the article is not able to function properly in special cases. First, the displacement of the cell in two consecutive images should not be more than the diameter of the bacterium. Since this algorithm is based on the overlapping of the bacterium in two consecutive images, the trajectory of the bacterium will be lost if the displacement of the bacterium is more than the diameter of the bacterium. Second, the bacterium should not leave the field of view of the microscope for at least 2 seconds.

Prasad et al. [27] present a solution for real-time multi object tracking. This algorithm is based on a hierarchical adaptive merge split mesh-based technique (HAMSM). HAMSM makes the mesh topology from both fine-to-coarse and coarse-to-fine techniques, while hierarchical adaptive structured mesh (HASM) is a coarse-to-fine technique.

The segmentation process identifies the presence of cell motion and the tracking process specifies the characteristics of the motion.

In the segmentation process, a frame is divided into several patches. The vertices of the patches are the mesh nodes. Then, the patches with several motion components are divided into smaller sized patches while patches without any motion components are merged to construct larger patches. Three levels of split and merge are performed in this algorithm. Assuming that each patch is 16 × 16 pixels at the first level, the final small patches are 4 × 4 pixels. In order to obtain the moving objects, the smallest patches should be merged. Therefore, all the eight neighbor patches of each small patch are compared to see if there is any connectivity. If no patch containing a moving object was found in the neighborhood, the patch is ignored. After all moving objects are detected, they are listed in a database and a rectangular boundary is defined around each of them. This segmentation stage is repeated for each frame to detect all new objects that enter or leave the microscope field of view.

For the tracking process, the motion vector should be defined. The motion vector of the mesh is estimated by generating an 8 × 8 block of pixels (macroblock) around the upper-left pixel of the mesh. The macroblock which produces the least SAD gives the motion vector with the least difference. Several SADs should be computed to find the best fit of the macroblock.

New boundary nodes are obtained from old boundary nodes added by motion vectors. These new nodes are the position of the object in the next frame.

This project concentrates on the bacterium motion in two dimensions (axes x and y). The bacterium motion in the third dimension (axis z) causes the bacterium to become larger or smaller during its motion and sometimes it causes the bacterium to disappear and go out of the field of view of the microscope. Therefore, the trajectory of the object may be lost.

Tracking a moving object in 3-D requires the application of other techniques. In other words, some more specifications of the cell motion should be considered for a constant tracking. Some publications explain the application of a second camera to detect the motion of the bacterium in z axis. The application of a second or more cameras introduce another concept in tracking which is known as stereo cameras.

Some other techniques change the zooming of the microscope to solve the problem of tracking in third dimension. Manipulation of the zooming of a microscope is not covered in this thesis.

The articles presented in this section study the cell tracking algorithms which are the subject of research in this project. The simulations of the above algorithms are done by Matlab software and the results are illustrated and discussed in chapter 5.

2.4 Proposed cell tracking algorithm

Each of the cell tracking algorithms which were described in the previous section propose solutions for the motion problem and matching problem presented in sections 2.1 and 2.2. Even if the implementation of any two tracking algorithms seems similar, they are different in details such as search window size, window of interest size, thresholds, etc. Thus, we can say each tracking algorithm has its own methodology and approach during its development and implementation.

Of course, each tracking algorithm has advantages and disadvantages. However some are powerful in one or more aspects. We tried to develop an algorithm which is efficient, reliable, and stable while low in calculations.

Our algorithm uses the adjacent regions method as a solution for motion problem. We considered a search window of 15 pixels around the target bacterium. This value is about two times the size of the bacterium in each frame. With this value, we guarantee that the bacterium will stay in the search window in the next frame. In other words, with our settings and according to the frequency of our camera (60 Hz), the movement of the bacterium is not more than its size during two consecutive frames.

In order to reduce the calculations, we convert the color video to gray scale video before processing the images. This will decrease the calculations to one third the requirements for color images. This is because for color images, we have to execute the algorithm on three channels of red, green, and blue, while our algorithm performs the calculation on just one channel which is the gray channel. Then, the algorithm uses the window tracking method as the solution for the matching problem. To be more precise, we use a threshold to segment the pixels which belong to bacterium from the background pixels. Since the bacterium looks like a round white shape and the image background is dark, the pixels which belong to bacterium can be easily distinguished by a threshold. For this project, a threshold value of 100 works very well and can recognize the bacterium pixels.

As we will explain in chapter 5, we simulated different algorithms. None of the algorithms are as simple as our algorithm and none of them can track the bacterium as fast as the suggested algorithm. The precision of all the algorithms are so high that none of them miss the trajectory of the bacterium.

We again note that although we simulated the algorithms which were proposed by other designers, we applied some extent of creativity for each algorithm, because none of the publications provided all design details to the reader. Furthermore, the circumstances under which the original algorithms and our algorithm are implemented differ in some parameters, such as the bacterium shape, microscope lighting, and color or gray imaging.

More details about the advantages and disadvantages of each tracking algorithm are provided in chapter 5 where we have provided a comparison of the algorithms in a table.

CHAPTER 3 TRACKING SYSTEM DESIGN

In this chapter we will present the hardware of the project. The chapter provides a detailed description of the circuit, its FPGAs, and the architecture of the MicroBlaze soft processor. This part is essential because the developer should have good knowledge about the hardware specifications of the parts of hardware which he decides to develop. The simulation methods come after. The simulation allows the designer to verify the functionality of the tracking system. It also allows the designer to analyze the hardware under different circumstances. Two simulation methods are explained in this chapter. We will design the components of the system. The design procedure of the object tracking module is illustrated and explained in detail. Then, it is generated and exported in order to be applied in the video processing system. The necessary software functions and documents about this module are generated. Finally, we will explain how to generate necessary files to download on the FPGAs and to start working with the ML402 video starter kit.

3.1 Hardware of the project

The hardware we have used in this project consists of an LVDS video camera, a Zeiss microscope, and the ML402 video starter kit (VSK). Some other items like cables, connectors, memory cards, etc. are mentioned during the sections in this chapter, but they are not explained in detail as they are not of high importance.

3.1.1 LVDS camera

The LVDS camera is a CMOS image sensor of the Irvine Sensors Corporation. It captures frames of 752 × 480 pixels in a frequency of 60 Hz. Its lens is manually adjustable in order to focus on near and far objects. The camera is connected to the Video Input/Output Daughter Card via a standard Cat-6 Ethernet cable. It should be noted that this interface is not compatible with the Ethernet standard and it does not support the Ethernet protocol.

3.1.2 ML402 Video Starter Kit

The ML402 is a starter kit specialized for implementing video processing applications [28]. This board is manufactured by Xilinx and supported by CMC Microsystems for academic and industrial research. The block diagram of the ML402 is depicted in Figure 3.1. The most

important part of the ML402 is a Virtex-4 Xilinx FPGA which functions as its heart. The board also contains different peripherals.

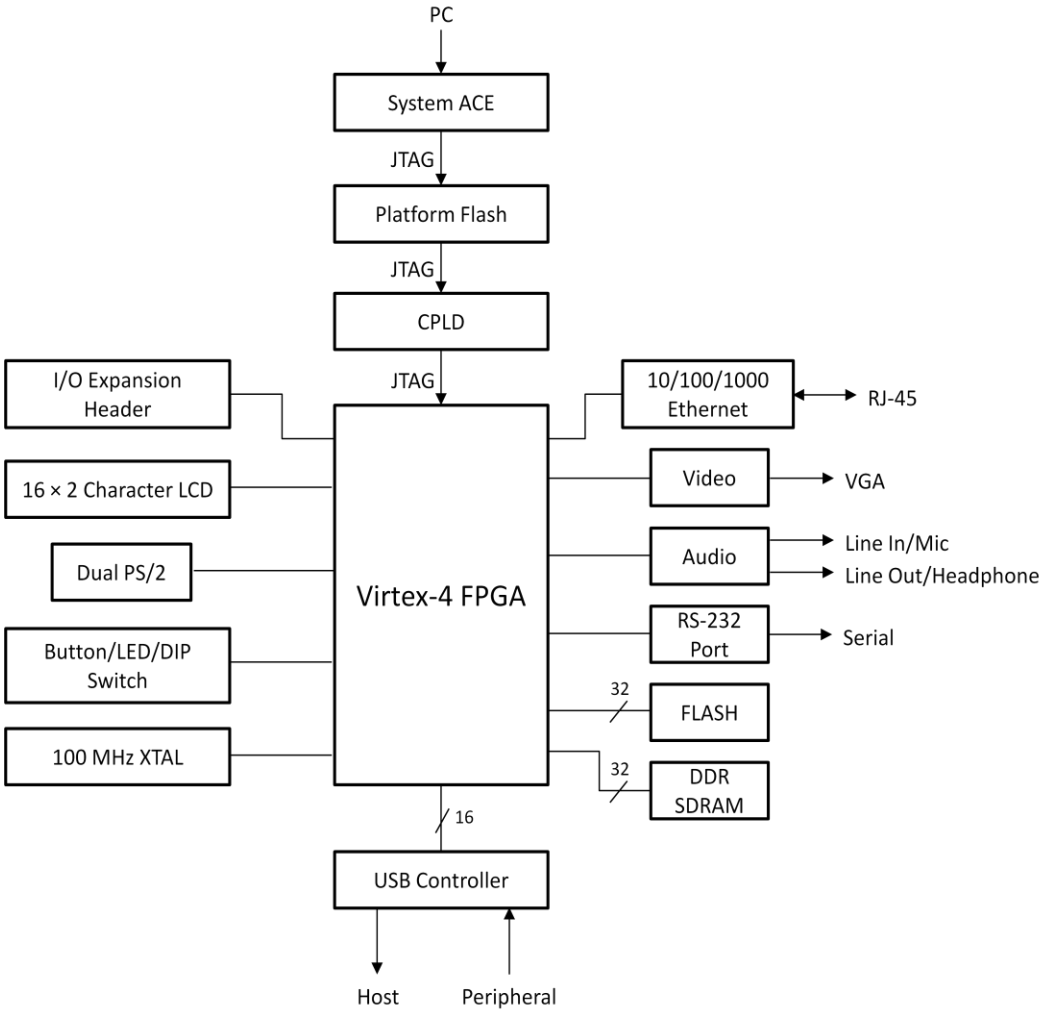


Figure 3.1 – Block diagram of ML402

3.1.2.1 RS-232 Port

The RS-232 serial port connects the ML402 to the PC through a program, like HyperTerminal. The RS-232 communicates with the modem based on a serial protocol at different bit rates. The RS-232 on the ML402 supports a highest speed of 115200 bits per second. The user can control the application program which the MicroBlaze processor executes by entering the relevant inputs through the PC. The program is able to make decisions according to the inputs of the RS-232 serial port so that the values that the user enters as the program inputs, can define the flow of the

program. Generally, the RS-232 serial port can be used as a very simple interface by which the user can interact with the VSK.

3.1.2.2 System Ace Card Interface

The System Ace Card Interface is used to read the configurations from Compact Flash memory cards. As we will show later in this chapter, the designer can copy the configuration of both PFGAs on the Compact Flash card and as soon as the board is switched on, the configuration is downloaded on the ML402 and VIODC FPGAs. Indeed, the VSK functions according to the of the ML402 and VIODC FPGAs configuration. In this case, there is no need to JTAG to download the bit files.

3.1.2.3 Gigabit Ethernet

The designer can connect the VSK and a PC via the 10/100/1 Gigabit Ethernet port to profit from high speed video simulation. In this kind of simulation, which is known as Hardware-in-the-Loop Co-Simulation, the simulation rate reaches up to 600 megabits per second.

3.1.2.4 DDR Memory

A 267 MHz 32-bit wide DDR memory is used to store video frames. As an example, in this project the VSK can be designed so that the tracking is performed on a saved video sequence; in other words, non real-time tracking. The video can thus be saved on the DDR memory. The user may also save the video after tracking. The video sequence after tracking can also be recorded on the DDR memory.

3.1.2.5 USB Ports

Two ports for USB peripherals and one USB host port are provided for high speed connection between VSK and other systems. Since VSK designers has not predicted any other output port for ML402, the USB ports can be very useful through which the developer interfaces the necessary output to the system.

3.1.2.6 I/O Expansion Header

The 64-bit I/O bus interfaces the ML402 with VIODC. The communication between the ML402 and VIODC is fulfilled through this I/O bus. The ML402 receives the video signals from VIODC through this bus and sends them back again through this bus after necessary processes. More detail about the bus is provided in section 3.1.3.10.

3.1.3 Video Input Output Daughter Card

The ML402 is interfaced with a video input output daughter card (VIODC). The VIODC is a standard video card for Xilinx platforms [29]. If the protocol of the I/O expansion header is respected, the VIODC can be interfaced to any other compatible circuit. The VIODC supports different input and output video ports such as High Definition Component video input and output, Standard Definition S-video input and output, Standard Definition Composite video input and output, Digital Video Interface (DVI) input and output, VGA analog input and output, and SDI Serial Digital video input and output. As mentioned in section 3.1.1, the daughter board also has the input interface for the LVDS camera. The LVDS camera attaches to VIODC through the provided interface. The VIODC consists of some ICs connected to a Xilinx Virtex-2 FPGA. These ICs and FPGA contain the drivers of the video ports. The daughter card is connected to ML402 via a 64-bit bus called VIOBUS. The VIOBUS will be further described in section 3.1.3.10. The block diagram of the VIODC is shown in Figure 3.2. More details about the inputs and outputs of the VIODC follow.

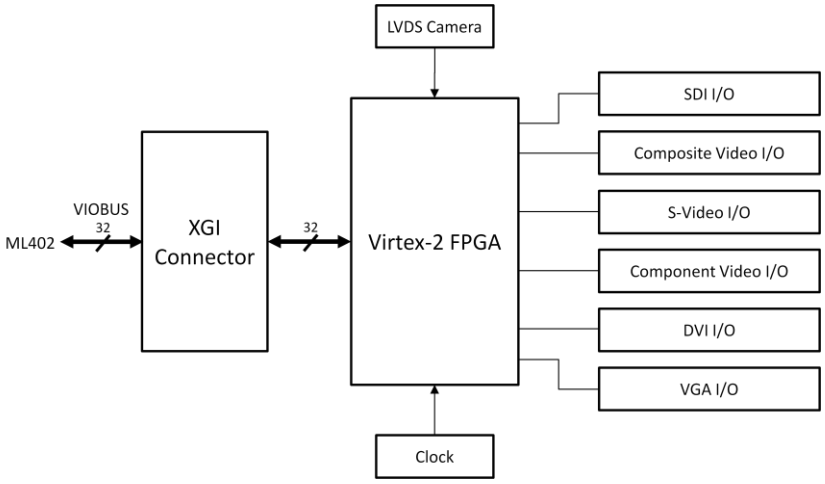


Figure 3.2 – Block diagram of VIODC

3.1.3.1 LDVS Camera Input Port

The LDVS camera input port supports the LDVS RGB camera of the Irvine Sensors. This camera contains a CMOS image sensor which captures the 752 × 480 pixel frames in the frequency of 60 Hz. The capture sensor is a kind of low noise, high quality sensor. The port interface is a standard Cat-6 Ethernet, but it should be noted this port is not compatible with Ethernet.

3.1.3.2 S-Video and Composite Video

The S-Video and Composite Video input and output interface is able to support the NTSC, PAL, and other standard definition (SD) formats. The interface consists of encoder integrated circuit (IC), decoder IC, and analog filters.

3.1.3.3 Component Video

Component Video supports High Definition (HD) video. It is encoded as YPbPr video channels and supports 1080i, 720P, and 525P video standards. The component video also consists of an encoder IC, a decoder IC, and some analog filters.

3.1.3.4 DVI Digital Video

DVI is used in flat panel displays and computer graphics cards. It carries HD video in high-end consumer video equipment like plasma displays and some DVD players. Its port supports HDMI by using a DVI/HDMI convertor. The DVI on this VIODC is able to support the pixel clock up to 165 MHz.

3.1.3.5 VGA Input and Output

The VGA input and output ports are used to connect the VIODC to the screen. The VGA output port is supported by the DVI output. The user should use a DVI-to-VGA connector to connect the screen to the DVI port.

3.1.3.6 SDI Video Input and Output port

The SDI Video interface can support both SD and HD videos on a high-speed serial protocol using a coaxial cable.

3.1.3.7 Clock Generator

The ICS 1523 clock generator IC provides the clock signal for the VIODC.

3.1.3.8 Virtex-2 FPGA

The VIODC contains a Xilinx Virtex-2 FPGA which holds the drivers for the video inputs and output ports. It also contains the video interfaces as well as the interface to the ML402 VSK.

The Virtex-2 family is a powerful and high performance FPGA designed to fulfill a wide range of applications from low-density to high-density designs, such as video, networking, DSP applications, and telecommunications. Its architecture is based on 0.15 µm CMOS and is optimized for designing the high speed and low power consuming customized modules and intellectual properties (IPs). The Virtex-2 family provides solutions which can reduce the time-to-market of applications. The density of the gates up to 10 million gives the flexibility to the designer to execute unlimited changes during the application design and development. High calculation cost, fast and complex routing of wide buses which is implemented in networking applications and extensive pipeline and FIFO memory requirements applied by the video processing applications make the Virtex-2 family a good choice for this kind of applications.

The Virtex-2 FPGA is made of 40 thousand to 8 million system gates, considering 11 different types of its family. The Virtex-2 family members include 256 to 46592 configurable logic blocks (CLBs). The CLBs are used to implement the combinational and synchronous logic elements and basic storage elements. Virtex-2 family has 72 to 3024 Kilobits of RAM blocks and 4 to 12 digital clock managers (DCMs). The DCMs provide flexibility in using the system clock, different delays, different multiplications and divisions of the system clock, and coarse-grained and fine-grained clock phase shifting. The Xilinx Virtex-2 XCV2P7 FPGA which is used in VIODC includes Multi-Gigabit Transceivers (MGTs) which are used to support the SDI interface.

The VIODC uses the Virtex-2 FPGA to implement the stand-alone mode. In this mode, the Virtex-4 FPGA on the ML402 does not act as the system processor. Instead, the Virtex-2 of the VIODC functions as the main processor. It can implement all the functions of the video input and output on the VIODC in a stand-alone mode.

3.1.3.9 XGI Connector

The ML402 VSK provides the 64-pin XGI connector so that other Xilinx standard cards are integrated into it. Therefore, the VIODC is connected to ML402 through the XGI connector. This connector interfaces the buses for data signals to travel from ML402 to VIODC and vice versa. The necessary 5V power for VIODC to function is also passed through the XGI connector.

3.1.3.10 VIOBUS

As mentioned in the previous section, the XGI connector is the interface which the ML402 uses to connect with its standard boards. The standard bus to fulfill this connectivity is named the VIOBUS. The VIOBUS consists of two 27-bit buses to carry the 27-bit digital video signals from ML402 to VIODC and from VIODC to ML402. Each 27-bit bus involves a 24-bit video bus plus a one bit horizontal synchronization signal, one bit vertical synchronization, and one bit clock enable signal. The VIOBUS has also a clock, a reset signal, an I2C interface, and a serial bus.

3.2 Simulation

Simulation is necessary for studying the functionality of all hardware systems. It is very risky to start designing a system without verifying the flawless functionality of the system. In addition, the designer obtains a good measure of system reliability. He can also analyze the system with different inputs and under variable circumstances to make sure whether any fault occurs during simulation. There are very powerful simulators nowadays to simulate the systems for different applications.

The VSK provides two methods of simulation for the designers. The designer can perform completely in simulation which is called Software Simulation or he may apply the hardware and a sequence of real video and perform a real-time simulation. The second method is named Hardware-in-the-Loop simulation. Hence, the VSK and accompanying softwares allow the designers to develop and test the video processing modules on real hardware, applying real-time video stream.

Software simulation of the video processing system presents some challenges and limitations. Controlling the real-time video stream is not easy; on the other hand, saving much amount of video data to process needs much memory and domination on memory management. It should be

mentioned however that the software simulation is able to simulate the video processing component. This simulation is not performed as fast as the real-time Hardware-in-the-Loop.

In the Hardware-in-the-Loop mode, the VIODC and its peripherals provide video streams for the ML402. The ML402 executes the real-time video processing on the video stream. The ML402 is equipped with a Virtex-4 FPGA as the main processor which supports the MicroBlaze processor architecture, a good amount of DDR SDRAM memory to buffer the video data, communication means like RS-232 serial port, and design environment softwares such as Simulink. The Simulink and System Generator design environments introduce a design method which is named Hardware/Software Co-simulation. Because both the hardware architecture of the FPGAs can be configured by the designer and a software program can be executed on it, the Co-simulation can be applied for developing the ML402. After the designer develops a video component by Simulink, he can produce an independent version of his design by Xilinx System Generator. This component is compiled in Simulink. When the simulation is started in Simulink, the component is integrated into ML402 FPGA. A real-time or non-real-time video stream is passed through VIODC and it is processed by the processing component. Finally, the results are sent back to Simulink to be shown on the scope.

3.2.1 Hardware-Software Co-Simulation

In the previous section we became familiar with two simulation methods and we discussed some pros and cons of each method. Now we will see how we can design a system, simulate its functionality, and import it for further development and download it on a real hardware. For a video processing system, we need to design a video processor which executes a function on the video stream. The video processor should be accompanied by necessary memory and peripherals. Xilinx System Generator and Embedded Development Kit (EDK) are very powerful software platforms which help the designer to implement the system. System Generator provides a good amount of useful tools to carry out a variety of applications in different domains, as well as in video processing. Figure 3.3 shows the block diagram of the video processing system, the microprocessor, its standard peripherals, and its custom peripherals. The peripherals in Xilinx processor systems are called Processor Cores or Pcores. System Generator is a software program with different libraries for designing microprocessors and pcores.

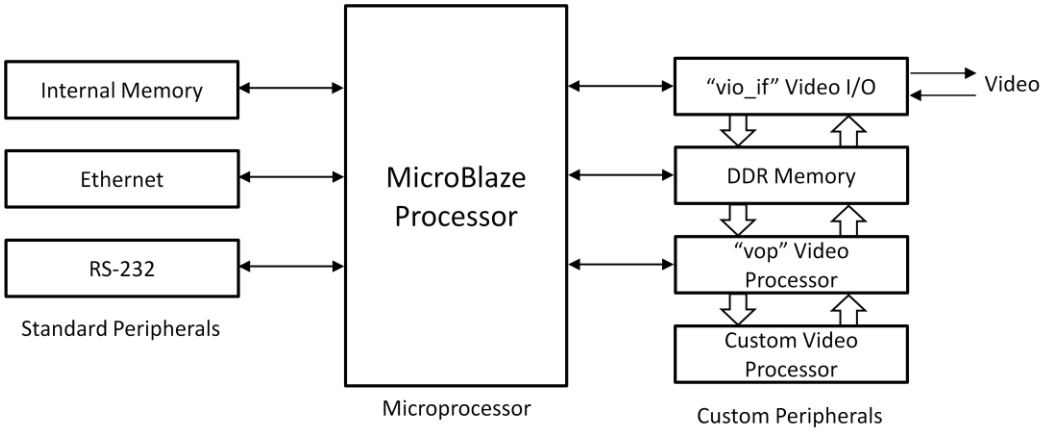


Figure 3.3 – Block diagram of video processing system

The above system is suitable to implement and test the video inputs/outputs of the VSK. One of the most important and most powerful methods that System Generator and EDK provide is that after compilation and validation the functionality of the pcores, each of the pcores can be integrated in any EDK system. Therefore, the pcores are highly reusable. Later in this chapter we will see how to design a desirable video processor. In the next chapter we will modify this pcore in order to achieve the objectives of this project. We will discuss how we can use the existing pcores and how to modify them in order to reuse them in other projects. The architecture of the FPGAs on ML402 and on VIODC is described in the following subsections.

3.2.1.1 ML402 FPGA

In this project, the system architecture implemented on the Virtex-4 FPGA consists of a MicroBlaze soft processor and four pcores. The MicroBlaze is a Reduced Instruction Set Computer (RISC) processor and is optimized for Xilinx FPGAs. MicroBlaze contains some fixed features which should exist in all designs. Some of its parts can also be configured according to the design requirements. The MicroBlaze consumes the logical gates of the FPGA chip and is downloaded on the FPGA as bitstream format. This is in contrast to hard core processors such as PowerPC which are real physical processors and are built on the FPGA chip by the manufacturer. The flexibility and configurability of the MicroBlaze allows it to be used in a wide range of applications. Compared to the Application Specific Integrated Circuits (ASICs), the FPGAs and their standard processor architectures (one of which is MicroBlaze processor) have better performance regarding the cost and time-to-market. They have shorter time-to-market according to its configurability. The FPGA systems can be integrated and reused easily in other projects and

the designer can profit from different design tools and softwares. Three peripherals which accompany the MicroBlaze processor as its standard peripherals are RS-232 serial port, 64 kilobyte internal memory, and the Ethernet protocol network interface. These peripherals can simply be chosen when the MicroBlaze is being configured during FPGA system design.

The vio_if pcore is the module through which the ML402 and VIODC communicate. The communication means is the VIOBUS. As mentioned earlier, VIOBUS carries a video input port and a video output port.

The ddr_if pcore supports the access to DDR memory. The DDR memry is designed to save video sequences for further processing or playback.

A video processor implements some operations on the video stream by the vop pcore. This pcore can be modified to fulfill new custom processes. The pcore labeled custom video processor is also available for the designer to perform any desired video processor to the system. Any number of custom video processors can be added.

3.2.1.2 VIODC FPGA

The Virtex-2 FPGA on the VIODC is in charge of configuring different video interfaces. It also passes the video stream toward ML402 and vice versa. A bit file which contains the drivers of the video inputs and outputs is loaded in the VIODC FPGA. When the user can choose a video input/output, the VIODC is configured so that the video is generated by the proper video source and be conducted toward the indicated video sink. The video source can be a live video input which was explained in sections 1.1.1 to 1.1.6. Another video source is the test pattern which functions like a live video stream. Moreover, the Matlab matrices and Matlab Simulink Image Processing Blocksets are other video sources and video sinks during simulation.

3.3 MicroBlaze System Design

The most important part in any embedded video processor system is the processor. The standard processor in ML402 is MicroBlaze. MicroBlaze soft-processor is configured simply by means of EDK and System Generator for Virtex-4 FPGAs. In addition, System Generator produces the interfaces to read and write data and reduces much design time and effort. There are two methods to develop a MicroBlaze microprocessor system:

1. A system developed in System Generator is exported into EDK.
2. A design developed in EDK is imported into System Generator.

We know that the System Generator supports the designer in developing a system and simulating it. In these two methods, it is again possible to develop the system in System Generator, simulate it, and export to EDK. Or, in the second method, develop in EDK, import it into System Generator and perform the simulation. Hence, the Hardware-Software Co-Simulation can be applied in both above methods.

In this project, a system is first designed in System Generator. It is then exported in an EDK. In this section we will study in more detail the design steps.

After the hardware of a system is designed in a Co-Simulation approach, a software program should control the functionality of this hardware system. Therefore, we need the means by which the data passes from hardware to software and from software to hardware. Shared memories are the tools that fulfill this communication. Shared memories in the form of FIFOs, Registers, and RAMs are in charge of transferring data from EDK processor to System Generator designs and from peripherals to EDK processor. When we export the design to EDK later on this project, we will see in one of the approaches that these shared memories are available through the functions of the C program. There are some standard functions by which the shared memories can be accessed. There is a read function and a write function for each shared memory. The data can be read through the read function and it can be modified by the write function. The shared memories, memory map, and read and write functions are automatically generated by System Generator. Memory map is the interface between the shared memories and the EDK embedded processor. Finally, the data can be shown on the screen through the RS-232 port and a program like Hyper Terminal.

3.3.1 EDK Processor

The EDK processor is the principal part of the system which will be translated to the Microblaze embedded processor after translation to hardware. An EDK processor block is shown in Figure 3.4:

Figure 3.4 – EDK processor block

MicroBlaze is configurable and like other configurable processors has a fixed part as well as a customizable part. The fixed part consists of thirty-two 32-bit general purpose registers, 32-bit instruction word, 32-bit address bus, and single issue pipeline. There are many configurable sets on the MicroBlaze processor. Some of them are: three to five processor pipeline depths, On-chip Peripheral Bus (OPB) data side interface, OPB instruction side interface, Local Memory Bus (LMB) data side interface, LMB instruction side interface, hardware divider, up to seven Fast Simplex Link (FSL) interfaces, Instruction cache over CacheLink (IXCL) interface, Data cache over CacheLink (DXCL) interface, 4 or 8-word cache line on XCL, Floating Point Unit (FPU), hardware multiplier 64-bit result, and Lookup Table (LUT) cache memory. The MicroBlaze supports word, half word, and byte data types. All of the MicroBlaze instructions are 32 bits and are categorized into two groups. The first group have up to two source register operands and one destination register operand. The second group have one source register and a 16-bit immediate operand. Instructions are classified as arithmetic, logical, branch, load/store, and special groups.

The MicroBlaze has thirty-two 32-bit general purpose registers. Depending on how the designer has configured the MicroBlaze, it can contain up to eighteen 32-bit special purpose registers. Special purpose registers are Program Counter (PC), Machine Status Register (MSR), Exception Address Register (EAR), Exception Status Register (ESR), Branch Target Register (BTR), Floating point Status Register (FSR), and 12 different Processor Version Register (PVR). The PVR gives back information about most parts of the system.

The MicroBlaze profits from a three stage pipeline architecture. Each processor performs the execution task in some steps. Pipelining permits the processor to perform this task in fewer steps than a normal execution. Most of instructions take one cycle to be executed by MicroBlaze. So, each instruction takes three cycles for Fetch, Decode, and Execute to be completed. Therefore, it takes 6 cycles for two instructions to be completed without pipelining; whereas, it takes 4 cycles for the same instructions in a three stage pipeline as depicted in Figure 3.5:

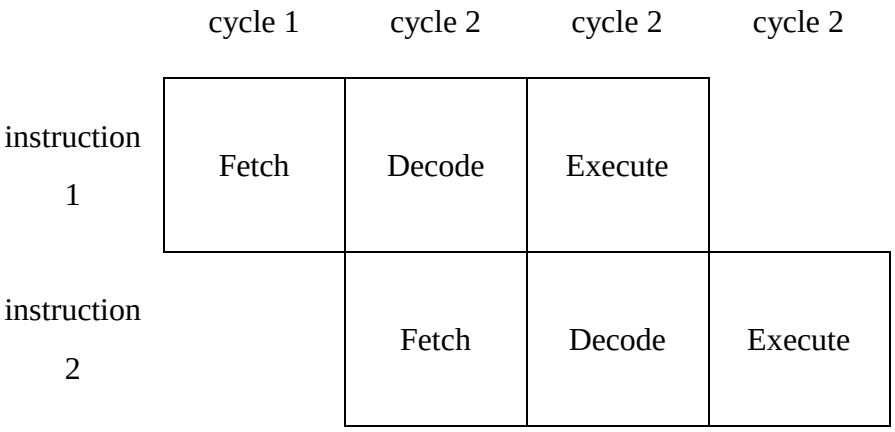


Figure 3.5 – Pipelining in 3 stages of fetch, decode, and execution of an instruction

A few instructions among the MicroBlaze instruction set take more than one cycle to be completed. During the execution of these instructions, the other instructions which are ready to be executed should wait until the execution of this task is terminated. Then, the execution of waiting instructions is resumed. The fetch stage may sometimes take more than one cycle due to slow reading of memories. The MicroBlaze applies a pre-fetch method to reduce the effect of the memory read latency. In this method, when the processor is stalled by execution stages which take more than one cycle, the processor implements a pre-fetch of the instruction into a buffer. So, as soon as the execution of the previous instruction is completed, the next instruction is ready at the pre-fetch buffer. Therefore, the implementation time of the program is reduced about a memory access time.

The MicroBlaze provides two separate address spaces for instruction and data. As mentioned in the MicroBlaze specifications, each of instruction and data have a 32-bit register and the data memory can be addressed in word, half word, or byte size. The addressing space that the MicroBlaze uses to access memory and I/O is the same. It means that the memory and I/O are not separate. The access to memory and I/O is done in three modes: through LMB, through OPB, or through XCL.

3.3.2 FIFO

The “To FIFO” and “From FIFO” are the shared memories used for high rate of data travelling between EDK processor and the peripherals. The FIFO memory is also used for co-simulation.

The FIFO uses independent clock than the system clock; so, the FIFO memory is asynchronous with the system clock. The shared FIFO which consists of a “To FIFO” and a “From FIFO” is shown in Figure 3.6.

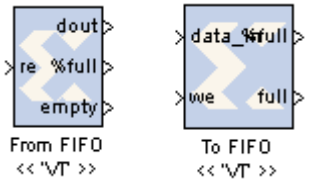


Figure 3.6 – Shared FIFO block

In real hardware, the shared FIFO is implemented with an asynchronous FIFO block and the ports are connected to other blocks of the design. Thus, there is no connection between the FIFO blocks and the PC. On the contrary, each of the shared FIFO blocks (To FIFO or From FIFO) can be connected to PC. The designer can connect one side of a To FIFO or From FIFO block to PC and connect the other side to the user logic. Then, the FIFO can be controlled by the PC. Moreover, the data passing by the FIFO can be shown on the PC screen. In order to explain the functionality of the single FIFO blocks and shared FIFO, the write port of a To FIFO is connected to the user logic. Then, the data which is sent from the design is shown on the PC screen during simulation. Conversely, when the read side of a From FIFO is connected to the user logic, the write side will be connected to PC interface and the design can be controlled from outside of the system (from PC) during the simulation.

3.3.3 Register

A “To Register”, “From Register”, or shared register is compiled as a hardware register in System Generator. Like the FIFO which was explained in previous section, a register can also used in a pair or in single block. A shared register pair is shown in Figure 3.7:

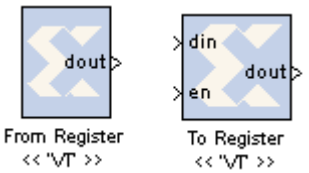


Figure 3.7 – Shared register block

If the designer connects both “To Register” and “From Register” blocks to the user logic, there will not be any communication with outside of the system. Hence, a single To Register or From Register block is used to establish a communication between PC and FPGA system. Therefore, the data passing through hardware can be read and/or controlled by the PC. When a To Register block is connected to the user logic, the other side of the register will be connected to the PC interface. Thus, the data can be written from outside of the system and it will be passed to the FPGA system. On the other hand, when a From Register is connected to the user design, the data is modified by the user logic and it can be read by the outside of the hardware system.

3.3.4 RAM

Two kinds of shared memories regarding the security in accessing the memory are available in System Generator: lockable and unprotected access. Both RAM blocks look like Figure 3.8. You can set the access mode of the shared memory in its configuration setting.

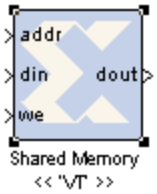


Figure 3.8 – RAM block

In the lockable memory, either software process or hardware design is going to access the memory, must first release the lock. If the hardware has locked the memory first, the software cannot access it until a hardware release happens. When a software lock takes place, the hardware cannot as well read or write the shared memory unless the software unlocks it.

In the unprotected access shared memory, there is not any limitation on accessing the memory. Either hardware or software can read or write the RAM during the simulation.

3.3.5 Port

Ports are used to establish the communication between the components in a Simulink design or between System Generator components. They are the simplest and the most common input/output interfaces. The ports also provide the way for data transmission between processing cores or between the processor and the pcore. The ports also provide the interface between the

VSK and PC during the hardware co-simulation in which the simulation is performed with live video. The ports for connecting two Simulink components are different from the ports used to connect two pcores.

In figure 3.9 you see the input and output ports which are applied a lot in Simulink designs:

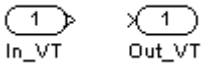


Figure 3.9 – Input and output port blocks

The schematic blocks for communication between pcores or the EDK processor and a pcore is depicted in figure 3.10:

Figure 3.10 – Input and output port of pcores

3.4 Developing the VSK

In this section we explain how we used the previous design tools and design necessary components to develop the ML402 VSK in order to implement the project. Some reference designs are provided by Xilinx and help the designers to reduce the design time. These reference designs which are available on the Xilinx web site [30] develop the ML402 or the VIODC using different tools and different approaches. In some designs a MicroBlaze or PicoBlaze EDK processor is developed, whereas some of them develop standalone peripherals in VHDL or Verilog language. They apply different input and output ports so that the developers will be able to profit from these designs as much as possible. In the VSK_pcore example, the components depicted in Figure 3.3 except the custom peripheral have already been designed. For this project, we developed a video processor as a custom peripheral. This peripheral performs the main functionality of the system, i.e. bacterium tracking.

3.4.1 Designing the PCore using System Generator

At the top level, the simulation system looks like Figure 3.11. The “vid_tracking” module is the main part of the system which performs the tracking algorithm. This module needs a video sequence on which to execute the tracking. The video sequence is provided by the “test pattern”

module. The test pattern module is the leftmost in Figure 3.11. The test pattern simulates the video signals. In the VSK_pcore reference design, it produces frames which consist of 10 lines. Each line then contains 100 pixels. We can see the produced test pattern in Figure 3.12. In the tracking algorithm top level, we will have an “EDK pcore” whose name is “vid_tracking” in this project. We will study the vid_tracking pcore in detail later in this section. Then, we have an “unpack” block. The unpack block separates the video signals into its elements. Each of the separated video signal elements are sent to the scope. Scopes are very useful output interfaces in Simulink.

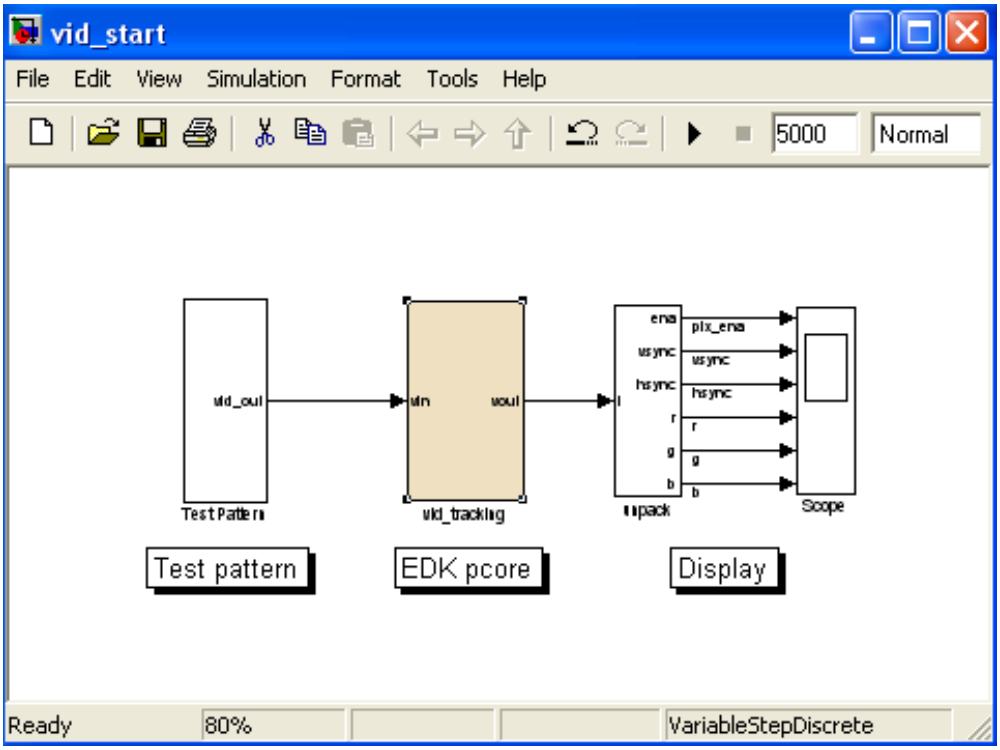


Figure 3.11 – Tracking algorithm top level design

The “scope” element in Matlab functions like an output screen. The designer can study the behavior of the output signals which are shown on the scope. The scope gives the possibility to zoom in or zoom out in all or just a selected part of the signal in order to study better the signals. Simulation results are shown in Figure 3.12. As mentioned, the scope shows the contents of the test pattern as well.

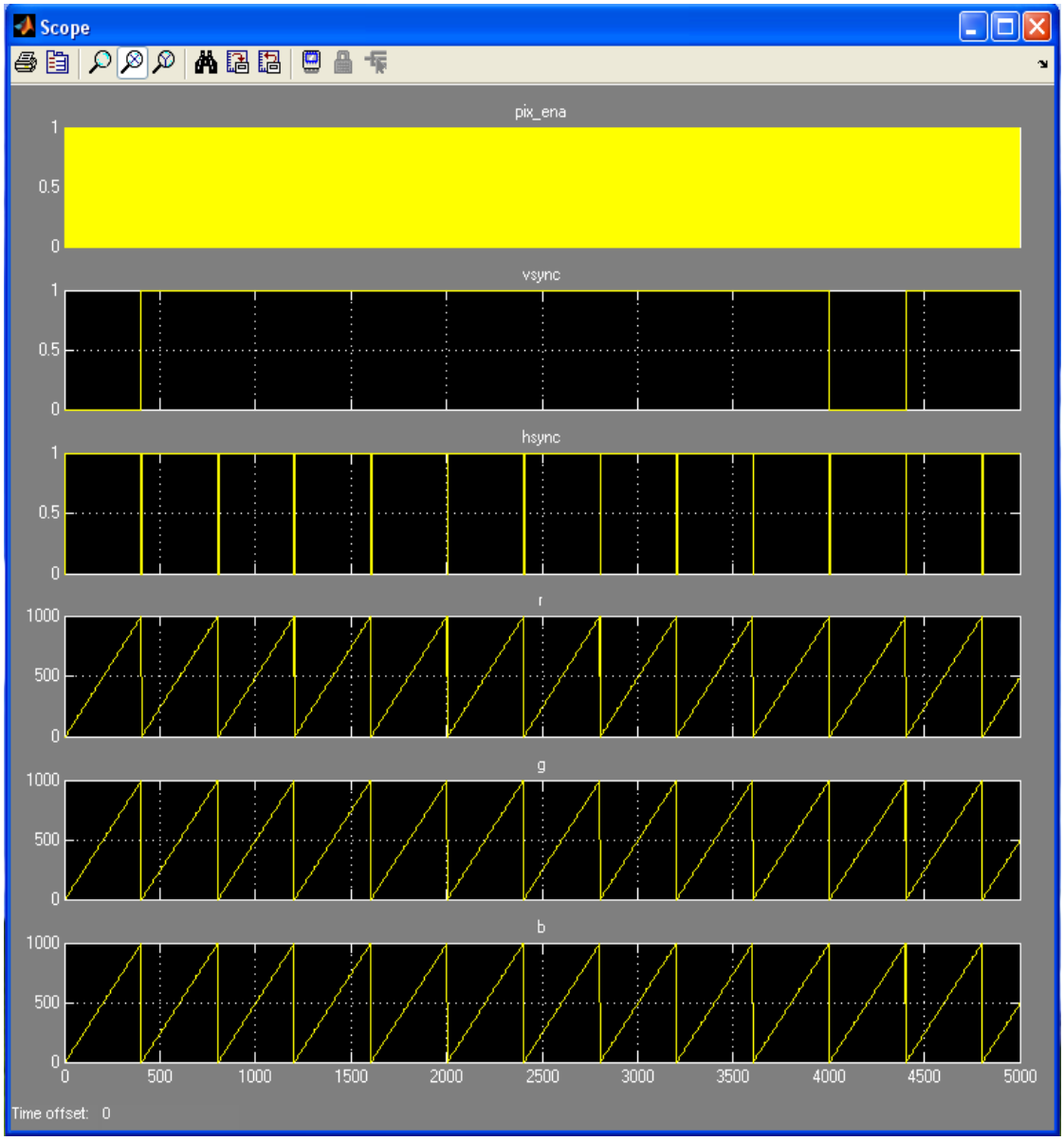


Figure 3.12 – Tracking design simulation result

Six signals exist on the scope view of the figure 3.12. The r, g and b signals simulate the three channels of the RGB color pixel. The r, g and b values of each two consecutive pixel are different. The pixel enable signal is low when a new pixel is detected. At the moment of shifting the pixel enable from 1 to 0, the new pixel arrives and the values of r, g and b change. According to the system specifications, the system clock functions on 100 MHz frequency and the pixel enable on 26.6 MHz. At the end of each line of a frame, the horizontal synchronization signal goes to 0 and it takes one pixel until it goes back to 1. The vertical synchronization signal becomes 0 at the arrival of a new frame. It stays 0 for the duration of one line; then, it comes back

to 1. Each channel of red, green, and blue which make the RGB pixel are 10 bit long. Therefore, each pixel is 30 bit long. Since the difference between the frequency of the signals in figure 3.12 is much, studying all the signals in one figure is almost difficult and it needs to zoom in some high frequency signals such as pixel enable to study it in more detail.

So, we need to design the vid_tracking block to implement our tracking algorithm. Our tracking system is implemented in a top-down design procedure. We started with the main components of the system and we continue to design the smaller parts in a hierarchical method.

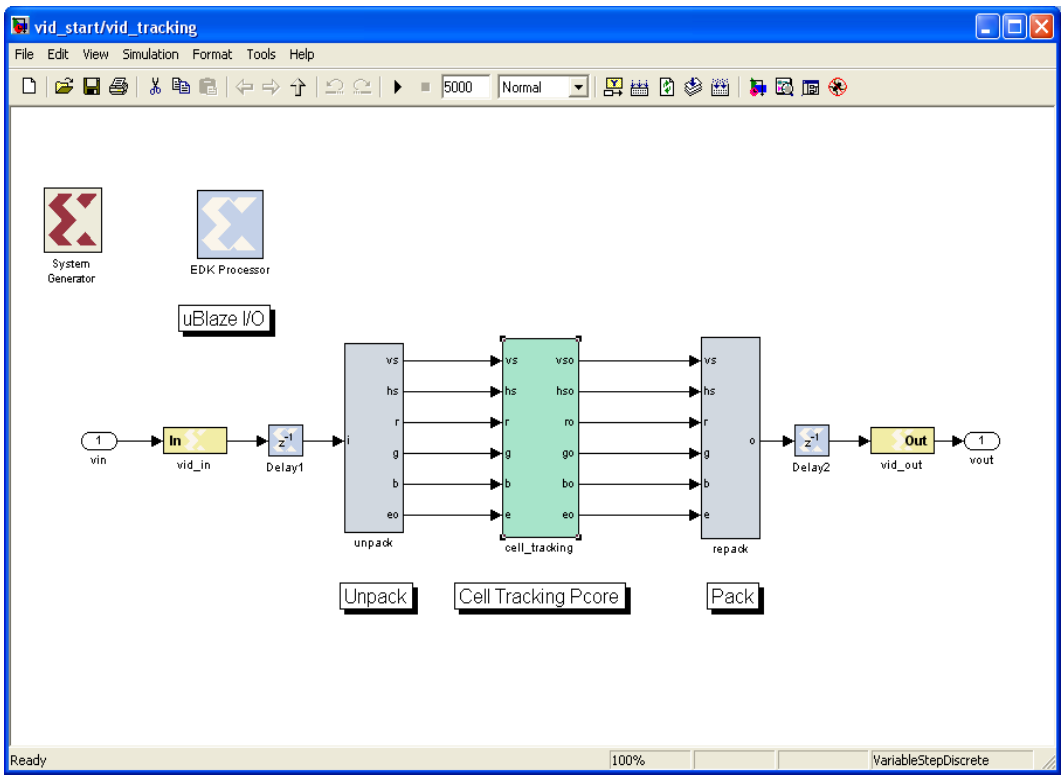


Figure 3.13 – Block diagram of the cell_tracking pcore

The components of cell_tracking pcore are depicted in Figure 3.13.

System Generator translates the Simulink block diagrams to physical hardware. It is applied in our project to generate the vid_tracking pcore. The EDK processor generates the MicroBlaze processor. The "vid_in" and "vid_out" are the input and output ports of the pcore, respectively. These ports provide the input from the test pattern and output toward the scope during simulation. They are input and output of the pcore for the video stream in the real hardware. The "unpack" and "repack" blocks are needed to separate and unite the elements of the video signal which are the red channel, green channel, blue channel, horizontal synchronization, vertical

synchronization, and pixel enable. The “delay” blocks are used to provide the necessary delays to make the signals ready to be processed. A “To Register” block is added to read the value of the red channel. The Simulink elements can be found in the Simulink library, since it is a very useful library which is added to Matlab software to reduce the design time. Simulink contains the tools for many applications such as image and video processing, digital signal processing, control systems, etc. As we explained in section 3.3, a “To Register” single block is used to read the value of a data from outside of the system, for example a PC. The other single block of the register pair is connected to the hardware of the system. Reading the value of a register needs a software function which is automatically generated by the System Generator. More details on registers and their functions will be explained later.

Now we should generate the EDK processor interface for the “red_channel_reg”. As we add the existing memories to the design, they will be listed in the “Memory Maps” section. We have just one register which is “red_channel_reg” in our design. We need the processor interface for the “red_channel_reg” register to be generated so that the MicroBlaze processor would be able to interact with the register. After we designed the pcore, we should generate it in order to export it into the EDK. As indicated earlier in this thesis, any pcore which is designed can be connected to and used in any EDK project. The hardware description (VHDL code) of the pcore is also generated by System Generator.

The “Synthesis Tool” for the EDK which we use is XST. Xilinx Synthesis Technology (XST) is the built-in synthesizer on Xilinx products such as XPS and ISE. XST supports VHDL and Verilog hardware description languages. It synthesizes the HDL designs to produce the netlist files of the design. A netlist file contains the hardware architecture and the design constraints.

3.4.2 Exporting the PCore

In order to develop the system either in C or VHDL language, we have to export the pcore into an FPGA design tool such as Xilinx Platform Studio (XPS). This will integrate the pcore which we designed in System Generator into a video processing project. As we will see in this chapter, the designer can write the functional code for the pcore in C language. In the next chapter we will explain how a designer can modify the pcore in VHDL language and define the system functionality in a very flexible and powerful manner.

In the VSK_pcore directory which is a reference design for the ML402 VSK provided by Xilinx, we find an XPS project system file, named system.xmp. This file contains the basic settings for the system which is going to be downloaded on the ML402 FPGA. A MicroBlaze processor and the driver interface for the basic peripherals which exist on the hardware of ML402 such as RS-232 serial port, memories, and LCD screen.

We rescan the user pcores to refresh them and see the pcores which are already added to the design and to detect the recently added pcore which is the vid_tracking pcore designed in previous section. The pcores are addressed as coprocessors in the XPS software.

We connect the vid_tracking pcore to the MicroBlaze processor via an FSL channel. The interfaces and details about the FSL channel connection are implemented by the XPS software. After adding the pcore to the MicroBlaze processor, we should connect the ports of the pcore to the appropriate ports of other modules. The vid_tracking pcore has 4 ports. One port is connected to power source, one to the system clock. Two other ports are video buses, one is input and one output. The video bus is a 33-bit bus which consists of 10 bit red channel, 10 bit green channel, 10 bit blue channel, 1 bit horizontal synchronization, 1 bit vertical synchronization, and 1 bit pixel enable. The vid_tracking pcore is connected to other components through the video bus. Therefore, the connection between the pcores should be configured according to Figure 3.17. In other words, the input port of vid_tracking is connected to vchan_1, its output port is changed to vchan_2, and the input port of vio_if module is modified to vchan_2.

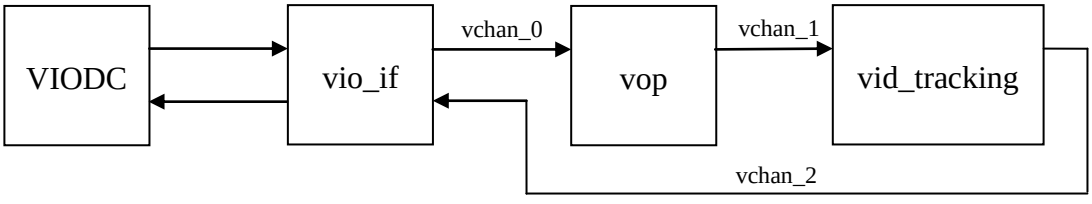


Figure 3.14 – Connection of vid_tracking pcore and other components of the system

3.4.3 Adding software functions

The hardware of the system is designed up to this step. We should now write the software program by which we control the flow of the program. We can read or write the data which is processing by the video processing pcore. Two files are already generated by the system generator during generating the vid_tracking pcore. These generated files are "vsk_vid_tracking.h" and "vsk_vid_tracking.c". The former contains the header of the functions while the latter holds the body of the functions. The main functionality of the functions is described in the second file.

As explained in section 3.4, the "Generate Netlist" tab activates the XST to synthesize the design and gives an error if any fault in design has occurred. Finally, a bit file is generated in order to be downloaded on the ML402 FPGA.

The software files should be copied in the "src" directory of the project directory. This directory contains the software functions which should be executed by the MicroBlaze processor. The software functions usually interact with the shared memories. We define the signature of two functions in the vsk_vid_tracking.h header file and their bodies in the vsk_vid_tracking.c source file. These functions read the data from and write the data to the red_channel_reg register which we generated in section 3.4. After these files are modified and added to the project, we should compile these files and add them to our project. While implementing the software files on the hardware, these files are loaded on the memory of the VSK in order to be executed by the MicroBlaze processor. Finally, we update the bit file. After updating the bit files of the project, a bit file which is named "download.bit" is generated in the "implementation" subdirectory of the project directory. This file should be copied on the SystemAce Flash memory card in order to download the bootable FPGA configurations and the structure of the FPGAs on the VSK as described in the next section.

3.4.4 Downloading the design

There are two ways in order to download the bit files on the FPGAs. You can either use the JTAG cable or the SystemAce Compact Flash memory to download the files.

For downloading the file with the JTAG cable, you need the iMPACT software. If the JTAG cable is connected to the ML402 and the VSK is on, iMPACT will start detecting the existing

FPGAs on the board by selecting the “Initialize Chain” from the “File” menu. You just need to right-click on the target FPGA, then select the file and download it on the board.

The other method is to download the configuration files on the VSK FPGAs through the SystemAce Flash memory. There is a three-state switch on the VSK which is called “Configuration Source Selector Switch” with the “CPLD Flash”, “Plat Flash”, and “SYS ACE” choices. In order to make VSK read the configuration of the FPGAs from the SystemAce Flash card, the user should set the “Configuration Source Selector Switch” on “SYS ACE”. We need a Flash card reader/writer to be able to connect it to a PC and write the files on the SystemAce Flash card. Eight different designs can be recorded on the Flash memory and the user can choose any of the eight designs. Each of the design configurations must have an address. The Flash memory contains a system file on which the given name of these eight configurations and their address is copied. This system file is a notepad file as depicted in Figure 3.18. Our project is saved with the name “vskdiag” and the address 2 on the Flash memory card. When the user chooses one of the designs, the address of the design is detected by the boot loader of the VSK and the configurations of the ML402 and the VIODC FPGAs are loaded.

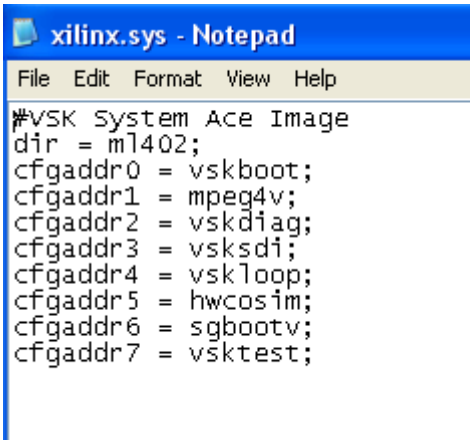


Figure 3.15 – Different configurations on the SystemAce Flash Card memory

There are three ways to choose a desired configuration to be loaded on the VSK. There are three switches on the VSK by which the user can select the desired program between 8 possible programs to be loaded on the ML402 and the VIODC daughter card. For example, if the user configures the switches as 010, the Cfgaddr2 of the system file is called and the vskdiag design is loaded on the VSK (Figure 3.18).

The user can also choose his favorite configuration by selecting among the programs shown on the LCD. When the VSK is switched on, this system file is read and an interface is shown on the LCD screen. After the user selects his desired configuration, the configuration files which are already copied on the SystemAce Compact Flash memory are loaded on one or both FPGAs, according to the project. According to the selected design, the design configuration may engage one FPGA or it may make both FPGAs function simultaneously.

The third way to choose a design is through the serial port. We can connect the VSK to a PC and choose a configuration through a software program such as Hyper Terminal.

For this project, two configuration files should be copied on the Flash card memory. One is necessary for the Virtex-4 FPGA on the ML402, the other for the Virtex-2 FPGA on the VIODC daughter board. The mentioned system file which is required to be copied on the Flash memory is named “Xilinx.sys”. If any new directory containing a new configuration is added to the Flash card, the “Xilinx.sys” file should be edited for the new directory.

The configuration files should be in a defined format in order to be read by the VSK. This format is produced by entering a command in the windows command line as follows:

```
ml40x_bit2ace bitfiles\download.bit bitfiles\viodc_wrapper_11b.bit vskdiag.ace
```

The “ml40x_bit2ace” is a program which produces the appropriate “ace” file for the FPGAs. This program accepts one or two bit files and generates an “ace” file suitable for the ML402 FPGA and VIODC FPGA. The “bitfiles” directory contains the bit files which are necessary for the generation of “ace” file. The “download.bit” file is generated by the XPS after bitstream update. As mentioned in the previous section, this file contains the configuration of the FPGA of the ML402. Both hardware and software settings are contained in this file. The architecture of the ML402 which consists of a MicroBlaze processor and its accompanying pcores, in addition to the software program which should be loaded on the memory and be executed by the processor form the “download” bit file. The “viodc_wrapper_11b.bit” file is the other bit file which is required for the VIODC FPGA. It contains the drivers for the input and output interfaces on the daughter card. This file is already provided by Xilinx and it is applied in this project as it is. We copy this file also in “bitfiles” directory. The “ml40x_bit2ace” considers the last operand of the instruction line as the target file. The generated “ace” file – vskdiag.ace – should be copied in a folder with the same name on the Flash card memory. In addition to the configuration of the FPGAs which

are located in separate directories on the Flash card memory, the SystemAce Compact Flash card contains the bootable FPGA configurations. It means that when you switch on the VSK in the case that the setting is read from the SystemAce card, the VSK will decide the architecture and the software program in which directory will be loaded on the memory.

3.5 Conclusion

In this chapter the pcore design process was studied in detail. We saw how the Matlab software and the Simulink library components could help us to design a pcore. Then, we simulated the design and discussed about the video signals. The pcore was then translated to hardware description by the System Generator tool of Simulink. The pcore was exported to XPS and the software functions were written to be able to interact with the design.

The Matlab/Simulink software is very powerful and flexible to design a hardware project but some features of the hardware are difficult to be dealt with through Matlab. For example, reading and comparing signals need a more professional tool. In the next chapter we will study how we can modify the pcore which we designed in chapter 3 in order to implement our tracking algorithm.

CHAPTER 4 CELL TRACKING PCORE MODIFICATION

We learned how to design a pcore in Matlab/Simulink in the previous chapter. Then, we studied how to export the design in XPS and how to download it on SystemAce Flash Card memory. We saw that the result was not the ideal that we are looking for and we expect in this project. This chapter starts by outlining some limitations of the design described in previous chapter. Then, it guides us through the modification of the pcore which was designed in the previous chapter. Synthesis of the hardware is an interesting subject which was performed in previous chapter but its potential problems did not appear. The problems we encountered during synthesis of the hardware are explained. We will see the solutions for every single problem introduced in the chapter. The solution to find an output port on the ML402 in order to connect a peripheral is also explained.

4.1 Survey of previous results

According to the pcore whose design process was explained in Chapter 3, we succeeded to send the pixel values of the entering video frames to PC. We could read the values on a HyperTerminal application through the RS-232 serial port. The stream of either the red, green, or blue channels of each pixel was shown on the PC screen. In order to make decision about the values of the pixels and implement our tracking algorithm, we need to have access to all 33 bits of entering pixels at the same time. In other words, the pixel enable signal, the values of red, green, and blue channels of each pixel, and the vertical and horizontal synchronization signals are necessary to be processed simultaneously by our tracking algorithm. Simulink, which was the main environment of designing the tracking system hardware in the previous chapter, has limited capacity for the implementation of the tracking system. In order to fulfill the requirements of the hardware system in Simulink, we have to use the available components in the Simulink libraries. It can be a good idea; however, it may happen in every stage of the design that we need a component which does not exist among the Simulink libraries. Since Simulink does not provide a low level script programming facility – as VHDL provides for designers – it is predictable that Simulink will not be perfect to perform all the details of a tracking algorithm. On the other hand, it should be mentioned that when the System Generator exports a pcore to XPS, it generates the VHDL equivalent of the design. We know that the code which is produced by System Generator is not optimized considering the speed of execution and number of VHDL lines which are

generated. The latter causes an increase in the hardware cost. Hardware cost means the number of logical gates which form our hardware. If the designer decides to program all the tracking system in VHDL without the aid of Simulink, of course he will finish the work with fewer lines of code than automatic code production by Simulink. This is obvious by taking a look at the code which is generated by Simulink. The advantage of Simulink in tracking system design is that it can be used to produce the main components of the system. As we saw in section 3.4, besides the tracking system, there are the EDK processor, the delay components, unpack and repack modules, and input and output ports. Simulink is useful in generating these modules, especially the pcore ports generation which is an error-prone process. Thus, the VHDL code which is generated by System Generator can be modified after it is exported in XPS. However, the modification should be performed very carefully because it is possible that the designer damages the code as it is automatically generated according to the predefined algorithms. In order to modify the code, the designer should have enough mastery of digital systems and he should thoroughly understand the generated code.

As mentioned earlier, the low level abstraction description of our hardware permits us to know about the details of video signals. The C programming language and the features that this language provides to designers are not enough for our goals. In the next section we will briefly study the VHDL language and the reasons which explain why it is a better fit for our project than the C language. We then continue with modifications of the vid_tracking pcore.

4.2 Hardware Description Languages

The output of the pcore which we introduced in the previous chapter did not fully satisfy the requirements of the project. C language is very efficient in memory management, creating functions and developing the applications of higher level abstractions than what we expect. We have to look for programming languages which provide the access to the details of a hardware design. Hardware Programming Languages (HDLs) are better fits to our requirements. VHDL is one of these programming languages. The designer can describe the hardware system in different hardware abstraction levels such as gate level, register level, and behavioral design. VHDL offers various features for designing in all above abstraction levels. VHDL is an efficient language for a wide range of applications from small simple designs to multiprocessor complicated ones. Newer versions of simulators and synthesizers of VHDL code have recently emerged so that they have

made this language more powerful and popular among hardware designers. The general method of developing the VHDL code in this project is that the VHDL code is developed in ModelSim simulation software and the syntactic errors are removed. After the code is proved to be syntactically correct, a testbench is developed to test the functionality of the design. Then, the code is copied into XPS in order to be compiled and synthesized. XPS has itself a built-in VHDL compiler as well as a synthesizer which produces the files for downloading on the hardware. Every time the code is changed, all the modules should be compiled and synthesized. The problem is that the synthesis of the code takes a relatively long time – 10 to 20 minutes, compared to some seconds of compilation time. Therefore, we prefer to profit from an independent VHDL compiler. That is the reason why we used ModelSim to compile the code, debug it if necessary, and verify its functionality before we copy it to XPS and study the synthesis results.

4.3 Pcore VHDL code modification

We explained that three custom peripherals are already designed by Xilinx and implemented on the ML402 (Figure 3.3). These three are necessary for configuring the input and output interfaces. They prepare the input and output video ports so that our own custom peripheral which is vid_tracking pcore will be functional. We can imagine that it is impossible for vid_tracking to work without these peripherals. Then, we tried to design the vid_tracking through Simulink and we saw that our expectations were not respected. As mentioned in the previous section, the solution is the modification of the pcore in a lower level of abstraction. The hardware description of all the four pcores in Figure 3.3 exists in four separated VHDL files. These VHDL files are generated through the pcore export tool from Simulink to XPS. The generation is performed by System Generator. As it was explained earlier, it is actually better to let the vid_tracking pcore not contain any logical component. Thus, in order to have the proper pcore for modification, we have to go some steps back from what we designed in previous chapter. As a result, in contrary to previous chapter that we added a “To register” register to the vid_tracking pcore, in this modified version we don’t add any component to vid_tracking pcore. So vid_tracking will be left like an empty module. An inspection of the VHDL file of vid_tracking reveals that when the pcore is generated, just the ports are generated and there will not be any Simulink block in the pcore. The input ports are mapped to the output ports. This means that the

arriving signals are connected on the leaving signals and no processing takes place on the signals. As mentioned earlier, the generated VHDL code is large and complex. The VHDL code consists of the equivalent hardware description of all the modules in the design; in other words, the EDK processor, pack and unpack modules, and delay blocks. The description of each module begins with the entity description followed by the ports definition. The architecture of each module follows its entity description and port definition. After we remove all the Simulink blocks from vid_tracking pcore, we see that vid_tracking only contains the ports of the pcore and it does not contain any more connections or any logical blocks. However, the port mapping of the input and output ports is done in its architecture description. This means that each input port is connected to its pertinent output port to keep the flow of data which enters and leaves the module. There is the hardware description of at least a logical component in other modules. In other words, there is at least a logical block in the schematic Simulink design of other components whose VHDL equivalent exists in its hardware description script.

We find the "vid_tracking" entity in vid_tracking VHDL description. We know that this entity consists of port definition and port mapping. We find the port which we look for in order to implement our tracking algorithm. When we study the input ports of the entity, we notice that the "vid_in" port carries the video signals from outside of the pcore into the entity, since it is 33 bits long. This port is defined by std_logic_vector data type in VHDL. Std_logic_vector is a data type which can be assigned eight possible values. The most important ones are: zero, one, and high impedance. This signal is the target of our video processing algorithm. The important advantage that the low level abstraction description of the pcores gives us, is that because we have access to all hardware features of the design, it is extremely flexible to perform the events at specific times. These principles can be better illustrated by ways of an example. Suppose that it is necessary to count the clock ticks of a system and perform an action according to this count. Or, in this project, we would like to be aware of the arriving frame by detecting the falling edge of "vertical synchronization" signal. We have to hold the state of the signal for a very short period (some nanoseconds). Then, we want to compare the value of the pixels with a threshold and make a decision according to this comparison. We have to consider the delay of the flip-flops and logic gates. The delay of the wires cannot be ignored either. Synchronization and timing of the logic gates and signals is one of the features that does not exist in high level languages like C or Java

while hardware description languages like VHDL provide very flexible features in order to process the signals.

According to the simulation which was performed in section 3.4 and its result depicted in Figure 3.12, the video signal should have the following order in its 33-bit bus: 10 bits for red channel, 10 bits for green channel, 10 bits for blue channel, 1 bit for horizontal synchronization, 1 bit for vertical synchronization, and 1 bit for pixel enable. This order is also obvious from the unpack and repack modules of vid_tracking pcore. We need to know the right order of signals which form the video signal bus. If the order of signals is mistaken, all calculations based on the correct detection of video signals will be invalid. For example, if the horizontal synchronization bit is supposed instead of vertical synchronization bit, the beginning of each line of frame will be supposed as the beginning of the frame. Then, the second line of a frame would be the second frame and so on. The ML402 manual does not precisely provide us the details of the video signal. We supposed the video signal to have such an order; however, after implementation of the tracking algorithm, it was proved that the above supposition was true. We could also verify the order of signals practically by using oscilloscope. As mentioned earlier in the chapter, because the VHDL code is generated automatically by the System Generator software, we should be careful while we are modifying the code. Thus, we keep the arriving video signal in a temporary signal and start implementing the video processing algorithm on this temporary signal.

According to our developed tracking algorithm, the frame should be converted to gray scale. Therefore, as the first step to process the video signal, the system should generate a gray video sequence from the entering RGB video. A video sequence is a series of video frames which arrive at regular intervals. The frame frequency depends on the frame generation of camera and the frame capturing of circuit. The frame frequency in our project is 60 hertz. During the gray video conversion, all the frames should be converted from color style which is called RGB video into gray video. A gray video has only one channel instead of three red, green, and blue channels of color video. Gray image consists of only black and white colors. In this kind of image only the intensity is variable. White pixels have the highest intensity and black pixels have the lowest. In order to have a gray image all the pixels of an image should be transformed from RGB to gray.

We have to detect the pixel_enable signal to be able to find the entering pixels. To detect a falling edge of a signal, we should detect a ‘0’ followed by a ‘1’ in two consecutive clock signal.

```
if (pixel_enable_1 ='0' and pixel_enable_2 = '1') then
      -- output; the falling edge is detected, you can perform the task
else
      -- do nothing, wait for next clock to detect a consecutive 0 and 1
end if;
```

Since we need to keep the states of the signal for some clock cycles, we have to use flip-flops. Therefore, the real hardware circuit which will be downloaded on the FPGA looks like this circuit. The two flip-flops keep the states of the pixel_enable signal in two consecutive clock cycles and the output contains the "if" condition of the above VHDL code.

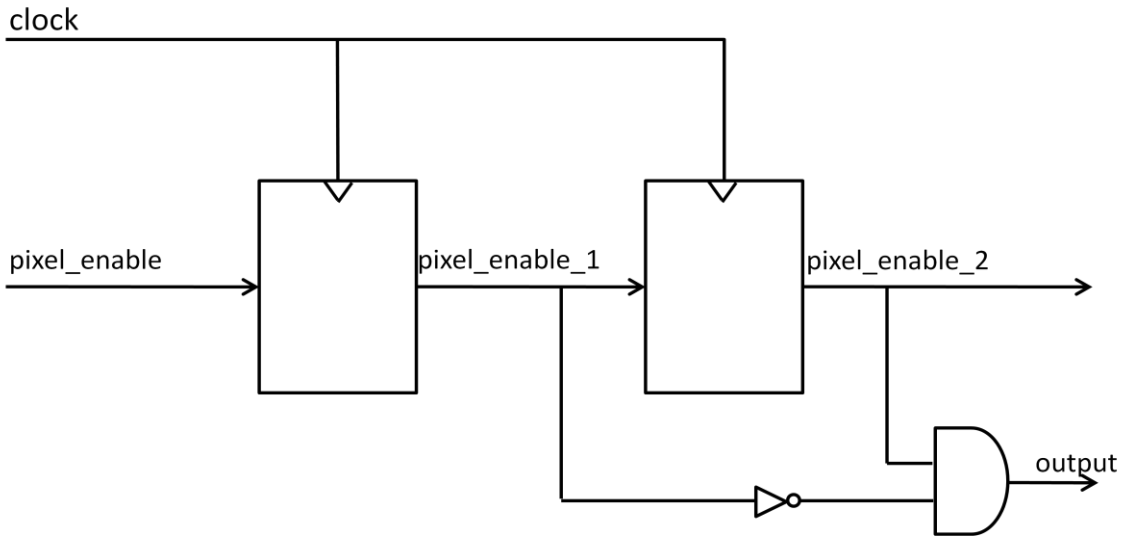


Figure 4.1 – Falling edge detection circuit

Thus, we detect each entering pixel and execute the conversion formula on it. The RGB to gray image conversion is defined by the following formula.

$$gray\_channel = 0.3 * red\_channel + 0.59 * green\_channel + 0.11 * blue\_channel \quad (4-1)$$

The above formula simply converts an RGB color image to a grayscale image. This conversion is necessary in our algorithm. Our algorithm considers the intensity of a pixel and compares it with

the intensity of another pixel. Therefore, one intensity factor is enough for comparison; however, three intensities of red, green, and blue make the algorithm more precise. It should be noticed that three comparisons for each pixel cause three times of processing time and three times of hardware costs compared to one intensity comparison. On the other hand, our algorithm has good performance on grayscale image. It was also proved that the comparison of pixel intensities in grayscale format gives us good results for tracking the bacterium. During the development of tracking algorithm, we try to use the simplest practical methods in which the lines of code, the complexity of the code, and the hardware cost will be as few as possible during both simulation and real hardware implementation.

As it was explained earlier in this chapter, like all the development process of VHDL code, we debug each piece of code in ModelSim software and verify the correctness of its functionality with a simple testbench. After the code is proved to function well, we copy it into XPS. Since compilation and synthesis of the VHDL code takes time in XPS, the above procedure reduces the development time. Defining the variables of *real* data type seems to be suitable to implement formula (4 – 1). As we imagined and we verified in ModelSim, the formula could be implemented by defining four variables and a single line of instruction in which three multiplications, three additions, and one assignment are performed. The result of the formula which is kept in the gray_channel variable should be a value between 0 and 1024. Since the red, green, and blue channels are 10-bit buses, their value should be between 0 and $2^{10} – 1$. The coefficients of these three variables cause the gray_channel value also have a value in this range. The result in ModelSim was exactly what we expected of the VHDL code which we developed. However, some errors appeared while compiling the code in XPS. The real data type is not acceptable in XPS. As the VHDL code compilers of different platforms like Foundation, XPS, Sinplify, etc are different, the errors like we encountered are possible. Some compilers are stronger and more flexible and are able to handle more programming features of the VHDL language.

In order to solve problems associated with the real data type, and in order to remove the need to perform real addition and multiplication, the formula was multiplied by 100 and divided by 100. So, we have a new formula that is depicted in (4 – 2).

$$gray\_channel = (30 * red\_channel + 59 * green\_channel + 11 * blue\_channel)/100 \quad (4-2)$$

This way, we define the red_channel, green_channel, and blue_channel variables as integer data types and remove the need of the real data type. The result will finally be truncated and assigned to gray_channel which is also defined as an integer. There are some important problems in this stage of code development. In order to not face many syntactic and functional problems during the code development, we decided to develop, debug, and synthesize each step of code step by step and study the hardware functionality of the system. So, if there is any error or fault during development of tracking algorithm, we are able to easily discover in which part of the code the errors have taken place.

After implementing and synthesizing the modified formula (4 – 2), the expected results of the system are not respected and the system does not function as we expect. This serious problem is due to inability of the synthesizer to efficiently synthesize our hardware. Since each line of the VHDL code should be translated to its hardware equivalent, the platform must be powerful enough to understand the VHDL code and generate the proper hardware architecture. In this regard, the designer plays the major role by the means that he should program a VHDL code which is highly synthesizable and avoid applying the instructions which are difficult to be synthesized or a set of instructions which are not synthesizable. In fact, the designer who dominates the digital design concepts is programming while he processes each line of code in his mind and imagines the hardware equivalent of the VHDL code. In other words, he synthesizes the code before he orders the machine to do so. The designer should also have experience in STA (Static Timing Analysis) in order to be able to calculate the delays of the gates and the effect of gate delays on other modules of the system, especially when the circuit is functioning in a high clock frequency rate. The wire latency is also an important factor in timing of the system and should be considered. Therefore, in next section we decide to program synthesizable code and pay attention to the points that the designer should take care about during VHDL code development.

4.4 Synthesizability

We explained that the real data type could not be recognized by the XPS platform. This was an example of inability of XPS to synthesize the code. Division is also an operation which is not synthesizable by XPS. This section is just an introduction to synthesizability. As we proceed in this chapter, we encounter different synthesis problems and we introduce the solutions for each one. Later in this section we will show one of the most important solutions. It will explain how to apply the Virtex-4 library components to make our VHDL code synthesizable and faster. It also helps to reduce the hardware costs by means of reducing the gate numbers of the hardware.

Two solutions which are possible to solve the multiplication problem will be presented. These solutions also reduce the hardware cost and the operation delay. The first idea is that we write a function that simulates the multiplication operator. We know that the multiplication is a definite number of additions. Then, if we want to add the binary number 1101 to 1001, we can add 1101 to itself 1001 times. Again, the designer should be careful with the size of the registers and carry bit. A function can be defined to perform the multiplication operator. Whenever there is a need to multiplication, this function is called and the operation is performed.

The second solution which is much faster than the previous one and reduces the hardware cost (gate numbers) is applying the components from Virtex-4 library. There are some hardware components which are already designed by Xilinx for different FPGAs of Xilinx family [31]. Each Xilinx FPGA generation such as Virtex-II, Virtex-4, etc has its special library. Some common hardware components which are highly used in FPGA-based systems are already designed in this library. Since these components are implemented completely with logical gates, their delay is very little. Actually, the delays of these components are not indicated in the related documents of the Xilinx library. The designer can simply add the library to the design and use any of its components. The DSP48 component is a very powerful module for implementing multiplications and additions. DPS48 is designed to be used for digital signal processing applications rather than simple multiplications and additions. The multiplier of DSP48 can multiply two 18-bit numbers which results in a 36-bit number. The result can be extended to 48-bit signed number and can be added to two 48-bit signed numbers. In other word, the adder accepts three 48-bit signed numbers and produces a 48-bit signed number. DSP48 is highly flexible; for example the adder can be cascaded to multiplier and to other DSP48 modules, and

the operands can be signed or unsigned. Therefore, a wide range of operations can be implemented by DSP48 module such as signed-signed, signed-unsigned, and unsigned-unsigned multiplications, logical, arithmetic, and barrel shifter. The block diagram of a DSP48 of Virtex-4 library is illustrated in Figure 4.1.

The ports of the block DSP48 are also shown. The ports A and B are used to connect the input for multiplication, or high and low significant bits for addition, respectively. Port C can be used as another operand for addition. Ports BCIN and PCIN can be applied for cascading this module with another DSP48. BCIN and PCIN are, respectively, cascaded multiplier and cascaded adder from previous DSP48. Some other inputs configure the module for executing each of the operations which were explained. In this project, it is not necessary to connect the configuration pins to any external signal. Thus, the components functions the multiplications which we expect for this algorithm. We will explain how the DSP48 should be configured in order to perform the desired operation.

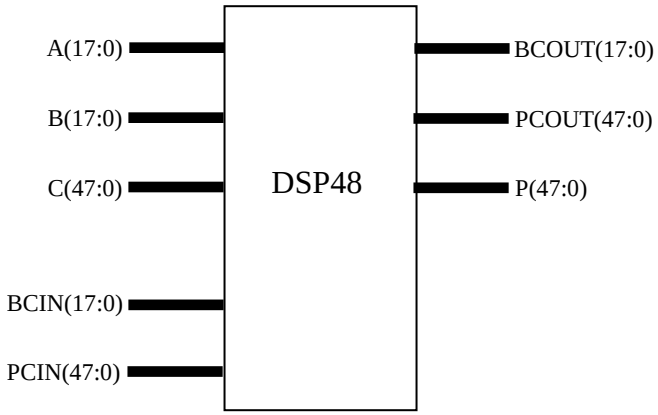


Figure 4.2 – Block diagram of DSP48 component of Virtex-4 library

Before applying the DSP48 and connecting the signals to its ports, we will perform some modifications on the formula (4 – 2). In order to reduce the calculations, we multiply the formula by 2.56. Thus, as we see in formula (4 – 3) the divisor will change from 100 to 256.

$$gray\_channel = (76.8 * red\_channel + 151.04 * green\_channel + 28.16 * blue\_channel)/256 \qquad (4-3)$$

As a result, we remove the need of a division operation. In the digital logic conceptions, one bit shift to the right of a binary number will result in the number divided by 2, two times of right

shift results in 2^2 or 4, and so forth. Therefore, we can simply execute the right shift 8 times to divide the formula by 256 (2^8). As depicted in formula (4 – 3), the new values of the coefficients after multiplying by 2.56 are 76.8, 151.04, and 28.16. Since the numbers which we can apply as inputs to DSP48 cannot be floating point, we have to change the above coefficients to integer numbers. Therefore, we round down the values to 77, 151, and 28, respectively. Thus, the new form of the formulas (4 – 2) and (4 – 3) is like (4 – 4).

$$gray\_channel = (77 * red\_channel + 151 * green\_channel + 28 * blue\_channel)/256 \quad (4-4)$$

After performing the above modifications to the RGB to gray formula, we connect the signals and apply the constant values to DSP48 modules. Three DPS48 modules are used in this project to implement the multiplication operations. Although the addition operation could also be implemented by DSP48 modules, we let the XPS to synthesize the three addition operations. The number of DSP48s can be reduced in order to decrease the hardware cost. The tracking algorithm in this project does not consume much hardware resources. We will see in next chapter that less than fifty percent of the logical gates of the Virtex-4 FPGA are used for all the tracking system. As we do not have constraints in the number of used logic gates, reducing the number of DSP48 components is not covered in this project.

In order to produce the gray video, we connect the least significant ten bits of vid_in to port A of DSP48 and apply the binary number 1001101 to port B. The first ten bits of vid_in carries the red channel of a pixel and the above binary number is equal to 77 in decimal system which is the coefficient of red channel in formula (4 – 4). Respectively, we connect the second 10 least significant bits to port A of another DSP48 and the number 10010111 to port B. Again, the second 10 least significant bits carry the green channel and 10010111 is equal to 151 in decimal system. The third 10 least significant bits of vid_in and the binary number 11100 are connected to port A and B of another DSP48 respectively. These are the blue channel and the binary equivalent of decimal number 28.

We have now the gray entering pixels and we are able to continue the implementation of the algorithm based on the gray channel. The entering pixels from camera to VIODC arrive without any identifier which indicates the position of the pixel in the frame. In other word, the first pixel of a line does not have any difference with the first pixel of a frame. Of course, the intensity

values of the pixels are different, but they do not have any difference according to their location in the frame. Even the pixels in different frames are the same. The pixels are simply a flow of data which arrive in a constant frequency. There are three signals which help to recognize the location of the entering pixel. The “pixel enable” signal becomes active when a new pixel enters the VIODC. The “horizontal synchronization” signal becomes active when a new line occurs in the frame. It means that in the beginning of each line, the horizontal synchronization signal is active. The “vertical synchronization” signal becomes active when the first pixel of a frame arrives. Therefore, the beginning of each frame can be detected by recognizing the activation of vertical synchronization signal. The processing clock of the ML402 in 100 Mhz and the pixel clock of the camera is 26.6. Therefore, each four clock carry almost one pixel. All the three above signals are active low. It means that when the event occurs, these signals change from logical one to logical zero, in other word, a falling edge occurs. They stay zero for a certain time and go back to one. According to the simulation which was explained in the previous chapter and was illustrated in figure (3 – 12), the horizontal synchronization signal stays zero for the duration of one pixel. Thus, when a new line occurs, the horizontal synchronization signal becomes zero, remains zero for one pixel, and goes back to one. Likewise, the vertical synchronization signal becomes zero when a new frame takes place, remains zero for the duration of one line, and returns to one.

As it was explained earlier, the pixels do not have any index or any label which indicates the pixel is located in which position of which line. The solution is to count the lines and the pixels to define the coordinates of the pixels in the frames. Indeed, we have to always keep the number of the lines and pixels. Thus, the first step is to detect the entering pixel to the VIODC. In order to detect a signal with a high frequency like the pixel enable in ML402, VHDL cannot simply look for a rising edge or falling edge of the signal by the language instructions. The signal should be studied in two consecutive clocks and these two situations should be compared. After we detect the pixel enable and count the pixel enable and horizontal synchronization signal to find the 20-pixel by 20-pixel window in the middle of the frame, we look for the pixels which belong to bacterium. We apply a simple threshold to detect the bacterium pixels. We compare the intensity of the entering pixels with the constant threshold equal to 100. If the pixel belongs to a bacterium, we scan the pixels around to find the whole pixels belonging to bacterium. In the next

frame, we look for the bacterium in the pixels around the current position of the bacterium, since it is more probable that the bacterium to be in the adjacent region.

A Random Access Memory (RAM) block is also applied in this project to hold the position of the bacterium in previous and current frame. This block is also available in the Virtex-4 library with the name of RAMB32_S64_ECC. The applied RAM is 64-bit wide and can hold 512 entries. It has two separate input and outputs and profits from a built-in error correction system.

Finally, the project is compiled and synthesized and implemented on the ML402 VSK. The results and statistics will be discussed in the next chapter. The possible future work of this project will also be presented.

CHAPTER 5 RESULTS AND DISCUSSION

This chapter explains the tracking algorithms which were simulated by the Matlab software. The algorithms are compared and briefly discussed about their implement simplicity, robustness and precision. The statistics related to the hardware implementation of the algorithm are also provided. In addition, some possible future works which can use this project as a start point are presented.

5.1 Simulation results

The MTB bacteria which are subject of tracking in this project were followed according to certain algorithms. The algorithms used for simulation are the ones we studies in previous section. Of course, each algorithm requires some modifications because neither the subject of tracking nor the hardware equipments and software programs are the same in this experiment and the mentioned articles. However the modifications are minor and they do not change the basics of the tracking.

In one of the simulations, the bacterium is recognized according to its intensity. First, the algorithm changes the color picture to a gray one. Then, it considers a bacterium as its tracking object. The algorithm looks for the pixels in a search window around this object with the condition that the intensity of these pixels is approximately equal to the intensity of target object pixels. The length and width of the search window is almost three times the length and width of the bacterium, respectively. Since in our experiment the bacteria do not move more than their size in two consecutive frames, the search window size of about three times of the size of bacteria is a good approximation. Indeed, the fact that a single bacterium covers itself in two consecutive frames helps the algorithm for some decisions in certain conditions.

The mean shift algorithm looks for the distribution of the pixels with the same color in the search window. This means that the algorithm finds how many times each color is used in the pixels of target object. Then, it searches for objects with pixels of the same color in the search window in the second frame. When the least distance (difference between the colors of the target object and the colors of the candidate objects in the search window) is detected, it will be the target object in the next frame. This algorithm is very efficient in general because it works with not only bacterium as tracking object but also with many other kinds of objects. The problem of this

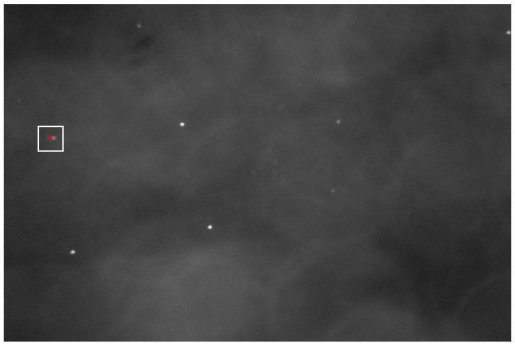
algorithm is in the scenes with the same colors in the foreground and background. Since the mean shift algorithm calculates the histogram of the search window in order to find its color distribution in each iteration, this algorithm is more complicated to implement. It is also longer than other simulated algorithms in terms of number of lines. This algorithm is not implemented on the hardware in this project. Furthermore, the simulation is performed on a recorded video sequence. Thus, we do not have any documented result about the speed, hardware costs and software costs of this algorithm.

As another experiment, the algorithm proposed in [9] was implemented. The algorithm finds a pixel of bacterium. Then it examines the adjacent pixels to decide whether they are bacterium pixels or not. This algorithm continues until all the pixels of a bacterium are recognized. This algorithm is very simple, low in calculation, and easy to implement, while it is not efficient and powerful in various situations.

Now, we see one of the real trade-offs explained in section 1.2 which happens in our project. This is taking place between the efficiency of the algorithm and the implementation time, hardware cost, and tracking speed.

The implementation of the above tracking algorithms in Matlab software on a recorded video sequence shows their robustness. The velocity and orientation of bacterium which are measured by data retrieved while tracking in Matlab are compared to the numbers provided by Martel et al. in [32], which concerns magnetotactic bacteria (MTB) as nanorobots in medical treatments. They have registered a record of 300 μm/s as the velocity of MTB. Their study shows that about 50 percent of these bacteria move in a velocity between 180 to 240 μm/s. The other 50 percent move between 30 to 180 μm/s and between 240 to 300 μm/s. The diameter of this type of bacterium is between 1 to 2 micrometers.

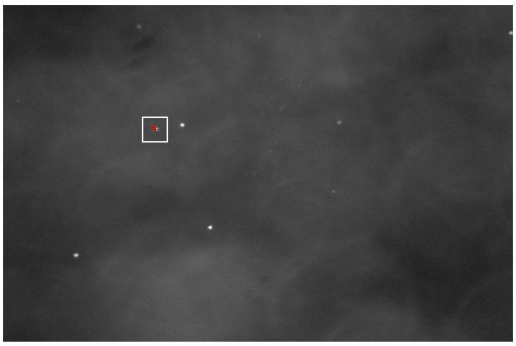
It should be noted that the velocity and angle indicated in figure 2.7 are according to the definitions of velocity and orientation explained in section 2.3 and formula (2 – 17). The velocity (micrometers per second) and orientation (degree) of the target is shown under each section in Figure 2.7(a) to 2.7(d).



(a) frame number : 5

Velocity = 212

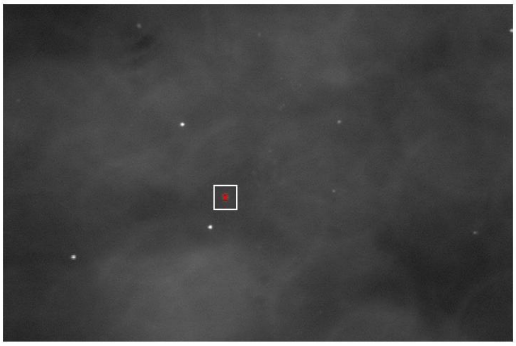
Angle = -8.13



(b) frame number : 25

Velocity = 228

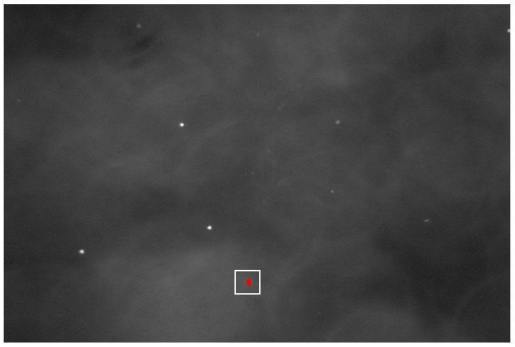
Angle = -23.2



(c) frame number : 45

Velocity = 162

Angle = -68.2



(d) frame number : 65

Velocity = 180

Angle = -90

Figure 5.1 – Velocity and orientation of the target is shown under the frame number at four frames of a 70-frame sequence

The speed of the target bacterium in all 70 consecutive frames is calculated and depicted in Figure 5.1. As indicated in [32], the velocity of the bacterium should be between 180 to 240 micrometers per second. According to Figure 2.8, the velocity in 60 frames of 70 frames is calculated between 180 and 240. Since the tracking subject is an alive creature, the increment or decrement of its velocity is dependent to its living situations such as the liquid where it lives, temperature, light, etc. Thus, the increment or decrement in the velocity of bacterium seems to be normal. It should be mentioned that the descending graph in figure 5.2 covers less than 3 seconds.

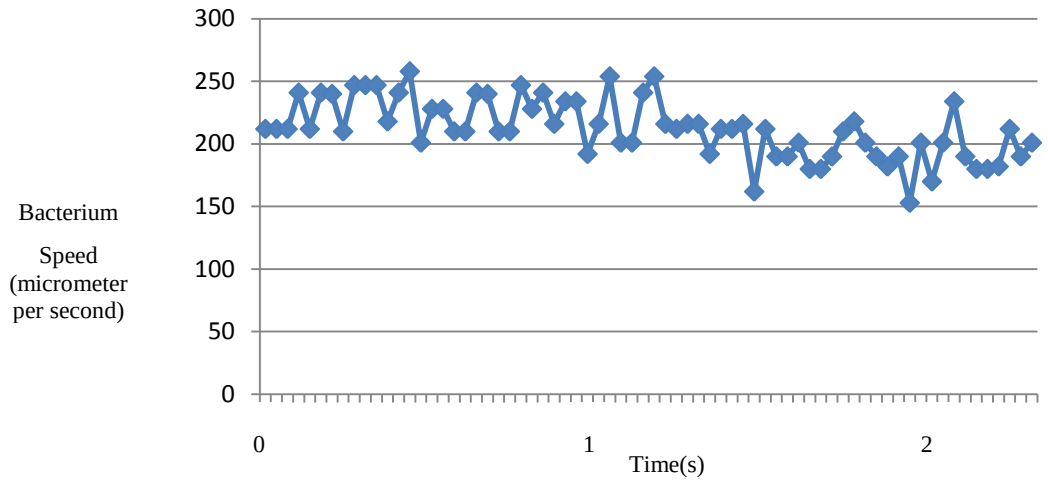


Figure 5.2 – Velocity of a target bacterium in 70 consecutive frames.

The simulation of MTB tracking is 100% robust since the bacterium was not missed in any of the above frames and the algorithm is able to follow the trajectory of the bacterium during all of the simulation time. The simulation indicates 100% precision in 2-dimensional movements of the bacterium. Two dimensional movements are possible if the chamber where the bacterium moves is manufactured so that the bacterium moves only in two dimensions (the bacterium does not move along the microscope view axis). Supposing the above condition, all the pixels which are recognized by the human eye as the bacterium pixels are also detected as the bacterium pixels by the simulator algorithm.

Exceptional cases like occlusion and movements in the third dimension need further study of the solutions and more precise algorithms. This can be considered as one of the future works related to this project. Some other special cases can be the situations with more than one tracking object or the existence of changes in the condition of the tracking scene, illumination, etc.

5.2 Analysis of tracking algorithms implementation

As we explained in sections 2.3 and 2.4, all tracking algorithms are different because the conditions under which they are applied are different. Furthermore, the designers do not provide all the details of their work in the scientific articles or journals. Thus, each algorithm has to introduce some measures of creativity. Of course, every work has some advantages and disadvantages. The following table summarizes the algorithms which were explained in previous section and it compares some of their development and implementation specifications.

Table 5.1 – Comparison of some tracking algorithms and their implementation.

Algorithm	Advantage	Disadvantage
Wang et al. [24]	1. Tracking in two imaging modes: bright-field and phase contrast. 2. Low in calculation algorithm.	1. The algorithm works just for the cells with indicated conditions of lighting (hasn't discussed about color images, multi-tracking, etc). 2. Implemented on personal computer (big size of hardware).
Oku et al. [25]	1. Different imaging methods: bright-field, dark-field, differential interference contrast. 2. The algorithm is implemented with two different magnifications of 5x and 20x. 3. Low in calculation algorithm.	1. The algorithm works just for the cells with indicated conditions of lighting 2. Implemented on personal computer (big size of hardware). 3. As it captures gray scale images, it may have constraints with color images.
Taboada et al. [26]	1. The algorithm is able to perform tracking for the images of low quality and in the conditions where too much noise exists. 2. The algorithm is low in calculation.	1. The algorithm works just for the cells with indicated conditions of lighting. 2. Implemented on personal computer (big size of hardware).
Prasad et al. [27]	1. Multi-object tracking. 2. It seems that any object (not limited to cells) can be tracked by suggested algorithm; however, it is not indicated in the article.	1. High in calculation algorithm. 2. Needs a great amount of memory for implementation.
Meanshift	1. A general algorithm for tracking any moving object. 2. The algorithm does not depend on color, size or form of tracking object.	1. High in calculation algorithm. 2. Misses the target if there are other objects with the same color in the scene.
Our tracking system	1. Independent of the colors in the scene. 2. Very low in calculation. 3. Implemented on special purpose processor, smaller size and faster processor.	1. It works for tracking a single cell. 2. It's designed for moving of the cell in two dimensions.

5.3 Tracking algorithm hardware consumption

This section provides the statistics about the hardware utilization of the tracking algorithm. The number of logical modules which the Virtex-4 FPGA provides, the number which are consumed in this project and the consumption percentage are presented in Table 5.1. The statistics are extracted from the XPS software after configuration file generation.

Table 5.2 – Utilization of the Virtex-4 FPGA device

Logical modules	Number consumed out of number provided	Percentage consumed
Number of Slices	6152 out of 15360	40%
Number of Slice Flip Flops	5872 out of 30720	19%
Number of 4 input LUTs Number used as logic Number used as Shift registers Number used as RAMs	9570 out of 30720 8757 557 256	31%
Number of bonded IOBs	181 out of 448	40%
Number of FIFO16/RAMB16s Number used as RAMB16s	64 out of 192 64	33%
Number of DSP48s	15 out of 192	7%

5.4 Future works

As mentioned in section 5.1, the variations of the tracking algorithm under different environmental circumstances can be developed. In addition to this tracking object, any kind of movements may be the subject of tracking. The moving items can be more than one.

The ML402 has USB and Ethernet RJ-45 port hardware interfaces. Thus, the VSK can be connected to other circuits to work under a network on a very high speed up to 1000 mbps. The VIODC has different video input and output ports. It can be connected to different analog or digital video devices.

With the current settings, the bacterium tracking can also be developed to perform artificial intelligence algorithms.

5.5 Conclusion

This project motivates the designer to study different video processing algorithms, especially tracking algorithms. In order to understand the tracking algorithms the designer should have mastery on the subject.

The Xilinx ML402 VSK is a very powerful board for implementation of the video applications. However, the documentation about the VSK and about the video interface details is not sufficient. Therefore, much work is needed to discover it. In some periods we had to make an assumption and further accept it as true or ignore it as a false theory. It is generally a good starter kit for video applications since the driver interfaces of the ports are already provided by the company.

Finally, we believe this work will be useful for students and researchers who decide to continue any of the proposed future works or projects individually defined.

REFERENCES

[1] D. Jung, Y. Yun, and J. Choi, "Foot motion tracking for motion capture from stereo cameras," *ICCE '09. Digest of Technical Papers International Conference on Consumer Electronics,* 10 – 14 Jan. 2009

[2] D. Gehrig, H. Kuehne, A. Woerner, T. Schultz, "HMM-based human motion recognition with optical flow data," *Humanoids 2009. 9th IEEE-RAS International Conference on Humanoid Robots*, pp. 425 – 430, 7 – 10 Dec. 2009.

[3] Lili Huang, "Real-time multi-vehicle detection and sub-feature based tracking for traffic surveillance systems," *2nd International Asia Conference on Informatics in Control, Automation and Robotics (CAR),* pp. 324 – 328, 6 – 7 March 2010.

[4] J. Zhu, Lao Yuanwei, Y.F. Zheng, "Object tracking in structured environments for video surveillance applications," *IEEE Transactions on Circuits and Systems for Video Technology*, pp. 223 – 235, Feb. 2010.

[5] B. Moufarraj and S. Martel, "System for the validation of cell-tracking algorithms using on-demand simulated optical microscope images," *Proceedings of the 28th IEEE EMBS Annual International Conference*, 30 Aug. – 3 Sept. 2006

[6] K. Fukushima, "Recognition of Occluded Patterns: A Neural Network Model," *Proceedings of the IEEE-INNS-ENNS International Joint Conference on Neural Networks*, pp. 135 – 138, 2000.

[7] X. Liu, B. Zou, J. Sun, "A New Approach to Separating Touching Spots in Particle Images," *Proceedings of 2004 International Symposium on Intelligent Multimedia, Video and Speech*, pp.133 – 136, 2004.

[8] S. J. McKenna, S. Jabri, Z. Duric, A. Rosenfeld, H. Wchsler, "Tracking Groups of People," *Computer Vision and Image Understanding*, Volume 80, pp. 42 – 56, October 2000.

[9] M. Mankiewicz, M. Mohammadi, S. Martel, "Motion Tracking and Analysis System for Magnetotactic Bacteria," *International Symposium on Optomechatronic Technologies*, 2007

[10] S. Martel, C. C. Tremblay, S. Ngakeng, and G. Langlois," Controlled manipulation and Actuation of Micro-objects with Magnetotactic Bacteria," *Applied Physics Letters*, Vol. 89, No.1, pp. 233804-6, 2006

[11] S. Martel and M. Mohammadi, "HIGH THROUGHPUT CONTROLLED BACTERIAL TRANSPORT USING GEOMETRICAL FLUIDIC MICROCHANNELS OR 3D MICROFIBERS STRUCTURES," *11th International Conference on Miniaturized Systems for Chemistry and Life Science*, 2007.

[12] Z. Lu and S. Martel, "CONTROLLED BIO-CARRIERS BASED ON AGNETOTACTIC BACTERIA," *14th International Conference on Solid-State Sensors and Actuators and Microsystems*, pp. 683 – 686.

[13] E. Trucco and K. Plakas, "Video tracking: A concise survey," *IEEE Journal of Oceanic Engineering*, vol. 31, no. 2, pp. 520 – 529, Apr. 2006

[14] P. S. Maybeck, *Stochastic Models, Estimation, and Control*. London, U.K.: Academic, 1979, vol. 1, 2

[15] Z. Hua-ping, W. Zhan-qing, W. Chao-zhong, W. Chuan-ting, F. You-fu, "Target tracking using Kalman Filter Embedded Trust Region," ICTM '09. *International Conference on Test and Measurement*, pp. 119 – 122, 5 – 6 Dec. 2009.

[16] A. Ali, S. M. Mirza, "Object Tracking using Correlation, Kalman Filter and Fast Means Shift Algorithms," *2nd International Conference on Emerging Technologies*, pp. 174 – 178, Nov. 2006.

[17] H. Wang and M. Brady, "Real-time corner detection algorithm for motion estimation," *Image and Vision Computing*, vol. 13, no. 9, pp. 695 – 705, 1995.

[18] T. Vieville and O. Faugeras, "Robust and fast computation of edge characteristics in image sequences," *International Journal of Computer Vision*, vol. 10, no. 2, pp.153 – 179, 1994.

[19] R. Deriche and O. Faugeras, "Tracking line segments," *Proceedings of the First European Conference on Computer Vision*, 1990, pp. 259–268.

[20] B. Bascle and R. Deriche, "Region tracking through image sequences," *Proceedings of IEEE International Conference in Computer Vision,* 1995, pp. 302–307.

[21] P. Wunsch and G. Hirzinger, "Real-time visual tracking of 3-d objects with dynamic handling of occlusions," *Proceedings of IEEE International Conference on Robotics and Automation,* April 1997, pp. 2868–2873.

[22]T. Cootes, A. Hill, C. Taylor, and J. Haslam, "The use of active shape models for locating structure in medical images," *International Journal on Computer Vision,* vol. 12, no. 6, pp. 356–366, 1994.

[23] N. Mekuz, K. G. Derpanis, and J. K. Tsotsos, "Adaptive Step Size Window Matching for Detection," *18th International Conference on Pattern Recognition ICPR 2006,* vol. 2, pp. 259 – 262, 2006

[24] Pengbo Wang, Chenglu Wen, Wei Li, and Ying Chen, "Motile microorganism tracking system using micro-visual servo control," *Proceedings of the 3rd IEEE international conference on Nano/Micro Engineered and Molecular Systems,* pp. 178 – 182, 6 – 9 Jan. 2008.

[25] H. Oku, N. Ogawa, K. Hashimoto, and M. Ishikawa, "Two-dimensional tracking of a motile micro-organism allowing high-resolution observation with various imaging techniques," *Rev. Sci. Instr.,* vol. 76, no. 3, Mar. 2005

[26] Taboada B., Poggio S., Camarena L., and Corkidi G., "AUTOMATIC TRACKING AND ANALYSIS SYSTEM FOR FREE-SWIMMING BACTERIA," *Proceedings of the 25th Annual International Conference of the IEEE EMBS, Cancun, Mexico,* September 17-21, 2003

[27] B. Prasad, S. Du, W. Badawy, and K. V I S Karel, "A real-time multiple-cell tracking platform for dielectrophoresis (DEP)-based cellular analysis," *Meas. Sci. Technol. 16,* pp. 909 – 924, 24 Feb. 2005

[28] Xilinx Inc., Video Starter Kit, UG217, v1.5, October 26, 2006.

[29] Xilinx Inc., Video Input/Output Daughter Card, UG235, v1.2.1, October 31, 2007.

[30] Xilinx Inc., Video Starter Kit Reference Guides [Online]. Available: http://www.xilinx.com/products/devboards/reference_design/Video_Starter_Kit.htm

[31] Xilinx Inc., Virtex-4 Libraries Guide for HDL Designs, UG619, v 11.3, September 16, 2009.

[32] S. Martel, M. Mohammadi, O. Felfoul, Z. Lu, and P. Pouponneau, “Flagellated magnetotactic bacteria as controlled MRI-trackable propulsion and steering systems for medical nanorobots operating in the human microvasculature,” *The International Journal of Robotics Research*, Vol. 28, No. 4, April 2009, pp. 571 – 582.