République Algérienne Démocratique et populaire

Ministère de l'Enseignement supérieur et de la Recherche Scientifique

Université A.MIRA de Bejaia

Faculté de Technologie

Département de génie Electrique

Projet de fin d'études

En vue de l'obtention du diplôme de Master en Electronique
Option : Automatique

THEME :

Commande par carte FPGA d'une modélisation SolidWorks appliquée à une main artificielle DLR/HIT II

Présenté par :

Mr. BIZRICHE Bilal

Mr. AIBOUT Mansour

Encadreur :

Mr. B.MENDIL

Mlle. S.MEZZAH

Examiné pas :

Mr. H.HADDAR

Mr. M.SABI

Promotion 2013-2014

Nous dédions ce travail à

Nos parents

Nos frères et sœurs

Nos amis

Ainsi qu'à tous ceux que nous aimons

Remerciements

Nous tenons, avant tout, à remercier DIEU, tout clément, tout puissant, de nous avoir donné la force de réaliser notre travail.

Nos remerciements vont exceptionnellement à Monsieur B.MENDIL, chargé de cours à l'université abdrahmane mira de bejaia, pour son aide, son suivi, ses conseils et directives et pour son dévouement.

Un spécial remerciement pour Issaad Massinissa qui nous a beaucoup aidés.

Nos remerciements vont aussi aux membres de jury, d'avoir bien voulue accepter d'examiner notre travail.

Nos remerciements, vont au personnel de l'université et à toute personne dévouée au service de l'université abdrahmane mira de bejaia.

Enfin, nos remerciements vont à toute personne ayant contribuée, de près ou de loin, à réaliser ce travail.

Résume

Notre projet a pour objectif, la commande d'une modélisation solidworks par carte FPGA en prenant comme application la commande de la main DLR/HIT II et en utilisant le système d'exploitation temps réel Xilkernel embarqué sur un système sur puce (SoC). Et ceci après avoir interfacé SolidWorks avec le circuit FPGA à l'aide d'une application développée en Visual Basic en exploitant l'API SolidWorks.
Mots clés : Main artificielle, API SolidWorks, SoC, FPGA, système d'exploitation temps réel.

Summary

Our project aims to control a solidworks modeling with FPGA board, taking as an example, the control of the hand DLR / HIT II using the real-time operating system "Xilkernel" embedded on a system on chip (SoC). And this after interfacing SolidWorks with FPGA Using an application developed in Visual Basic by exploiting the SolidWorks API.
Key words : Artificial hand, SolidWorks API, SoC, FPGA, RTOS.

Liste des figures

Table de matières

Chapitre III : Ressources hardware et software de l'application

Chapitre IV : Implémentation

Abréviations

API : Application Programming Interface

BLDC: BrushLess Direct Current

BRAM: Block of Random Access Memory

BSB: Base System Builder

BSP: Board Support Package

DLR/HIT : German Aerospace Center/Harbin Institute of Technology

EDK: Embedded Development Kit

ELF: Executable and Linkable Format

FPGA: Field Programmable Gate Array

ISE: Integrated Software Environment

LibGen: Library Generator

LMB: Local Memory Bus

MHS: Microprocessor Hardware Specification

MSS: Microprocessor Software Specification

OPB: On chip Peripheral Bus

PlatGen: Platform Generator

PLB: Processor Local Bus

RTOS: Real Time Operating System

SDK: Software Development Kit

SoC: System on Chip

UCF : User Constraint File

XMK : Xilinx Micro-Kernel

XMP: Xilinx Microprocessor Project

XPS: Xilinx Platform Studio

Introduction générale

La représentation du modèle réel s'est depuis longtemps imposée comme déterminante pour la compréhension des phénomènes complexes, pour leur prévision et leur éventuel contrôle. Cela vaut dans bien des domaines, qu'il s'agisse des sciences environnementales, de la Physique, de la Médecine, de la Biologie ou de la conception de nouvelles technologies comme c'est le cas pour la Robotique. L'évolution récente des outils de modélisation et des moyens de calcul a permis des avancées significatives en matière de performance. [1]

Dans tous les domaines cités, la construction de modèles virtuels ainsi que leur utilisation pour une simulation réaliste et prédictive représentent un réel défi quant à la maîtrise de la complexité. De nos jours, les prototypes virtuels sont devenus une étape incontournable avant d'aboutir au modèle réel. [2]

La création de modèles virtuels, reposant sur la visualisation d'objets ou de phénomènes complexes, est un domaine en plein essor. Leur utilisation se généralise dans les milieux industriels avec le développement du prototypage virtuel rapide, du design collaboratif et l'utilisation d'images virtuelles pour les études d'impact (projets architecturaux par exemple). La possibilité d'utiliser ces maquettes pour planifier et simuler des interventions humaines constitue un apport déterminant, par exemple dans le domaine biomédical. [2]

D'un point de vue méthodologique, modélisation, simulation et visualisation de mondes réels ou virtuels demandent une gestion intelligente de la complexité et doivent s'appuyer sur des approches efficaces visant leurs maîtrises. Simulations du monde réel et créations de prototypes virtuels présentent donc des lignes de convergence fortes, aussi bien sur le versant applicatif que sur celui des outils utilisés. [1]

D'une autre part, les exigences que doit remplir la partie électronique pour commander le prototype virtuel vont être les mêmes que celles pour commander le prototype réel. On peut citer par exemple : la précision, le travail en temps réel, le parallélisme dans le traitement de données et la miniaturisation, qui sont des caractéristiques importantes des systèmes de nos jours.

Sur le plan matériel, l'apparition des Systèmes on Chip a révolutionné le monde de la microélectronique.

Le prototypage rapide permet d'ajouter ou d'enlever des périphériques et des ports d'entrées/sorties selon nos besoins sans avoir à modifier ou à ajouter du matériel qui pourra coûter une fortune. [8]

Notre travail consiste à commander un prototype virtuel de la main DLH/HIT II conçu sous SolidWorks en utilisant un système embarqué implémenté sur carte FPGA à base du processeur soft MicroBlaze en utilisant le système d'exploitation temps réel Xilkernel .
Pour une meilleure présentation de notre travail, nous avons organisé le mémoire en quatre chapitres :

Le premier chapitre présente l'état d'art, le modèle choisi, le modèle mathématique, l'environnement Solidworks et les critères de choix de cet environnement, la conception du modèle de la main sous Solidworks. Ainsi que la relation entre le modèle sous SolidWorks et le modèle mathématique.

Le deuxième chapitre prendra en compte l'API (**A**pplication **P**rogramming **I**nterface) Solidworks et la possibilité d'interaction avec le milieu extérieur ainsi que le flot de conception de notre API en Visual Basic.

Dans le troisième chapitre, un tour d'horizon sera donné sur les systèmes on chip, l'environnement de développement EDK, l'approche suivie pour la conception de l'architecture matérielle.

Le quatrième chapitre portera sur la mise en œuvre des parties hardware et software du projet.

Enfin, nous terminons par une conclusion générale et les perspectives envisagées.

I.1. Introduction

La conception de mains artificielles est un thème qui est devenu très intéressant pour plusieurs laboratoires de recherche à travers le monde et ce, après la deuxième guerre mondiale, où plusieurs soldats étaient amputés de leurs bras. Ce qui a fait croître la demande des prothèses dites primaires d'une forme très modeste qui n'avait qu'un joint équipé d'un simple crochet, et qui, en réalité, ne représentaient qu'une consolation morale. Il fallait attendre jusqu'en 2004 pour assister à un modèle réaliste mis au point par le "*Rehabilition Institute of Chicago*". Ce nouveau bras est une merveille technologique. Car, après avoir connecté les nerfs à des périphériques, il peut être commandé par la pensée [1].

Ce chapitre traitera l'état de l'art de la conception d'une main artificielle. On y abordera l'outil de conception ***SolidWorks*** et ses différentes fonctionnalités. On présentera les étapes qui ont été suivies lors de la conception de notre prototype et enfin la modélisation géométrique de ce dernier, étape primordiale pour la commande électronique qui sera l'objet des chapitres suivants.

I.2. Etat de l'art

Créer une main robotique, semblable plus ou moins à la main humaine, est le rêve de tout chercheur dans ce domaine. Dans ce qui suit, quelques mains robotiques seront présentées ainsi que leurs caractéristiques. On se concentrera ensuite du modèle choisi pour notre étude.

I.2.1. Modèle de Barrett

Produite par « BarrettTechnology », elle est basée sur une conception développée à l'université de Pennsylvanie en 1988. C'est une main mécanique composée de trois doigts. Un doigt est fixe et les deux autres peuvent s'écarter jusqu'à 180 degrés par rapport à la paume (le doigt 3 est le doigt fixe et les doigts 1 et 2 tournent autour de la paume), et ceci d'une manière synchrone. Chaque doigt possède deux articulations rotoïdes (*Fig.I-1*). La main est commandée par quatre moteurs qui activent sept liaisons au total. [4]

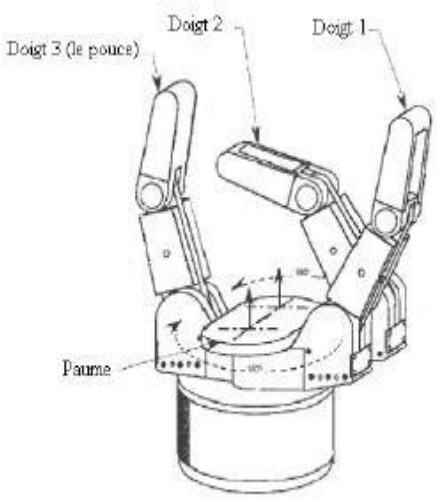

Figure I-1 Modèle de Barrett

I.2.2. La main SARAH

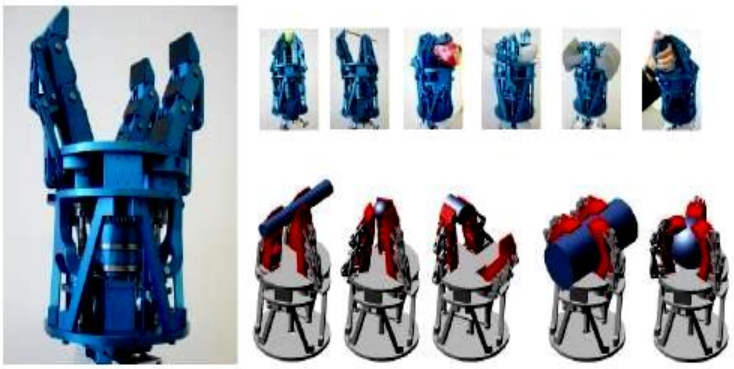

Figure I-2 Main SARAH

La main SARAH (Self-Adaptive Robotic Auxiliary Hand), est une main à trois doigts comme celle de Barrett (Figure I-2). Cette main est installée sur la station spatiale internationale depuis 2008. Le but de cette main est la saisie des objets dans l'espace. Les trois doigts sont synchronisés de telle sorte qu'ils s'ouvrent ensemble ou se ferment ensemble. Cette main est motorisée par seulement deux moteurs indépendants, un pour l'ouverture/fermeture et un autre pour l'orientation des doigts. Cette main possède 10 articulations. [4]

I.2.3. La main Robonaut

La main *Robonaut* a été développée par la NASA au centre de l'espace Johnson. Cette main possède 5 doigts et un total de 14 degrés de liberté. Sa taille est équivalente à 95% à celle d'une main humaine (Fig. I-3). L'index, le majeur et le pouce sont considérés comme les doigts primaires de manipulation. Ils sont capables de réaliser les mouvements d'abduction et d'adduction. L'annulaire et l'auriculaire sont décalés par rapport aux autres doigts ce qui leur donne la capacité de s'enrouler autour de l'objet à saisir. Ils sont capables d'exercer des efforts de serrage importants. Cette main comprend 14 moteurs au total situés tous au niveau de l'avant-bras. [4] [5]

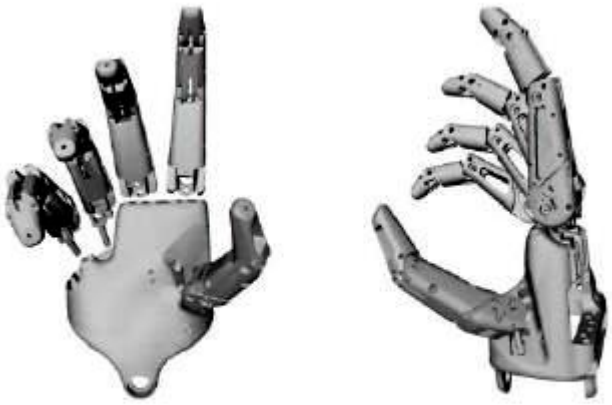

Figure I-3 Main Robonaut

I.2.4. La main de Shadow

Cette main créée par la compagnie *Shadow Robot* en Angleterre est, à ce jour, la main robotique la plus proche de la main humaine. Elle contient 24 degrés de liberté qui sont contrôlés par des muscles pneumatiques situés tous au niveau de l'avant- bras (Figure I-4). Cette main robotique peut être utilisée dans plusieurs applications : télé-présence, mobilité virtuelle, industrie…etc. Les liaisons des doigts sont actionnées par desmuscles pneumatiques

antagonistes. Par contre, quelques articulations sont actionnées par un seul muscle et le mouvement opposé se fait à l'aide d'un ressort. [4]

Beaucoup d'autres modèles de mains artificielles ont été établis dans différents laboratoires à travers le monde. Le tableau suivant donne les caractéristiques fondamentales de quelques modèles de mains robotiques déjà réalisées, telles que : le nombre de doigts, le nombre de degrés de liberté et la taille par rapport à la main humaine.

Figure I-4 Main Shadow

Tableau 1-1 : Résumé des caractéristiques principales de quelques modèles [6]

Main	**Nombre de Doigts**	**Taille de la main par rapport à la main humaine**	**Degrés de liberté**	**Nombre d'actionneurs**	**Poids en Kg**	**Force (N)**
Main humaine	5	1	22+2 poignet	38 ext+int	Environ 0.4	>300
Salisbury	3	1.2+contrôle	9	12 ext	1.1	44
Utah/MIT	4	2+contrôle	16	32 ext		31
DIST	4	1.5+ contrôle	16	20 ext	1	
Sugiuchi	5		17			
Anthrobot	5	1+ contrôle	20	16 ext	4.5	
SDSU	5		15	6 ext	2	15/doigts
NTU	5		17		1.57	
Shadow	4	1.2+ contrôle	22+2 poignet		4	
BUAA	4	1+ contrôle	16		1.4	
Goldfinger	4	1.5+ contrôle	12		2.27	
Barrett	3	1.2	8	4 int	1.18	20/doigt
Laval 1	3	2	12	6 ext	9	687
Laval 2	3	1.5	10	2 ext		
Robonaut	5	1.2+ contrôle	12+2 Poignet	12+2 ext		
DLR II	**4**	**1.5+ contrôle**	**13**	**13 int**	**1.8**	**30/doigt**
SARAH	3		2	2		

Après la comparaison entre plusieurs mains robotisées par rapport à leurs nombres de doigts, de degrés de liberté (dextérité) d'actionneurs et leurs emplacements, leurs volumes et leurs poids, nous avons choisi un modèle qui est le **DLR/HIT II,** du *centre aérospatial Allemand,* pour sa modularité, son volume et son poids faibles. Une modèle, par simulation virtuelle sous Solidworks, de cette main sera présentée dans le prochain point.

I.2.5. La main DLR/HIT II

La main *DLR II* a été conçue pour qu'elle soit puissante et très fiable (Figure I-5). Le nombre de câbles entre la main et le microprocesseur principal a été considérablement réduit (de plus de 400 à 12 seulement). La combinaison optimale des moteurs BLDC (Brush Less DC), du réducteur harmonique, des courroies et des pignons différentiels ont fait que la force de bout du doigt atteint les 30 N. Le degré de liberté supplémentaire du pouce a augmenté non seulement la puissance de la saisie mais également permet des manipulations fines [7].

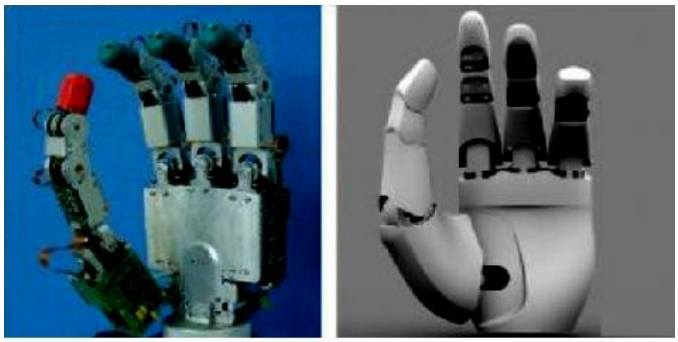

Figure I-5 Main DLR/HIT II

La main DLR/HIT II, est définie par :

- La modularité : ce qui signifie que tous les doigts sont identiques. Ce qui facilite la simulation et la fabrication.
- Un total de 4 doigts et 13 degrés de liberté.
- Chaque doigt comporte 3 degrés de liberté et 4 articulations. Les deux premières se trouvent à l'extrémité de la paume et assurent le mouvement d'abduction/adduction et flexion/extension. Les deux autres articulations sont mécaniquement couplées et ont un seul degré de liberté.
- Les actionneurs sont des moteurs sans brosses à courant continu et sont directement intégrés soit à la structure du doigt soit au niveau de la paume. Ce qui est une caractéristique importante en les comparants aux structures qui emploient des actionneurs pneumatiques dont l'emplacement est au niveau de l'avant-bras, ce qui mène à utiliser plus de volume.

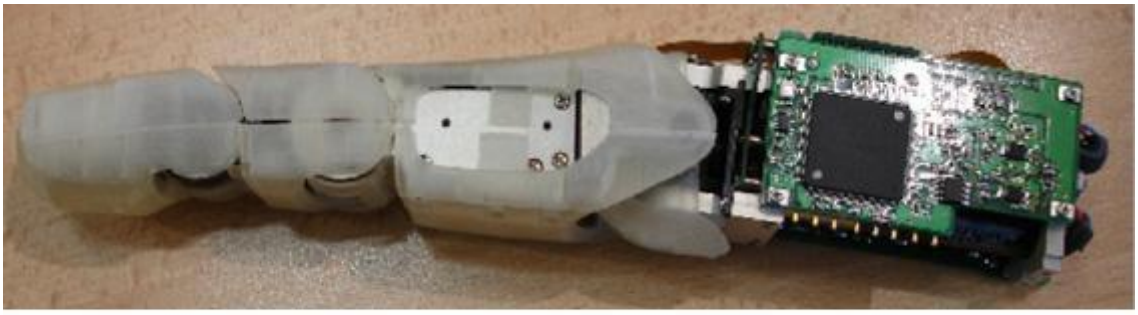

Figure I-6 Un doigt de la main DLR/HIT II

I.3. La conception de la main dans l'environnement SolidWorks

I.3.1. L'environnement SolidWorks

Solidworks est un logiciel commercial largement utilisé pour la modélisation et la conception assistée par ordinateur. Il est basé sur la définition des paramètres des composants et des fonctionnalités. Il peut être utilisé d'une façon très intuitive [8]. La figure I-7 donne une vue d'ensemble de l'environnement SolidWorks ainsi que les différentes parties le constituant

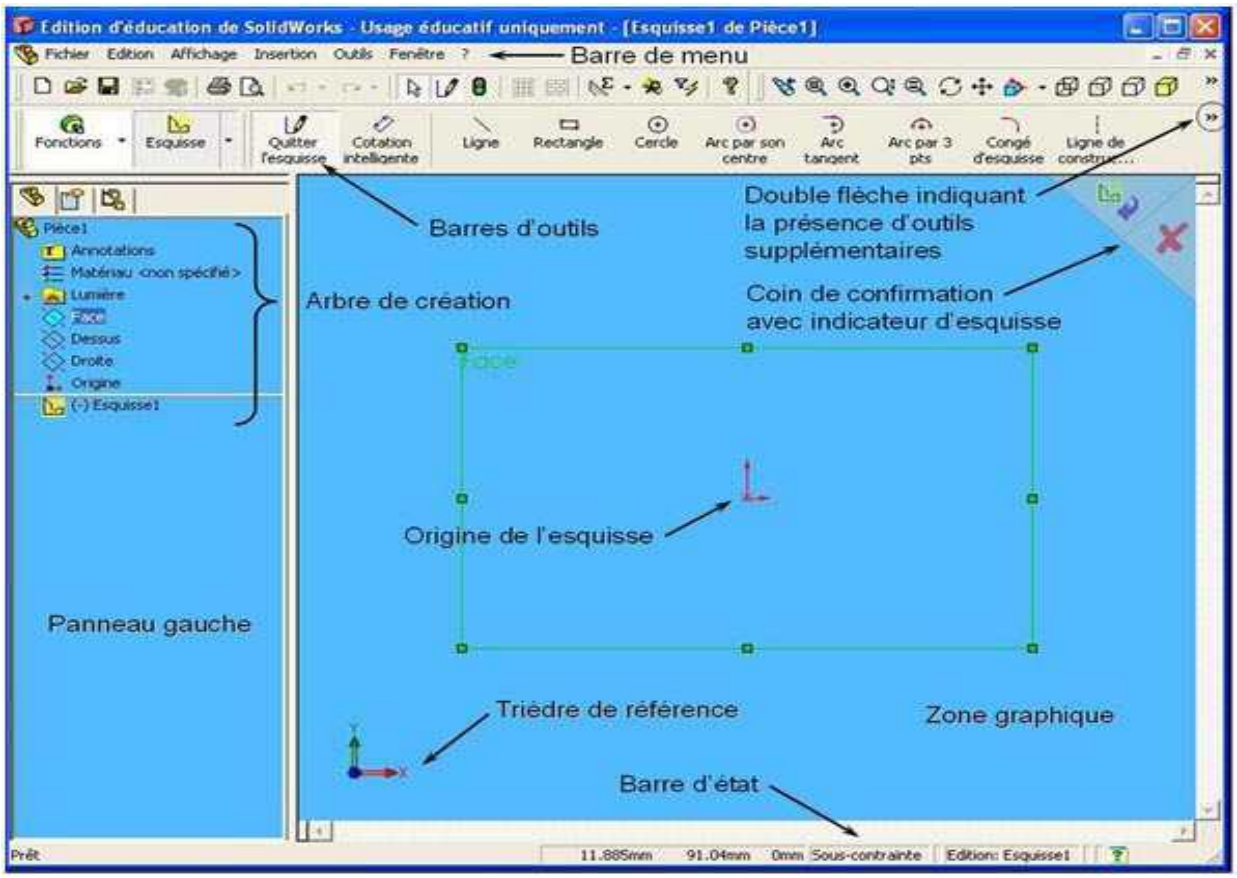

Figure I-7 Vue d'ensemble de l'environnement SolidWorks [8]

I.3.2. Le choix de SolidWorks

Notre choix pour le logiciel SolidWorks était fait pour les raisons suivantes :

- La facilité d'utilisation du logiciel.
- La bibliothèque assez complète pour le choix des matériaux utilisés pour les différentes pièces et aussi des pièces pré-utilisables.
- Les outils performants de simulation des contraintes imposées aux pièces et aux assemblages conçus.
- La simulation des mouvements des pièces et des assemblages les unes par rapport aux autres, sans oublier la détection de collision et l'interaction mécanique entre les volumes.
- La simulation de la masse et le calcul des centres d'inerties et aussi la simulation des forces appliquées, la rigidité des modèles …

Toute ces fonctionnalités permettent de s'approcher le plus possible du modèle réel souhaité qui est l'un des buts soulignés dans notre travail.

SolidWorks nous permettra donc non seulement de concevoir notre prototype mais aussi de l'animer, de l'étudier et de le valider, nous l'utiliserons aussi pour calculer les différentes caractéristiques mécaniques de notre main robotisée comme : la masse, le centre de gravité et le tenseur d'inertie.

Rappelons juste que cette main a été déjà réalisée sous cet environnement au niveau de l'Ecole National Polytechnique. Néanmoins nous donnerons les principales étapes qui ont été suivies :

La construction de la main DLR II revient en la construction d'un seul doigt, ce dernier est dupliqué pour créer la main. Il est composé de deux parties essentielles : la base et l'unité.

I.3.3. La base

Elle se compose des deux premiers moteurs, leurs supports et le pignon différentiel. Elle assure au doigt les mouvements d'abduction/adduction et flexion/extension, qui seront assurés par le couplage, à l'aide d'un pignon différentiel (Figure I-9), de deux moteurs sans bagues à courant continu identiques (Figure I-8). Les deux pièces de cette base sont donc le pignon différentiel et les deux moteurs ainsi que leurs réducteurs inclus. Les figures I-8 à 10 illustrent ces pièces sous Solidworks :

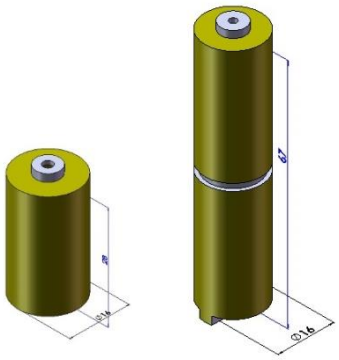

Figure I-8 Les Moteurs

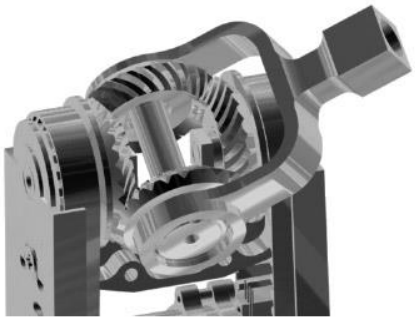

Figure I-9 Le peignon différentielle

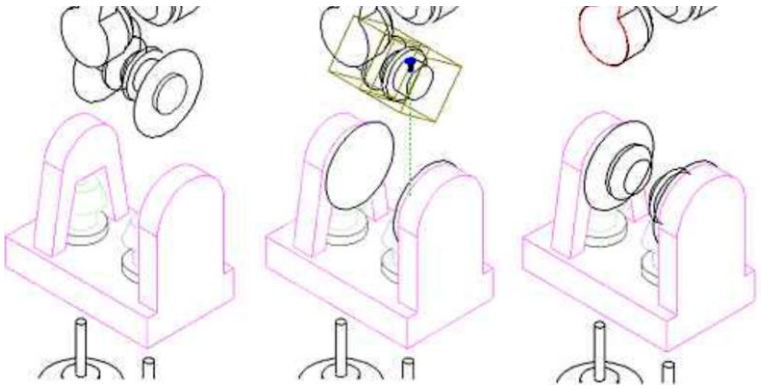

Figure I-10 : Montage du pignon différentiel

Le pignon différentiel est fixé au support supérieur des moteurs, ces derniers sont aussi fixés par le support inférieur pour les empêcher de tourner sur eux-mêmes une fois sous tension. La figure 1.10 défile les étapes du montage des différents éléments, le pignon satellitaire est fait de sorte que la fixation de l'unité du doigt soit assurée. Figure I-11 montre l'aspect final de la base du doigt.

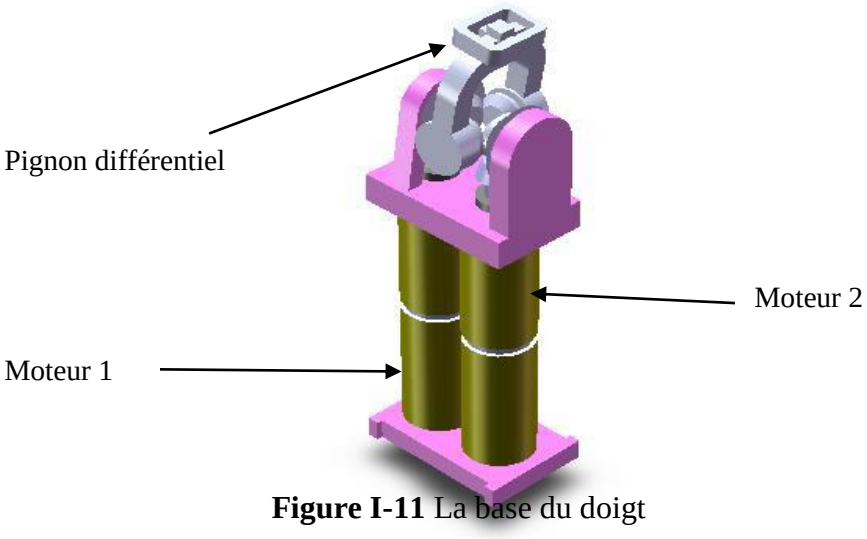

Figure I-11 La base du doigt

I.3.4. L'unité

C'est la partie motrice du doigt. Elle a un degré de liberté et deux articulations. L'actionneur est identique à ceux utilisés dans la base du doigt sauf que, pour des raisons de contraintes d'espace, le réducteur n'est pas intégré. Pour remédier à cette contrainte, l'utilisation d'un réducteur harmonique serait indispensable. La caractéristique fondamentale d'un réducteur harmonique est qu'il réduit les vitesses à l'aide seulement de trois pièces à axe creux. La première est fixe. Les deux autres tournent sur le même axe à des vitesses différentes. Les trois sont imbriquées l'une dans l'autre ; d'où l'espace nécessaire est réduit.

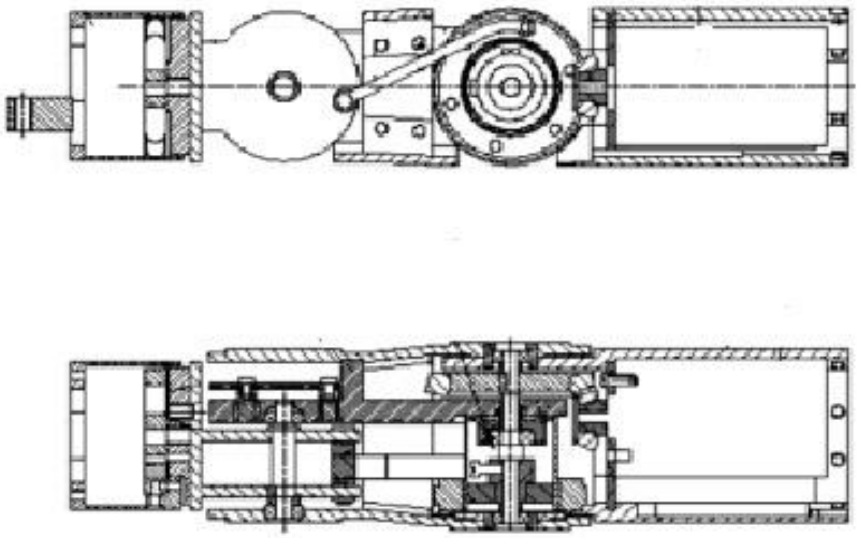

Figure I-12 L'unité du doigt

Figure I.13. Le réducteur harmonique

La vitesse du moteur est transmise au réducteur harmonique à l'aide du pignon noir (Figure I-14). Le réducteur harmonique réduit la vitesse. Celle-ci sera transmise ensuite au mécanisme par l'axe jaune qui est à l'intérieur de la première transmission.

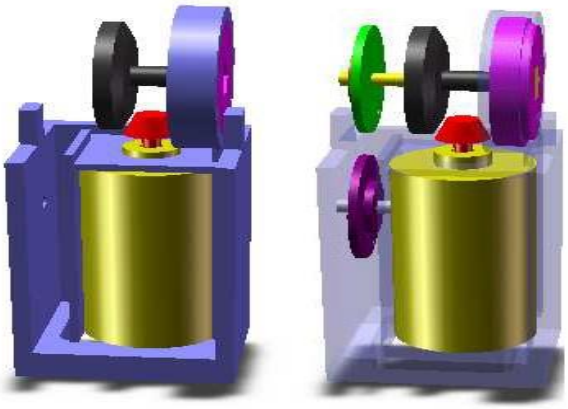

Figure I-14 Le pignon de transmission et le réducteur harmonique

Le mouvement de la phalange distale est piloté par le mouvement de l'articulation de la phalange moyenne à l'aide d'une barre rigide. Si nous couplons ces deux dernières directement nous aurons deux sens inversés de mouvement. Ce qui nous a menés à inverser le sens de la vitesse transmise à la phalange distale avant la transmission. Pour effectuer cette opération, on utilise une courroie dentée directe (le pignon marron même vitesse que l'axe jaune vers la pièce mauve) en premier lieu et puis une autre croisée pour retransmettre la vitesse à la barre qui est soudée sur un roulement (la pièce verte) du même axe que le réducteur harmonique (Figure I.15). Le rapport de réduction de tout le mécanisme d'inversion est de 1 : 1.

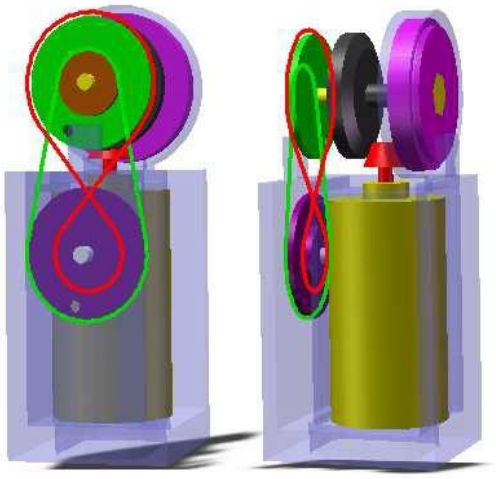

Figure I-15 Les courroies

La structure et les paramètres de la barre de transmission sont optimisés par la simulation.

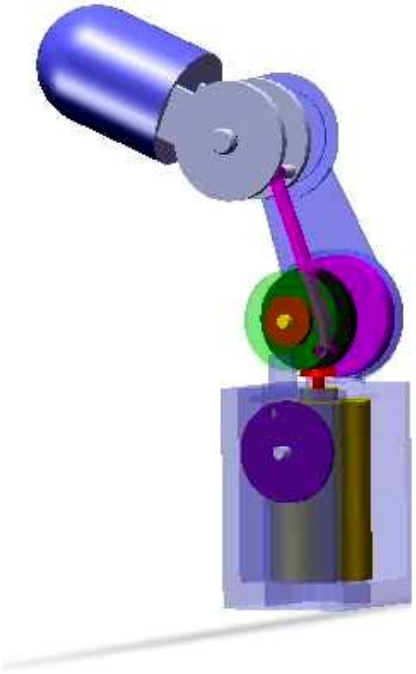

Figure I-16 Placement de la barre de transmission

Une fois que les phalangettes sont placées, il ne nous reste que la fixation de cette unité à l'unité de base, grâce à la structure du pignon satellitaire.

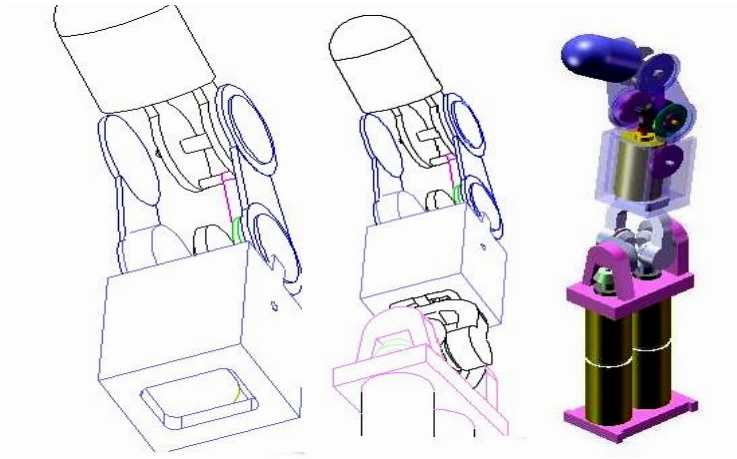

Figure I-17 Fixation des deux unités et aspect final

I.3.5. La conception de la main DLR/HIT II

La conception de la cinématique de la main robotique prouve que le mouvement du pouce, par rapport à la paume est absolument nécessaire pour améliorer l'exécution de la saisie avec une contrainte de précision ou d'effort. Par conséquent, la main sera conçue avec un degré de liberté additionnel afin de réaliser le mouvement du pouce relatif à la paume. Ceci permet d'utiliser la main dans différentes configurations. Les doigts seront fixés à la paume à l'aide des vis. Le pouce est équipé d'une nouvelle articulation qui précède sa base qui fera le mouvement de la pronation / et la supination, en d'autres termes le mouvement de rotation, si nous prenons l'avant-bras comme axe de rotation.

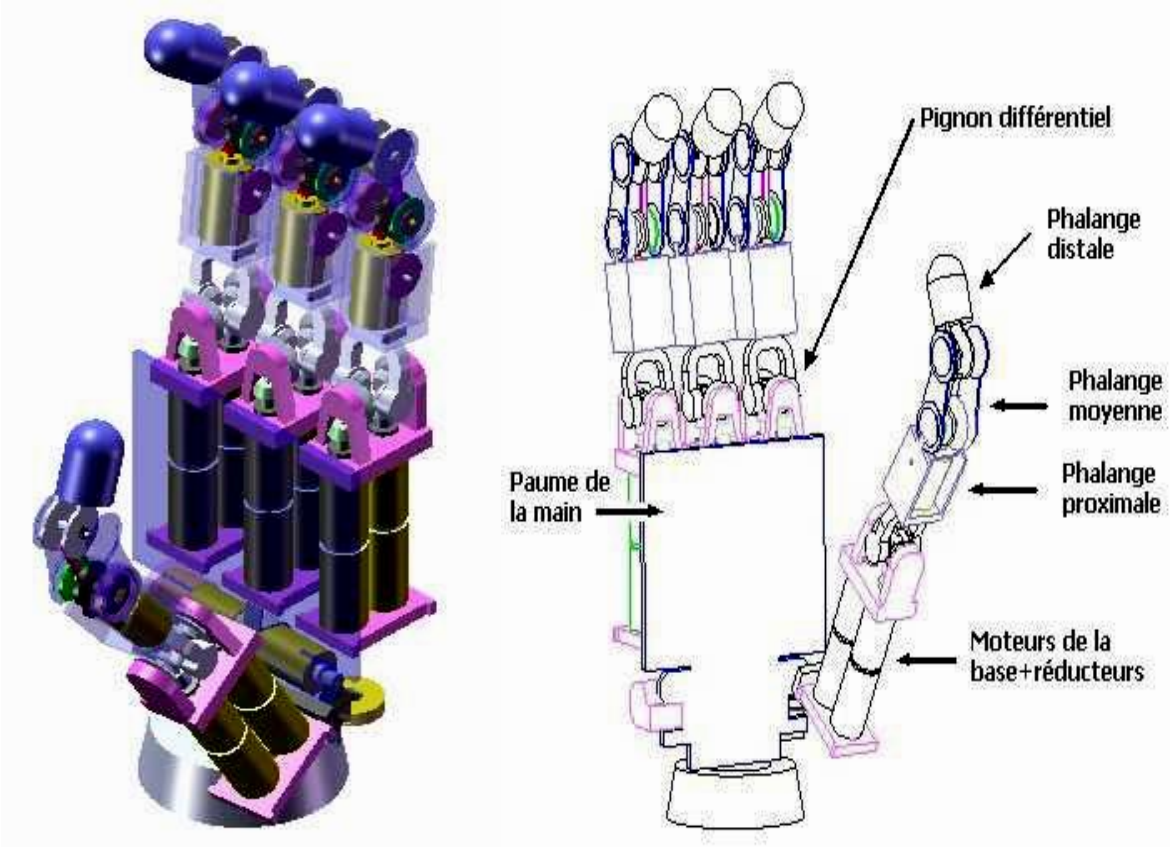

Figure I-18 La main DLR/HIT II proposée

I.4. Modélisation géométrique de la main

I.4.1. Introduction

La modélisation est un principe ou technique qui permet d'établir un modèle explicatif d'un phénomène ou comportement en recensant les variables ou facteurs explicatifs et l'importance relative de chacune de ces variables. On aboutit ainsi à un modèle mathématique décrivant le modèle physique étudié.

Dans notre étude, la modélisation nous servira aussi à traduire les signaux de commandes établis par l'FPGA, soit pour simuler la main en mouvement soit pour observer le comportement des moteurs.

Plusieurs niveaux de modélisation sont possibles selon les objectifs, les contraintes de la tâche et les performances recherchées : modèle géométrique, cinématique et dynamique.

I.4.2. Description de la géométrie du doigt DLR/HIT II

La cinématique du doigt de la main DLR/HIT II est du type RRRR (Figure I.19). On note que la quatrième articulation est pilotée par la troisième. Ce qui ramène le nombre de degrés de liberté à trois.

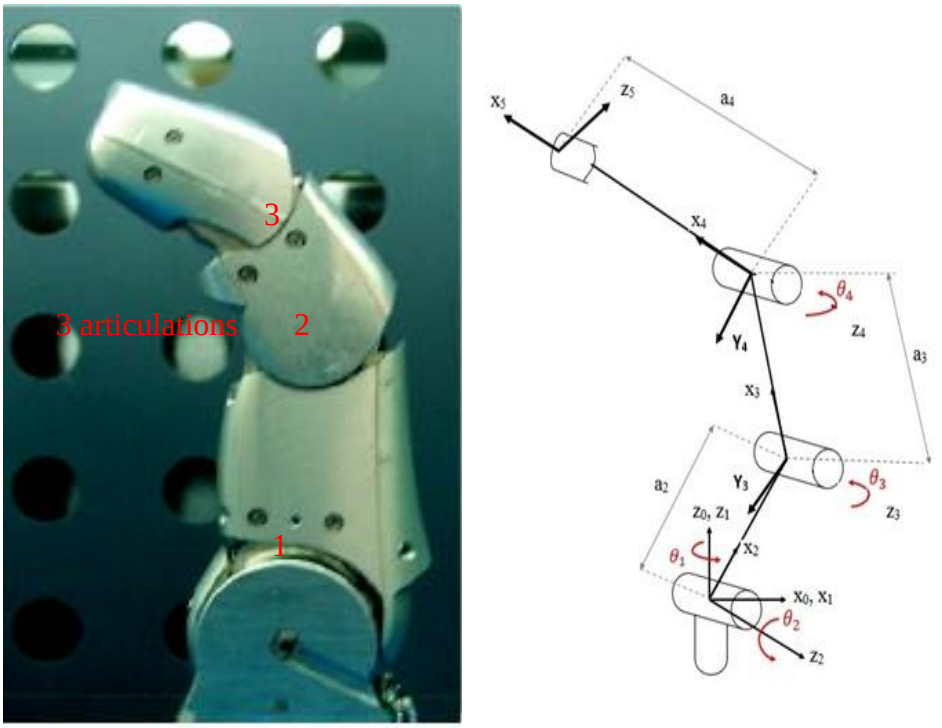

Figure I-19 cinématique du doigt de la main DLR/HIT

D'un point de vue méthodologique, la méthode de DENAVIT-HARTENBERG est nécessaire pour élaborer le modèle géométrique. Résumons cette méthode : nous placerons d'abord les axes z_i sur les axes articulaires. Les axes x_i seront portés par la perpendiculaire commune aux axes z_j et z_{j+1}. Puisque dans notre cas tous les axes z sont parallèles, nous avons considéré les axes x parallèles aux corpsC_j. Les origines des repères sont prises aux centres des articulations et le repère R_0 est confondu avec R_1 aux positions du repos ; C'est-à-dire quand tous les $\theta_j$sont nuls. Nous déterminons ensuite la table des paramètres géométriques du doigt.

Tableau I-2 Paramètres géométriques du doigt de la main DLR/HIT II

j	α_j	d_j	θ_j	r_j
1	0°	0	θ_1	0
2	90	0	θ_2	0
3	0°	a2	θ_3	0
4	0°	a3	θ_4	0
5	0°	a4	0	0

Tableau I-3 Contraintes géométriques au niveau des articulations

Les angles des articulations (en degrés)	Les distances entre les articulations (en mm)
$\theta_1[0° \; 90°]$	a1 0.0
$\theta_2[-20° \; 20°]$	a2 67
$\theta_3[0° \; 90°]$	a3 30
$\theta_4[0° \; 90°]$	a4 30

I.4.3. Modèle géométrique direct

Le modèle géométrique direct nous permettra d'exprimer les coordonnées opérationnelles de l'organe terminal, (dans notre cas c'est la fin du doigt) en fonction des coordonnées articulaires dans le repère de base R_0. Ce qui revient à exprimer le repère R_t dans le repère R_0. Pour cela, nous faisons appel à la matrice de transformations homogènes notée ${}^0T_t$.

$$ {}^0T_t = {}^0T_1 * {}^1T_2 * {}^2T_3 * {}^3T_4 * {}^4T_t $$

$$ {}^0T_t = \begin{bmatrix} C1 & -S1 & 0 & 0 \\ S1 & C1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} C2 & -S2 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ S2 & C1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} C3 & -S3 & 0 & 67 \\ 0 & 0 & 1 & 0 \\ -S3 & -C3 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} C4 & -S4 & 0 & 30 \\ S4 & C4 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} 1 & 0 & 0 & 30 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} $$

Notre système est à 3 degrés de liberté nous n'effectuerons que la commande en position, l'orientation ne sera qu'observée et est représentée par les cosinus directeurs de la matrice ${}^0T_t$ (les trois premières lignes et les trois première colonnes de ${}^0T_t$). Le vecteur position est représenté par la 4éme colonne de ${}^0T_t$.

$$ {}^0T_t = \begin{bmatrix} sx & nx & ax & px \\ sy & ny & ay & py \\ sz & nz & az & py \\ 0 & 0 & 0 & 1 \end{bmatrix} $$

Tel que :

$$ \begin{cases} sx = C1C2C34 - S1S34 \\ sy = S1C2C34 + C1S34 \\ sz = S2C34 \\ \\ nx = -C1C2S34 - S1C34 \\ ny = -S1C2S34 + C1C34 \\ nz = -S2S34 \\ \\ ax = -C1S2 \\ ay = -S1S2 \\ az = C2 \\ \\ px = 30C1C2C34 - 30S1S34 + 30C1C2C3 - 30S1S3 + 67C1C2 \\ py = 30S1C2C34 + 30C1S34 + 30S1C2C3 + 30C1S3 + 67S1C2 \\ pz = 30S2C34 + 30S2C3 + 67S2 \end{cases} $$

avec :

$$ C_i = cos(\theta_i)\ ,\ Si = sin(\theta_i),\ Cij = cos(\theta_i + \theta_J),\ Sij = sin(\theta_i + \theta_J) $$

Remarque : pour la commande du modèle sous Solidworks seul le modèle géométrique est nécessaire vu que l'asservissement sera de type position.

I.5. Conclusion

Dans ce chapitre, nous avons donné l'état de l'art de la conception d'une main artificielle. Après avoir comparé plusieurs modèles de mains artificielles, notre choix s'est porté sur la main DLR/HIT II qui, grâce à la modularité qui est une caractéristique importante de ce modèle, nous permettra de construire la main en se basant sur la construction d'un seul doigt. Un modèle géométrique de celle-ci a été exposé. Il restera ensuite à concevoir l'électronique de commande et l'application d'interfaçage entre cette dernière et le prototype sous SolidWorks.

II.1. Introduction

Due à la diversité des fonctionnalités des ordinateurs de nos jours, le développement d'une application qui répond aux besoins des utilisateurs dans différents domaines rend cette dernière très gourmande en termes de ressources : mémoire, espace sur disque dur … etc. Pour contourner ce problème, les développeurs des solutions concentrent leurs applications sur le domaine prioritaire tout en laissant la possibilité soit d'intégrer des modules complémentaires ou de développer des modules personnalisés en utilisant une API propre à l'application.

Dans ce chapitre on va donner les notions fondamentales concernant les API en général puis les caractéristiques de l'API de SolidWorks utiles pour développer l'interaction entre la main DLR/HIT II et une carte de commande.

II.2. Les API

II.2.1. Définition de l'API

Une **API** (Application Programming Interface, traduisez « interface de programmation » ou « interface pour l'accès programmé aux applications) est un ensemble de fonctions permettant d'accéder aux services d'une application, par l'intermédiaire d'un langage de programmation.

II.2.2. Utilité de l'API

Une API permet de fournir un certain niveau d'abstraction au développeur, c'est-à-dire qu'elle lui masque la complexité de l'accès à un système ou à une application en proposant un jeu de fonctions standard dont seuls les paramètres et les valeurs retournées sont connus.
Ainsi, par analogie avec une voiture, le conducteur n'a pas à connaître le fonctionnement mécanique du moteur d'un véhicule pour pouvoir le conduire, il s'agit d'une certaine façon de l'interface proposée à l'utilisateur.

Grâce aux API, un développeur n'a donc pas à se soucier de la façon dont une application distante fonctionne, ni de la manière dont les fonctions ont été implémentées pour pouvoir l'utiliser dans un programme. Une API peut être disponible pour un langage particulier ou bien être disponible pour plusieurs langages de programmation.

II.2.3. Utilisation de l'API

L'ADD-IN ou l'application standalone représente l'accès programmé aux services de l`application cible (figure II-1). Pour développer une ADD-IN ou une application standalone le développeur accèdera aux services de l'application cible par l'intermédiaire de API (fournis avec l'application cible).

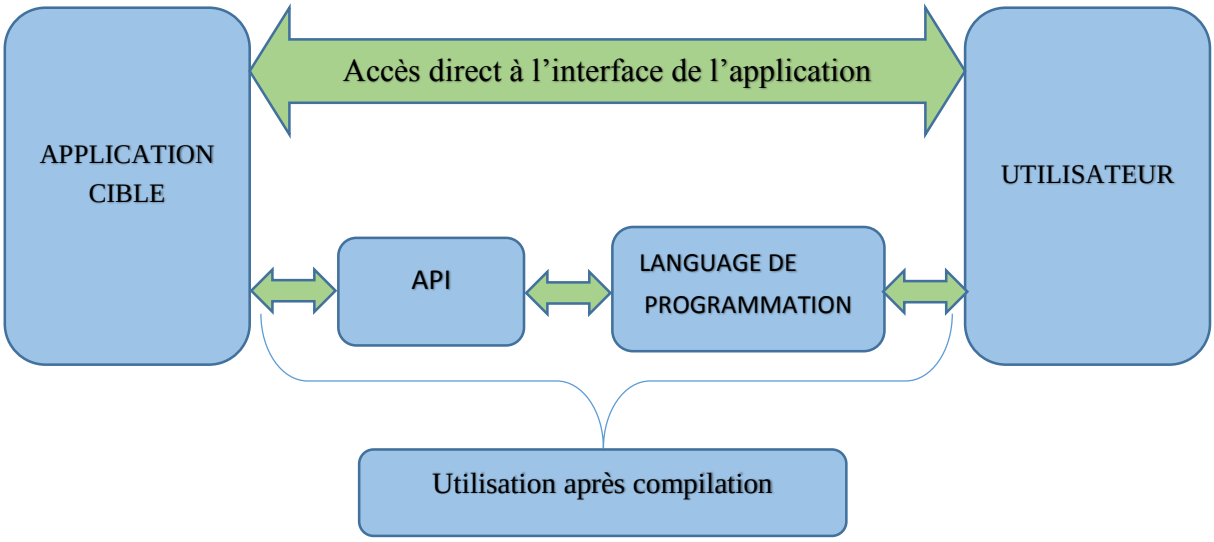

Figure II-1 Flot de données lors de l'utilisation d'une API

II.3. L'API de SolidWorks

II.3.1. Problématique

SolidWorks est un logiciel de conception non équipé d'une interface interactive de communication avec l'extérieur, comme la carte FPGA dans notre cas. Mais les développeurs de cette solution l'on dotée d'une API très riche en fonctions permettant d'accéder par programmation aux différentes fonctionnalités et services de SolidWorks. Ces fonctions peuvent être appelées par différents langages de programmation comme Visual Basic 6.0, Visual Basic for Applications (VBA), Visual C# .NET, Visual C++ 6.0, Visual C++ .NET Visual Basic .NET

II.3.2. Visual Basic

Pour notre travail, le choix de Visual Basic comme langage de programmation est du à sa grande compatibilité avec l'API de SolidWorks par rapport aux autres langages surtout celle entre les types des paramètres des différentes fonctions définies par l'API. Même les développeurs de ce logiciel proposent le (Visual Basic for Application) comme langage de programmation pour les Macros d'automatisation des conceptions.

II.3.3. Interaction avec SolidWorks

Pour la création d'une application personnalisée de SolidWorks, il faut bien comprendre le modèle d'objet de l'API SolidWorks.

SldWorks étant l'objet de plus haut niveau de l'API, il fournit un accès direct ou indirect à tout autre objet donné dans cette particulière API. Pour SolidWorks, la librairie de référence est nommée « Type Librairie ».Elle définit les caractéristiques de la structure d'objet du logiciel et les fonctions « methods» appropriées à chaque objet et permet un accès direct aux fonctionnalités de SolidWorks, par exemples : la création d'une ligne, l'insertion d'une pièce dans un document, la vérification des paramètres d'une surface et autres. [7]

La figure II-2 illustre le modèle d'objet de l'API de SolidWorks

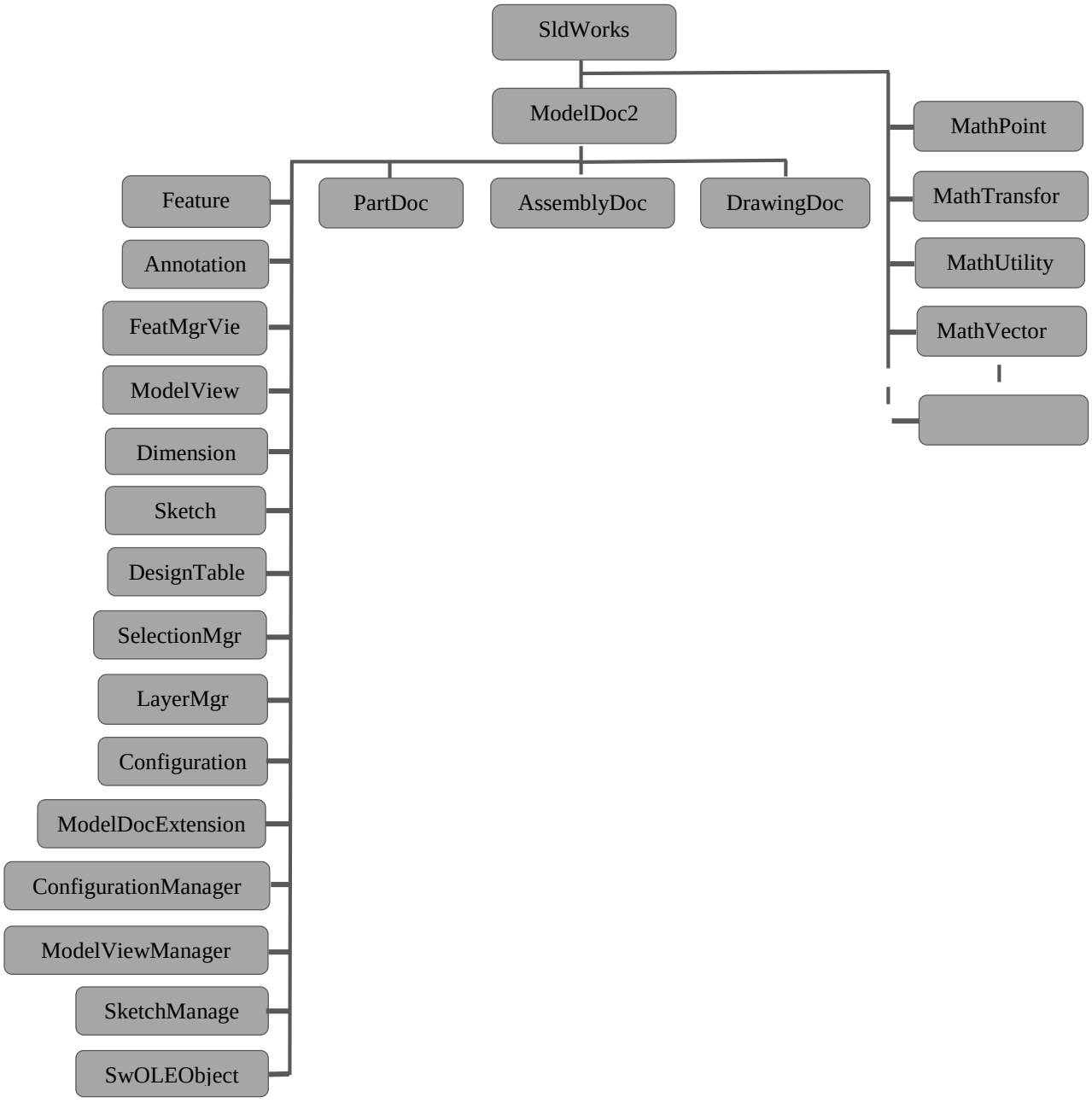

Figure II-2 Modèle d'objet de l'API SolidWorks [7]

II.3.3.1. Interface SldWorks

Une application créée doit inclure une variable de référence pour manipuler le transfert de données au/du logiciel SolidWorks .

Cette interface fournit un ensemble général de fonctions qui permet des opérations au niveau de l'application comme la création, l'ouverture, la fermeture des documents, l'arrangement des icônes et des fenêtres, le changement du document actif, et la création des définitions d'attributs.

Elle donne aussi l'accès aux objets et aux interfaces suivantes utilisées au cours de notre travail :

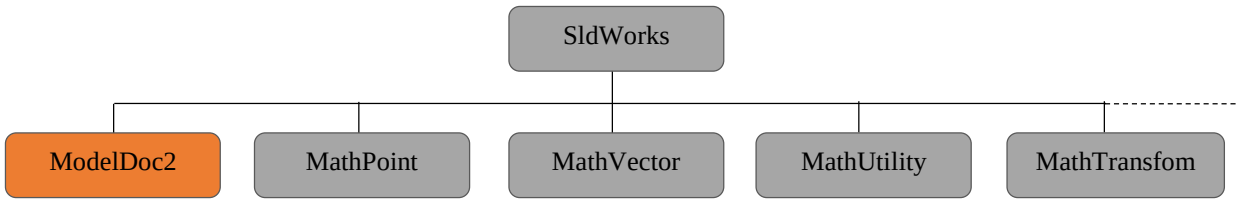

Figure II-3 Objets accessibles par SldWorks[7]

- MathTransform

Elle fournit une interface simplifiée pour la manipulation des matrices de transformation.

- MathPoint

Elle fournit une interface pour la manipulation des points et elle donne accès à des méthodes pour la création d'autres objets.

- MathVector

Elle fournit une interface pour la manipulation des vecteurs et elle donne accès à des méthodes pour la création d'autres objets.

- ModelDoc2

Elle sera détaillée dans les paragraphes suivants

L'interface SolidWorks contient aussi d'autres objets qui ne seront pas détaillés dans ce chapitre car ils ne sont pas utilisés.

II.3.3.2. Interface ModelDoc2

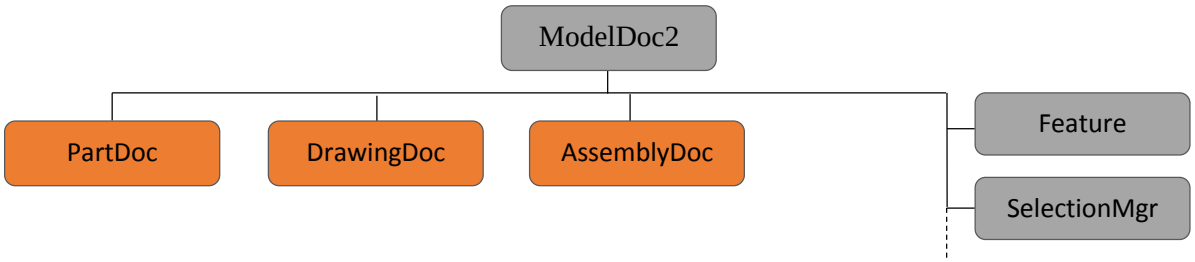

Figure II-4 Objets accessibles par ModelDoc2[7]

Elle permet un accès aux documents SolidWorks. Trois types principaux sont à distinguer :

- Piece «Part».
- Schema «Drawing».
- Assemblage «Assembly».

Et aussi aux objets servant à d'autres opérations sur ces modèles de documents comme exemple :

- Feature

Permet l'accès au type de caractéristiques d'un objet (nom, données de paramètre), et au prochain élément dans l'arbre de conception de FeatureManager.

- SelectionMgr

Permet d'obtenir des informations sur les objets choisis, d'obtenir des objets de l'API représentant l'élément choisi. D'obtenir les coordonnées de la selection dans l'espace de modèle ou de l'esquisse.

II.3.3.3. Interface PartDoc

Cette interface fournit des fonctions permettant de :

- Créer des corps et des éléments.
- Exécuter les opérations de désactivation.
- Obtenir les mesures des pièces.
- Localiser les entités par leurs noms.

Les objets fils de PartDoc sont illustrés par la figure II-5

II.3.3.4. Interface AssemblyDoc

En règle générale, l'objet d'AssemblyDoc permet d'accéder aux fonctions qui effectuent des opérations d'assemblage ; par exemple, ajouter de nouveaux composants, ajouter les contraintes entre les composants, cacher et présenter l'assemblage en mode éclaté.

Les objets fils d'AssemblyDoc sont illustrés par la figure II-6

II.3.3.5. Interface DrawingDoc

Permet l'accès aux fonctions qui effectuent des opérations sur le schéma. Par exemple, la création, l'alignement.

Les objets fils de DrawingDoc sont illustrés par la figure II-7

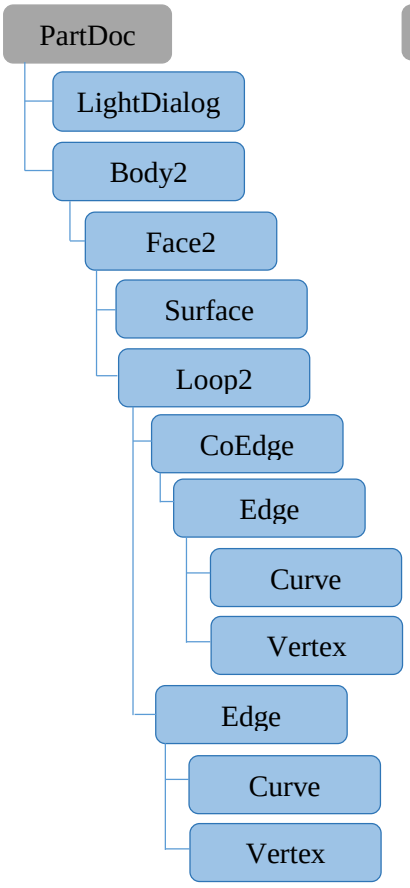

Figure II-5 objets filsde PartDoc

Figure II-6 objets fils de AssemblyDoc

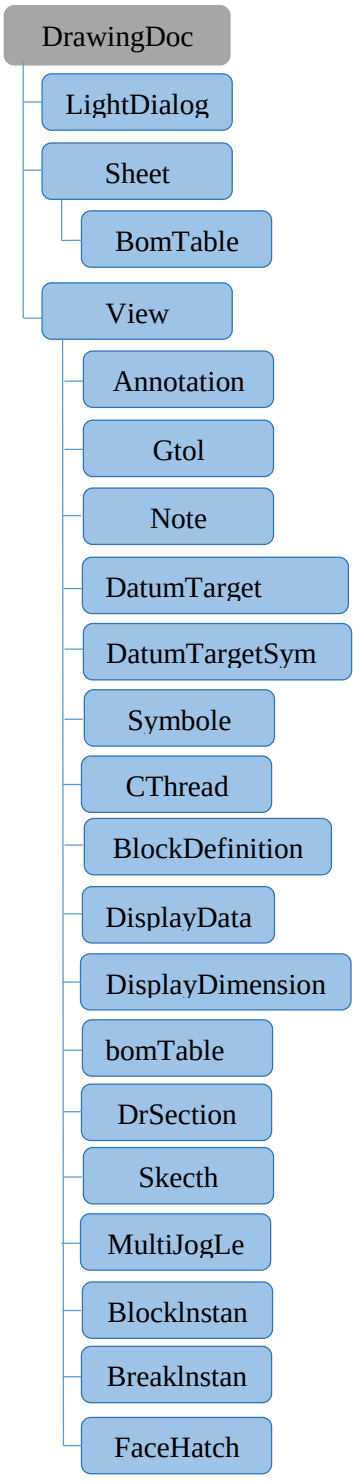

Figure II-7 objets fils de DrawingDoc

II.3.3.6. Les fonctions «Methods»

Généralement plusieurs fonctions appropriées pour chaque objet du diagramme sont disponibles.

La méthode la plus simple pour obtenir les fonctions à utiliser pour une tache donnée est l'enregistrement des macros. Elle convertira la chaine d'actions à des appels en Visual basic correspondantes aux actions exécutées au niveau de l'interface utilisateur. Après, ces appels sont ajustés pour être utilisés dans le code de l'application à développer.

II.4. Conclusion

Dans ce chapitre, nous avons donné les notions fondamentales des API : définition, rôle et flot de données lors de l'utilisation. Nous avons précisé par la suite les spécificités de l'API SolidWorks et donné la structure hiérarchique du modèle d'objet de ce dernier, ce qui nous a permis de voir et d'intégrer l'utilisation de l'API SolidWorks dans notre travail.

III.1. Introduction

Le présent chapitre porte sur les *Systems On Chip* et leurs avantages par rapport aux autres solutions, ainsi que les différents outils de développement qui nous permettent de réaliser notre système embarqué lequel se chargera de l'échange des signaux de commande vers l'application de liaison pour le pilotage du prototype sous Solidwork, et enfin de l'architecture des deux plateformes matérielle et logicielle qui seront utilisées.

Dans la première partie de ce chapitre, on exposera la solution SoC et ses principaux avantages. On proposera, ensuite dans la deuxième partie, un tour d'horizon dans l'environnement de développement EDK ainsi que ses principales fonctionnalités. A travers la troisième partie, on découvrira les différentes approches qu'on a faites au niveau de l'architecture matérielle, l'incontournable processeur utilisé par la famille *Spartan-3* «*MicroBlaze*» et ses principales caractéristiques, la carte *Spartan-3E* utilisée. Dans la quatrième partie, quelques concepts relatifs aux systèmes temps réel sont abordés afin de faciliter la description de la partie software sous EDK. Enfin, on donnera un aperçu sur la partie software sous Windows.

III.2. Les Systems on Chip (SoC)

III.2.1. Définition

Un système sur puce (SoC) est un dispositif caractérisé par l'intégration de toutes les parties du système informatique en un seul circuit. Cela permet de réduire la dissipation de l'énergie, les interconnexions et la taille du dispositif [8]. Un système on SoC typique contient un ou plusieurs processeurs, un nombre arbitraire de périphériques, une mémoire On Chip et un bus qui interconnecte tous ces composants. L'avantage de la solution On Chip est d'utiliser un seul circuit pour tout le système [8].

L'élaboration d'un système SoC peut passer par l'utilisation d'un circuit FPGA. Ce dernier nous permet une grande flexibilité et un faible coût par rapport aux autres solutions telles que les ASIC qui sont généralement caractérisés par une conception figée, mais qui sont plus rapides.

Pour notre travail, nous avons choisi une carte FPGA du constructeur *Xilinx* c'est la *Spartan-3E*, modèle *XC3S500E* et qui sera présentée plus loin dans ce chapitre.

III.2.2. La configuration du circuit FPGA

Dans le monde des circuits intégrés, chaque constructeur a développé un ensemble d'outils est de logiciels indispensables à la configuration de leurs circuits. Cette configuration se ramène au chargement d'un fichier appelé le *Bitstream*. Ce dernier décrit les interconnexions entre les constituants du FPGA (CLB et IOB). Dans notre cas (Constructeur Xilinx), la configuration du circuit FPGA peut se faire de deux manières :

1- L'utilisation d'un langage de description matérielle (VHDL). Ce langage permet de synthétiser comment un matériel doit être implémenté. Dans ce cas, on est amené à utiliser l'outil de développement *ISE* de Xilinx.
2- L'utilisation d'un processeur soft qui offre plus de flexibilité et de rapidité de reconfiguration que les processeurs hard. Ce qui nous a conduit à l'utilisation de l'outil EDK de Xilinx.

III.3. L'environnement de développement EDK

III.3.1. Introduction

EDK (*Embedded Development Kit*) est un ensemble d'outils et de Propriétés Intellectuels (IP) qui permettent à l'utilisateur de concevoir des systèmes embarqués à base de microprocesseurs. Cet outil, développé par Xilinx, nous permettra de configurer notre carte FPGA et ce après avoir généré le *Bitstream* qui contient la configuration des deux parties software (les applications) et hardware (la configuration de *MicroBlaze* et de ses périphériques) de notre système qui sera ensuite chargé sur la carte Spartan-3E [9]. Parmi les outils fournis par EDK on trouve :

1- L'environnement de développement intégré *Xilinx Platform Studio* (XPS) qui se charge de la partie matérielle (définition du matériel et sa configuration) de notre système embarqué.
2- *Software Development Kit* (SDK) qui s'occupe, comme son nom l'indique, de la partie software (écriture d'application en langage C ou C++).

L'avantage fourni par EDK est que ce dernier nous assiste durant toutes les étapes de conception de notre système embarqué. EDK inclus aussi d'autres éléments, tels que :

- Des bibliothèques de périphériques « *Hardware IP* » pour les processeurs embarqués Xilinx.
- Des pilotes et des librairies pour le développement de la partie software.

- Le compilateur GNU ainsi qu'un debugger pour le développement software visant les processeurs MicroBlaze et PowerPC.
- Documentation.
- Exemples de projets [9].

EDK dépend aussi d'un autre environnement de logiciel intégré (ISE) fourni aussi par Xilinx. ISE est un produit de Xilinx nécessaire pour implémenter une conception sur le circuit FPGA. L'accès aux différents composants supportés par EDK est impossible sans cet environnement. Car de nombreux outils présents dans ISE sont appelés à partir d'EDK pour effectuer diverses tâches (par exemple la génération de la Netlist grâce au flot ISE) [9].

III.3.2. Le flot de conception sous EDK

La figure III-1 illustre le flot de conception simplifié sous EDK/

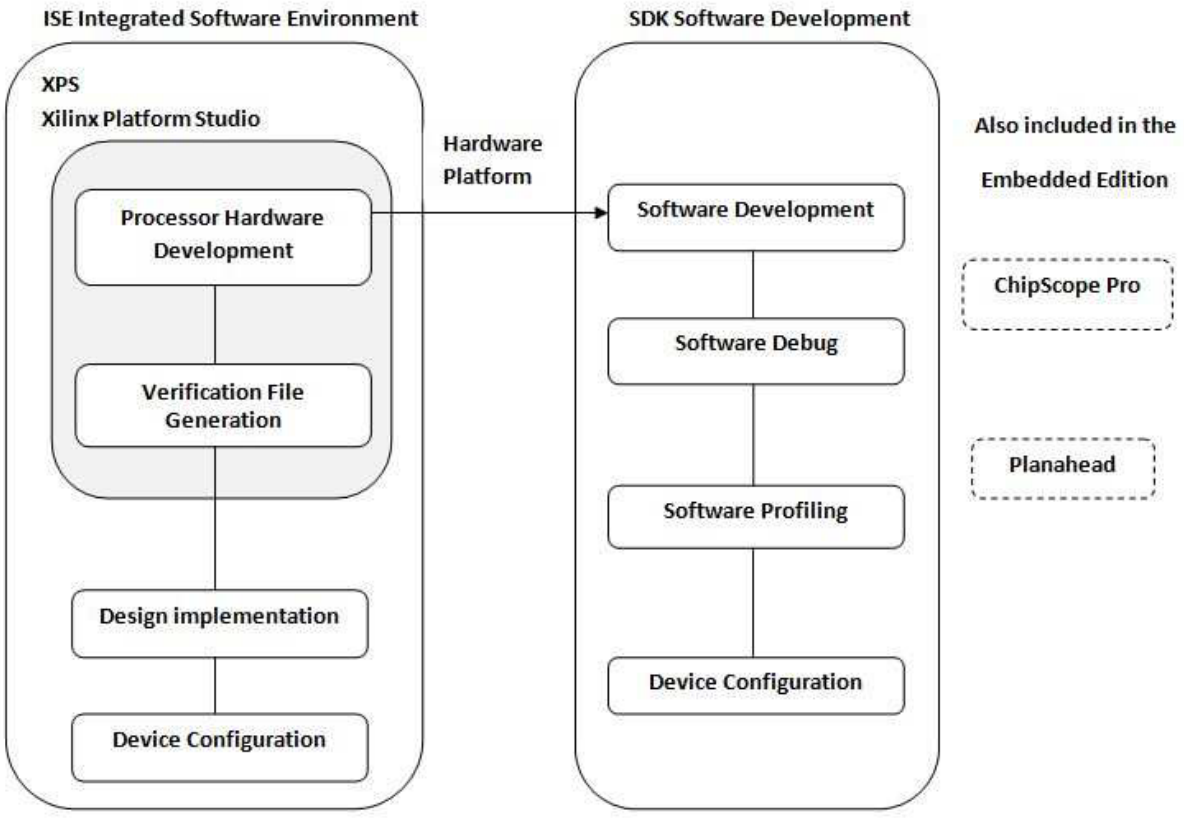

Figure III-1 Flot de conception simplifié sous EDK [9].

D'abord, on utilise XPS pour créer la partie hardware de notre système (spécifications et configuration du processeur, définition des périphériques et leurs interconnexions). Puis, on lance SDK qui se chargera de la partie software (Figure III-1). On peut, ensuite, vérifier le fonctionnement de la plateforme matérielle à travers le simulateur intégré dans ISE. Une fois la vérification faite, on génère le fichier Bitstream qui se chargera de la configuration du circuit FPGA. EDK nous donne aussi l'opportunité de travailler avec des outils tels que : «*PlaneAhead design analysis* » et « *ChipScope Pro*» qui sont très utiles pour le débogage. La figure précédente illustre le flot de conception sous EDK mais sous un aspect superficiel, en réalité ce flot est beaucoup plus complexe.

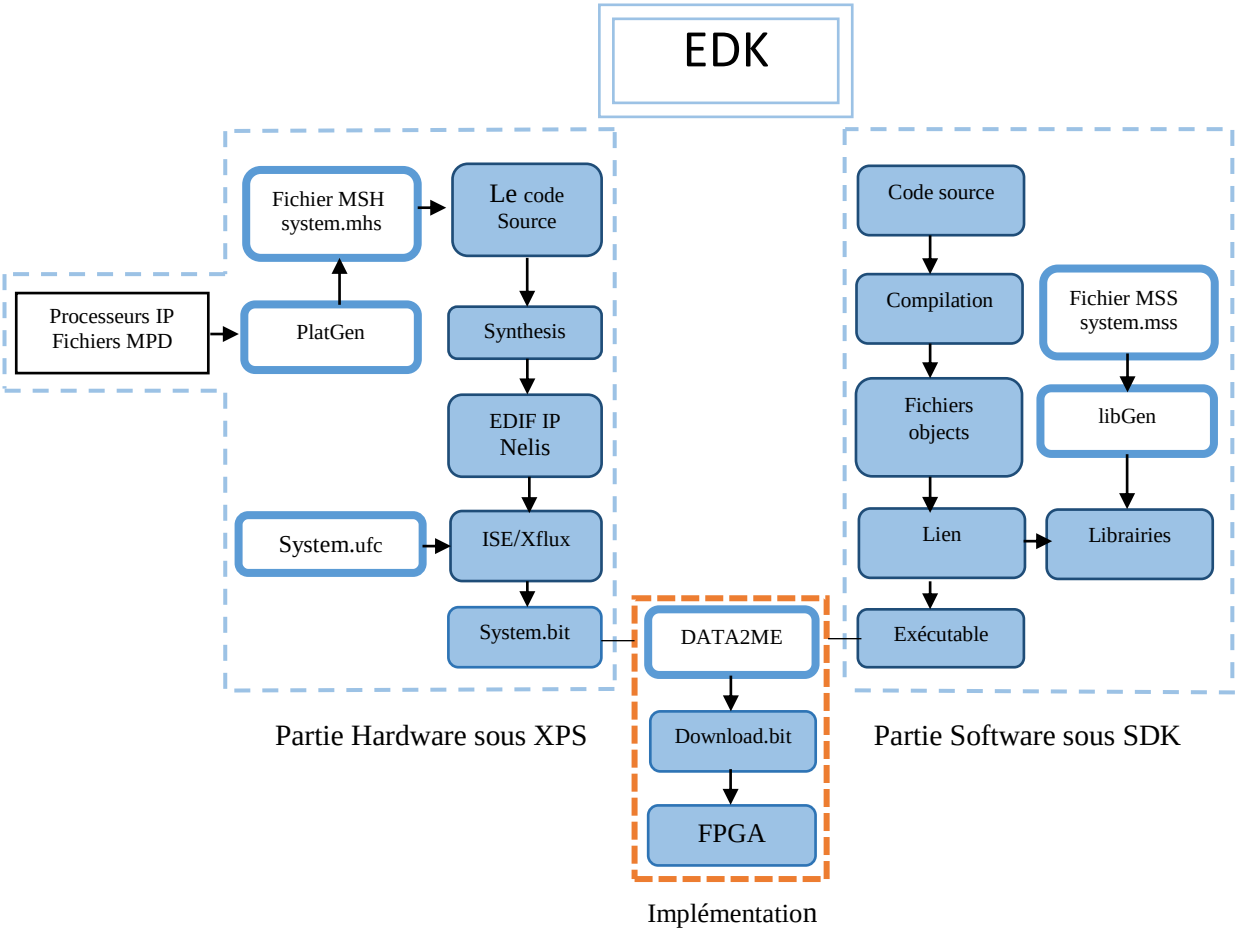

Figure III-2 Flot de conception détaillé sous EDK [9].

Comme illustré ci-dessus, EDK va nous permettre de développer la partie hardware en parallèle avec la partie software. On parlera alors du Co-design. Dans le flot de conception de la partie Hardware, on procède comme suit :

- On choisit un processeur IP et les différents périphériques qui lui seront connectés.
- On exécute l'outil *PlatGen* d'EDK qui les configure en se basant sur les caractéristiques spécifiées dans le fichier MHS.
- XPS synthétise une *Netlist* à partir des codes sources des IP cores propriétaires fournis par l'environnement ISE en langage VHDL.
- Génération d'un fichier UCF qui décrit les connexions des différents périphériques avec le circuit FPGA qui sera lié à la Netlist pour former un fichier *.bit* qui est le fichier de toute la configuration de la plateforme matérielle [9].

Dans le flot de conception software, le programme de l'application doit être écris soit en langage assembleur soit en langage C ou C++. Le code source obtenu doit être compilé afin de récupérer des fichiers en code objet et qui seront liés avec des librairies spécifiques. Ces dernières sont générées lors de l'exécution de l'outil PlatGen de EDK pour les spécifications du software qui sont données dans le fichier MSS. On obtient à la fin du flot software, un fichier exécutable [9].

Le fichier *.bit* obtenu à la fin du flot matériel et le fichier exécutable de l'application, obtenu à la fin du flot software, sont liés par l'outil DATA2MEM pour former un seul fichier *download.bit* qui configure tout le système en entier et qui sera chargé sur le circuit FPGA[9].

III.4. Architecture de la partie matérielle

Dans cette section, on présentera les différentes approches qui ont attiré notre attention concernant l'architecture matérielle qui sera utilisée pour la commande de notre prototype. Comme nous l'avons déjà mentionné précédemment, concernant le Hardware, notre choix c'était porté sur un circuit FPGA de la famille Spartan-3E : c'est le XC3S500E utilisant le processeur soft (MicroBlaze).

La carte Spartan-3E, qui inclut le circuit FPGA ainsi que d'autres périphériques (switches, leds, écran LCD, mémoire flash,… etc.), représentera la plateforme matérielle de notre implémentation (notre carte de contrôle).

III.4.1. Les différentes approches d'architectures de commande

Pour la commande de la main artificielle sous Solidworks, on peut suivre deux approches : une commande centralisée et une commande répartie :

III.4.1.1. La commande centralisée

Dans ce type de commande, les doigts de notre main artificielle seront pilotés avec un seul processeur implémenté sur SoC. Le parallélisme est assuré par la partie software à l'aide d'un système d'exploitation temps réel (Figure III-3). Ce qui conduit à l'architecture software de la figure III-4.

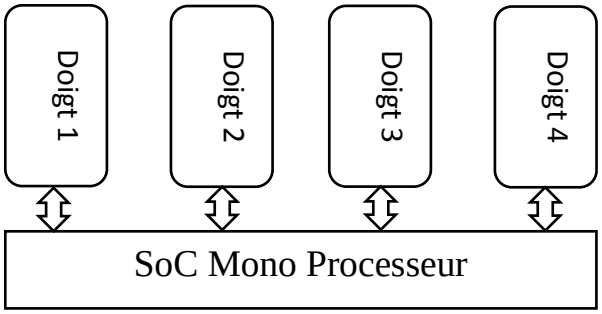

Figure III-3 L'architecture hardware de la commande centralisée

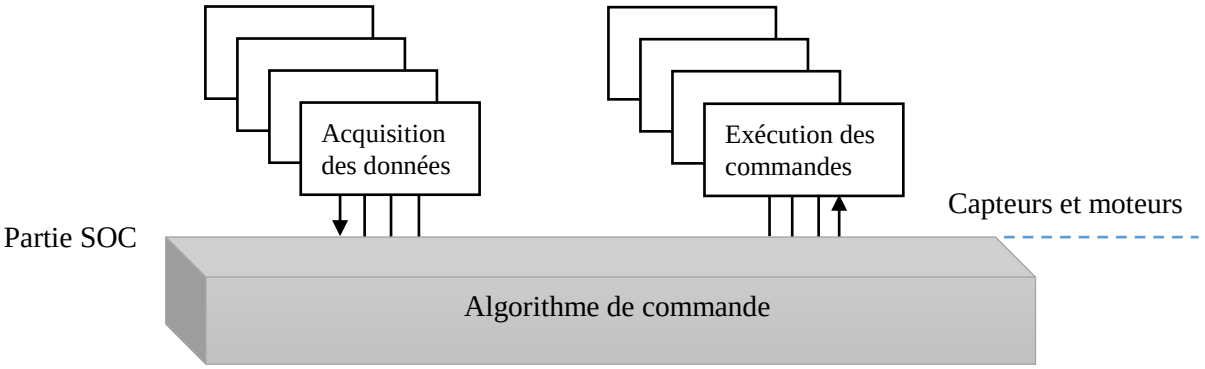

Figure III-4 L'architecture software de la commande centralisée

III.4.1.2. Commande répartie

La deuxième approche est la commande repartie dans laquelle on implémente pour chaque doigt, un processeur qui traitera les données (Figure III-5). Mais, cette approche impose l'élaboration d'une nouvelle partie qui gère le transfert de données entre les processeurs et les doigts ou entre les processeurs eux même. L'architecture software associée est donnée par Figure III-6.

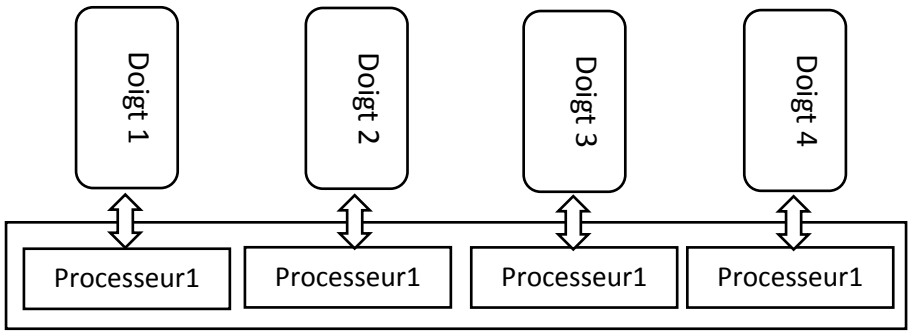

Figure III-5 L'architecture hardware de la commande répartie.

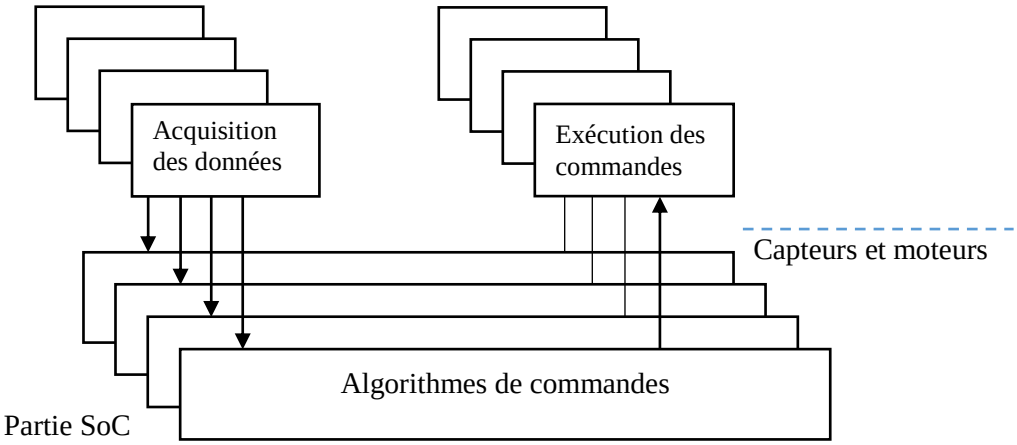

Figure III-6 L'architecture software de la commande répartie

III.4.2. Le processeur MicroBlaze

III.4.2.1. Définition

Le processeur soft MicroBlaze est un processeur à jeux d'instructions réduit (RISC), optimisé pour l'implémentation dans les circuits FPGA de Xilinx. Le processeur soft MicroBlaze est disponible dans l'environnement EDK sous forme d'une description en langage HDL (Hardware Description Language). De nombreux périphériques peuvent lui être connectés via le bus PLB, tel un module permettant la gestion du port série [10].

Conçu selon une architecture dite Harvard séparant le chemin des instructions de celui des données et permettant un parallélisme dans le transfert des données et des instructions, MicroBlaze réalise l'exécution des instructions en un cycle d'horloge, grâce à son architecture pipeline. La figure III-7 illustre un aperçu de la constitution interne du processeur MicroBlaze.

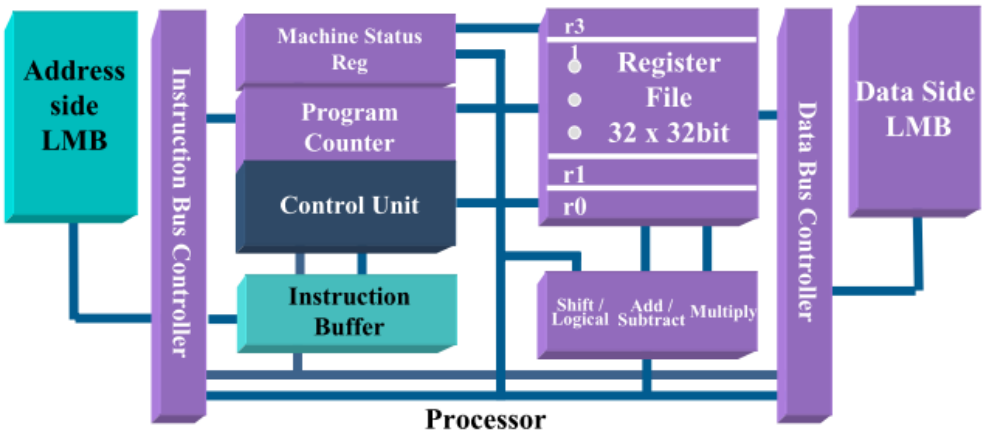

Figure III-7 Architecture interne du processeur MicroBlaze[10]

III.4.2.2 Caractéristiques de MicroBlaze

Voici les principales caractéristiques de MicroBlaze :

- Un mot d'instruction à 32 bits, avec trois opérandes et deux modes d'adressage.
- Des bus de données et d'instructions séparés de 32 bits connectant le processeur aux blocks mémoires internes via le bus LMB.
- Des bus d'adresse à 32 bits.
- Une architecture pipeline qui exécute l'instruction en un cycle d'horloge.
- Un cache d'instructions et de données qui permet d'augmenter la bande passante entre le processeur et la mémoire.
- Un debugger logique du matériel.
- Un support FSL (Fast Simplex Link) qui permet une seule fonction à la fois soit la lecture soit l'écriture [10].

III.4.3. La carte de contrôle Spartan 3E

III.4.3.1 Définition

De part la présence de son FPGA très largement dimensionné (près de 200 K portes) et de ses dispositifs de commandes et de visualisation divers (boutons poussoirs, afficheurs, Leds, port PS2, Port VGA, Port USB...), cette platine convient tout aussi bien pour la réalisation d'applications de décodage logique très simple comme pour la mise au point de réalisations extrêmement complexes et puissantes. L'ensemble est livré avec un bloc d'alimentation et un câble de programmation USB (Figure III-8) [15].

Figure III-8 La carte Spartan-3E XC3S500E [15].

III.4.3.2. Caractéristiques principales du kit

Voici les principales caractéristiques du kit de développement :

- 128 Mbit de mémoire.
- 64 Mbit DDR SDRAM.
- Port Ethernet.
- Port VGA.
- Port PS pour clavier et souris.
- 4 boutons switches.
- 8 leds.
- Convertisseur analogique/numérique de 4 bits **[15].**
- 8 Mbit SPI flash.
- 4 Mbit de mémoire flash.
- Port USB pour la programmation de la carte.
- Port RS232.
- Afficheur LCD de deux lignes.
- 2 boutons poussoir.
- Un oscillateur variable a cristal.

III.5. Les systèmes temps réel

Avant d'aborder l'architecture de la partie software qu'on a adopté, il est important de donner quelques concepts relatifs aux systèmes temps-réel nécessaires à la compréhension de la suite de ce chapitre.

III.5.1. Définition

On peut définir un système temps-réel comme suit : « U*n système temps réel est une association logiciel/matériel où le logiciel permet, entre autre, une gestion adéquate des ressources matérielles en vu de remplir certaines tâches ou fonctions dans des limites temporelles bien précises*» [11].

Une confusion classique est de mélanger temps-réel et rapidité de calcul du système donc puissance du processeur. On entend souvent : " *être temps Réel, c'est avoir beaucoup depuissance* " et qui en réalité, n'est pas toujours vrai.

III.5.2. Quelques concepts relatifs aux systèmes temps réel

Voici quelques notions relatives aux systèmes temps réel [11] [12] :

- **La section de code critique (région critique) :** C'est un code indivisible, c'est à-dire qu'une fois son exécution lancée, elle ne doit pas être interrompue (avant de lancer l'exécution du code critique, il faut mettre les interruptions invalides et puis les valider une fois le code critique terminé).
- **Les ressources :** Une ressource est toute entité utilisée par une tâche. Elle peut donc être des dispositifs d'entrée/sortie, tel qu'une imprimante, un clavier, etc.
- **Ressources partagées :** Ce sont des ressources qui peuvent être utilisées par plusieurs tâches. Chaque tâche peut avoir un accès exclusif à la ressource pour éviter la corruption des données. Ce qui est appelé *exclusion mutuelle.*
- **Une tâche :** Du point de vue du microprocesseur, une tâche est une activité qui consomme des ressources de la machine informatique.
- **Le multitâche :** C'est le processus de l'ordonnancement de la CPU. La CPU se partage entre les tâches. L'aspect le plus important d'un système multitâche est qu'il permet aux programmeurs d'applications de gérer les applications en temps réel.

- **L'ordonnanceur de tâches (scheduler) :** Un Ordonnanceur de tâches, scheduler, ou superviseur a pour rôle de faire basculer le traitement d'une activité sur une autre. C'est lui qui gère le temps CPU, C'est l'Ordonnanceur de tâches qui prend la décision d'allouer la ressource « temps CPU » à une activité plutôt qu'à une autre (d'activer une tâche ou non).
- **Fonctionnement en tourniquet (round robbin) :** Dans ce mode de fonctionnement, le scheduler est capable de gérer plusieurs activités et alloue le temps CPU à une tâche pour un temps infini, tant qu'une tâche de priorité plus grande ne devient pas prête, comme dans un fonctionnement basé sur la priorité mais fonctionne en temps partagé à l'intérieur d'un même niveau de priorité. L'ordonnanceur considère deux classes de niveau de priorités : une classe où le fonctionnement est basé sur la priorité et une autre classe où le fonctionnement est en temps partagé.
- **Le noyau préemptif :** La plus prioritaire tâche prête à être exécutée prend le contrôle de la CPU dans un noyau préemptif. Quand une interruption met une tâche plus prioritaire dans son état prêt, l'exécution de la tâche active est suspendue et c'est la tâche prioritaire qui prend le contrôle de la CPU.

III.6. Architecture de la partie Software

III.6.1. Soft sous EDK

Dans cette partie, on présentera le système temps réel qu'on a utilisé pour notre travail, à savoir: *Xilkernel*. Ce dernier est un système d'exploitation disponible dans l'environnement EDK. Il est embarqué sur la carte Spartan 3E, pour piloter notre prototype virtuel.

III.6.1.1. Le but de l'utilisation d'un système d'exploitation temps réel

Plusieurs facteurs peuvent nous amener à choisir un noyau temps-réel pour une conception donnée, en particulier pour les systèmes embarqués. Parmi ces facteurs, on peut citer les suivants :

- Les boucles de contrôle dans un système embarqué sont très difficiles à développer quand le nombre de tâches nécessaires est assez grand. En plus, les performances de telles applications diminuent dramatiquement quand les complexités du système augmentent.

- Diviser les tâches en plusieurs applications individuelles et les implémenter dans un système d'exploitation est plus intuitif, spécialement quand la complexité des applications augmente.
- Plusieurs applications communes dépendent des services d'un système d'exploitation, tel que le management du temps (l'ordonnancement).

Dans notre cas, l'utilisation d'un système d'exploitation temps-réel s'avère plus que nécessaire vu le nombre importants de tâches à effectuer.

III.6.1.2. Le choix Xilkernel

III.6.1.2.1. Présentation du noyau Xilkernel

Xilkernel est un noyau embarqué qui peut être personnalisé pour une large gamme de systèmes. Il possède les caractéristiques clefs d'un noyau temps-réel comme le multitâche, un ordonnanceur préemptif piloté par les priorités, une communication inter processus, une facilité de synchronisation, une prise en charge des interruptions, etc [13].

XilKernel est un noyau modulaire. On peut sélectionner et personnaliser les modules du noyau nécessaires pour une application donnée. La figure III-9 montre les différents modules de XilKernel.

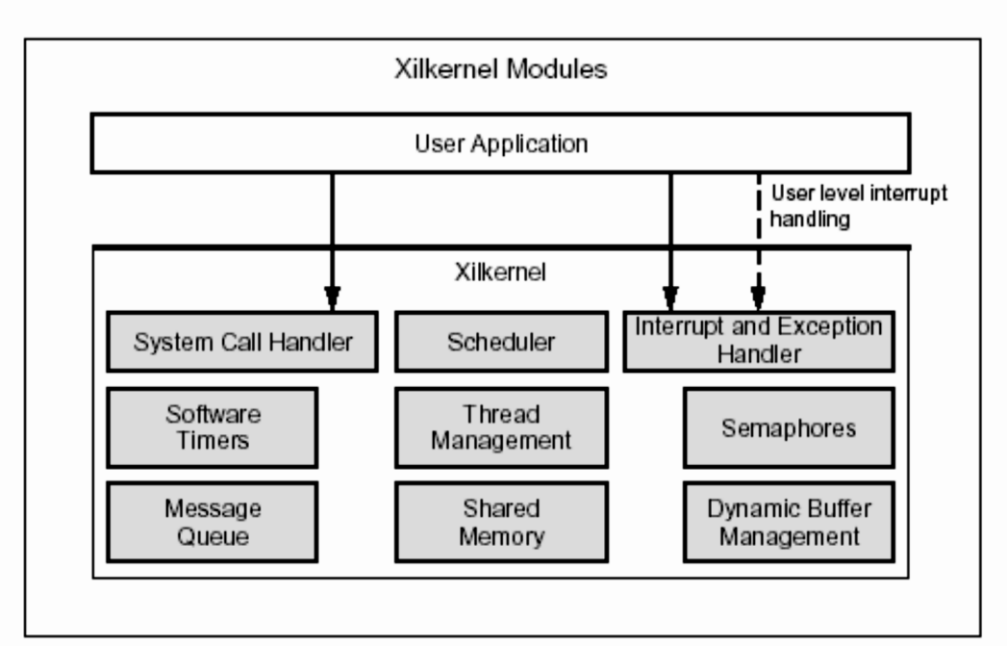

Figure III-9 Organisation de Xilkernel[13]

III.6.1.2.2 Gestion des processus [13]

Les systèmes d'exploitation exécutent plusieurs applications utilisateurs comme des processus indépendants. Un *timer* matériel interrompe périodiquement le processeur et fait appel à l'Ordonnanceur qui détermine quel processus doit être exécuté. L'intervalle de temps entre ces appels est appelé *time slice.* A chaque *time slice,* l'Ordonnanceur sélectionne un processus à exécuter.

Sans un système d'exploitation, les applications doivent être combinées dans une seule fonction énorme, difficile à mettre en œuvre. Cependant, avec un noyau, chaque application devient un processus indépendant qui peut être développé et testé indépendamment de l'autre.

L'algorithme d'ordonnancement le plus simple pour Xilkernel est l'Ordonnanceur en tourniquet, qui met tous les processus dans une seule queue et accorde le *time slice* pour la première tâche sans prendre en compte aucune priorité. Dans ce mode, une seule queue est prête à être exécutée et chaque processus s'exécute durant un *time slice* avant de louer l'exécution au processus suivant.

Lors de l'exécution d'un programme sous XilKernel, le processus qui est à la tête de la queue la plus prioritaire prête est tout le temps prévu pour être exécuté. Si un ensemble de processus a une priorité identique, l'ordonnancement est en tourniquet. Les processus bloqués ne sont pas dans les queues prêtes appropriées, mais plutôt dans la queue en attente. Cette dernière est configurée comme une queue de priorité si l'ordonnancement est basé sur les priorités. Autrement, elle est configurée comme une queue FIFO.

Chaque processus peut être dans l'un des états suivants :

- PROC_NEW : un nouveau processus crée
- PROC_READY : un processus prêt à être exécuté
- PROC_RUN : un processus en exécution (actif)
- PROC_WAIT : un processus bloqué dans une ressource
- PROC_DELAY : un processus en attente d'un *timeout*
- PROC_TIMED_WAIT : un processus bloqué dans une ressource et ayant un *timeout.*

La figure III-10 illustre les différents états des processus :

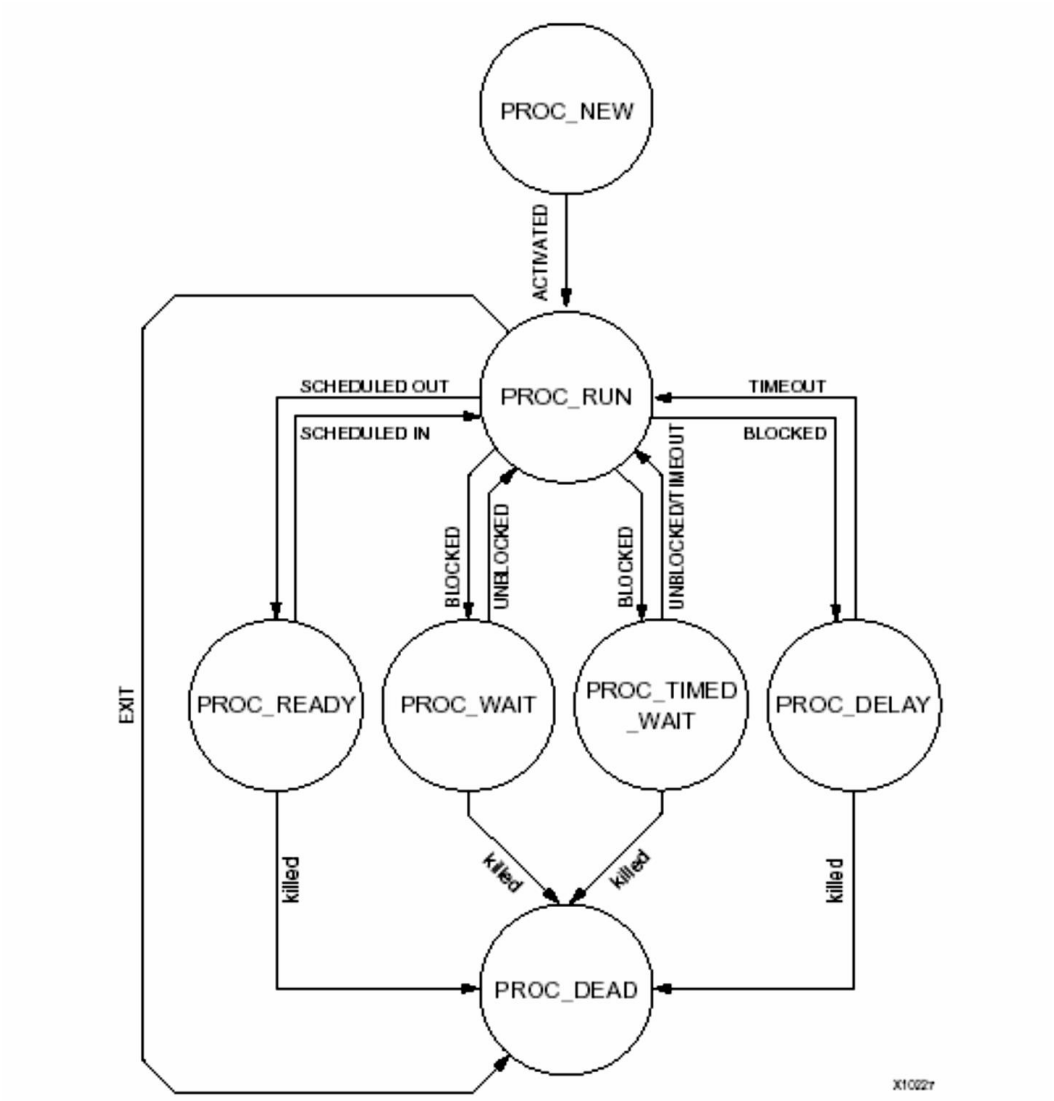

Figure III-10 Les états des processus.

III.6.3. Flot de conceptions sous Xilkernel [14]

Xilkernel est organisé sous la forme d'une librairie de fonctions. Pour construire l'image du noyau, Xilkernel doit être inclus dans la plateforme software. Il doit être configuré convenablement, pour exécuter le générateur de librairies d'EDK LIBGEN et obtenir la librairie LibXilkernel.a.

Les applications peuvent être développées séparément ou bien au sein même de la plateforme software SDK. Une fois ces applications construites, pour compiler l'image finale du noyau Xilkernel, un lien doit être fait avec la librairie LibXilkernel.a. Par conséquent, toutes les fonctionnalités du noyau y seront incluses. Figure III-11 montre ce flot de développement.

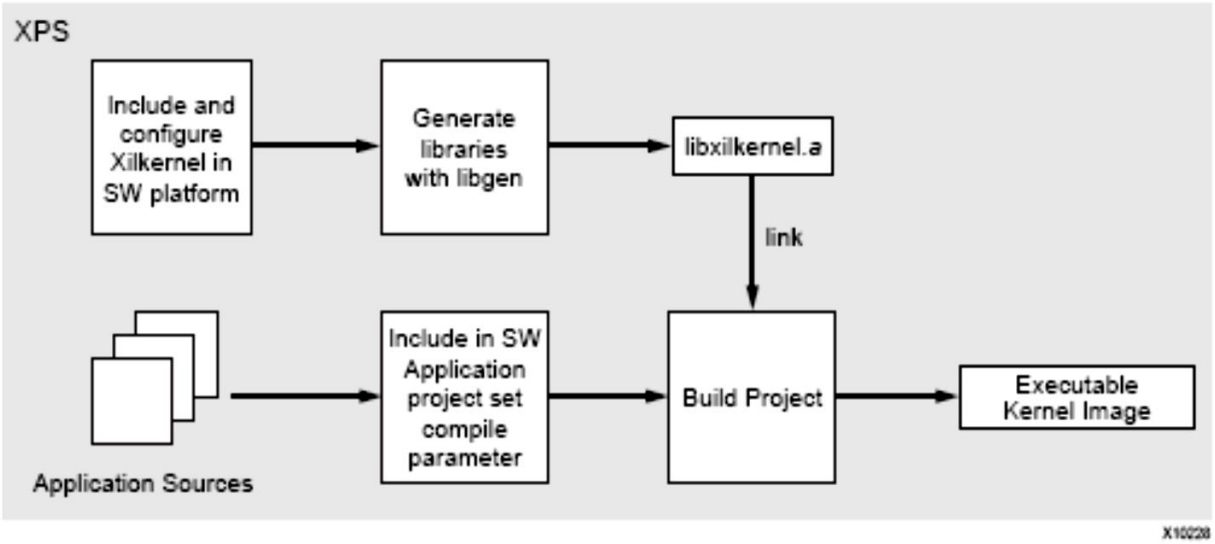

Figure III-11 Construction d'application pour Xilkernel

Contrairement aux systèmes d'exploitation classiques, *Xilkernel* crée un même exécutable contenant et l'image du noyau et les exécutables des applications.

III.6.2 Soft sous Windows

Dans ce qui suit, nous aborderons la description de l'application Windows chargée de la liaison entre la carte de commande et la conception sur SolidWorks. Cette application est devisée en 3 parties :

- **La première partie :** assure le transfert de données, via le port USB, entre la carte FPGA et la 2eme partie en gérant cette dernière à l'aide des librairies appropriées.
- **La deuxième partie :** s'occupe du codage des données à envoyer ou le décodage des données reçues par d'une ou l'autre des deux autres parties.
- **La troisième partie :** c'est au niveau de cette partie qu'on utilise les fonctions et les objets définis par l'API SolidWorks pour le transfert de données au/du prototype virtuel.

La figure III-12 donne un aperçu sur le flot de données de la partie software sous Windows.

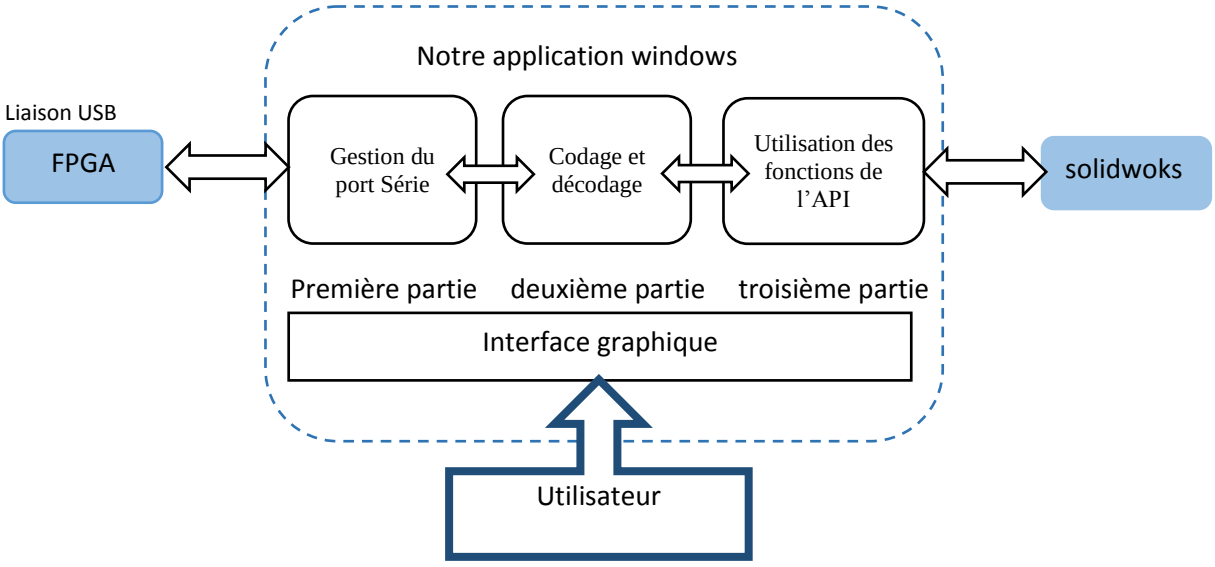

Figure III-12 Flot de données de la partie software sous Windows

L'interface graphique

Le rôle de cette interface est de fournir à l'utilisateur les fonctionnalités suivantes :

- Le lancement du logiciel SolidWorks et l'ouverture du document contenant le prototype à commander.
- Le choix du port série sur lequel la carte de commande est branchée.
- L'établissement de la connexion avec la carte ou éventuellement la déconnexion.
- Quitter l'application.

III.7. Conclusion

Ce chapitre nous a permis d'avoir un aperçu général sur les outils de développement utilisés dans notre projet, ainsi que le flot de conception et les algorithmes de fonctionnement des différentes parties : hardware et software. Nous avons vu aussi que l'utilisation du noyau Xilkernel permet une bonne gestion des différents processus.

IV.1. Introduction

Notre travail consiste à interfacer SolidWorks avec une électronique externe qui se chargera de piloter le prototype virtuel de la main DLR.

Un des grands avantages du prototypage virtuel est sa flexibilité qui nous a permis une dissociation entre la partie mécanique et la partie motorisation de la main. Ce qui est impossible dans une réalisation physique du prototype, ce concept est illustré dans les figures IV-1 et IV-2.

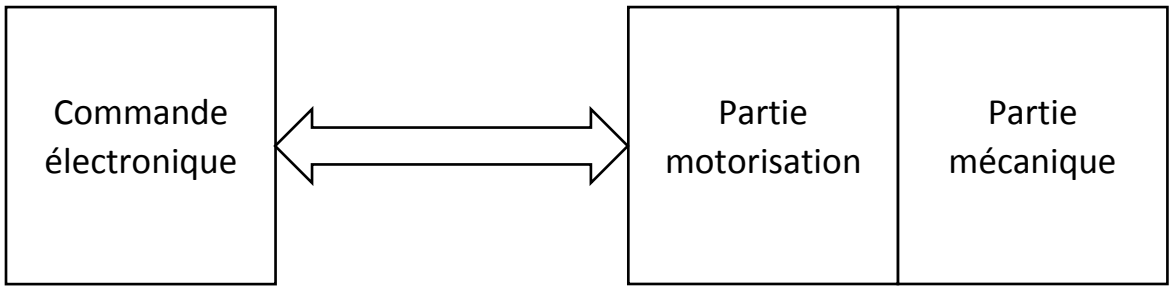

Figure IV-1 Cas d'une réalisation physique

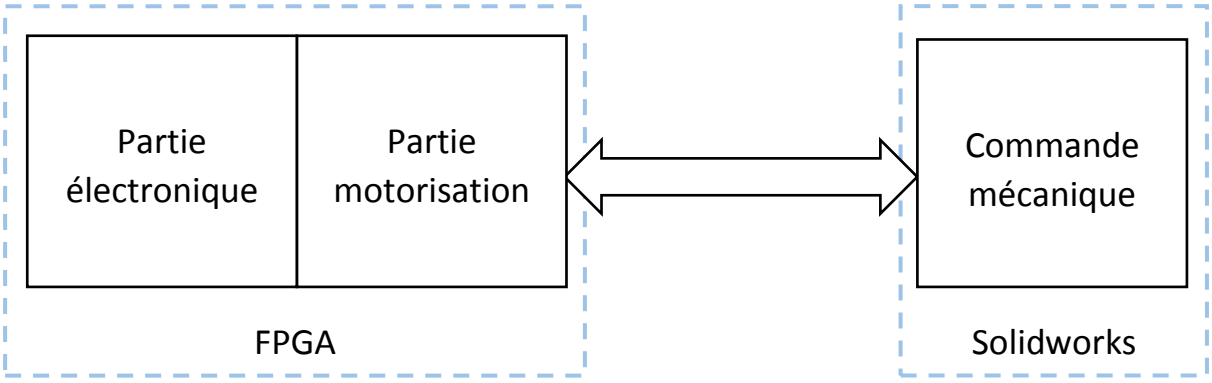

Figure IV-2 Cas d'une réalisation virtuelle

Dans notre cas, cette approche se traduit par l'intégration de la partie motorisation dans le circuit de commande sous forme de tableaux contenant les angles décrivant la trajectoire du mouvement du doigt. Ces angles sont les résultats d'une simulation sous MATLAB des différents moteurs déjà modélisés sous cet environnement.

Remarque : Un moteur est modélisé par une fonction de transfert ayant comme entrée une tension 'V' et un angle θ comme sortie.

Notre travail peut être exposé comme suit. Pour chaque doigt de la main, on a des tableaux de valeurs décrivant une trajectoire pour ce dernier tel que le mouvement de chaque doigt est géré par un *thread* généré par le système d'exploitation Xilkernel et qui est indépendant des autres threads. Le déclenchement du mouvement d'un doigt est fait avec l'un des boutons poussoirs disponibles sut la carte FPGA. Le travail structuré selon les parties suivantes : la mise en œuvre de la plateforme matérielle, la réalisation de la partie software sous EDK, la réalisation de la partie software sous Windows et, enfin, les résultats et conclusions.

IV-2 La mise en œuvre de la plateforme matérielle

IV.2.1 Configuration de la carte

La plateforme matérielle est créée à l'aide du BSB en suivant les étapes suivantes :

- Lancement du logiciel XPS.
- Sélection du modèle de la carte qui est dans notre cas Spartan 3E.
- Choix du processeur MICROBLAZE, seul processeur disponible pour ce modèle de la carte, ainsi que la fréquence d'horloge qu'on la prend égale à 50 Mhz et la taille de la BRAM qui sera 8Ko.
- Sélection et configuration des interfaces d'entrée/sortie du système qui sont :
    - RS232_DCE qui sera utilisé comme périphérique d'entrée/sortie standard.
    - BUTTON_4BIT qui sera utilisé pour déclencher les différents processus.
    - DDR_SRAM, son utilisation est nécessaire pour charger notre application, vue sa taille importante.
    - Un timer qui va générer les *ticks* pour le MicroBlaze.
- L'étape finale de cet assistant nous donne un résumé pour tous les périphériques choisis ainsi que leur adressage au niveau du processeur. Après confirmation, le BSB se ferme et nous renvoie vers l'interface XPS. La figure IV-3 représente le bloc-diagramme de la plateforme crée.

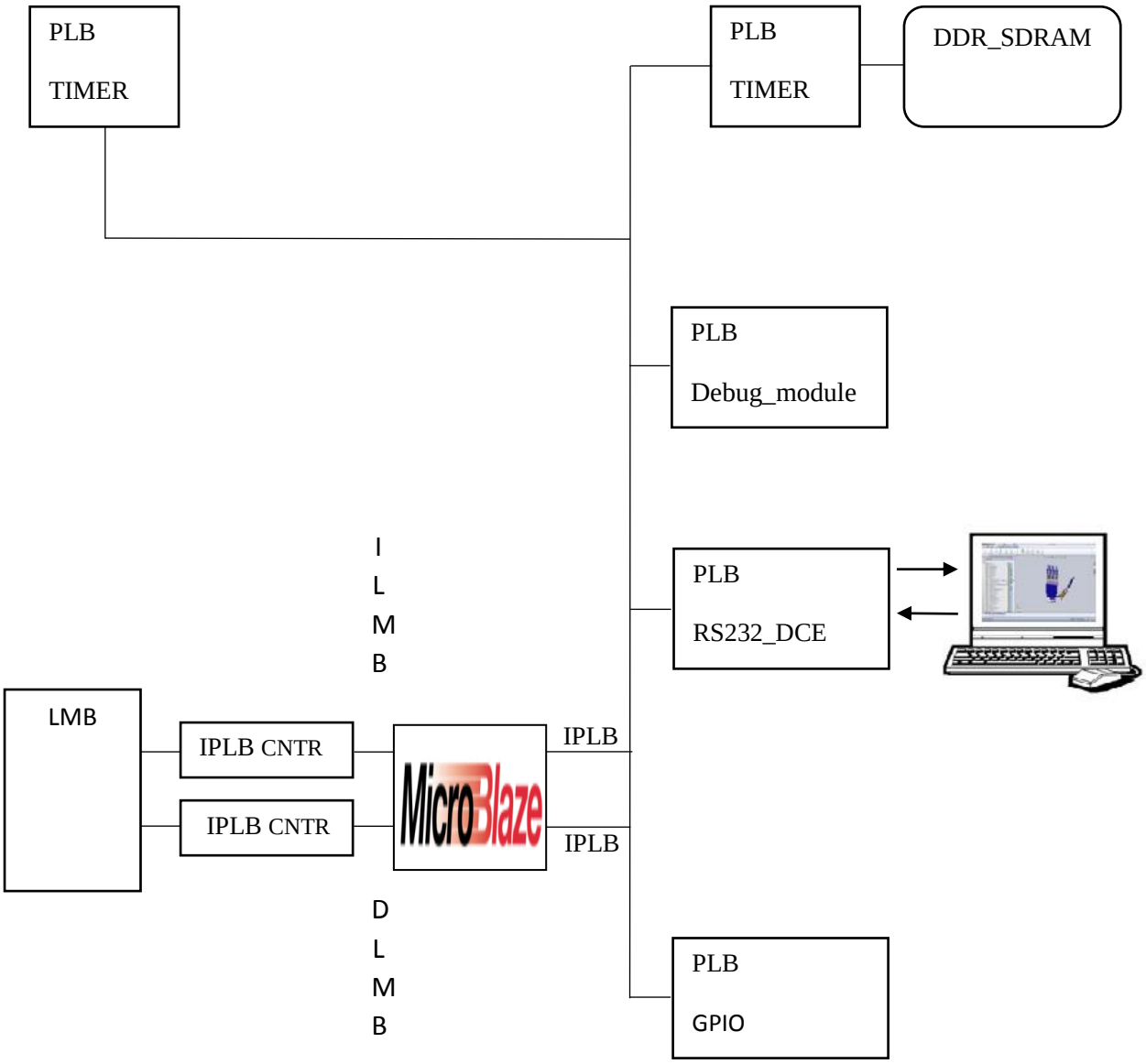

Figure IV-3 Bloc-diagramme de la plateforme matérielle.

Pour l'adressage, EDK se charge d'attribuer à chaque périphérique l'espace mémoire approprie. Le tableau VI-1 résume le système d'adressage de la plateforme adoptée ainsi que les bus de connexions.

Instance	Name	Base address	High address	Bus connexion
dlmb_cntlr	C-BASEADDR	0x00000000	0x00001fff	dlmb
ilmb_cntlr	C-BASEADDR	0x00000000	0x00001fff	ilmb
debug_module	C-BASEADDR	0x84400000	0x8440ffff	mb_plb
button_4Bit	C-BASEADDR	0x81400000	0x8140ffff	mb_plb
xps_timer_1	C-BASEADDR	0x83c00000	0x83c0ffff	mb_plb
RS232_DCE	C-BASEADDR	0x84000000	0x8400ffff	mb_plb
DDR_SDRAM	C_MPMC_BASEADDR	0x8c000000	0x8fffffff	mb_plb

Tableau IV-1 Résumé du système d'adressage

IV.2.2. Génération de la plateforme matérielle

La plateforme matérielle est créée en utilisant l'outil de génération de plateforme PlatGen. Cet outil a pour entrée le fichier MHS qui définit l'architecture du système, les périphériques, les processeurs enfouis, les connections du système, la configuration des adresses de chaque périphérique dans le système et les options de configuration de chaque périphérique. PlatGen crée donc une Netlist qui décrit le matériel utilisé en plusieurs formes (NGC, EDIF). Après l'exécution de cet outil, les outils d'implémentation sur FPGA (ISE) s'exécutent pour compléter l'implémentation du matériel. A la fin du flot ISE, un fichier de configuration (Bitstream) est généré. Le tableau suivant donne un résume sur les ressources utilisées par la plateforme ainsi générée.

Type	Utilisée	Disponible	%
Slices	2579	4656	55
Slice Flip Flops	3083	9312	33
4 input LUTs	3492	9312	37
IOs	15751	NA	NA
bonded IOBs	44	232	18

Tableau IV-2 Ressources utilisées par la plateforme matérielle

IV.3 La réalisation de la partie software sous EDK

Après la création de la partie hardware et la génération du Bitstream, on lance le SDK directement à partir de l'XPS qui nous permettra de créer notre partie software, à savoir : la plateforme software, la configuration et l'implémentation du noyau XilKernel et l'écriture de l'application software.

IV.3.1. La plateforme software

La plateforme logicielle est spécifiée par le fichier MSS (Microprocessor Software Specification) qui définit les librairies, les pilotes, les paramètres de personnalisation du processeur, les dispositifs d'entrées/sorties, les routines de traitement d'interruptions et d'autres caractéristiques du software. Le fichier MSS est utilisé comme entrée pour l'outil de génération de librairies (LibGen). Cet outil nous permet de configurer les librairies et les pilotes avec les adresses des périphériques du processeur enfoui.

Les librairies et les pilotes configurés et les programmes sources (d'extension .c ou bien .h) seront compilés grâce à mb-gcc qui est un compilateur GNU adapté à MicroBlaze.
Le résultat est un fichier exécutable qui sera chargé dans la BRAM ou la SRAM de MicroBlaze pour être exécuté.

IV.3.2. Configuration et Implémentation de XilKernel

L'un des avantages de l'utilisation du noyau XilKernel est sa disponibilité dans l'environnement EDK. Pour la configuration du noyau XilKernel on a suivi les étapes suivantes :

- Dans la première fenêtre de l'assistant "Software Platform Settings" on choisit XilKernel comme système d'exploitation. Et on ajoute –lxilkernel comme *extra_compiler_flag* nécessaire pour l'utilisation des fonctionnalités de XilKernel.
- Dans la fenêtre "OS and libraries" on configure le système d'exploitation comme suit :
  - On choisi RS232_DCE pour les paramètres de configuration *stdin* and *stdout*.
  - On choisi "*xps_intc_0* comme le "*sysintc_spec* pour spécifier le nom de l'instance du périphérique de contrôle d'interruption qui pilotera les interruptions du système.

- Dans le menu *systmr_spec* on choisit *xps_timer_1* comme systmr_dev pour spécifier le timer utilisé pour l'OS.
- Dans le menu *config_pthread_support* , on choisit *static_pthread_table* et on ajoute la fonction *test_start_func* avec une priorité de niveau 1. Cette fonction contiendra les threads crées lors du lancement de Xilkernel.
- Dans le menu *config_sched,* on sélectionne le mode tourniquet (SCHED_RR) comme mode de fonctionnement de l'ordonnanceur de tâches.
- Pour avoir la capacité d'arrêter le fonctionnement (KILL) d'un thread, on doit mettre l'option *enhanced_features* à la valeur *true*.
- On quitte l'assistant "Software Platform Settings" en cliquant sur le bouton OK, pour revenir à la fenêtre principale de SDK.

Une fois la configuration du Xilkernel est terminée, on doit générer le Linker Script qui sera utilisé pour lancer notre application directement à partir de la mémoire DDR-SDRAM et ajouter la librairie Xilkernel aux librairies du compilateur *gcc*, puis charger le Bitstream généré sur la carte FPGA pour avoir l'architecture suivante :

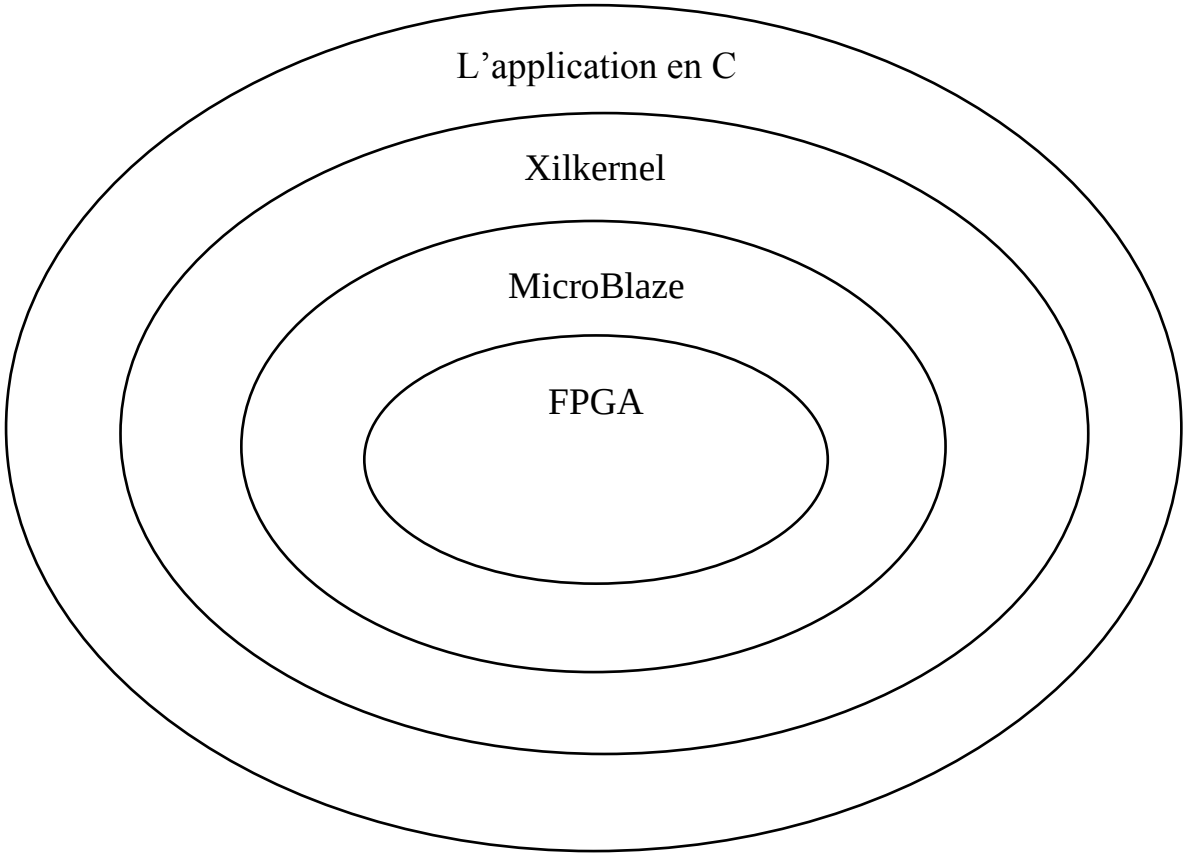

Figure VI-4 L'architecture de la partie software

IV.3.3. L'application software sous EDK

IV.3.3.1 Les threads

- Définition

Un *thread,* ou tâche ou processus léger, est similaire à un processus. Car, tous deux représentent l'exécution d'un ensemble d'instructions du langage machine d'un processeur. Contrairement aux processus, les threads partagent le même espace mémoire ce qui rend le basculement d'un thread à un autre beaucoup plus rapide dans un système d'exploitation. On aura, en conséquence, une réduction dans la consommation du temps CPU et de la mémoire utilisée.
D'une autre part, si un thread endommage le contenu de sa mémoire, les autres threads pourraient rencontrer des problèmes car ils partagent le même espace mémoire.

IV.3.3.2. Architecture de l'application

L'application doit assurer l'échange de données avec la partie software sous Windows, ce qui nous a menés à l'utilisation des threads gérés par l'OS Xilkernel. Après avoir adapté la carte et compilé le noyau temps réel, on procède à l'étape finale qui est l'écriture du programme qui doit s'exécute comme suit.

Au début, la fonction principale `main()` est exécutée automatiquement, dans laquelle on fait appel à la fonction `xilkernel_main()`, qui ce chargera de lancer le système d'exploitation Xilkernel.

Une fois le noyau appelé et initialisé, il appel son Ordonnanceur pour ordonnancer les différentes tâches du programme.

La première fonction qui sera exécuté dans notre programme est la fonction `start_test_func` qui a la priorité la plus grande et qui sera donc exécutée dès le démarrage de Xilkernel. Pour chaque doigt de notre main un thread est créé tout en lui affectant une fonction approprié (lanched_pthread1, lanched_pthread2 ou lanched_pthread3). Cette dernière contient des instructions qui traduisent lors de leurs exécutions l'organigramme de Figure IV-5.

Une priorité entre les fonctions fingerX_func a été établit pour contourner le problème du temps de réponse relativement long du logiciel SolidWorks. Finalement, on aboutit à l'architecture de Figure IV-6.

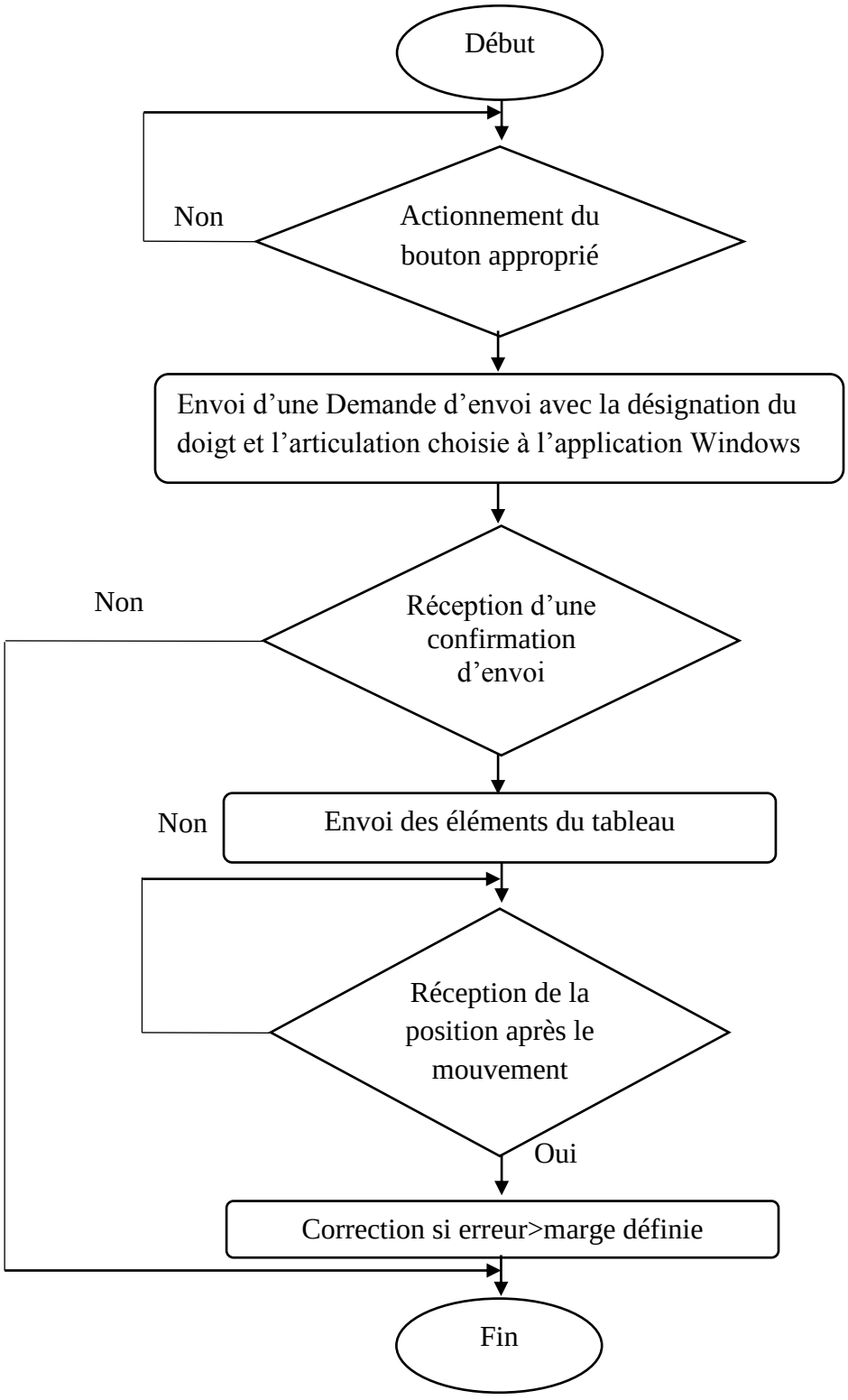

Figure IV-5 L'organigramme de la fonction gérante du doigt.

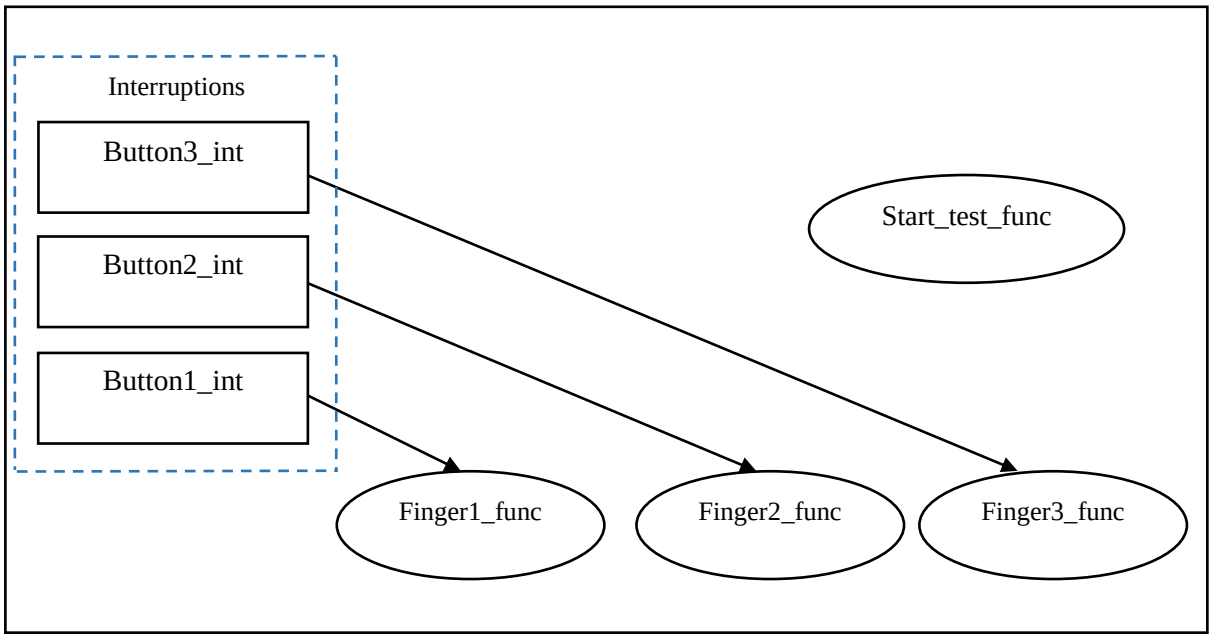

Figure IV-6 Architecture de l'application sous EDK.

IV.4.La réalisation de la partie soft sous Windows

L'application réalisée sous Visual Basic est du type *Windows Form*. Elle comporte donc une interface graphique qui permet une meilleure interactivité avec l'utilisateur et une partie code pour rendre cette dernière fonctionnelle.

Le code est structuré sous forme de fonctions et de subroutines permettant une modularité qui a pour avantage de ne pas changer tout le code lors de la modification d'une fonctionnalité.

IV.4 .1 Le fonctionnement de l'application

- Lors du lancement de l'application, une subroutine est chargée d'inspecter les ports accessibles sur la machine et les afficher à l'utilisateur afin qu'il puisse choisir le port sur lequel la carte est branchée et la configuration de ce dernier.

- La grande partie du programme est écrite sous la subroutine qui se déclenche automatiquement lors de la réception des données sur le port RS232. Elle fait appel aux fonctions suivantes :

  - `SelComponent()`

  Cette fonction permet de sélectionner le composant dont le nom est transmis comme paramètre de type chaîne de caractères. Le type d'objet de retour est Component2.

  - `Origine()`

  Cette fonction a aussi comme paramètre le nom du composant, est donne en retour les coordonnées du point d'origine du composant voulu. Le retour est du type tableau de valeurs.

  - `createTrans()`

  Cette fonction a comme retour un objet du type *MathTransform* qui sera crée à partir des paramètres suivants : coordonnées de l'origine, Axe de rotation et de l'angle de rotation autour de cette axe.

  - `Difference()`

  Cette fonction permet d'avoir en sortie un tableau de valeurs qui donne les différences entre deux éléments successifs du tableau en entrée. L'Algorithme de la partie soft sous Windows est conçu comme suit :

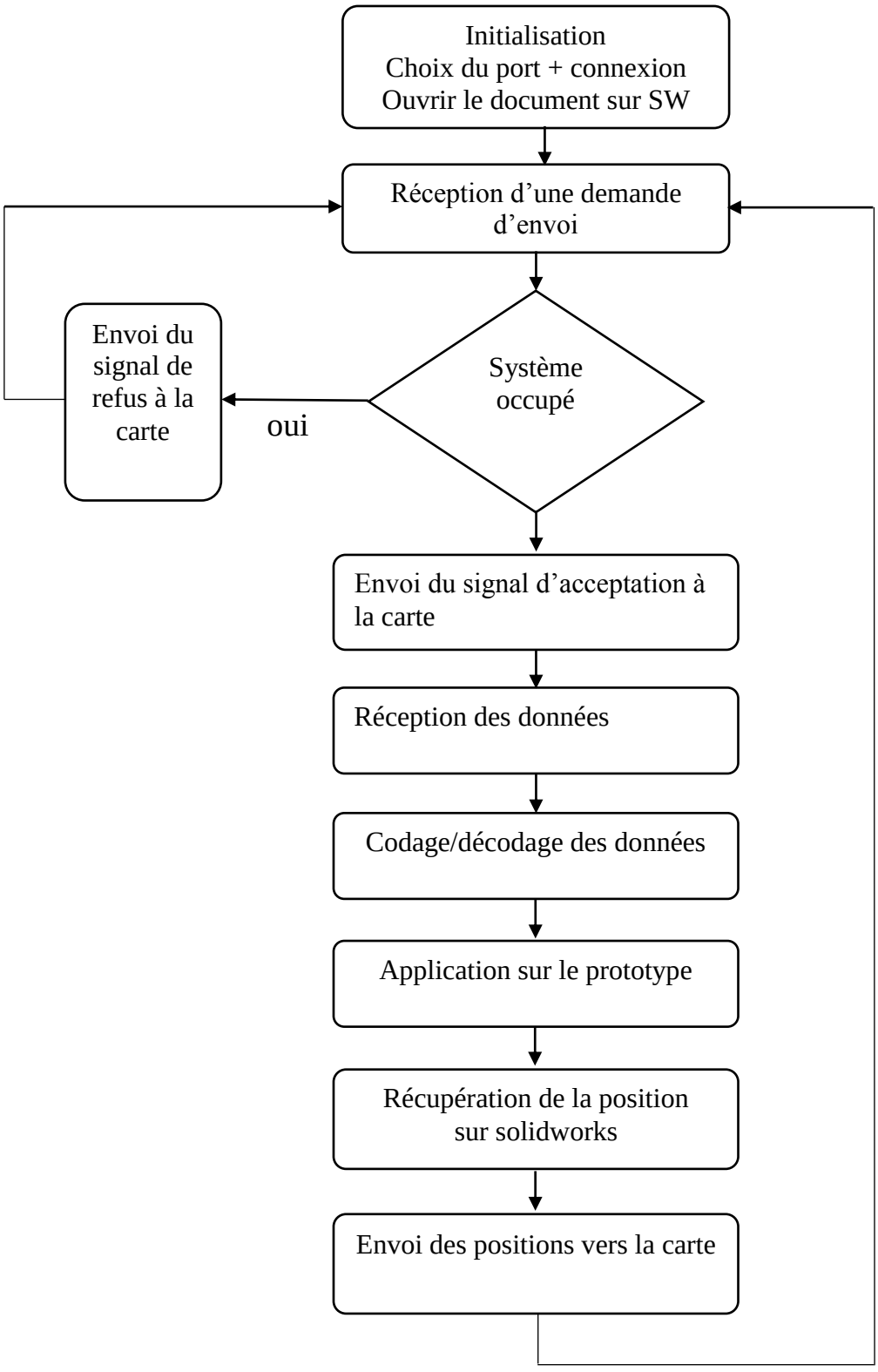

Figure IV-7 L'organigramme de la réception des données au niveau de la partie soft sous Windows

Le codage de l'information qui circule entre les deux parties software respecte les règles suivantes :

- Le transfert de données se fait par octets. Un octet envoyé par l'application sous EDK égal à 'D'est décodé comme demande d'envoi au niveau de l'application sous Windows et doit être suivi par deux octets portants le numéro du doigt et le numéro de l'articulation à piloter respectivement.
- Un octet envoyé par l'application sous Windows égal à 'O' est décodé comme confirmation d'envoi.
- Tous les octets différents de 'D' et 'O' seront considérés comme données et vont être ajoutés respectivement au tableau de données qui contient la trajectoire de l'articulation du doigt choisis.

IV.5. L'interface graphique

La figure IV-8 donne un aperçu sur l'interface de l'application développée sous Visual Basic

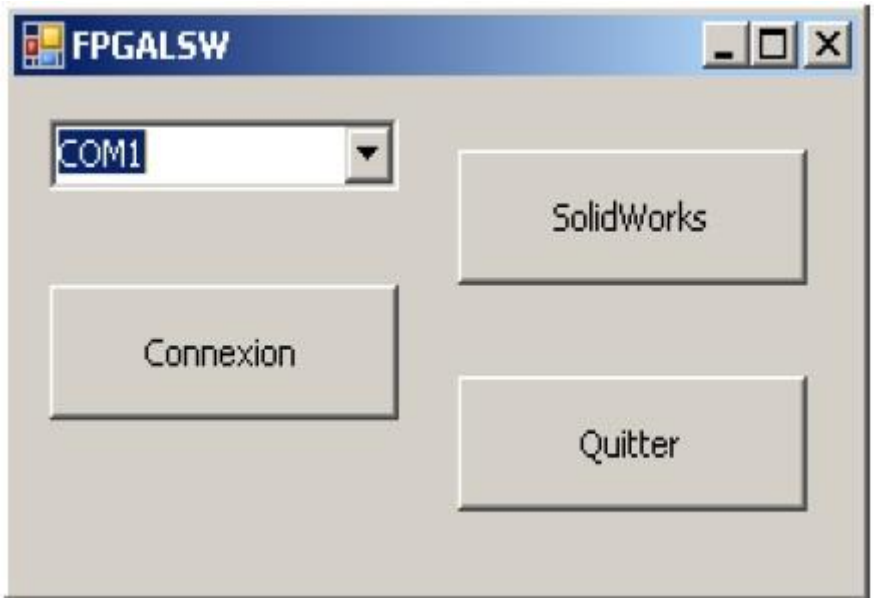

Figure VI-8. L'application Windows

IV.6. Conclusion

A travers ce chapitre, nous avons pu exploiter l'API pour établir une liaison entre la carte FPGA et solidworks et de configurer les plateformes matérielle et logicielle pour réussir la commande de la modélisation.

Il est clair aussi que l'aspect modulaire qu'offre la programmation RTOS permet d'avoir plusieurs tâches dans le même programme au lieu d'un seul grand programme ; ce qui rend le programme plus facile à comprendre, à développer et à maintenir.

Conclusion générale

Dans ce projet, nous avons abordé la réalisation d'une commande électronique, sous forme d'un SoC, d'une modélisation SolidWorks. Cette commande électronique est réalisée à l'aide d'une carte FPGA Spartan 3E XC3S500E. La modélisation SolidWorks concerne une main artificielle DLR/HIT II. Pour ce faire, l'étude préalable des caractéristiques géométriques de cette main est nécessaire.

Une application Windows a été développée assurant l'interfaçage entre la carte électronique et la conception de la main DLR/HIT II. Cette application est basée essentiellement sur l'API SolidWorks.

La première contrainte était le temps alloué au projet qui a limité notre travail à la réalisation d'une commande pour une architecture SoC à un seul processeur. Le fonctionnement en temps réel était assuré par l'utilisation d'un système d'exploitation temps réel (XilKernel), pilotant un processeur soft (MicroBlaze) fourni avec l'outil de développement EDK de Xilinx.

La deuxième contrainte était le temps de réponse relativement long de SolidWorks, lors de l'exécution des commandes reçues par la carte. Ce qui nous a poussé à envoyer les commandes séquentiellement pour chaque doigt.

L'extension de ce travail est possible sur plusieurs plans

- L'Intégration de la partie motorisation dans l'application Windows pour avoir une commande plus réaliste.
- Développement d'une architecture MPSoC pour la commande pour avoir une indépendance entre le fonctionnement des doigts.
- Développement d'une application Windows de sorte qu'elle devienne indépendante du modèle SolidWorks choisi.

Bibliographie

[**1]** : M. C. Carrozza, B. Massa, S. Micera, M. Zecca, P, « A "Wearable" Artificial Hand for Prosthetics and Humanoid Robotics Applications » , Dario_Scuola Superiore Sant'Anna, Pisa, Italy and Centro INAIL RTR, Viareggio (Lu), Italy,2001.

[2]: Prédiction des efforts musculaires dans le système main avant-bras : Modélisation, simulation, optimisation et validation. Joe CHALFOUN. Thèse de doctorat, 29 avril 2005.

[3] : The modular multisensory DLR-HIT-Hand, H. Liu a, P. Meusel a, N. Seitz a, B. Willberg a, G. Hirzinger a, M.H. Jin b, Y.W. Liu b, R. Wei b, Z.W. Xie b. Proceedings of the 2005 IEEE/ASME, International Conference on Advanced Intelligent Mechatronics Monterey, California, USA, 24-28 July, 2005

[4] : Joe CHALFOUN **, «**Prédiction des efforts musculaires dans le système main avant-bras : Modélisation, simulation, optimisation et validation», Thèse de doctorat en robotique, Université de Versailles Saint-Quentin-en-Yvelines,2005.

[5] : Jimmy W. Soto Martell and Giuseppina Gini, « Robotic Hands: Design Review and Proposal of New Design Process», Conference,World Academy of Science, Engineering and Technology, Milano, 27th 2 2007.

[6] : M.C. Carrozza, F.Vecchi, S. Roccella, L. Barboni, E. Cavallaro, S. Micera, P. Dario, « The adah project: An astronaut dexterous artificial hand to restore the manipulation abilities of the astronaut»,7th ESA advanced Space Technologies for Robotics and Automation "ASTA 2002" ESTEC, Noordwijk , The Netherlands, November 19-21 2002.

[7] : J. Butterfa_, M. Grebenstein, H. Liu and G. Hirzinger, « DLR-Hand II: Next Generation of a Dextrous Robot Hand »,International conference on robotics automation, Seoul KORIA, May 21-25 2001.

[8] : Maxime Nicole, « SolidWorks-Bases_et_Pieces»,Cours master , Université de Sherbrooke, CANADA, 2005.

[9] : SolidWorks API Help 2008.

[10] : Peter Magnusson , «Evaluating Xilinx MicroBlaze for Network SoC solutions » , *master thesis in Computer Engineering*, SWEDEN, 10th January 2004.

[11] : “EDK OS and libraries Reference Manual”, E*mbedded Development Kit 6.3i.* UG 114(V3.0), 20 aout 2004.

[12] : MicroBlaze Processor Reference Guide Embedded Development Kit EDK 10.1i UG081 (v9.0).

[13]: jean j.LABROSE, “MicroC/OS-II the real-time kernel”, CMP books Lawrence, kansas66046, USA, 1999.

[14] : Torstein Wroldsen Ståle Tveitane, *“Real Time Operating System for embedded platforms”,* Thèse de magister en technologie de communication et d’information. Grimstad, Mai 2004.

[15] : “Embedded System Tools Reference Manual”, E*mbedded Development Kit 6.3i.* UG 111(V3.0), 20 aout 2004.

[16]: Platform Studio User GuideXPS , E*mbedded Development Kit 6.3i.* UG 113(V3.0), 20 aout 2004.

[17]: Spartan-3E FPGA Starter Kit Board User Guide UG230 (v1.1) June 20, 2008.