République Algérienne Démocratique et Populaire

Ministère de L'Enseignement Supérieur et de la recherche scientifique

Ecole Nationale Polytechnique

Département d'Electronique

Projet de fin d'études

Pour l'obtention du titre

d'Ingénieur d'Etat en Electronique

THEME :

Utilisation d'une carte FPGA dans la contribution à la réalisation d'un simulateur de vol

Présenté par :

Mr. CHIKOUCHE Mohamed Mahmoud

Mr. DJOHOR Yakoub

Proposé par :

Mr. R.SADOUN

Promotion juin 2011

Ecole Nationale Polytechnique

10, Avenue Hacen Badi, El-Harrach, Alger

الملخص

الهدف من هذا المشروع، القيام بعملية وصل بين شريحة FPGA Altera DE2 التي تحتوي على بروسيسور محمل من نوع Nios II مع محاكاة الطيران Microsoft Flight Simulator X . استعمال الشريحة سيكون على شكل HIL من أجل أن تلعب دور لوح القيادة لمقصورة الطيران متفاعلة مباشرة مع محاكاة الطيران عن طريق API الخاصة به.

الكلمات المفتاحية : علم الطيران ، محاكاة الطيران ، Flight Simulator X API ، FPGA ، SoC (SoPC) ، HIL.

Résumé

Notre projet à pour objectif, d'interfacer la carte FPGA DE2 d'Altera contenant le processeur embarqué Nios II avec le simulateur de vol de Microsoft Flight Simulator X. la carte sera utilisée comme HIL pour jouer le rôle d'un mini cockpit interagissant directement avec le simulateur de vol en faisant appel à ses API.

Mots clés : aéronautique, simulateur de vol, API Flight Simulator X, FPGA, SoC (SoPC), HIL.

Abstract

Our project aims to interface the FPGA Altera DE2 board that contains the Nios II embedded processor with the Microsoft Flight Simulator X. the DE2 board will be used as HIL to act as a mini cockpit, interacting directly with the flight simulator using its API.

Keywords: aviation, flight simulator, Flight Simulator X API, FPGA, SoC (SoPC), HIL.

Remerciements

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

Nos remerciements vont exceptionnellement à Monsieur R.SADOUN, pour son aide, son suivi, ses conseils et directives ainsi que pour son dévouement et ses sacrifices. Qu'il trouve ici notre sincère gratitude.

Nous exprimons notre profonde reconnaissance aux membres du jury d'avoir accepté de juger notre travail.

Que tous nos professeurs qui ont contribué à notre formation trouvent ici notre plus profonde gratitude.

Nous adressons nos plus sincères remerciements à tous ceux qui ont contribué, de près ou de loin, à l'aboutissement de ce travail.

Enfin, nous souhaitons dédier ce mémoire à nos parents. Rien n'aurait été possible sans leur soutien, confiance et générosité.

DEDICACES

A mes parents, mes parents, mes parent, et encore mes parents qui m'ont toujours soutenu et poussé à devenir ce que je suis aujourd'hui

A mes grands-parents

A mes deux sœurs : Amina et Rym qui resteront toujours ma source d'inspiration

A mes tantes et oncles

A mes cousins et cousines

A tous mes amis

A mon promoteur M. Sadoun

A mon binôme : Yakoub

A tous ceux qui ont su croire en moi

A tous ceux qui me sont chers

Je dédie ce modeste travail

Mohamed Mahmoud

DEDICACES

Au deux personnes qui comptent le plus à mes yeux dans ce monde, mes très chers parents, qui on fait de moi ce que je suis

A mes grands-parents

A mes deux frères : Mohamed et Nafaa

A ma sœur Zahra qui occupe une place unique dans ma vie

A mes tantes et oncles, cousins et cousines qui ont donné un sens au mot « famille »

A tout mes amis et spécialement ceux qui sont présents en ce jour mémorable

A mon promoteur R.SADOUN

A mon binôme Mahmoud

A toute personne présente dans ma vie et qui m'est chère

Je dédie ce modeste travail.

Yakoub

Table des notations

API	: Application Programming Interface
ASIC	: Application-Specific Integrated Circuit.
FPGA	: Field-Programmable Gate Array.
FSX	: Flight Simulator X
HIL	: Hardware-In-the-Loop
HILS	: Hardware-In-the-Loop Simulation
IDE	: Integrated Development Environment
JTAG	: Joint Test Action Group
MFS	: Microsoft Flight Simulator
MIPS	: Million Instructions per Second
PLL	: Phase-Locked Loop.
RISC	: Reduced Instruction Set Computer.
SDK	: Software Development Kit
SDRAM	: Synchronous Dynamic Random Access Memory.
SOC	: System On Chip.
SOPC	: System On Programmable Chip.
UART	: Universal Asynchronous Receiver Transmitter.
VHDL	: Very High Speed Integrated Circuit Hardware Description Language

Liste des figures

Sommaire :

Introduction générale

Depuis la nuit des temps, l'homme cherche toujours à comprendre l'univers qui l'entoure pour mieux l'exploiter, et ceci se manifester la plupart du temps par des équations mathématiques. Leur extension ne couvre pas uniquement la Physique, bien au contraire, on les trouve aussi dans plein d'autre domaines tels que : la Médecine, la Chimie et bien d'autre. L'évolution des calculateurs et les outils de modélisation a permis d'avoir une approche de plus en plus réelle ce qui a donné naissance à une nouvelle science connue sous le nom de : réalité virtuelle.

Cette science est souvent définie comme étant une simulation informatique interactive immersive, visuelle et sonore, d'environnements réels ou imaginaires. Son but est de permettre à une personne (ou à plusieurs) une activité sensori-motrice et cognitive dans un monde artificiel, créé numériquement, qui peut être « imaginaire, symbolique ou une simulation de certains aspects du monde réel ».

A l'heure actuelle, les prototypes virtuels sont considérés comme la partie la plus importante avant d'aboutir au modèle réel, nécessitant ainsi une gestion intelligente des ressources, aussi bien sur le versant applicatif que sur celui des outils utilisés. Les exigences que doit remplir la partie électronique pour commander le prototype virtuel doivent 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.

Notre travail consiste donc à interfacer la carte DE2 d'Altera à Microsoft Flight Simulator X en utilisant un système embarqué à base du processeur NIOS II, mettant en œuvre ainsi notre contribution à la réalisation d'un simulateur de vol.

Pour atteindre cet objectif, nous avons organisé notre travail comme suit :

- Le premier chapitre donne une idée générale sur les simulateurs de vol, et les différents instruments de bord, ainsi passer à la méthodologie de sa conception en utilisant la technique HIL.

- Le second chapitre portera sur le logiciel Flight Simulator X, ainsi que la possibilité d'interagir avec lui à travers ses API, en utilisant le langage C++.

- Le troisième et dernier chapitre expliquera en détails les démarche nécessaires à la contribution de la réalisation d'un simulateur de vol, la mise en œuvre des parties hardware et software du projet ainsi que le codage des données pour assurer l'interfaçage.

CHAPITRE I

Simulateur de vol et Hardware-in-The-Loop

I-1 Introduction

Le domaine de l'aéronautique est l'un des domaines les plus complexes dans le développement et la conception, mais aussi l'un des plus couteux. Des solutions pour remédier à cette lacune ont été trouvées dans le concept de la virtualisation.

Dans ce chapitre on va aborder pour commencer les simulateurs de vol ainsi que leur constitution pour passer ensuite à la présentation des différents simulateurs de vol et la méthodologie de conception de ces derniers. On terminera par la présentation des instruments présents dans l'aéronautique.

I-2 Rétrospective sur les simulateurs de vol

I-2-1 Les pionniers

La difficulté du pilotage a fait qu'on a eu recours très rapidement à des entraîneurs pour les pilotes et cela dès les débuts de l'aviation, c'étaient des simulateurs très simplifiés au service de l'apprenti pilote où il pouvait répéter les manœuvres de base sur des commandes fictives. L'idée de base était de restituer les effets aérodynamiques des commandes sur un avion simplifié fixé au sol et placé dans le lit du vent. L'un des premiers entraîneurs fut le « tonneau Antoinette » construit en France en 1910, mais le premier vrai simulateur de vol a vu le jour en 1929, réalisé par Edwin Link et connu sous le nom de Link Trainer, il se composait d'une cabine posée sur un mouvement électropneumatique dont les positions répondaient aux commandes du pilote.

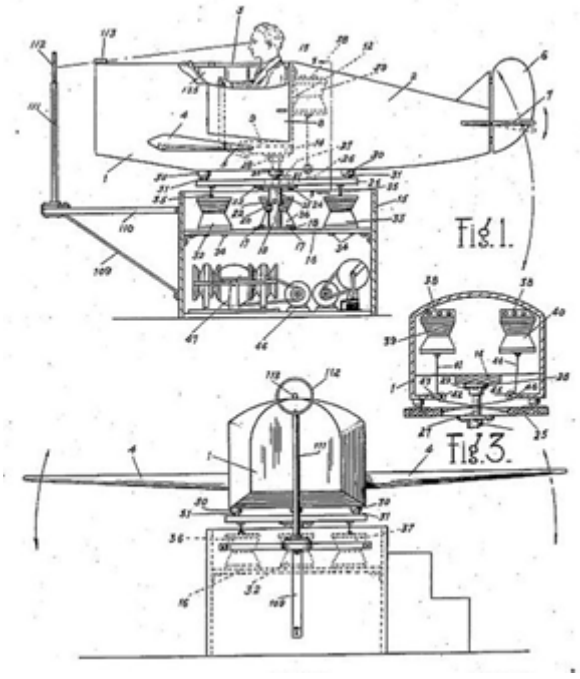

Figure I.1 : Brevet d'Edwin Link

I-2-2 L'approche scientifique

Le développement technologique tel que la mise en œuvre de modèles de vol sur calculateurs analogiques puis numériques, a donné un tout autre aspect au simulateur de vol. Le grand projet Whirlwind du Massachusetts Institute of Technology en 1946 fut de concevoir et mettre au point un calculateur numérique en temps réel (nécessaire à un simulateur de vol militaire) permettant de représenter plus fidèlement le comportement d'un aéronef en vol.

Outre la puissance de calcul souvent insuffisante, l'aspect visuel n'était pas très développé, il se limitait à des images rendues par une caméra vidéo survolant mécaniquement une maquette réelle, l'introduction d'images de synthèses n'a fait son apparition qu'au milieu des années 1970, l'évolution des images de synthèses a considérablement avancée lors des dernières années pour arriver jusqu'aux images de synthèses 3D texturées.

I-3 Constitution d'un simulateur de vol

Un simulateur de vol comporte : un calculateur numérique et un ensemble matériel permettant au pilote d'introduire les commandes pour ensuite percevoir les réactions de l'aéronef.

I-3-1 Commandes

Les commandes mises à la disposition du pilote doivent obligatoirement reproduire l'opération réelle le plus fidèlement possible autrement dit aussi identiques que possible à celles de l'aéronef réel. Dans certains simulateurs on peut même trouver des éléments réels intégrés, si ce n'est des cabines entières découpées d'un prototype ou une maquette.

I-3-2 Modèle

L'aéronef est représenté par un modèle numérique, utilisant les équations de l'aérodynamique. À partir d'un état donné, l'ordinateur calcule :

- la position : coordonnées et altitude ;
- l'attitude : position par rapport à l'horizon ;
- le vecteur vitesse : déplacement par rapport au sol, taux de montée ;
- d'autres éléments liés à l'avion : paramètres moteur par exemple.

Le calcul de ces variables se fait sur des périodes très petites de l'ordre du dixième de seconde pour les avions à évolutions normales, ces périodes peuvent être raccourcies pour les avions à manœuvres brutales tel que les chasseurs militaires. Le calculateur doit être capable d'effectuer tous ces calculs dans un temps inférieur à celui de la période de référence : il travail en « temps réel ».

I-3-3 Restitution de l'environnement

C'est la partie la plus difficile à réaliser et la plus gourmande en capacité mémoire et en puissance de calcul. Il s'agit de reconstituer le plus fidèlement possible l'environnement réel du pilote lors d'un vol [1].

I-3-4 Restitution visuelle :

La restitution visuelle comporte l'affichage des instruments de bord (indispensable), la vue du sol (indispensable pour les phases du vol telle que l'atterrissage, mais moins utile pour le vol aux instruments)et l'environnement tel que l'éblouissement par le soleil.

la restitution visuelle se fait à travers des écrans spécifiques, une projection est faite sur un écran placé en face du cockpit pour projeter une vue du sol, la restitution de l'environnement se fait en utilisant des moniteurs dont l'image est vue par l'intermédiaire d'un miroir sphérique.

I-3-5 Restitution physique :

les impressions physiques ressenties par le pilote sont liées à l'assiette de l'aéronef, aux accélérations et aux efforts sur les commandes.

La restitution physique consiste dans le fait de créer des effets artificiels reproduisant par la les sensations physiques lors d'un vol tel que les différentes accélérations ressenties par le pilote durant le pilotage. L'effet artificiel peut ne pas être produit que par les actionneurs de la plate-forme, d'autre méthodes sont utilisées comme des coussins gonflables censés simuler les sensations d'écrasement.

On peut parler aussi dans la restitution de l'environnement de la restitution auditive qui consiste a reproduire les bruits émis durant un vol [1].

I-4 Les différents simulateurs de vol

Les simulateurs de vol peuvent être classés en deux grandes familles : les simulateurs de vol professionnels et les simulateurs en jeu vidéo.

I-4-1 Simulateurs de vol professionnels

Les simulateurs de vol professionnels sont utilisés soit pour des études, soit pour des entraînements, leur constitution dépends principalement de leur utilisation.

I-4-1-1 Simulateurs d'études

Les simulateurs de vol d'études sont utilisés pour le développement de nouveaux avions, leur utilisation peut se faire durant toute les étapes de développement. Pour un même

avion, différents simulateurs de peuvent utilisés, pour chaque étape de développement on utilise le simulateur approprié et on néglige les autres paramètres, par exemple, lors de la conception du calculateur de bord on pourra se passer du simulateur de la visualisation externe.

I-4-1-2 Simulateur de formation/entraînement

Le nom de simulation de formation/entraînement peut induire en erreur le lecteur, en effet ce genre de simulateur ne sert pas à l'apprentissage du pilote mais plutôt pour faire des entraînements très spécifiques destinés à un personnel déjà pilote. L'objectif principal de ce simulateur est l'acquisition de nouvelles connaissances pour les nouveaux avions ainsi que pour les procédures à suivre lors de certaines phases de vol bien précises.

I-4-1-3 Simulateur d'enquête liée aux accidents

Les enquêteurs cherchent à reconstituer l'enchaînement des faits ayant conduit à un accident grâce aux enregistrements des boîtes noires. Le simulateur permet une approche qualitative de la situation à laquelle l'équipage s'est trouvé confronté et d'en tirer des enseignements pour l'amélioration éventuelle des interfaces ou des procédures.

I-4-2 Simulateurs en jeu vidéo

Les simulateurs de vol ont été parmi les premiers types de simulateurs développés dans le domaine des jeux vidéos pour les ordinateurs individuels. Le mot « ouvert » peut être associé à la plupart de ces simulateur, car ils permettent à l'utilisateur d'ajouter ses propres créations. Avec une certaine maitrise de logiciels 3D l'utilisateur peut devenir « créateur d'avion » en réalisant ses propres modèle d'avion [1].

I-4-2-1 Simulateurs de vol spatial

Les simulateurs de vol spatial peuvent être considérés comme une extension des simulateurs de vol précédant vu que l'espace est le prolongement logique de l'espace aérien. A la différence des autres simulateurs de vol, le simulateur de vol spatial développe un plus grand réalisme du rendu de la haute atmosphère et des voyages interplanétaires.

I-4-2-2 Poste de pilotage

Certains amateurs de jeux de simulations de vol ne se contentent pas de manipuler les avions via leurs ordinateurs personnels, ils vont encor plus loin et réalisent des postes de pilotages personnalisés, pour se faire, des constructeurs mettent au service du client des gadgets se trouvant réellement sur les cockpits. L'utilisateur peut aller encor plus loin dans sa réalisation de poste de pilotage en interagissant directement avec le simulateur de vol (cas de simulateur ouvert).

I-5 Méthodologie de conception d'un simulateur de vol

I-5-1 Schéma général

la conception d'un simulateur de vol peut être séparer en trois grands blocs principaux : bloc électronique, bloc mécanique et le bloc software.

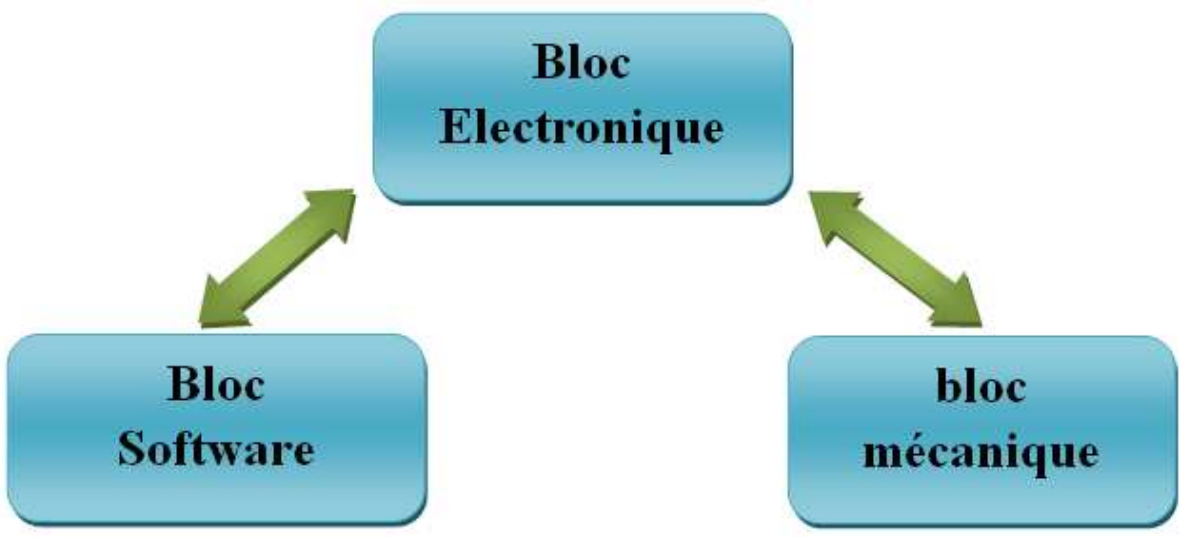

Figure I.1 : Schéma général de conception d'un simulateur de vol

le bloc électronique contient les composants hard utilisés comme outils de commandes de notre simulateur, il envoie la commande au bloc software qui est représente la partie virtuel du simulation de vol, ce dernier interagie aussi avec le bloc électronique en lui envoyant les données de vol qui seront affichées sur notre dispositif.

le bloc mécanique est le bloc qui donne les effets artificiels tel que les turbulences et inclinaisons de l'avion, la liaison entre ce bloc et le bloc électronique est bidirectionnelle elle aussi, il reçoit le données de pour faire bouger les actionneurs et il agit à son tour sur le bloc électronique avec les effets accompagnants la situation (comme une force supplémentaire pour redresser le manche lors d'une décente). Il peut être partie intégrante du bloc électronique [1].

I-5-2 Hardware-In-the-Loop « HIL »

I-5-2-1 Concept

Le Hardware-In-the-Loop, traduit en langue français, donne « matériel dans la boucle », il est concrétisé par l'ajout d'un élément dans la boucle de notre système pour pouvoir faire des tests. Il y'a plusieurs formes de HIL, la plus utilisée est le Hardware-In-the-Loop Simulation « HILS ».

La simulation Hardware-in-the-loop est une méthode de simulation caractérisée par l'association de véritables composants, connectés à une partie temps-réel simulée. Le HILS peut être assimilé à un simulateur temps réel, sauf qu'il a la particularité d'avoir des composants réels dans la boucle.

Habituellement, les systèmes de contrôles matériel et logiciel sont identiques à ceux retenus pour la production en série. Le processus à contrôler (actionneurs, système physique, et ses capteurs) peut être composé soit d'éléments simulés, soit d'éléments réels. En général, un mixe des deux est réalisé.

Fréquemment, les actionneurs sont réels, et le système physique ainsi que les capteurs sont simulés. Cela s'explique par le fait que les actionneurs et systèmes de contrôle font, la plus part du temps, partie du même sous-ensemble, ou bien que les actionneurs sont difficilement modélisables précisément et pas très simulable en temps-réel.

Pour des raisons évidentes, l'utilisation de véritables capteurs dans un système simulé peut demander de considérables efforts de réalisations. La donnée du capteur (déplacement, vitesse, température, pression etc.) doit être alors générée artificiellement. Ce qui est effectivement plus commode [2]. En d'autres termes, l'utilisation de capteur artificielle nous donne la possibilité de pousser les conditions météorologiques à l'extrême, par exemple la simulation du mois de Février, au lien d'attendre son vrai climat (Froid).

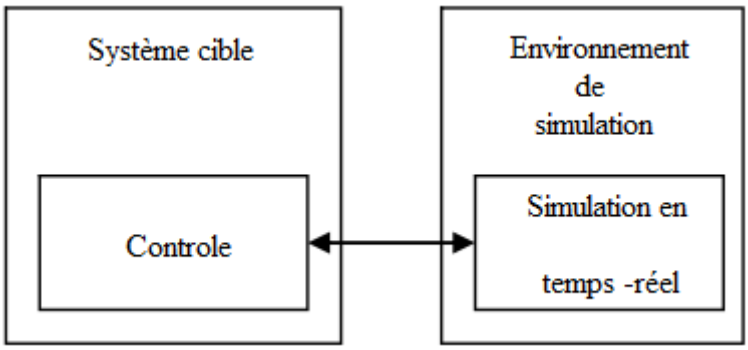

Figure I.3 : Exemple d'un système de simulation HIL

I-5-2-2 Intérêts

Les avantages offerts par la simulation Hardware-in-the-loop sont multiples :

- La conception et les tests du système de contrôle peuvent être faits sans le système réel (le travail peut s'effectuer en « labo »).
- Les conditions d'essais sur système de contrôle matériel peuvent être poussées à l'extrême (exemple : haute/basse température, fortes accélérations et choques mécaniques, compatibilité électromagnétique).
- Il est possible d'effectuer des essais sur les effets de défauts et pannes des actionneurs, capteurs et ordinateurs sur l'ensemble du système.
- Opérations et essais en conditions extrêmes et dangereuses. Expériences reproductibles et répétables à souhait.
- Opérations facilitées avec différentes interfaces hommes-machines (conception de cockpit et formation des opérateurs).
- Économie de temps et d'argent dans le processus de développement [2].

I-5-2-3 Approches et outils de développement

Il existe plusieurs types d'approches dans la technique de la simulation Hardware-In-the-Loop, cela dépend du composant ajouté à la chaine de simulation, on peut avoir des systèmes physiques tel que des actionneurs ou des moteurs, ou bien des systèmes embarqués tel que les FPGA. Le choix des concepteur se porte de plus en plus dans le développement des simulateur HIL vers les FPGA pour leur rapidité de conception, un FPGA reprogrammable

possède un avantage déterminant pour la simulation grâce à sa souplesse. Au fur et à mesure que les contraintes du système de contrôle changent et que des composantes sont ajoutées ou retirées, le FPGA est susceptible d'être reconfiguré sans modifications matérielles significatives.

L'approche FPGA dans la validation des HIL est la mieux adaptée grâce à leurs rapidités, mais la présence d'inconvénients est toujours existante. En effet, la mise en œuvre de FPGA nécessite un apprentissage des outils de développement associés, pour la conception des systèmes de contrôle embarqués on doit avoir une certaine connaissance du langage de développement VHDL, un langage pas très maniable du point de vu facilité. Pour faciliter le travail, divers outils ont fait leur apparition comme le System C afin d'élever le niveau d'abstraction du VHDL à celui des langages de type C [2].

I-5-3 HIL et le simulateur de vol

L'industrie aéronautique et spatiale est soumise à des normes et règles parmi les plus rigoureuses tous domaines confondus y compris dans la conception de systèmes de contrôle électronique. Le développement de tel systèmes incombe un coût total très élevé. L'efficacité est donc un enjeu majeur. C'est pourquoi les solutions de prototypage rapide, la génération de code automatique et les systèmes Hardware-In-the-Loop sont utilisées dans de nombreuses applications dans l'industrie aérospatiale.

Comme exemple de conception de simulateur HIL on peut citer le projet de conception de pilote automatique pour l'avion, un simulateur HIL peut nous éviter des pertes de temps et surtout d'argent dans le cas de problème de crash . . . Un autre exemple dans d'utilisation, la conception d'un cockpit en entier pour qu'il interagit avec un simulateur de vol software.

Figure I.4 : Exemple d'un simulateur HIL appliqué à l'avionique

Notre étude va porter sur un simulateur de vol de Microsoft « Flight Simulator 2006 » (présenté dans le chapitre suivant), ce simulateur de vol est un système software constituant une boucle fermée fonctionnant uniformément sous forme de simulation. L'intégration d'un maillant en plus dans cette chaine revient à faire du HIL, i.e. intégration d'un composant hard à cette boucle soft déjà prédéfinie. Pour cela, on devra utiliser une carte FPGA du concepteur Altera (vu plus en détail dans le chapitre III), de plus, on exploitera les fonctions simuler sur Flight Simulator tel que les instruments à bord du cockpit qui nous donnent les indications du vol mais aussi les fonctions de commande de l'avion qui seront gérées par la carte et pour cela on fera appel au API de Flight Simulator. Ces fonctions pourront être utilisées grâce à différents langages de programmation tels que Visual Basic, Visual C#, Visual C++ . . .

I-6 Instrumentation aéronautique

L'aéronautique est basée essentiellement sur l'utilisation des principes d'électronique dans divers instruments de commande, de communication, de navigation et de contrôle de l'avion pendant son vol.

Ces instruments de bord ou bien pour une appellation plus précise ' instruments de mesures' de bord ont pour rôle de fournir à l'équipage des indications généralement chiffrées sur les paramètres nombreux dont la connaissance est nécessaire, la navigation, la conduite et la surveillance des groupes motopropulseurs, et des servitudes de bord.

I-6-1 Classification des instruments du poste de pilotage

- **Instruments de conduite :**
  - Anémomètre (Machmètre).
  - Altimètre.
  - Variomètre.
  - Indicateur de virage.
  - Horizon artificiel.
- **Instruments de contrôle :**
  - Contrôle des pressions et/ou dépressions.
  - Contrôle des vitesses de rotation.
  - Contrôle des débits.
  - Contrôle des températures.
  - Contrôles des niveaux.
  - Contrôles divers.
- **Instruments de navigation et de radionavigation :**
    - ❖ **Navigation :**
  - Compas.
  - Conservateur de cap.
  - Centrales inertielles.
    - ❖ **Radionavigation :**
  - VOR (VHF Omnidirectional Range).
  - DME (Distance Measuring Equipment).
  - ADF (Automatic Direction Finder).

- ILS (Instruments Landing System)/MLS (Microwave Landing System) .
- GPS (Global Positioning System).

I-6-2 Instruments de vol

Ces instruments regroupent les instruments qui présentent au pilote toutes les informations nécessaires pour maintenir le bon déroulement du vol. Ils sont placés dans le tableau de bord en face du pilote le plus prés possible et cela dans une disposition bien définie (en configuration du T basique), l'horizon artificiel au centre, à sa gauche l'anémomètre, l'altimètre à sa droite et enfin le gyro directionnel (plateau de route) en dessous. Ils peuvent être présentés dans un tableau de bord classique ou bien dans un écran électronique (Glass Cockpit) .

Figure I.5 : Tableau de bord sous ses deux formes

I-6-3 Equipements de navigation

Les équipements de navigation sont les équipements qui aident le pilote pour se diriger vers sa destination. Les équipements de navigation ont un point commun spécifique propre à eux qui se manifeste dans le fait qu'ils sont pas indépendants, chaque équipement de navigation a besoin d'un signal qui doit lui être transmis pour pouvoir se positionner et indiquer la bonne direction.

I-7 Conclusion

Après avoir présenté les simulateurs de vol, leur constitution et la méthodologie de leur conception, ainsi qu'une familiarisation avec le HIL et son application dans le domaine des simulateur de vol, on peut faire un perspective de ce que sera l'architecture générale de notre application qui consistera à intégrer notre HIL (carte FPGA) dans un simulateur de vol privé.

CHAPITRE II

Flight Simulator et API

II-1 Introduction

La technologie de nos jours, évolue selon l'effet d'avalanche, chaque application engendre un ensemble d'applications ainsi de suite, d'où le développement de cette dernière propre à soi est devenue très fréquent, et indispensable pour créer des modules personnalisés, et pout cela les développeurs ont ajouté des outils permettant d'apporter un tuning personnel, connu sous le nom d'API.

Lors de ce chapitre, on aura une présentation de Flight Simulator, ensuite, on verra les principales notions d'API, et plus précisément celle du simulateur pour qu'on puisse enfin développer l'interaction avec la carte.

II-2 Présentation de Flight Simulator

Flight Simulator est un logiciel de simulation de vol le plus réussi au monde, crée et développé par Microsoft depuis les années 80. Considérait dans la plupart du temps comme un jeu vidéo, MFS était l'un des premiers produits dans le portefeuille de Microsoft, grâce à son originalité, car il est plus orienté business [3].

Le programme *Flight Simulator* a été développé à partir de l'année 1977, sous la direction de Bruce Artwick; et sa société subLOGIC l'a vendu pour divers ordinateurs personnels. En 1982, la société d'Artwick accorda une licence de développement à Microsoft pour l'IBM PC qui fut commercialisée sous le nom de *Microsoft Flight Simulator 1.00*, cette ère représente le début d'un commerce juteux concernant les simulateurs de vol.

Le développement de *Flight Simulator* est passé par plusieurs étapes durant ces trois dernières décennies que l'on peut citer [4]:

- 1982 – Flight Simulator 1.0
- 1983 – Flight Simulator 2.0
- 1988 – Flight Simulator 3.0
- 1989 – Flight Simulator 4.0
- 1993 – Flight Simulator 5.0
- 1995 – Flight Simulator 5.1

- 1996 – Flight Simulator 95
- 1997 – Flight Simulator 98
- 1999 – Flight Simulator 2000
- 2001 – Flight Simulator 2002
- 2003 – Flight Simulator 2004: A Century of Flight
- 2006 – Flight Simulator X

L'un des points les plus fort de la dernière version est le faite qu'elle soit extensible grâce à des programmes développé par les clients, ceci est possible en utilisant la SDK.

II-3 SDK Flight Simulator

Microsoft Flight Simulator SDK « Software Development Kit » est un ensemble de fichiers et de programme [5], fourni à l'utilisateur lui donnant la possibilité d'interagir avec Flight Simulator. Elle est considérée comme un outil de développement qui permet de développer, créer ou modifier le contenu de la simulation. La majorité des outils sont sous forme de ligne de commande, mais certains sont accessibles via le menu du jeu Outils.

Le SDK de Flight Simulator contient 65 fonctions et une large documentation incluant des fichiers de travaille (working samples) pour bien comprendre leurs fonctionnements. Ces applications peuvent être nouvelles ou une simple modification pour améliorer des missions, des terrains, des aéroports, des effets spéciaux, les jeux de camera, et plein d'autre éléments pour la simulation.

Le SDK est divisé en quatre principaux domaines :

1. Core Utilities Kit: Ce kit couvre, la commande des avions, les données instantanées, la configuration camera, et la modification des évènements imposés par le client.

2. Environment Kit: Ce kit couvre la création des terrains, des scènes, et des effets spéciaux.

3. Mission Creation Kit: Ce kit couvre la création des missions (aventure et défis)

4. SimObject Creation Kit: Ce kit couvre la création de panneau pour l'avion, ainsi que tous les autres objets de simulation pouvant apparaître (animaux, bateaux, trains … et).

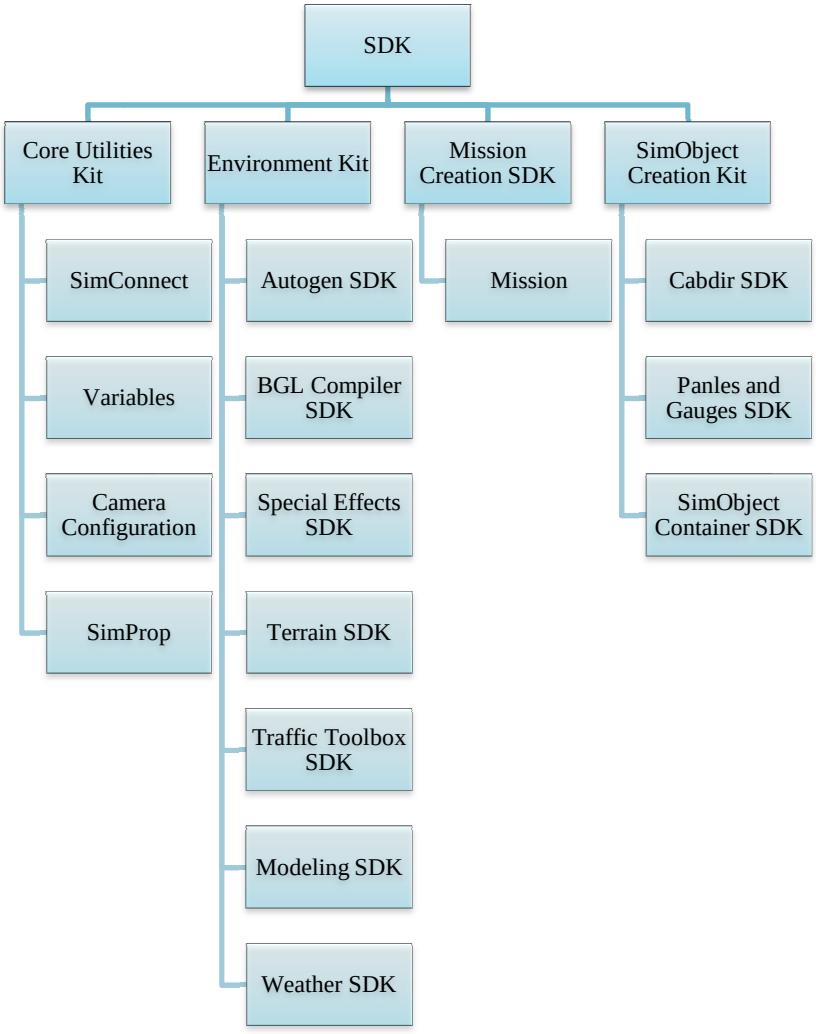

Figure II-1 : Composants SDK

Le développement d'une application propre à soi est possible, en utilisant les API propre à Flight Simulator.

II-4 Généralités sur les API

II-4-1 Définition de l'API

Une API (Application Programming Interface, traduite en langue française « 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-4-2 Principe de l'API et utilité

L'interface n'est pas un « décor » gérant l'aspect externe de l'application. Elle réalise une fonction d'adaptation bidirectionnelle entre l'usager et la partie fonctionnelle de l'application qui effectue les traitements, en d'autre termes, elle masque la complexité de l'accès à une application, en lui offrant un jeu d'instructions assez simple avec comme valeurs connus : les entrées et les sorties, par exemple : un téléspectateur n'a pas à connaitre le fonctionnement interne du téléviseur pour regarder ses chaines préférées, en gros un développeur n'a pas à s'inquiéter de la manière dont elle est implémentée pour pouvoir l'utiliser dans un programme.

Cette figure illustre le principe de fonctionnement d'une API

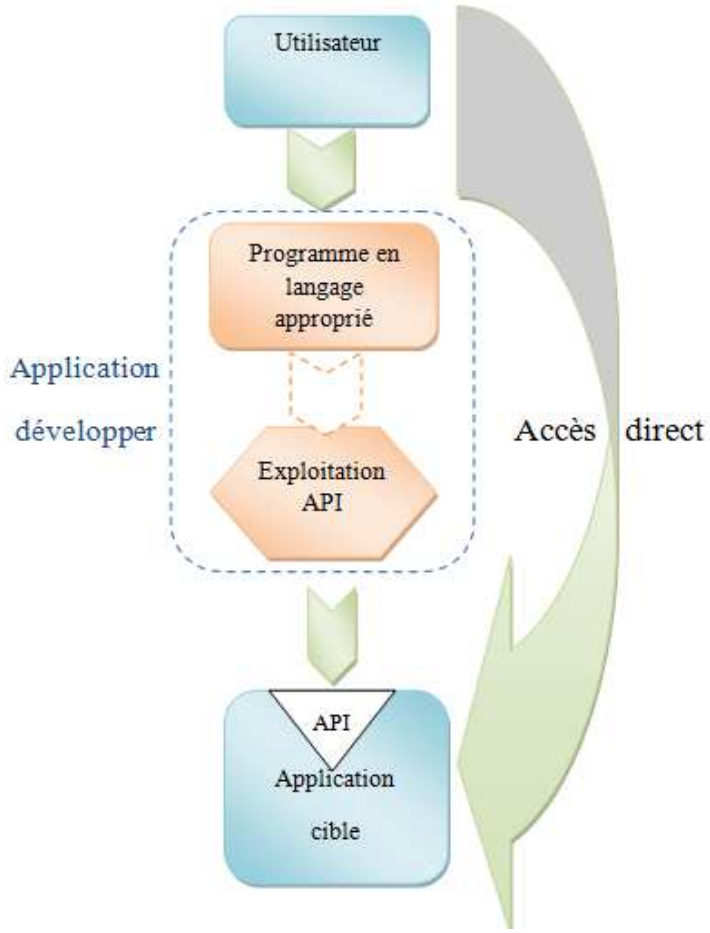

Figure II-2 : Principe de fonctionnement d'une API

Au finale, une API n'intervient que sur l'application propre à elle, de ce faite elle ne travaille pas seule, elle utilise toujours des fichiers DLL connus sous le nom « dynamique link library ».

II-4-3 Bibliothèques de liens dynamique

Une DLL est une bibliothèque contenant du code et des ressources nécessaires à d'autres applications, la raison pour laquelle on lui a attribué ce nom est le faite qu'elle soit appelée durant l'exécution des applications.

Les bibliothèques logicielles se distinguent des exécutables dans la mesure où elles ne représentent pas une application. Elles ne sont pas complètes, elles ne possèdent pas l'essentiel d'un programme comme une fonction principale et par conséquent ne peuvent pas être exécutées directement. Les bibliothèques peuvent regrouper des fonctions simples (par exemple le calcul d'un cosinus, ou l'inversion d'une matrice) comme des fonctions complexes

avec de nombreuses fonctions internes non accessibles directement. L'intérêt des bibliothèques réside dans le fait qu'elles contiennent du code utile que l'on ne désire pas avoir à réécrire à chaque fois.

Dans notre cas L'API Flight Simulator fait appel à la DLL propre à lui : SimConnect.dll, cette dernière interagie directement avec le logiciel Flight Simulator en récupérant les handles des API.

II-4-4 Handle

Cet élément représente l'immatriculation d'un objet informatique (e. g. une fenêtre ou un fichier) permettant sa « manipulation » ou sa gestion. Ce numéro est attribué par le système d'exploitation, pour différencier les processus actifs, ainsi en accédant au handle d'un objet, on peut modifier ses propriétés.

Dans notre cas, on a manipulé les objets à travers le handle, mais tout cela n'était pas réalisable sans les API Flight Simulator.

II-5 Les API Flight Simulator

Comme on l'a décrit auparavant, le SDK est composé de quatre principaux domaines pour faciliter son utilisation. Cependant, il faut souligner le fait que certaines taches telles que la création de nouveaux avions demande beaucoup d'implication ainsi qu'une grande connaissance sans oublier le facteur temps. Pour ces raisons, notre travail sera basé principalement sur la partie « Core Utilities Kit ».

II-5-1 Core Utilities Kit

Le Core Utilities Kit contient quatre composant : SimConnect SDK, Variables, Camera Configuration et le SimProp.

Le SimConnect SDK est une bibliothèque dédiée aux programmeurs permettant d'élargir Flight Simulator X, elle peut être utiliser par le programmeur pour écrire des add-on components (ajout de composant) pour le FlightSim. Ces composants peuvent être écrient en C, C++ ou voir même en C#.NET. Typiquement, les composants ajoutés peuvent réaliser un ou plusieurs éléments, qu'on peut citer :

- Remplacer le processus de traitement d'un ou plusieurs événements de Flight Simulator avec une nouvelle logique.
- Enregistrer ou surveiller un vol.
- Créer et configurer les plans de vol pour l'intelligence artificielle des avions.
- Configurer différents systèmes météorologiques
- Activer un nouveau matériel pour travailler avec Flight Simulator.
- Contrôle d'une caméra supplémentaire.

Comme son nom l'indique le Camera Configuration permet d'inclure des modifications sur les cameras utilisées, il a été complètement réécrit pour le Flight Simulator X pour donner une plus grande flexibilité et extensibilité.

II-5-2 Environment Kit

Durant le jeu, les développeurs peuvent étendre le monde réel à l'aide du kit Environment. Des textures de terrain et des bâtiments en 3D haute résolution peuvent être importés ou modifiés à l'aide des utilitaires fournis dans le SDK. Outre des objets statiques, des effets spéciaux dynamiques peuvent être ajoutés à un emplacement spécifique dans le monde réel, tels que des feux d'artifice, des fontaines ou des éclairs.

II-5-3 Kit Mission Creation

Ce kit permet la création de nouvelles missions, la difficulté est bien sur réglé par l'utilisateur. Un développeur peut créer des scénarios de missions ou de nouveau chalenge en les combinant avec l'API Core Utilities. Il peut aussi intégrer les expériences structurées dotées d'une prise en charge multiutilisateur avec un mode cockpit partagé, qui lui permettrai de prendre place « virtuellement » aux côtés d'un autre utilisateur dans le même avion, pour les cas des applications réseau; la communication serra établie par la VoIP (Voice over IP).

II-5-4 SimConnect Creation Kit

Ce kit offre la possibilité de personnaliser son avion, en modifiant ses attributs (nom, couleur, bruit, panneaux, mesures, et ainsi de suite), ainsi que la possibilité d'ajouter des objets (animaux, bateaux, trains … etc). Il permet aussi de customiser un tableau de bord, propre à soi, avec les indicateurs de notre choix (altimètre, anémomètre, horizon artificiel, indicateur de carburant …etc).

II-6 Utilisation des API Flight Simulator

L'accès aux API de Flight Simulator a été facilité par Microsoft SDK, en les séparant en plusieurs domaines bien spécifiques, l'utilisateur n'aura pas besoin de se perdre dans les applications inutiles, par exemple, pour le changement d'un traitement de Flight Simulator en une autre logique propre à lui, on n'aura pas besoins de chercher, on se dirige directement vers le SimConnect.

Apres la détermination du domaine de travail, une familiarisation avec les fonctions est nécessaire, on doit bien connaitre les paramètres des fonctions pour pouvoir les utiliser correctement par exemple : le handle, la valeur retourné, le nom et le type des variables … etc.

L'exploitation des API, tel que la construction d'un add-on doit se faire en utilisant le logiciel de Microsoft Visual Studio, ce dernier est un ensemble complet d'outils de développement, utilisant un environnement de développement intégré (IDE) incluant le Visual Basic, Visual C++, Visual C# . . . il permet ainsi de faciliter la création de solutions faisant appel à plusieurs langages, ce qui est le cas des API de Flight Simulator. L'utilisation des API Flight Simulator se fait principalement en utilisant le langage de programmation C++, chose qui est due essentiellement au développement de la bibliothèque SimConnect en ce langage.

II-7 Conclusion

Dans ce chapitre, nous avons présenté d'abords le logiciel Flight Simulator ainsi que son kit de développement SDK. Nous avons donné par la suite les notions fondamentales des API : définition, rôle et utilisations. Nous avons précisé par la suite les spécificités des API Flight Simulator, ce qui nous a permis de comprendre et d'intégrer leurs utilisations dans notre travail.

CHAPITRE III

Contribution à l'implémentation d'un simulateur de vol

III-1 Introduction

Notre travail consiste à faire un simulateur de vol en utilisant la technique du HIL déjà définie dans le premier chapitre, notre HIL est une carte FPGA d'Altera qui sera décrite par la suite, l'interaction avec notre simulateur se fera à travers cette carte. Comme support soft on utilisera le simulateur de Microsoft, Flight Simulator et ses API, chose déjà abordée dans le chapitre 2.

Dans ce qui suit, on traitera la liaison entre les deux chapitres précédents, on commencera par décrire la synthèse globale de notre application pour séparer ensuite le travail en deux parties : la partie HIL englobant la configuration de notre carte, la deuxième partie traitera l'interface entre le HIL et la boucle soft qui se trouve être le simulateur de Microsoft en utilisant ses API.

III-2 Structure générale de l'application

La figure qui suit donne la représentation de la structure générale de notre simulateur de vol.

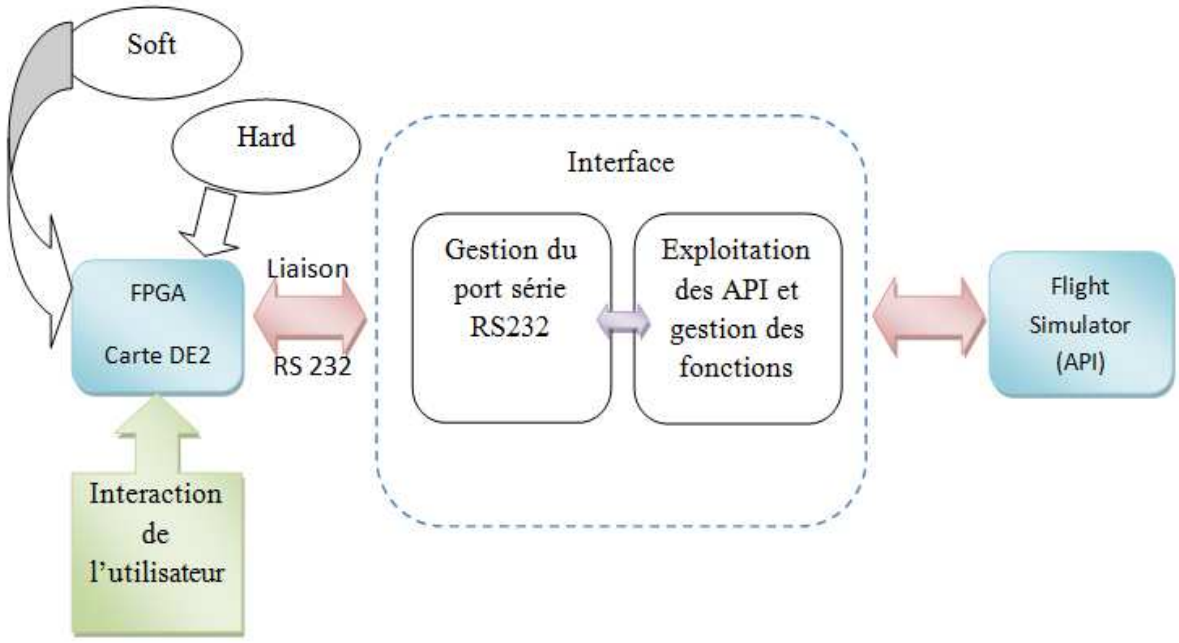

Figure III.1 : Structure générale de conception du simulateur de vol

la conception de notre simulateur de vol se présente sous deux parties principales comme le montre le diagramme au dessus. La première partie est la partie de l'intégration de notre HIL dans la boucle de simulation, pour cela on doit passer par la configuration hard de notre FPGA (déclaration des périphériques, leurs branchements . . .). La deuxième partie est le développement soft de notre interface entre le FPGA et le Flight Simulator.

III-3 Bloc HIL

III-3-1 Présentation de la carte de développement DE2

La carte DE2 fait partie du pack d'ALTERA DE2 Développent And Education Bord (carte de développement et d'éducation). Elle possède de nombreuses fonctionnalités qui permettent à l'utilisateur de mettre en œuvre un large éventail de circuits simples à divers projets multimédias [9].

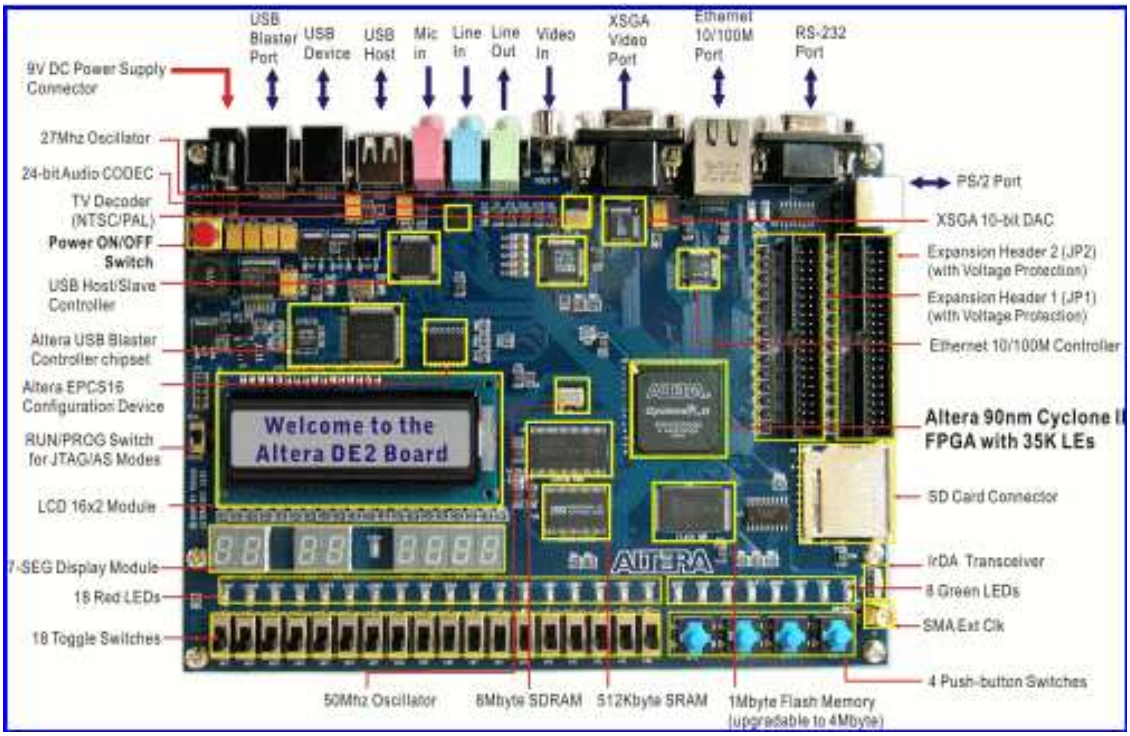

Figure III.2 : composants de développement et interfaces de la carte DE2

Les principales caractéristiques de la carte sont :

- **Cœur** :
  - FPGA : (Cyclone II EP2C35F672C6) : 35.000 cellules (LE) utilisables dans un boîtier de 672 broches (BGA) ; il s'agit du centre nerveux de la carte.
  - Flash de configuration : EPCS16 : cette mémoire permet de stocker de manière non-volatile la configuration du FPGA.
  - Horloges : Oscillateurs 27 et 50 MHz et entrée externe (SMA).
- **Mémoire** :
  - FLASH (S29AL032D).
  - SDRAM (IS42S16400-8).
  - SRAM (61LV25616).

En plus de ces caractéristiques hardwares, la carte DE2 a un support software pour les interfaces I/O standards ; elle possède aussi un panneau de contrôle pour faciliter l'accès a diverses composants. En outre, le software est fourni pour profiter d'un certain nombre de fonctionnalités avancées de la carte DE2. Pour utiliser la carte DE2 dans ce projet, l'utilisateur

doit être familier avec le logiciel Quartus II. Pour fournir un maximum de flexibilité pour l'utilisateur, toutes les connexions sont faites par le Cyclone II FPGA. Ainsi, l'utilisateur peut configurer le FPGA pour mettre en œuvre n'importe quel design pour son système. La figure qui suit représente le schéma block de la carte DE2.

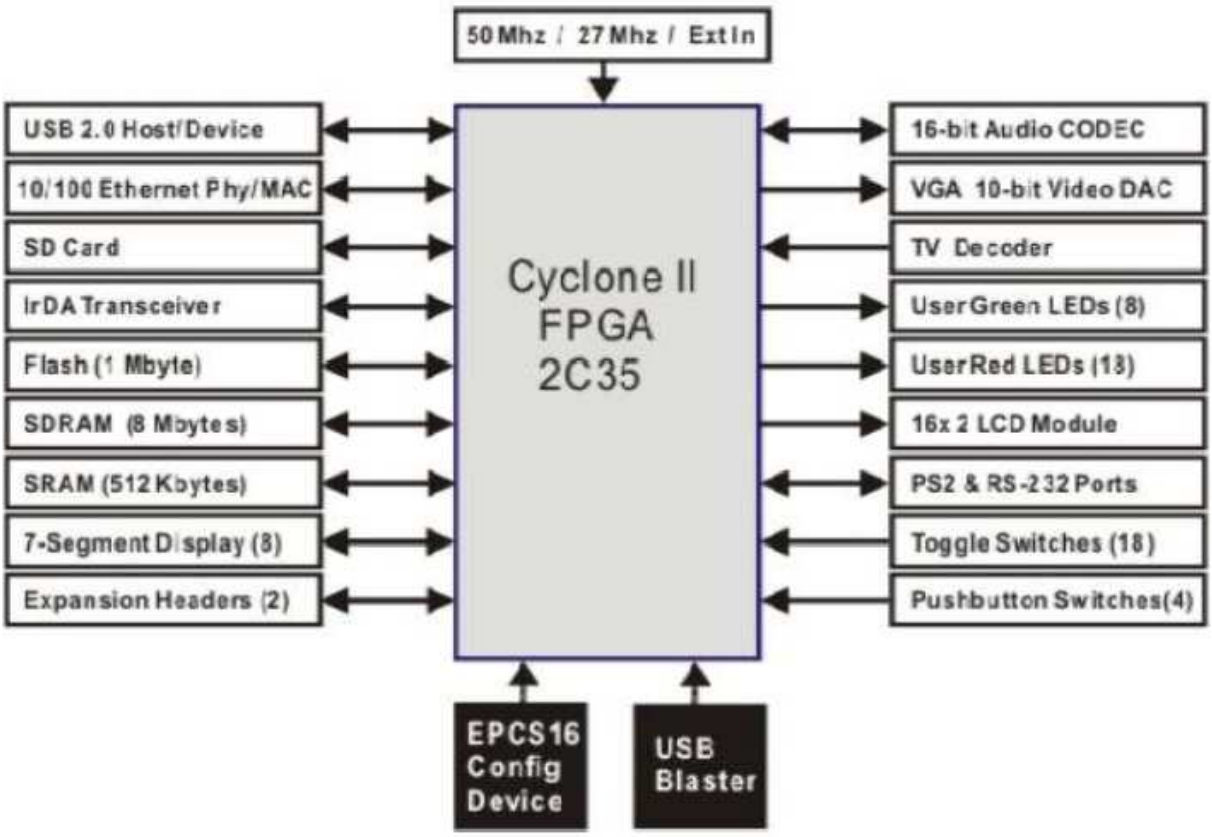

Figure III.1 : Schéma bloc de la carte DE2

III-3-1-1 Cyclone II EP2C35 dispositif logique programmable :

La famille du dispositif Cyclone II est la deuxième génération de la famille de la série à faible coût des Altera Cyclone [9]. Il a été conçu selon le dispositif à immense succès de la première génération la famille Cyclone ™, Altera Cyclone II FPGA fait étendre la densité de la plage des FPGA à faible coût à 68.416 éléments logiques (LEs : logic elements), il fournit jusqu'à 622 broches I/O utilisables mais aussi jusqu'à 1,1 Mbits de mémoire embarquée. En réduisant la surface de silicium, les circuits Cyclone II peuvent supporter des systèmes

numériques complexes sur une seule puce d'un seul cout, ce qui rivalise avec les circuits des ASIC.

Les FPGA Cyclone II offrent des performances supérieures de 60 pour cent et la moitié de la puissance de consommation des composants FPGA concurrents. Le faible coût et les fonctionnalités optimisées font des FPGA Cyclone II le choix idéal dans plusieurs domaines vastes et variés tels que l'automobile, les communications, le traitement vidéo, de test et mesure . . . etc. [13]

Les caractéristiques d'EP2C35 Cyclone II sont indiquées dans le tableau suivant :

Caractéristique	EP2C35
Eléments logiques (LEs)	33216
Blocs de RAM M4K (4Kbits plus 512 bits de parité)	105
Bits totale de la RAM	483840
Multiplicateurs embarqués	35
Phase locked loop « PLL » (boucle à verrouillage de phase)	4
Maximum de broches I/O utilisables	475

Tableau 1 : Caractéristiques du Cyclone EP2C35

Tous les FPGA Cyclone II sont reconfigurables. Apres avoir configuré le Cyclone II, il peut être reconfiguré en un autre circuit en réinitialisant l'appareil et puis en effectuant le chargement de nouvelles données.

III-3-2 Flot de conception

III-3-2-1 Environnement Quartus II

Le Quartus II est un logiciel de la société d'Altera, son utilisation se fait sous forme de projet. Il permet la simulation, la synthèse et la programmation des circuits logiques programmables [10].

La saisie du processus à synthétiser, ou autrement dit la fonction logique que l'on veut implanter dans le circuit peut se faire de deux façons différentes :

- Saisie sous forme d'un schéma à l'aide d'un éditeur graphique en assemblant des symboles.
- Saisie sous forme d'un ficher texte à l'aide de n'importe quel éditeur de texte (en langage VHDL, Verilog . . .).

III-3-2-2 Notion de SOC (SoPC) et IP

Les Systems On Chip sont des blocs fonctionnels intégrés dans des éléments électroniques ayant au moins un processeur comme élément de traitement. Les blocs de traitement particulièrement coûteux sont réalisés en tant que matériel, cependant, l'application est implémentée en logiciel et exécutée par le cœur du processeur.

Le SOC a été appliqué au développement des ASIC, il fournit de grande performance en se qui concerne la consommation, la vitesse et la surface, mais une lacune lui est associée celle du fait qu'il soit figé donc non réutilisable. La technologie des SOC a été prolongée au FPGA on parle alors de SoPC (System On Programmable Chip), se sont des composants reconfigurables à volonté permettant un développement et un prototypage rapide du système [14].

La grande avancée dans les SOC/SoPC a engendré une complexité dans leur développement, chose qui a conduit à la création d'un nouveau concept au service de l'utilisateur, des composants utilisables déjà élaborés ce sont les blocs IP.

III-3-2-3 SOPC Builder

Le SOPC Builder est une partie intégrée du Quartus II, elle se présente sous forme d'une interface graphique permettant à l'utilisateur de définir les différents éléments constituants le système souhaité. Mais aussi, il permet de faire le paramétrage des différentes caractéristiques des périphériques et de créer les interfaces de ceux-ci avec le processeur Nios. C'est le SOPC Builder qui gère la génération des éléments nécessaires au développement de la partie matérielle et logicielle.

III-3-2-4 Nios II et ses périphériques

Le Nios II est un processeur 32 bits, il est la deuxième génération de processeurs embarqués à cœur logiciel d'Altea, il a la plus grande flexibilité, les meilleurs performances 30 à 80 MIPS (Million Instructions Per Second) et le plus faible coût, les développeurs peuvent choisir un nombre infini de configurations système afin de répondre a leurs exigences de performances et de coût sans avoir recours à un ASIC [11] [12].

La figure qui suit représente le schéma synoptique d'un système de processeur NiosII.

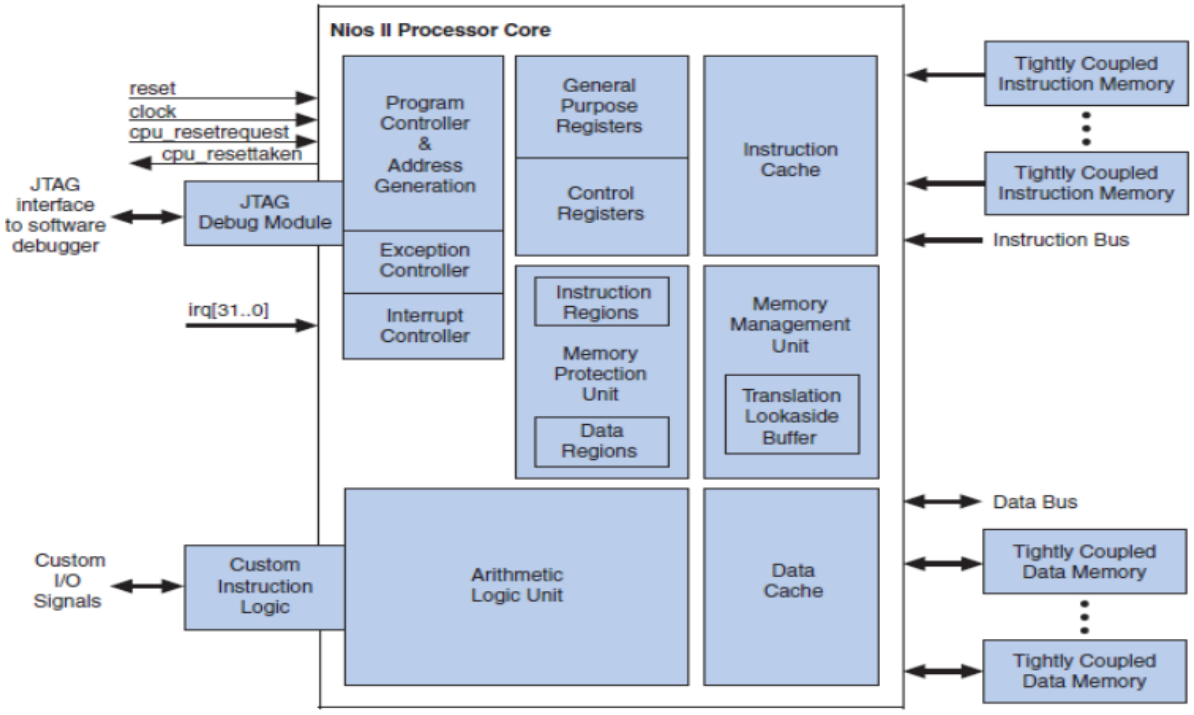

Figure III.2 : Architecture du processeur Nios II

L'architecture Nios II est une architecture **RISC** soft-core qui est mise en œuvre entièrement dans une FPGA. La nature embarquée de ce type de processeur permet au concepteur de spécifier et générer un processeur sur mesure pour des besoins d'applications spécifiques. Les concepteurs peuvent étendre les fonctionnalités de base en y ajoutant une unité de gestion de mémoire prédéfinie, définir des instructions personnalisées ou des périphériques personnalisés.

Il existe trois modes utilisables pour le processeurs embarqués Nios II :

- Economy (/e core): utilise moins de surface de silicium du composant FPGA 540 LEs (optimiser pour la taille).
- Standard (/s core): donne une bonne combinaison entre la surface et la rapidité (équilibre entre la vitesse et la taille). 1030 LEs occupés.
- Fast (/f core): c'est la version la plus rapide des trois processeurs au détriment de la mémoire, 1600 LEs occupés (optimiser pour la vitesse).

III-3-2-5 Flot de conception d'un système SoPC

La conception de notre projet passe par la configuration hard de la carte, puis par la programmation soft. La réalisation se fait suivant le flot de conception [13]:

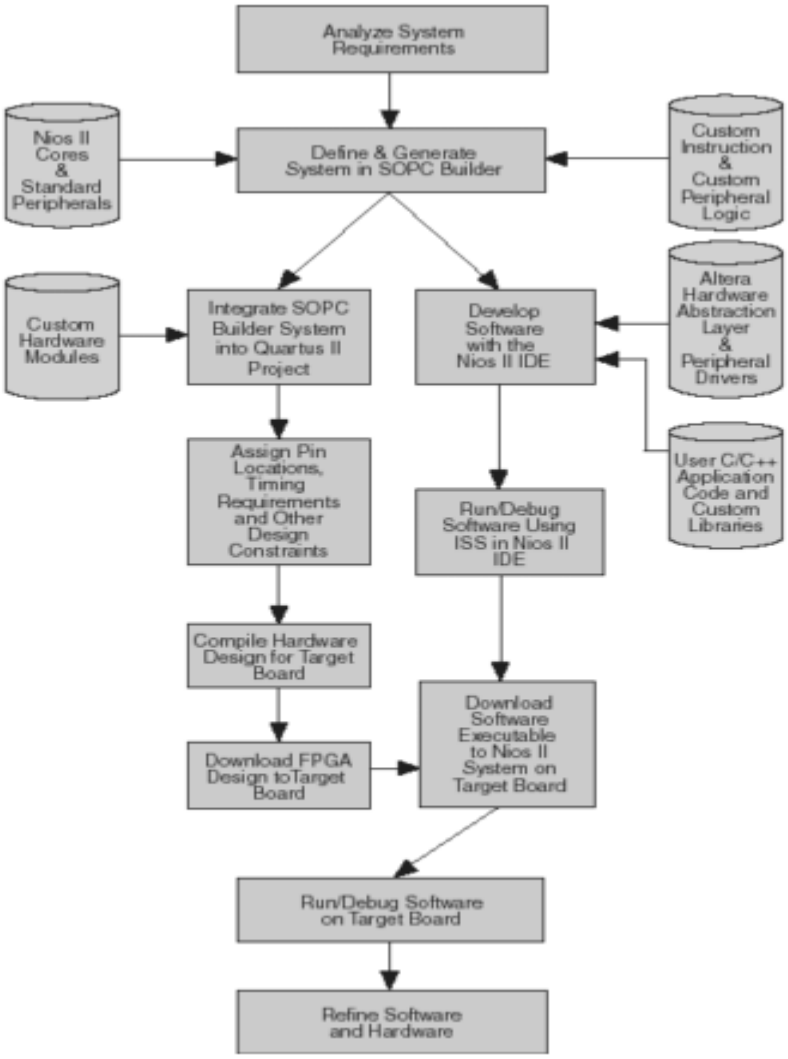

Figure III.3 : Flot de conception d'un système Nios II et son implémentation sur la carte

Le développement peut être défini en deux parties bien distinctes, le développement sous Quartus II et ses outils intégrés pour le processus de génération matériel (hard) , la deuxième partie étant le processus de développement logiciel (soft) et cela en utilisant l'IDE un logiciel du kit EDS.

III-3-3 Implémentation

III-3-3-1 Description des périphérique utilisés

La première étape de l'implémentation est l'intégration de notre système SOPC Builder dans le logiciel de développement Quartus II. Notre choix c'est porté sur le mode bloc

diagramme, tel que le cœur de notre système embarqué Nios est représenté par un seul et même bloc, comportant tout les périphériques standards et personnalisés déclarés au part avant dans le SOPC Builder. On peut classer ces composants selon deux aspects : on chip (propre au processeur) et off chip (composants faisant parties des éléments physiques de la carte).

Composants propres au processeurs :

- CPU : Nios II 32 bits.
- SysID : assistant pour le débogage.
- On chip memory : mémoire interne.

Composants physiques de la carte :

- JTAG _UART : pour l'implémentation du hard et du programme sur la carte.
- LCD : afficheur LCD, utiliser comme écran de sortie.
- LEDs : utilisées comme voyant lors de la commande de l'avion (tel que freinage, accélération, tourner à gauche ou à droite . . .)
- UART : pour la communication.
- SDRAM : utilisée comme mémoire externe (off chip memory), vu que la mémoire interne n'a pas une taille suffisante pour contenir notre programme.
- Switchs : des switchs pour l'envoi de commande à la carte.

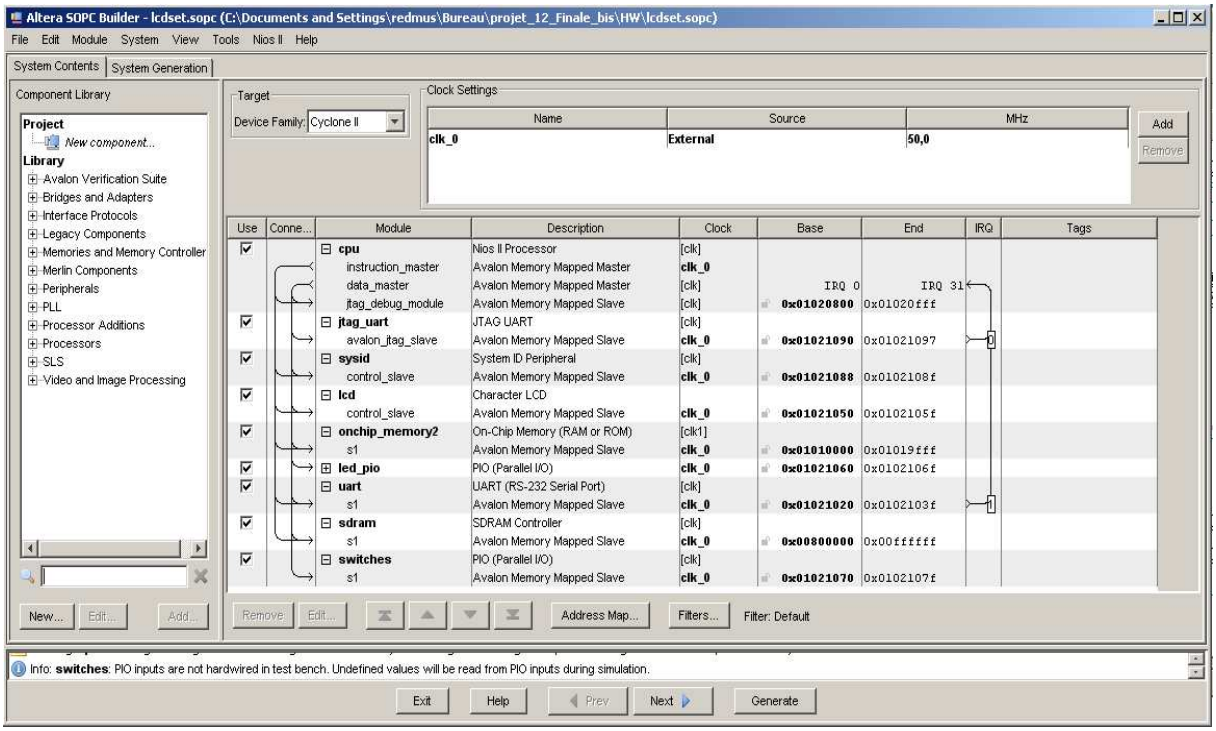

Figure III.4 : disposition des périphériques du bloc Nios sous SOPC Builder

Dans la conception de notre système on s'est heurté à un problème de mémoire, notre programme ayant une taille importante n'a pas pu être intégré dans la mémoire interne (on chip memory), chose qui nous a amené à faire appel à une mémoire externe, une SDRAM. Cette solution, nous a conduit vers une autre impasse, car la SDRAM a besoin d'une horloge bien spécifique à elle-même (différente de celle du bloc Nios), pour ce faire un autre bloc doit être ajouté pour que notre système soit opérationnel, c'est le bloc PLL. Ce bloc est généré par un outil de Quartus II appelé « MegaWizard Plug_in Manager », il permet d'agir sur l'horloge, on peut avoir plusieurs horloges à la sortie du bloc ayant plusieurs fréquences de travail et cela grâce à un facteur de multiplication et de division. Dans notre cas, on utilisera ce bloc pour faire avancer l'horloge du système de 3 ns (nanoseconde) pour le bon fonctionnement de la SDRAM qui aura de ce fait une horloge indépendant de celle du module Nios [6].

III-3-3-2 Implémentation de l'architecture du système hard

La déclaration des périphériques du Nios et leurs intégration dans le Quartus II est suivie par l'assignement des broches d'entrées / sorties du système, chaque broche de notre système doit être assimilée à son assignement prédéfini sur la carte DE2 [7].

Après La génération du système sur SOPC Builder, son intégration dans le Quartus II avec un assignement correct, on obtient la schématique juste au dessous [10]. L'étape suivante avant l'implémentation du hard sur la carte DE2 est la compilation de notre projet sur Quartus. Après une compilation réussie, on implémente notre système sur la carte cible, pour cela on utilise un autre outil du Quartus appelé « Programmer », l'implémentation se fait par le chargement d'un fichier *.SOF propre au projet généré lors de sa compilation, le chargement se fait en mode JTAG en utilisant le câble de liaison USB Blaster.

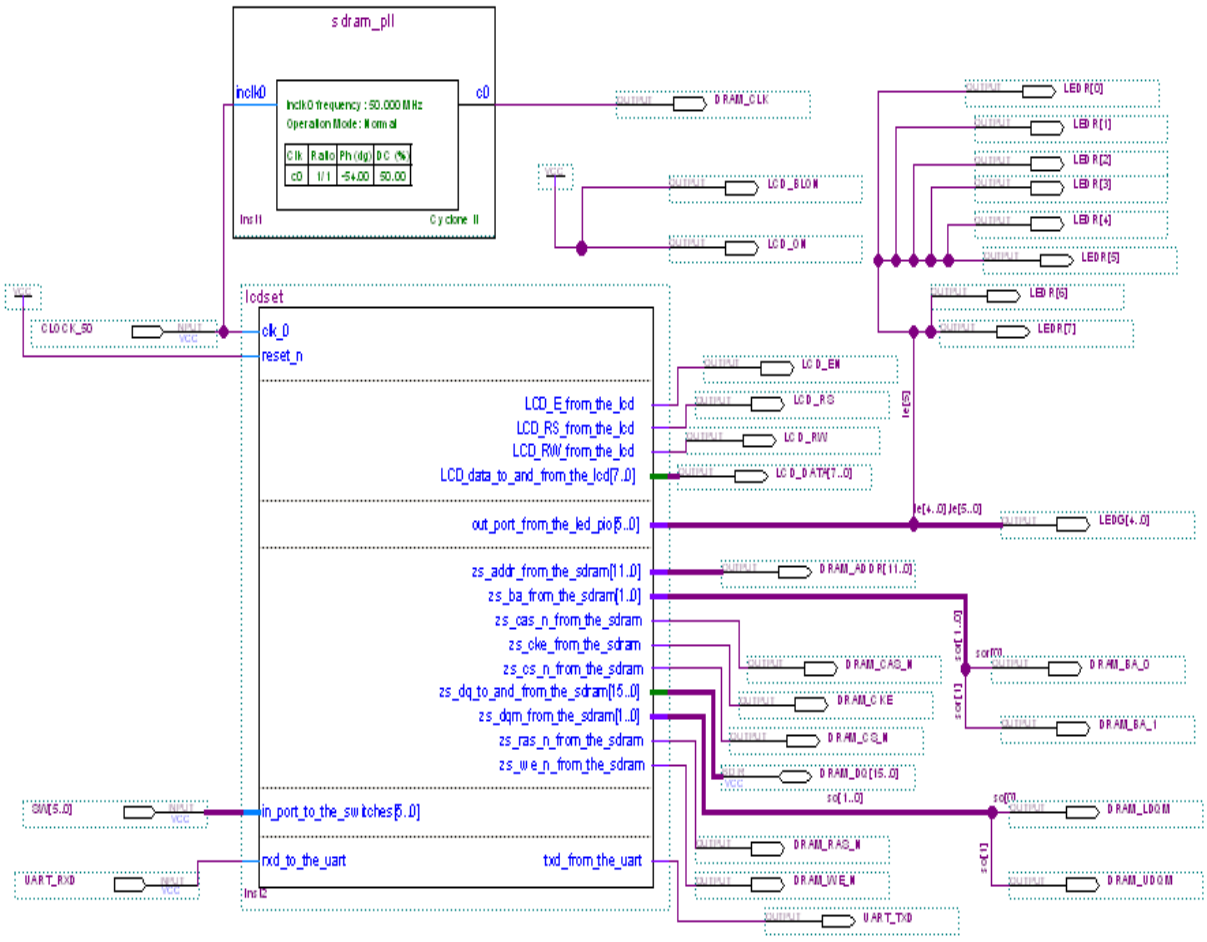

Figure III.5 : Schéma graphique du système conçu

III-3-4 Architecture applicative

Dans cette partie on gère le rôle des composants décrits dans le SOPC Builder et leurs manières de fonctionner, pour ce faire on utilisera le logiciel Nios II IDE « Integrated Development Environment » [15].

Après avoir lancé IDE, on ouvre un nouveau projet en choisissant dans la liste pour les nouveaux projets « Nios II C / C++ Application »,une fenêtre s'affichera dans laquelle on doit spécifier le nom de notre projet, l'emplacement de notre projet ainsi que le fichier .PTF généré par le SOPC Builder propre à ce dernier, l'ouverture d'un nouveau projet est accompagnée par la création d'une bibliothèque système portant le nom suivant « nom_du_projet_syslib[nom du ficher PTF] ». Avant de commencer notre programme, on passe à la définition des ses propriétés en cliquant sur le bouton droit sur le projet puis on choisit ' System Library Properties ' pour choisir l'emplacement de notre programme, dans notre cas on place le programme dans la SDRAM d'espace de 8 Méga pour pouvoir englober la taille importante de notre projet, on spécifie aussi les périphériques stdin, stdout et stderr, on leurs attribue le JTAG_UART. La connexion JTAG avec la carte cible doit être elle aussi précisée, dans notre cas la connexion c'est l'USB Blaster, on choisit JTAG_UART comme dispositif de communication avec le terminal Nios II.

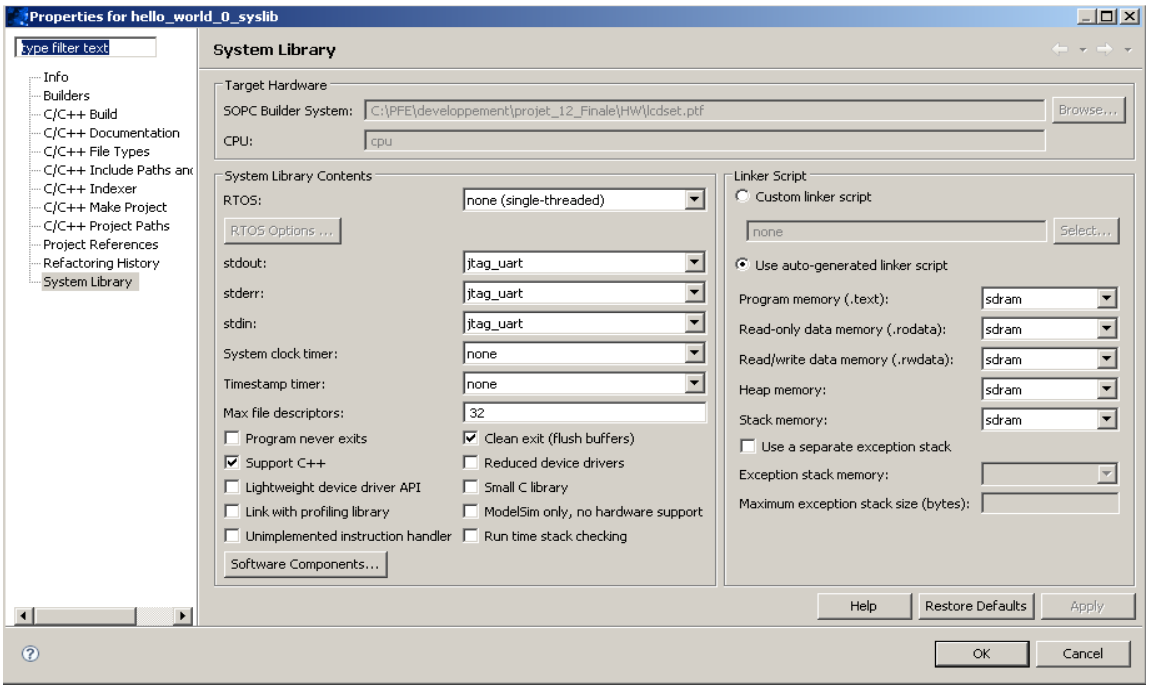

Figure III.6 : interface graphique pour les propriétés du système librairie

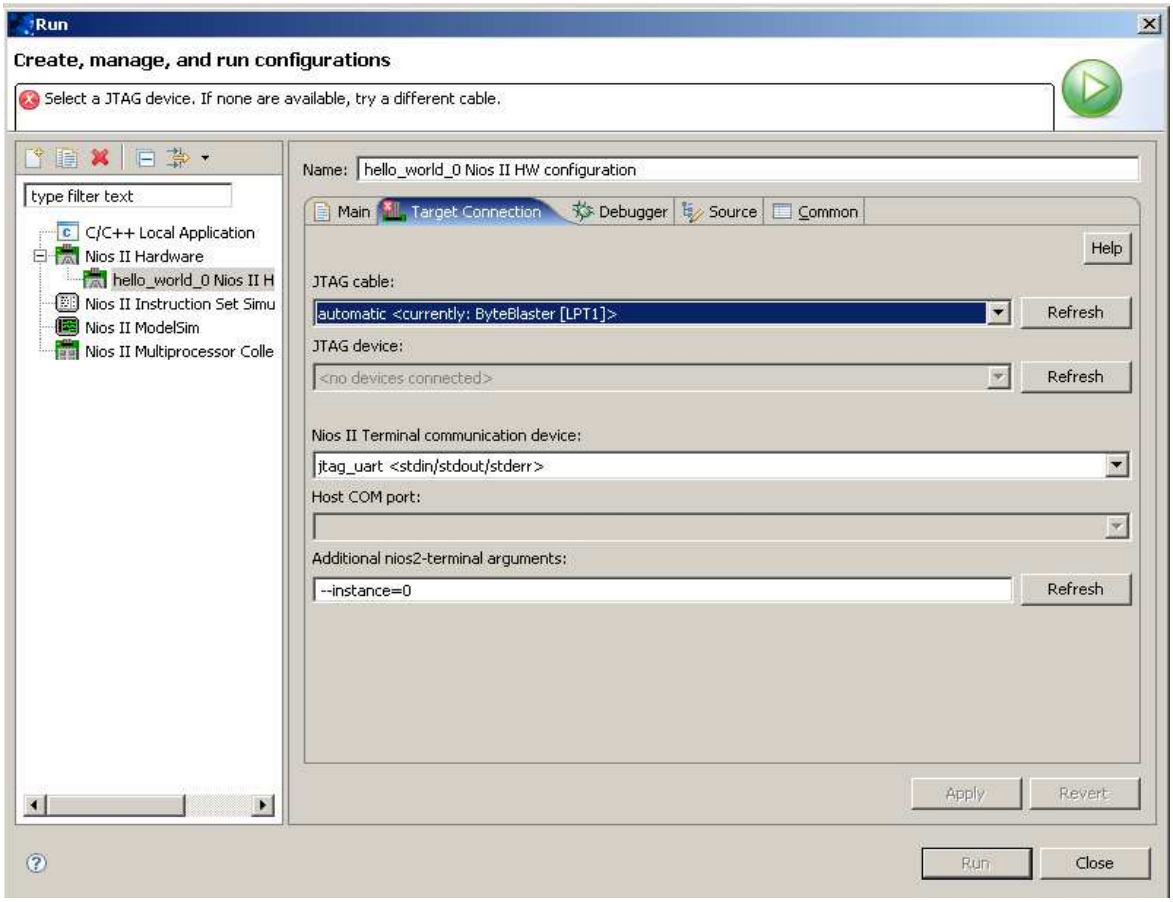

Figure III.7 : Spécification des connexions

Après avoir terminé le choix des propriétés, on passe à la programmation du fonctionnement de nos composants dans le compilateur C / C++, on construit notre programme avant de le charger sur la carte grâce à l'option « Build Project », dans cette étape le compilateur nous informera des erreurs existantes dans le script. Si le programme ne contient pas d'erreurs, on le charge et on exécute directement l'application compilée sur la carte, pour cela on a deux mode de travail :

1. Le mode débogue : pour suivre l'exécution étape par étape.
2. Le mode run : exécute l'application en entier.

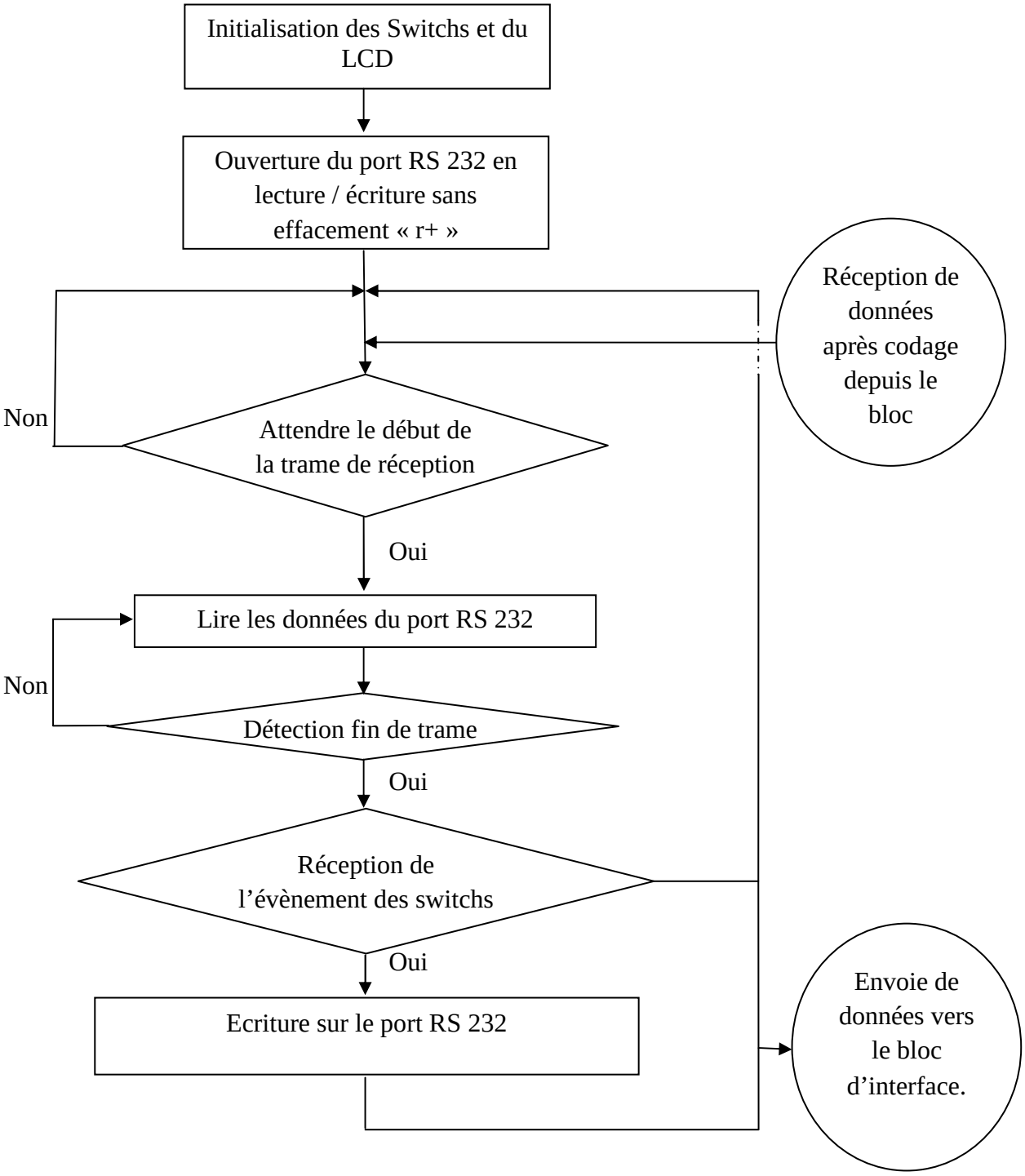

Figure III.10 : Organigramme de l'application Soft du SOPC

L'organigramme au dessus représente la structure globale de notre programme gérant les fonctionnalités de notre SOPC.

III-4 Bloc d'interface PC simulateur

Le bloc d'interface contient lui aussi à son tour deux partie, la partie la plus importante est celle de l'exploitation des API de Flight Simulator ainsi que la gestion de ses fonctions. L'autre partie concerne la gestion du port de transmission série RS232. L'exploitation des API ce fait par le billet du Microsoft Visual Studio utilisant le langage C++ selon l'organigramme représenté dans la figure III.11.

Les données de commandes envoyées par la carte (depuis les switchs) sont codées en forme de lettre majuscule, et peuvent s'identifier avec les différents événements nécessaires au vol, suivant le tableau 1.

Switch	Donnée envoyée vers le programme Flight Sim	Evénements de Flight Sim	Description
S0	A	AILERONS_RIGHT	Virer à droite.
S1	B	AILERONS_LEFT	Virer à gauche.
S2	C	INCREASE_THROTTLE	Gain en altitude.
S3	D	DECREASE_THROTTLE	Perte en altitude.
S4	G	GEAR_TOGGLE	Sortie-rentrer des roues.
S5	F	Brakes	Freinage des moteurs.

Tableau 1 : Identification des données envoyées par la carte, avec les événements Flight Simulator.

Tous les événements de Flight Sim cités dans la colonne 3, ont été tirés depuis le fichier Standard.xml [17]. Ce dernier représente tous les événements et leur correspondance avec les touches du clavier.

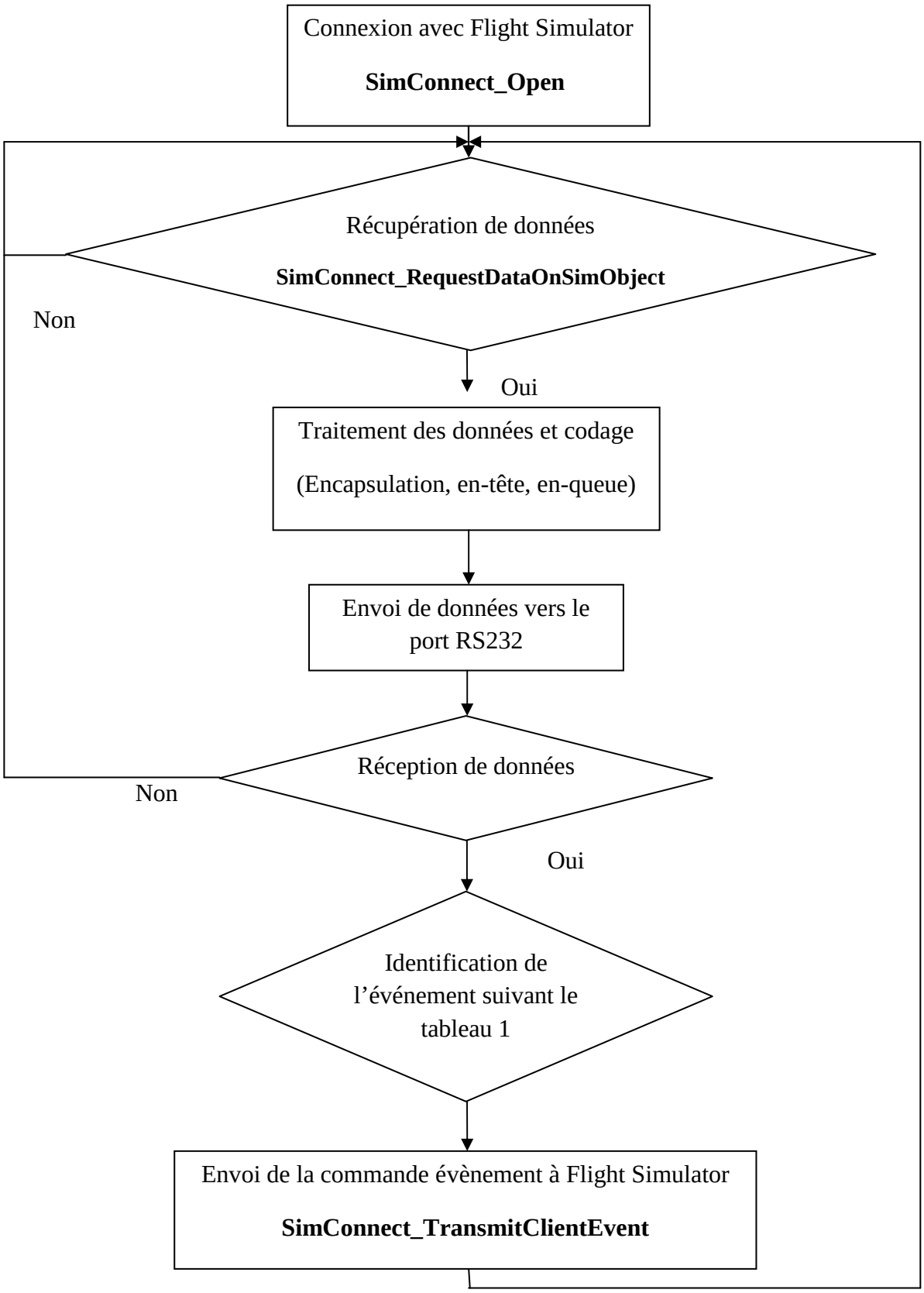

Figure III.11 : Organigramme de la gestion Soft du Simulateur de vol

La gestion du port RS232 ce fait avec un programme en langage C, on commence part choisir le port approprié pour notre carte, la gestion des processus d'envoi et de réception vient juste après la reconnaissance du port. Comme le port RS232 est un port série, on est limité par la contrainte d'effectuer une opération à la fois, le port commence part l'envoi des données obtenues du Flight Simulator vers la carte pour l'affichage, juste après, le port se met en mode réception, il reçoit les données de commande (événements) qui vont permettre à l'utilisateur d'interagir avec le simulateur de vol via la carte DE2 (le HIL) comme le montre l'organigramme au dessus.

III-4-1 Codage des données

Les données à envoyer vers la carte sont obtenues lors de l'appel à la fonction **SimConnect_RequestDataOnSimObject,** cette dernière renvoie :

- L'altitude (en pied).

- La longitude (en degré).

- La latitude (en degré).

La première donnée est l'élévation verticale d'un lieu ou d'un objet par rapport à un niveau de base (peut être le sol, ou le niveau de la mère, dans notre cas c'est le second), quant aux deux dernières sont des coordonnées géographiques représentées par une valeur angulaire, la première exprime la position d'un point sur Terre (ou sur une autre planète), au nord ou au sud de l'équateur qui est le plan de référence, la seconde quant à elle exprime la position à l'est ou à l'ouest du méridien de Greenwich[18].

Si la terre était considérée comme une sphère parfaite, on pourra dans ce cas attribuer à chaque point de la planète, une situation géographique et cela suivant la figure suivante :

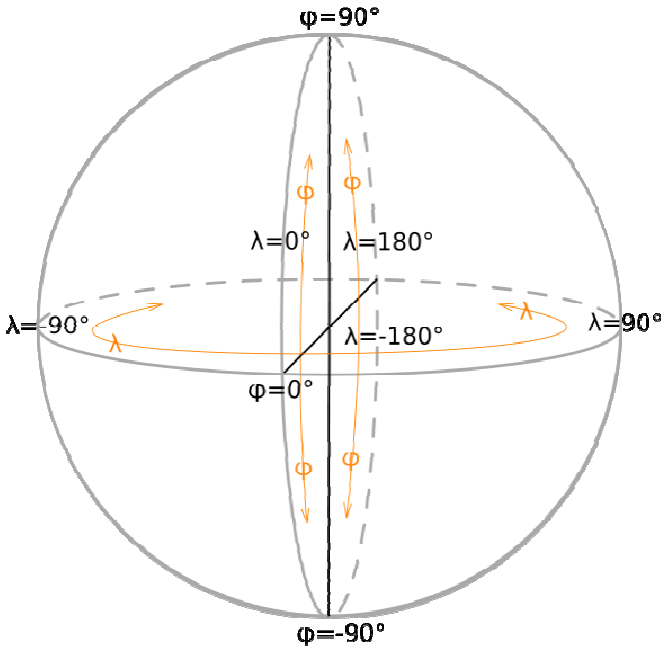

Figure III.12 : Coordonnées géographiques sur un globe.

Sur cette figure, La latitude correspond à la mesure de l'angle marqué *phi* (φ), la mesure de l'angle marqué *lambda* (λ) par rapport au méridien de référence donne la longitude

Le traitement de données intervient lors du calcul de la vitesse en utilisant la formule d'Haversine, cette dernière est très importante dans le domaine de l'aviation car elle permet de calculer la distance orthodromique séparant deux point se trouvant sur une sphère à partir de leur longitude et latitude, cette méthode est définie comme suit [19] :

Considérons un repère orthonormé Oxyz. Soient θ et φ les coordonnées angulaires d'un point P sur la sphère (voir la figure III.13), θ étant l'angle entre l'axe Oz et le rayon OP (c'est la colatitude de P : sa latitude est $\lambda=\pi/2 - \theta$) et φ l'angle azimutal, c'est-à-dire l'angle polaire que fait la projection de OP sur le plan xOy avec l'axe Ox [20].

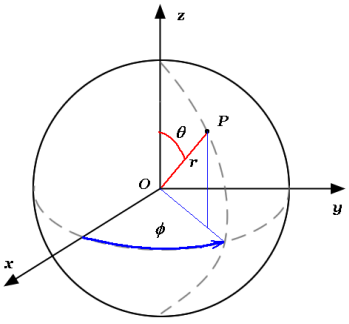

Figure III.13 : Repérage d'un point en coordonnées sphériques

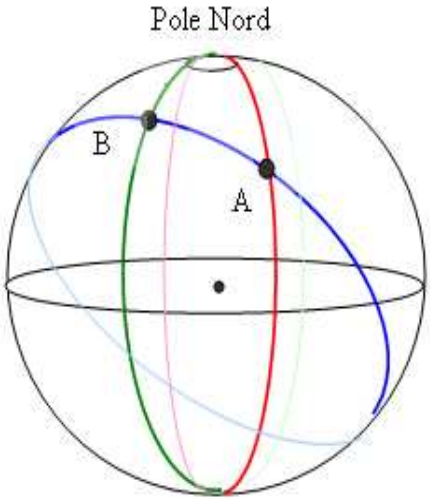

Figure III.14 : Représentation des points A et B sur le globe terrestre.

En prenant les points A et B de *lat1* et *lat2* (λ1 et λ2), et sachant que R représente le rayon terrestre (soit 6 378 kilomètres, rayon équatorial moyen) nous obtiendrons la formule finale donnant la distance orthodromique entre ces deux point avec φ comme étant la différence des deux longitudes (φ=φ2-φ1) :

$$arc(AB) = R\ \arccos[\cos\lambda 1\ \cos\lambda 2\ \cos(\varphi) - \sin\lambda 1\ \sin\lambda 2\ ]$$

Une fois la distance instantanée obtenue, on la multiplie par le temps d'une boucle, ayant ainsi la vitesse instantanée.

Dès que nos données à envoyer sont prêtes « altitude et vitesse », le codage peut commencer comme suit :

- Envoie comme début de trame la lettre « a ».

- Envoie de toute cette donnée « alt:valeur_de_altitude ».

- Envoie de fin de la première partie qui est représenté par la lettre « v ».

- Envoie de la seconde partie des données « spe:valeur_de_la_vitesse ».

- Envoie de fin de trame « f ».

Exemple :

aalt:valeur_de_altitudevspe:valeur_de_la_vitessef

Une fois que toute la trame soit envoyée, le décodage se fait comme suit :

- Vérification du début de trame.

- Affichage de la première partie 'altitude' sur la première ligne de l'afficheur LCD.

- Vérification de la fin de la première partie et ainsi le début de la deuxième.

- Déplacement du curseur vers la deuxième ligne de l'afficheur LCD.

- Affichage de la seconde partie des données 'vitesse' sur la seconde ligne de l'afficheur LCD.

- Détection de fin de trame.

- Retour du curseur vers la position initiale.

Grace à cette méthode, on peut dire qu'on a réalisé une communication bidirectionnelle reliant la carte FPGA DE2 à Flight Simulator, via le port RS232.

III-5 Conclusion

Dans ce chapitre, on a montré comment on a pu configurer notre carte FPGA en décrivant les composants hard nécessaires pour notre application ainsi qu'on gérant leurs rôles en programmant leurs soft. Aussi on a pu exploiter les API et les fonctions du Flight Simulator pour pouvoir interagir avec la carte FPGA. L'utilisation du port RS232 nous a permit de réaliser une liaison bidirectionnelle entre la carte et le simulateur.

Conclusion générale

Nous avons abordé dans ce projet la réalisation d'une commande électronique, sous forme d'un SoPC. Cette commande s'est faite à l'aide d'une carte FPGA DE2 Board Cyclone II EP2C35F672 qui joue le rôle d'un mini cockpit en l'interfaçant au logiciel : Microsoft Flight Simulator X. Pour ce faire, l'étude préalable des instruments de bord, est nécessaire.

Une application Windows a été développée assurant l'interfaçage entre la carte électronique et la commande de l'avion via le port RS232. Cette application est basée essentiellement sur l'API Flight Simulator.

La première contrainte était le temps alloué au projet qui a limité notre travail à la réalisation d'une commande pour une architecture SoPC à un seul processeur. Le fonctionnement en temps réel des commandes du vol était assuré par son processeur (NIOS II) fourni avec l'outil de développement Nios II IDE d'Altera.

La deuxième contrainte était le temps de réponse relativement long de l'afficheur LCD lors de la réception des données depuis l'hôte. Ce retard est causé au moment de l'effacement du contenu du LCD qui fait appel aux différentes fonctions nécessitant un temps de pause relativement important.

L'extension de ce travail est possible sur plusieurs plans :

- L'ajout d'un joystick pour faciliter et approcher le réalisme de la commande d'avion.

- Développement d'une architecture MPSoC pour avoir un affichage en temps réel, et une indépendance entre les différents périphériques.

- L'intégration de la carte au bord d'un vrai avion pour avoir un affichage et une commande plus réalistes.

Reference :

[1] http://fr.wikipedia.org/wiki/Simulateur_de_vol

[2] http://www.abmecatronique.com/la-simulation-hardware-in-the-loop-hil_540611/

[3] http://www.microsoft.com/france/jeux/flightsimulatorx/

[4] http://fr.wikipedia.org/wiki/Flight_Simulator

[5] C:\Program Files (x86)\Microsoft Games\Microsoft Flight Simulator X SDK\SDK\sdk overview.html

[6] Altera Corporation (2008). Using the SDRAM Memory on Altera'sDE2Board with VHDL Design

[7] Advanced Electronics Lab, Course Book 2010-2011, Mr. Araz Sabir Ameen, University of Sulaimani-College of Engineering-Electrical Engineering Department, Iraq.

[8]DE2 Programming using Quartus II, Eric Wheeler, Alex Edelsburg, Andrew Waterman, Duke University, August 4, 2010.

[9]ALTERA Corporation (2008). DE2 Development and Education Board User Manual.

[10]Altera Corporation (2008). QuartusII Introduction Using Schematic Design.

[11]Altera Corporation (February 2011). Nios II Software Developer's Handbook

[12]Altera Corporation (December 2009) NiosII Hardware Development Tutorial

[13]CHEN KAH YEE(2009), MEDIAN FILTER USING NIOS II PROCESSOR WITH SORT HARDWARE ACCELERATOR, UNIVERSITI TEKNOLOGI MALAYSIA

[14]Ahmed BEN ATITALLAH (2007), Etude et Implantation d'Algorithmes de Compression d'Images dans un Environnement Mixte Matériel et Logiciel, L'UNIVERSITE BORDEAUX 1.

[15]Altera Corporation (March 2009). Nios II IDE Help System

[16]Fahmi GHOZZI (2003), OPTIMISATION D'UNE BIBLIOTHEQUE DE MODULES MATERIELS DE TRAITEMENT D'IMAGES. CONCEPTION ET TEST VHDL, IMPLEMENTATION SOUS FORME FPGA, L'UNIVERSITÉ BORDEAUX I.

[17] C:\Users\"Nom de votre session"\AppData\Roaming\Microsoft\FSX\Controls

[18] http://fr.wikipedia.org/wiki/Latitude

[19] http://fr.wikipedia.org/wiki/Orthodromie

[20] http://fr.wikipedia.org/wiki/Coordonn%C3%A9es_g%C3%A9ographiques

ANNEXE

I Référence des API SimConnect :

I-1 Les fonctions :

Le tableau suivant liste toutes les fonctions nécessaires à la génération de l'application client :

I-1-1 Fonctions générales :

Fonctions	Description
DispatchProc	Ecrite par le développeur, comme une fonction d'appel pour gérer la communication avec le serveur.
SimConnect_CallDispatch	Utilisée pour traiter le prochain message de SimConnect reçut par la fonction de rappel.
SimConnect_ClearClientDataDefinition	Utilisée pour effacer la définition des données spécifiques de client.
SimConnect_ClearDataDefinition	Utilisée pour enlever toutes les variables de simulation d'un client.
SimConnect_Close	Utilisée pour demander l'arrêt de la communication avec le serveur.
SimConnect_GetLastSentPacketID	Renvoie l'identificateur ID du dernier paquet envoyé au serveur SimConnect.
SimConnect_GetNextDispatch	Utilisée pour traiter le prochain message de SimConnect reçut, sans utilisation d'une fonction de rappel de service
SimConnect_MapClientDataNameToID	Utilisée pour associer un identificateur ID à un nom de zone.
SimConnect_MapInputEventToClientEvent	Utilisée pour relier des événements d'entrée (tels que des mouvements de frappes, de joystick ou de souris) à des événements spécifiques appropriés.
SimConnect_Open	Utilisée pour envoyer une demande au serveur *Flight Simulator* d'ouverture des communications avec un nouveau client.
SimConnect_RequestClientData	Utilisée pour envoyer les données demandées par le client.
SimConnect_RequestDataOnSimObject	Utilisée pour spécifier quand le client de SimConnect doit recevoir des valeurs de données pour un objet spécifique.

I-1-2 Fonctions d'aide :

Fonctions	Description
SimConnect_InsertString	Utilisée pour aider à ajouter des variables de types string à des structures.
SimConnect_RequestResponseTimes	Utilisée pour fournir quelques données sur la performance de la connexion serveur-client.
SimConnect_RetrieveString	Utilisée pour aider à retrouver une variable depuis une structure.

I-1-3 Fonctions spécifiques aux objets d'AI :

Fonctions	Description
SimConnect_AICreateEnrouteATCAircraft	Utilisée pour créer une AI qui commande l'avion qui est sur le point de démarrer ou est déjà en cours de vol (exemple : le pilote automatique).
SimConnect_AICreateSimulatedObject	Utilisée pour créer une AI contrôlant des objets autres que les avions (exemple : building).
SimConnect_AIReleaseControl	Utilisée pour libérer l'AI de contrôle des objets simulables, (utilisée uniquement pour des avions, contrôlables par le client).
SimConnect_AIRemoveObject	Utilisée pour effacer n'importe quel objet crée par le client nécessitant l'utilisation d'une AI fonctions.
SimConnect_AISetAircraftFlightPlan	Utilisée pour modifier le plan de vol d'une AI contrôlant un avion.

I-1-4 Fonctions spécifiques à la caméra :

Fonctions	Description
SimConnect_CameraSetRelative6DOF	Utilisée pour ajuster la camera de l'utilisateur.

I-1-5 Fonctions spécifiques au menu :

Fonctions	Description
SimConnect_MenuAddItem	Utilisée pour ajouter un menu d'articles.
SimConnect_MenuAddSubItem	Utilisée pour ajouter un sous-menu d'articles.
SimConnect_MenuDeleteItem	Utilisée pour supprimer un menu d'articles.
SimConnect_MenuDeleteSubItem	Utilisée pour supprimer un sous-menu d'articles.

I-1-6 Fonctions spécifiques aux missions :

Fonctions	Description
SimConnect_CompleteCustomMissionAction	Renvoie des données en cas de réussite d'une mission.
SimConnect_ExecuteMissionAction	Utilisée pour exécuter une nouvelle mission.

I-1-7 Fonctions spécifiques au climat :

Fonctions	Description
SimConnect_WeatherCreateStation	Utilisée pour ajouter un état météorologique.
SimConnect_WeatherCreateThermal	Utilisée pour créer un état thermique pour un lieu bien précis.
SimConnect_WeatherRemoveStation	Utilisée pour supprimer un état météorologique crée auparavant.
SimConnect_WeatherRemoveThermal	Utilisée pour supprimer un état thermique crée auparavant.
SimConnect_WeatherRequestCloudState	Utilisée pour demander l'information sur la densité de nuages

I-2 Structures et énumérations de SimConnect :

SimConnect utilise les structures et énumérations suivantes :

Nom	Type	Description
SIMCONNECT_DATATYPE	Enum	Utilisée avec SimConnect_AddToDataDefinition pour spécifier le type de données que le serveur doit retourner au client.
SIMCONNECT_DATA_INITPOSITION	Struct	Utilisée pour initialiser la position de l'utilisateur.
SIMCONNECT_DATA_LATLONALT	Struct	Utilisée pour la géo-localisation.
SIMCONNECT_DATA_MARKERSTATE	Struct	Utilisée pour aider graphiquement le modèle de vol
SIMCONNECT_DATA_WAYPOINT	Struct	Utilisée pour obtenir toutes les informations nécessaires en un point donné
SIMCONNECT_EXCEPTION	Enum	Utilisée avec la structure SIMCONNECT_RECV_EXCEPTION pour retourner une information sur une erreur produite
SIMCONNECT_MISSION_END	Enum	Utilisée pour spécifier les trois résultats possibles d'une mission.
SIMCONNECT_PERIOD	Enum	Utilisée avec SimConnect_RequestDataOnSimObject pour spécifier combien de données

		doivent être envoyées au client.
SIMCONNECT_RECV	Struct	Utilisée avec l'énumération SIMCONNECT_RECV_ID pour indiquer quel type de structure a été retourné
SIMCONNECT_RECV_CLIENT_DATA	Struct	Est envoyée au client après appel à SimConnect_RequestClientData . Cette structure est identique à SIMCONNECT_RECV_SIMOBJECT_DATA.
SIMCONNECT_RECV_CLOUD_STATE	Struct	Utilisée pour retourner des données sur l'état des nuages.
SIMCONNECT_RECV_EVENT	Struct	Utilisée pour retourner un identificateur d'évènement "ID event".
SIMCONNECT_RECV_EVENT_FILENAME	Struct	Utilisée avec SimConnect_SubscribeToSystemEvent pour renvoyer un filename ainsi qu'un ID event au client.
SIMCONNECT_RECV_EVENT_FRAME	Struct	Utilisée avec SimConnect_SubscribeToSystemEvent pour renvoyer le taux d'armature et la vitesse de simulation au client.
SIMCONNECT_RECV_EVENT_OBJECT_ADDREMOVE	Struct	Utilisée pour retourner le type ainsi que l'ID de l'AI objet dont le client à supprimer ou ajouter.
SIMCONNECT_RECV_EXCEPTION	Struct	Utilisée avec l'énumération SIMCONNECT_EXCEPTION pour retourner les informations des erreurs produites durant la simulation.
SIMCONNECT_RECV_ID	Enum	Utilisée avec SIMCONNECT_RECV pour indiquer quel type de structure a été retourné.
SIMCONNECT_RECV_OPEN	Struct	Utilisée pour retourner des informations confirmant le succès de l'appel à la fonction SimConnect_Open.
SIMCONNECT_RECV_QUIT	Struct	Utilisée pour retourner des informations confirmant le succès de l'appel à la fonction SimConnect_Quit.
SIMCONNECT_RECV_SIMOBJECT_DATA	Struct	Est reçue par le client après appel aux fonctions SimConnect_RequestDataOnSimObject et SimConnect_RequestDataOnSimObjectType.
SIMCONNECT_RECV_SIMOBJECT_DATA_BYTYPE	Struct	Est reçue par le client après appel à la fonction SimConnect_RequestDataOnSimObjectType. Cette dernière est identique à celle de

		SIMCONNECT_RECV_SIMOBJECT_DATA.
SIMCONNECT_RECV_WEATHER_OBSERVATION	Struct	Utilisée pour renvoyée des données après appel, à l'une des fonctions suivantes: SimConnect_WeatherRequestInterpolatedObservation, SimConnect_WeatherRequestObservationAtStation, ou bien SimConnect_WeatherRequestObservationAtNearestStation.
SIMCONNECT_SIMOBJECT_TYPE	Enum	Utilisée avec SimConnect_RequestDataOnSimObjectType pour obtenir des informations sur un objet spécifique.

II Les fonctions utilisées :

II-1 SimConnect_Open :

Cette fonction envoie une requête au serveur *Flight Simulator* pour ouvrir une communication avec un nouveau client.

Syntaxe:

```
HRESULT SimConnect_Open(
  HANDLE*  phSimConnect,
  LPCSTR  szName,
  HWND  hWnd,
  DWORD  UserEventWin32,
  HANDLE  hEventHandle,
  DWORD  ConfigIndex
);
```

Paramètres :

phSimConnect

Ce premier paramètre est le handle de la fonction déclaré comme pointeur. C'est un numéro unique attribué par le système d'exploitation qui lui permet de l'identifier.

szName

Est le nom attribué par le client au programme.

hWnd

Est utilisé au cas où l'utilisateur utilisait un objet handle de Windows. Toujours mis à la valeur NULL s'il n'y en a pas.

UserEventWin32

Est un nombre que le client peut spécifier s'il veut présenter un code à cette fonction. Mettre égale à 0 s'il n'est pas utilisé.

hEventHandle
Est un handle d'évènement Windows. Le client peut l'ajouter pour synchroniser d'autre application windows qui peuvent interagir avec l'application Flight Simulator. Mettre à 0 s'il n'y en pas.

ConfigIndex
L'index de configuration, est toujours ajouté au cas où le client voudrait apporter des modifications au niveau du fichier SimConnect.cfg. Mettre à 0 s'il n'y a pas de modification à mettre.

Valeur retournée :
Cette fonction retourne un HRESULT qui peut prendre les trois valeurs suivantes :

S_OK	L'appel à la fonction a réussie.
E_FAIL	L'appel à la fonction a échouée.
E_INVALIDARG	Une erreur s'est produite lors de l'appel du dernier paramètre, probablement un argument manquant est signalé.

II-2 SimConnect_AddToDataDefinition :
Cette fonction est utilisée pour ajouter une nouvelle variable de simulation à l'application client.

Syntaxe :
```
HRESULT SimConnect_AddToDataDefinition(
  HANDLE  hSimConnect,
  SIMCONNECT_DATA_DEFINITION_ID  DefineID,
  const char*  DatumName,
  const char*  UnitsName,
);
```

Paramètres :
hSimConnect
Un handle vers un objet SimConnect.

DefineID
Spécifie l'ID de la définition de données établies au préalable par le client.

DatumName
Spécifie le nom de la variable de simulation.

UnitsName
Spécifie l'unité de la variable par exemple : (feet, degrees, …etc.).

Valeur retournée :
Cette fonction retourne un HRESULT qui peut prendre les deux valeurs suivantes :

S_OK	L'appel à la fonction a réussie.
E_FAIL	L'appel à la fonction a échouée.

Remarque :
Le nombre maximum de données à ajouter dans l'application client est de 1000.

II-3 SimConnect_MapClientEventToSimEvent :

Cette fonction associe un ID event définie par le client avec un événement attribué à Flight Simulator.

Syntaxe :

```
HRESULT SimConnect_MapClientEventToSimEvent(
  HANDLE  hSimConnect,
  SIMCONNECT_CLIENT_EVENT_ID  EventID,
  const char*  EventName
);
```

Paramètres:

hSimConnect
Un handle vers un objet SimConnect.

EventID
Spécifie l'ID de l'événement client.

EventName
Spécifie le nom de l'événement de *Flight Simulator* [1].

Valeur retournée :

Cette fonction retourne un HRESULT qui peut prendre les deux valeurs suivantes :

S_OK	L'appel à la fonction a réussie.
E_FAIL	L'appel à la fonction a échouée.

II-4 SimConnect_AddClientEventToNotificationGroup :

Cette fonction est utilisée à fin d'ajouter un événement -préalablement définie et associé à un événement Flight Simulator- à un groupe d'événement spécifique au client

Syntaxe :

```
HRESULT SimConnect_AddClientEventToNotificationGroup(
  HANDLE  hSimConnect,
  SIMCONNECT_NOTIFICATION_GROUP_ID  GroupID,
  SIMCONNECT_CLIENT_EVENT_ID  EventID,
  BOOL  bMaskable = FALSE
);
```

Paramètres :

hSimConnect

Un handle vers un objet SimConnect.

GroupID

Spécifie l'ID du groupe défini par client

EventID

Spécifie l'ID de l'événement client.

bMaskable

De type booléen, la valeur True indique que l'évènement a été masqué par le client et ne peux être transmit à d'autre client *Flight Simulator* au cas d'une application réseau. False, le contraire.

Valeur retournée :

Cette fonction retourne un HRESULT qui peut prendre les deux valeurs suivantes :

S_OK	L'appel à la fonction a réussie.
E_FAIL	L'appel à la fonction a échouée.

II-5 SimConnect_SetNotificationGroupPriority :

La fonction SimConnect_SetNotificationGroupPriority est utilisée pour ajouter la priorité pour un groupe d'événements (utilisée pour les applications réseau).

Syntaxe :

```
HRESULT SimConnect_SetNotificationGroupPriority(
  HANDLE  hSimConnect,
  SIMCONNECT_NOTIFICATION_GROUP_ID  GroupID,
  DWORD  uPriority
);
```

Paramètres :

hSimConnect

Un handle vers un objet SimConnect.

GroupID

Spécifie l'ID du groupe défini par client

uPriority

Spécifie la priorité du groupe.

Valeur retournée :

Cette fonction retourne un HRESULT qui peut prendre les deux valeurs suivantes :

S_OK	L'appel à la fonction a réussie.
E_FAIL	L'appel à la fonction a échouée.

Remarque :

Priorités SimConnect

Priorités	Valeur	Description
SIMCONNECT_GROUP_PRIORITY_HIGHEST	1	La priorité la plus haute
SIMCONNECT_GROUP_PRIORITY_HIGHEST_MASKABLE	10000000	La priorité la plus haute qui permet à des événements d'être masqués.
SIMCONNECT_GROUP_PRIORITY_DEFAULT	2000000000	Priorité par default
SIMCONNECT_GROUP_PRIORITY_LOWEST	4000000000	La plus basse priorité

II-6 SimConnect_TransmitClientEvent :

Cette fonction est utilisée pour transmettre des événements à l'application *Flight Simulator*

Syntaxe :

```
HRESULT SimConnect_TransmitClientEvent(
  HANDLE  hSimConnect,
  SIMCONNECT_OBJECT_ID  ObjectID,
  SIMCONNECT_CLIENT_EVENT_ID  EventID,
  DWORD  dwData,
  SIMCONNECT_NOTIFICATION_GROUP_ID  GroupID,
  SIMCONNECT_EVENT_FLAG  Flags
);
```

Paramètres:

hSimConnect
Un handle vers un objet SimConnect.

ObjectID
Par default égale à 0, utilisé pour les applications réseau.

EventID
Spécifie l'ID de l'événement client.

dwData
Double word contenant des données nécessaires à des événements spécifiques.

GroupID
Spécifie l'ID du groupe défini par le client.

Flags
Peut prendre une des identifications suivantes :

Flag	Description
0	Ne rien faire
SIMCONNECT_EVENT_FLAG_SLOW_REPEAT_TIMER	Ce flag remettra à zéro le temporisateur de répétition pour simuler la répétition lente. Employez ce drapeau pour remettre à zéro (activer) le temporisateur de répétition qui fonctionne avec divers événements de clavier et clics de souris.
SIMCONNECT_EVENT_FLAG_FAST_REPEAT_TIMER	Ce flag remettra à zéro le temporisateur de répétition pour simuler la répétition rapide.
SIMCONNECT_EVENT_FLAG_GROUPID_IS_PRIORITY	Indique au serveur SimConnect que le traitement commence d'abords par ordre de priorité, et que la vitesse de temporisation en dépondra aussi.

Valeur retournée :

Cette fonction retourne un HRESULT qui peut prendre les deux valeurs suivantes :

S_OK	L'appel à la fonction a réussie.
E_FAIL	L'appel à la fonction a échouée.

II-7 SimConnect_CallDispatch :

Cette fonction est utilisée pour traiter le prochain message reçut par une fonction de rappel (callback).

Syntaxe :

```
HRESULT SimConnect_CallDispatch(
  HANDLE  hSimConnect,
  DispatchProc  pfcnDispatch,
  void *  pContext
);
```

Paramètres :

hSimConnect
Un handle vers un objet SimConnect.

pfcnDispatch
Spécifie la fonction de rappel (the callback fonction).

pContext
Spécifie un indicateur que le client peut définir qui sera retourné dans le rappel de service.

Valeur retournée :

Cette fonction retourne un HRESULT qui peut prendre les deux valeurs suivantes :

S_OK	L'appel à la fonction a réussie.
E_FAIL	L'appel à la fonction a échouée.

II-8 DispatchProc :

Cette fonction est développée et appelée par le client, comme une fonction de rappel (callback fonction), à fin d'établir toutes les communications nécessaires avec le serveur Flight Simulator.

Syntaxe :

```
void CALLBACK DispatchProc(
  SIMCONNECT_RECV*  pData,
  DWORD  cbData
  void *  pContext
);
```

Paramètres :

pData
Pointeur de données traitées comme une structure SIMCONNECT_RECV.

cbData
La taille de la données en octet.

pContext
C'est le même paramètre utilisé dans la fonction précédente.

Valeur retournée :

Cette fonction ne retourne aucune valeur.

V-9 SimConnect_RequestDataOnSimObject :

Cette fonction est employée pour demander quand le client doit recevoir des données concernant un objet spécifique.

Syntaxe :

```
HRESULT SimConnect_RequestDataOnSimObject(
  HANDLE  hSimConnect,
  SIMCONNECT_DATA_REQUEST_ID  RequestID,
  SIMCONNECT_DATA_DEFINITION_ID  DefineID,
  SIMCONNECT_OBJECT_ID  ObjectID,
  SIMCONNECT_PERIOD  Period,
  SIMCONNECT_DATA_REQUEST_FLAG  Flags = 0,
  DWORD  origin = 0,
  DWORD  interval = 0,
  DWORD  limit = 0
);
```

Paramètres :

hSimConnect
Un handle vers un objet SimConnect.

RequestID
Spécifie l'ID des demandes définies par le client.

DefineID
Spécifie l'ID des données définies par le client.

ObjectID
Spécifie l'ID de l'objet de *Flight Simulator* dont les données lui sont appropriées.

Period
Représente l'unité de la période qui devrait s'écouler entre les transmissions des données (en seconde).

Flags
Peut prendre l'une des identifications suivantes :

Flag value	Description
0	Etablie par defaults.
SIMCONNECT_DATA_REQUEST_FLAG_CHANGED	Les données vont être envoyées uniquement si l'une d'elles a été change, dans le cas contraire, rien n'est renvoyé.

origin
Ce paramètre représente la période qui devrait s'écouler avant que la transmission des données commence. Si cette valeur est égale à 0, cela signifierai que les transmissions commenceront immédiatement.

interval

La période qui devrait s'écouler entre les transmissions des données. Si cette valeur est égale à 0, cela signifierai que les données vont être transmises chaque période.

limit
Le nombre de fois où les données devraient être transmises avant que cette communication soit finie. Si cette valeur est égale à 0, cela signifierai que les données devraient être transmises sans fin

Valeur retournée :
Cette fonction retourne un HRESULT qui peut prendre les deux valeurs suivantes :

S_OK	L'appel à la fonction a réussie.
E_FAIL	L'appel à la fonction a échouée.

II-10 SimConnect_WeatherRequestObservationAtNearestStation :

Cette fonction est utilisée pour envoyer une demande d'obtention d'informations concernant l'objet *Flight Simulator*, ces données sont la latitude et la longitude.

Syntaxe :
```
HRESULT SimConnect_WeatherRequestObservationAtNearestStation(
  HANDLE  hSimConnect,
  SIMCONNECT_DATA_REQUEST_ID  RequestID,
  float  lat,
  float  lon
);
```

Paramètres :
hSimConnect
Un handle vers un objet SimConnect.

RequestID
Spécifie l'ID des demandes définies par le client.

lat
Spécifie la latitude en degré.

lon
Spécifie la longitude en degré.

Valeur retournée :
Cette fonction retourne un HRESULT qui peut prendre les deux valeurs suivantes :

S_OK	L'appel à la fonction a réussie.
E_FAIL	L'appel à la fonction a échouée.

II-11 SimConnect_Close :

Cette fonction est utilisée pour envoyer une rupture de communication avec le serveur

Syntaxe :

```
HRESULT SimConnect_Close(
  HANDLE  hSimConnect
);
```

Paramètres :

hSimConnect
Un handle vers un objet SimConnect.

Valeur retournée :

Cette fonction retourne un HRESULT qui peut prendre les deux valeurs suivantes :

S_OK	L'appel à la fonction a réussie.
E_FAIL	L'appel à la fonction a échouée.

III Les structures et énumérations utilisées :

III-1 SIMCONNECT_RECV :

Cette structure est utilisée avec l'énumération SIMCONNECT_RECV_ID pour indiquer quel type de structure a été retourné.

Syntaxe :

```
struct SIMCONNECT_RECV{
  DWORD  dwSize;
  DWORD  dwVersion;
  DWORD  dwID;
};
```

Membres :

dwSize
La taille de la valeur retournée en octet.

dwVersion
Le nombre de version du serveur de SimConnect.

dwID
L'ID de la structure utilisée: doit être l'une des variables de SIMCONNECT_RECV_ID.

Remarque :

Cette structure utilise les paramètres suivants :

- SIMCONNECT_RECV_ASSIGNED_OBJECT_ID
- SIMCONNECT_RECV_CLOUD_STATE
- SIMCONNECT_RECV_EXCEPTION
- SIMCONNECT_RECV_EVENT (utilisé pour attribuer ce paramètre à un évènement spécifique à l'application client).
- SIMCONNECT_RECV_SIMOBJECT_DATA
- SIMCONNECT_RECV_SIMOBJECT_DATA_BYTYPE

- SIMCONNECT_RECV_OPEN
- SIMCONNECT_RECV_RESERVED_KEY
- SIMCONNECT_RECV_SYSTEM_STATE
- SIMCONNECT_RECV_WEATHER_OBSERVATION

III-2 SIMCONNECT_RECV_ID:

Cette énumérations est employée dans la structure précédente « SIMCONNECT_RECV » pour spécifier quels types de structure doit être retourné.

Syntaxe:

```
enum SIMCONNECT_RECV_ID{
  SIMCONNECT_RECV_ID_NULL,
  SIMCONNECT_RECV_ID_EXCEPTION,
  SIMCONNECT_RECV_ID_OPEN,
  SIMCONNECT_RECV_ID_QUIT,
  SIMCONNECT_RECV_ID_EVENT,
  SIMCONNECT_RECV_ID_EVENT_OBJECT_ADDREMOVE,
  SIMCONNECT_RECV_ID_EVENT_FILENAME,
  SIMCONNECT_RECV_ID_EVENT_FRAME,
  SIMCONNECT_RECV_ID_SIMOBJECT_DATA,
  SIMCONNECT_RECV_ID_SIMOBJECT_DATA_BYTYPE,
  SIMCONNECT_RECV_ID_CLOUD_STATE,
  SIMCONNECT_RECV_ID_WEATHER_OBSERVATION,
  SIMCONNECT_RECV_ID_ASSIGNED_OJBECT_ID,
  SIMCONNECT_RECV_ID_RESERVED_KEY,
  SIMCONNECT_RECV_ID_CUSTOM_ACTION,
  SIMCONNECT_RECV_ID_SYSTEM_STATE,
  SIMCONNECT_RECV_ID_CLIENT_DATA,
};
```

Membres:

Paramètres	Structures reçues
SIMCONNECT_RECV_ID_NULL	NULL
SIMCONNECT_RECV_ID_EXCEPTION	SIMCONNECT_RECV_EXCEPTION
SIMCONNECT_RECV_ID_OPEN	SIMCONNECT_RECV_OPEN
SIMCONNECT_RECV_ID_EVENT	SIMCONNECT_RECV_EVENT
SIMCONNECT_RECV_ID_EVENT_OBJECT_ADDREMOVE	SIMCONNECT_RECV_EVENT_OBJECT_ADDREMOVE
SIMCONNECT_RECV_ID_EVENT_FILENAME	SIMCONNECT_RECV_EVENT_FILENAME
SIMCONNECT_RECV_ID_EVENT_FRAME	SIMCONNECT_RECV_EVENT_FRAME
SIMCONNECT_RECV_ID_SIMOBJECT_DATA	SIMCONNECT_RECV_SIMOBJECT_DATA
SIMCONNECT_RECV_ID_SIMOBJECT_DATA	SIMCONNECT_RECV_SIMOBJECT

_BYTYPE	_DATA_BYTYPE
SIMCONNECT_RECV_ID_WEATHER_OBSERVATION	SIMCONNECT_RECV_WEATHER_OBSERVATION
SIMCONNECT_RECV_ID_CLOUD_STATE	SIMCONNECT_RECV_CLOUD_STATE
SIMCONNECT_RECV_ID_ASSIGNED_OJBECT_ID	SIMCONNECT_RECV_ASSIGNED_OBJECT_ID
SIMCONNECT_RECV_ID_RESERVED_KEY	SIMCONNECT_RECV_RESERVED_KEY
SIMCONNECT_RECV_ID_CUSTOM_ACTION	SIMCONNECT_RECV_CUSTOM_ACTION
SIMCONNECT_RECV_ID_SYSTEM_STATE	SIMCONNECT_RECV_SYSTEM_STATE
SIMCONNECT_RECV_ID_CLIENT_DATA	SIMCONNECT_RECV_CLIENT_DATA
SIMCONNECT_RECV_ID_QUIT	SIMCONNECT_RECV QUIT

I Méthode de calcul

Considérons un repère orthonormé Oxyz. Soient θ et φ les coordonnées angulaires d'un point P sur la sphère (voir la figure 3), θ étant l'angle entre l'axe Oz et le rayon OP (c'est la colatitude de P : sa latitude est $\pi/2 - \theta$) et φ l'angle azimutal, c'est-à-dire l'angle polaire que fait la projection de OP sur le plan xOy avec l'axe Ox.

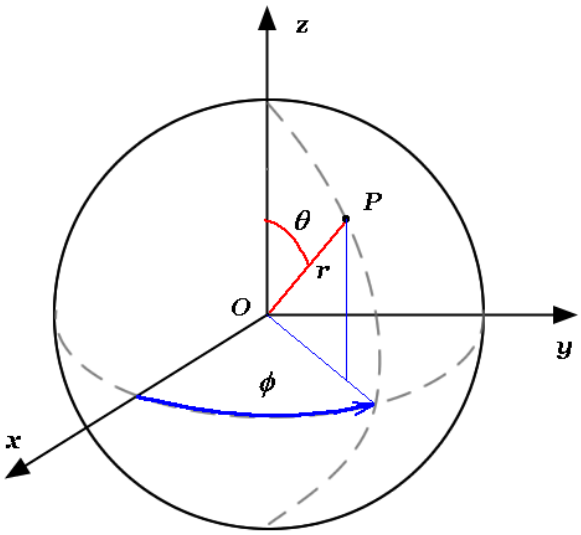

Figure I : Repérage d'un point en coordonnées sphériques

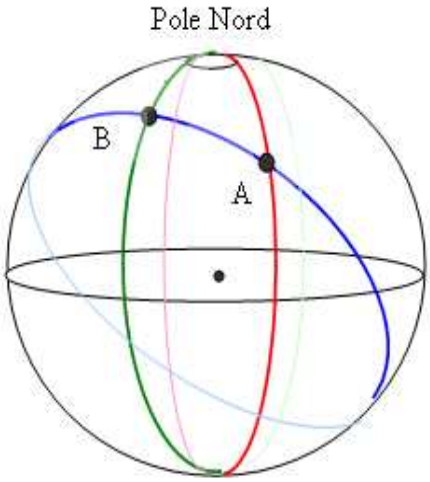

Figure II : Représentation des points A et B sur le globe terrestre

En prenant le rayon de la sphère pour unité, les coordonnées cartésiennes des points A (point 1) et B (point 2) s'expriment en fonction des coordonnées polaires par les formules classiques.

$$\mathrm{x1} = \sin\theta 1 \cos\varphi 1$$

$$\mathrm{y1} = \sin\theta 1 \sin\varphi 1$$

$$\mathrm{z1} = \cos\theta 1$$

Et

$$\mathrm{x2} = \sin\theta 2 \cos\varphi 2$$

$$\mathrm{y2} = \sin\theta 2 \sin\varphi 2$$

$$\mathrm{z2} = \cos\theta 2$$

Le carré de la longueur du segment AB (c'est-à-dire du tunnel virtuel sous Terre) est

$$AB^2 = (x2 - x1)^2 + (y2 - y1)^2 + (\mathrm{z2} - \mathrm{z1})^2$$

Soit

$$AB^2 = 2\,[1 - \sin\theta 1\ \sin\theta 2\ \cos(\varphi 2 - \varphi 1) - \cos\theta 1\ \cos\theta 2\,]$$

La distance rectiligne *h* = AB/ 2 représente le sinus de la moitié de l'angle au centre AOB recherché.

Dans ce cas là, la longueur de l'arc AB

$$arc(AB) = \theta\ R$$

Rappelons que le rayon R est pris pour unité, on aura donc

$$arc(AB) = 2\ \arcsin h$$

Où *h* est défini par son carré

$$h^2 = (1/2)\,[1 - \sin\theta 1\ \sin\theta 2\ \cos(\varphi 2 - \varphi 1) - \cos\theta 1\ \cos\theta 2\,]$$

Pour simplifier cette relation on part de

$$\cos 2\mathrm{u} = 1 - 2\ \sin^2 u$$

Qui entraîne

$$\cos(2\arcsin h) = 1 - 2\ \sin^2(\arcsin h)$$

Soit

$$\cos(2 \arcsin h) = 1 - 2\ h^2$$

En inversant cette dernière formule nous obtenons

$$2 \arcsin\ h = \arccos\ (1 - 2\ h^2)$$

Et l'expression de l'arc AB devient (toujours sur une sphère de rayon unité)

$$arc(AB) = \arccos[\ \sin\theta 1\ \sin\theta 2\ \cos(\varphi 2 - \varphi 1) - \ \cos\theta 1\ \cos\theta 2\ ]$$

Sur Terre on repère un point par sa latitude et sa longitude. La latitude *lat* est le complément à π / 2 de la colatitude θ, ce qui veut dire que

Lat= π / 2 – θ.

En notant R le rayon terrestre (soit 6 378 kilomètres, rayon équatorial moyen) et en notant *long* la différence de longitude (φ=φ2-φ1) entre les deux points, nous obtenons la formule finale donnant la distance orthodromique entre les points A et B de latitude *lat1* et *lat2* (λ1 et λ2) et de différence de longitude *long* comme :

$$arc(AB) = R\ \arccos[\ \cos\lambda 1\ \cos\lambda 2\ \cos(\varphi) - \ \sin\lambda 1\ \sin\lambda 2\ ]$$