SOMMAIRE

LISTE DES ABREVIATIONS

- AC : Alternative Current
- ALU : Arithmetic Logic Unit
- API : Application Programmable Interface.
- ASIC: Application Specific Integrated Circuit
- CAN : Convertisseur Analogique Numérique
- CAO : Conception Assistée par Ordinateur
- CNA : Convertisseur Numérique Analogique
- DC : Direct Current
- DTCM : Data Tightly Coupled Memories
- éch/s : échantillons par seconde
- FIFO : First In First Out
- FPGA : Field Programmable Gate Array
- GUI : Graphical User Interface
- IDE : Integrated Developpement Environment
- IEEE : Institut of Electrical and Electronics Engineers
- IP Cores : Intellectual Property Cores
- ITCM : Instruction Tightly Coupled Memories
- LCPS : Low-Cost Programming Stick
- LR: Link Register
- LUT : Look Up Table
- MFC : Microsoft Foundation Classes
- MIPS : Million d'Instructions Par Seconde
- NI : National Instruments
- NVIC: Nested Vectored Interrupt Controller
- PC : Personal Computer
- PC : Program Counter
- PCI : Peripheral Component Interconnect

- PLD : Programmable Logic Device
- PSR : Program Status Register
- PXI : PCI Extension for Instruments
- RAM : Random Access Memory
- RISC : Reduced Instruction Set Computer
- RVI : Reconfigurable Virtual Instrument, instrument virtuel reconfigurable
- SDRAM : Synchronous Dynamic RAM
- SP : Stack Pointers
- SRAM : Static RAM
- UML : Unified Modeling Language (langage de modélisation unifié)
- USB : Universal Serial Bus
- VHDL : VHSIC Hardware Description Language
- VHSIC : Very High Speed Integrated Circuit
- VME : Virtual Machine Environment.
- VXI : VME eXtensions for Instrumentation. Plate-forme spécialement conçue pour les instruments virtuels.

LISTE DES TABLEAUX

LISTE DES FIGURES

INTRODUCTION

Les instruments électroniques et scientifiques sont indispensables dans tous les instituts de recherches scientifiques et les établissements techniques. Ces instruments évoluent, se perfectionnent, s'adaptent aux besoins des utilisateurs. En parallèle leurs coûts augmentent suivant leurs précisions et leurs performances. Souvent, l'utilisateur a tendance à s'équiper d'un appareil fournissant des fonctionnalités et des performances supérieures à ce qui lui est nécessaire pour ses études en cours en prévision de ses besoins futurs. Il préfère s'assurer que l'instrument qu'il choisit propose les fonctionnalités et les outils qui seront adaptés à ses applications à venir. Des chercheurs ont donc travaillé pour mettre au point des matériels complexes reconfigurables pour simuler n'importe quel instrument spécifique, c'est-à-dire en quelque sorte un instrument polyvalent configurable en un instrument quelconque. En effet, la mise au point de ce système réduit le coût de développement et rend possible l'évolutivité de l'instrument en termes de performance, capacité, précision et fonctionnalité. La configuration de l'instrument se fait par un langage de description matériel de haut niveau, ce qui donne une grande souplesse à la conception. La conception se fait en effet de manière logicielle dans un environnement de développement intégré (IDE) semblable à ceux pour les conceptions des logiciels informatiques. La mise au point d'un tel système implique deux choix : un circuit logique programmable haute performance et un langage de description pour implémenter la description dans le matériel.

L'idée d'un instrument virtuel n'est pas nouvelle, mais c'est encore récemment que cette solution a gagné l'attention qu'elle mérite de la part des scientifiques grâce à l'évolution considérable des circuits logiques programmables (ASIC et FPGA en particulier) et la spécification des langages de description évolués (VHDL, Verilog). Cette solution est en fait très intéressante pour concevoir non seulement des instrumentations de mesure mais presque tous les instrumentations scientifiques utilisées dans des domaines très variés : en médecine, en biologie, en aéronautique. Ces principaux atouts étant la possibilité de faire évoluer l'instrumentation et la réduction des coûts de développement et de production du système d'instrumentation.

Cet ouvrage contient cinq chapitres. Le chapitre I explique le principe de base de l'Instrumentation Virtuelle. Le Chapitre II met en œuvre la programmation matérielle, le Chapitre III, la programmation logicielle. Les Chapitres IV et V détaillent respectivement la réalisation d'un oscilloscope et d'un générateur de forme d'onde.

CHAPITRE I. GENERALITES SUR L'INSTRUMENTATION VIRTUELLE

I.1.Introduction à l'instrumentation virtuelle

I.1.1. De l'instrumentation à l'instrumentation virtuelle

Au début, les premiers instruments électriques ont été contrôlés manuellement. Le domaine de l'instrumentation a fait un grand progrès vers les appareils de mesure sophistiqués gérés par ordinateur et programmés par l'utilisateur. L'instrumentation avait les phases suivantes :

- Appareils de mesure analogiques,
- Appareils d'acquisition et de traitement de données,
- Traitement numérique de données basé sur des systèmes électroniques et informatiques,
- Instrumentation virtuelle distribuée

La première phase est représentée par les premiers appareils de mesure analogiques purs tels que les oscilloscopes analogiques. L'usage supplémentaire de données ne faisait pas partie du paquet de l'instrument, et un opérateur devait copier physiquement les données à un cahier en papier ou à une fiche technique. Une représentation complexe ou des procédures de test automatisées ont été plutôt compliquées ou même impossible, comme tout devait être mis manuellement.

La deuxième phase a commencé en 1950, par suite de demandes du champ de commande industriel. Les instruments ont commencé à numériser les signaux mesurés en autorisant le traitement numérique de données et en introduisant un contrôle plus complexe ou des décisions analytiques. Cependant, les exigences du traitement numérique en temps réel étaient encore trop hautes. Les instruments étaient encore des boîtes définies par le vendeur.

Dans la troisième phase, c'est la phase qui nous intéresse le plus étant donné qu'elle concerne surtout notre projet, les instruments de mesure sont devenus basés sur ordinateur. Ils ont commencé à inclure des interfaces qui ont permis la communication entre l'instrument et l'ordinateur. Initialement, les ordinateurs ont été utilisés comme des instruments autonomes. Comme la vitesse et les capacités des ordinateurs croissent exponentiellement alors ils sont devenus plus rapides pour des mesures en temps réel complexes. Bien que la performance des

ordinateurs soit devenue assez haute, les ordinateurs n'étaient pas encore faciles à utiliser pour les expérimentations. Presque tous les premiers programmes de commande de l'instrument ont été écrit en BASIC, parce qu'il avait été le langage le plus utilisé avec des contrôleurs spécialisés pour l'instrument. Cela exige des ingénieurs et autres utilisateurs de devenir programmeurs avant d'être utilisateurs de l'instrument. Par conséquent, une borne importante dans l'histoire de l'instrumentation virtuelle s'est passée en 1986, quand «National Instruments » a introduit « LabVIEW 1.0 » sur une plate-forme PC (Personal Computer). LabVIEW a introduit des interfaces utilisateurs graphiques (GUI : Graphical User Interface) et une programmation visuelle de l'instrumentation informatisée, en joignant la simplicité d'une opération de l'interface utilisateur avec des capacités élevées d'ordinateurs. Aujourd'hui, le PC est la plate-forme sur laquelle la plupart des mesures sont faites, et l'interface utilisateur graphique a rendu des mesures conviviales. En conséquence, l'instrumentation virtuelle a rendu possible la baisse du prix d'un instrument.

La quatrième phase est devenue faisable avec le développement de réseaux locaux et globaux d'ordinateurs, depuis que la plupart des instruments ont déjà été informatisés. Les avances dans les télécommunications et les technologies de réseau ont fait des systèmes du télémédical pour fournir de l'information médicale et des services à distance, grâce à la distribution physique de composants d'instruments virtuels. L'infrastructure possible pour l'instrumentation virtuelle distribuée inclut l'internet, les réseaux privés et les réseaux cellulaires où l'interface entre les composants peut être équilibrée pour le prix et la performance**. [1]**

I.1.2. Notion sur les instruments virtuels

Du fait des besoins croissants en puissance, capacité, et précision des instruments utilisés dans divers domaines scientifiques, la conception d'un instrument est toujours révisée. Pour une nouvelle version d'un instrument, même si juste une petite partie de l'instrument est à remanier, il fallait remplacer l'instrument tout entier ! L'utilisateur était donc constamment contraint à acheter un nouvel instrument (souvent plus coûteux) et à laisser l'ancienne version. Face à cette situation, des chercheurs ont envisagé une solution plus flexible pour concevoir *des instruments reconfigurables,* en utilisant des circuits logiques programmables. La reconfigurabilité permet de faire évoluer les fonctionnalités de l'instrument. Ainsi l'utilisateur est rassuré que son instrument peut suivre les évolutions des techniques de mesure et des normes. Pour les concepteurs, cette solution leur offre une nouvelle manière de concevoir les instruments, une manière souple, conviviale et rapide. L'instrument

reconfigurable envisagé doit être capable d'émuler des configurations matérielles complexes. Cette technologie est possible grâce à des circuits logiques également complexes et reprogrammables, ainsi qu'un langage de haut niveau pour programmer ces circuits. Le but est de concevoir une plateforme matériel/ logiciel réutilisable pour l'émulation de différents instruments électroniques et scientifiques. L'architecture est basé sur des partages en blocs indépendants ce qui favorisent la réutilisabilité et l'aptitude à des évolutions complexes.

On peut donner la définition de l'Instrumentation Virtuelle comme l'association d'outils informatiques (ordinateurs, logiciels, réseaux…) et des moyens d'entrée/sortie (interface Série, Parallèle, USB, modules d'acquisition, instruments de mesure…) afin de réaliser des systèmes définis par l'utilisateur.

La définition que nous avons donnée plus haut est très générale. Ceci dit, certains termes sont importants. Par exemple, le fait que le système soit "défini par l'utilisateur" est tout à fait essentiel puisque cela place l'utilisateur au core des préoccupations. Il faut noter que par rapport à des systèmes fermés, prêts-à-l'emploi, propriétaires, l'Instrumentation Virtuelle propose une alternative dans laquelle le concepteur peut utiliser son libre arbitre. Certes, cela impose de faire des choix et des responsabilités mais en contre partie, l'Instrumentation Virtuelle est la seule solution qui permette de concevoir des systèmes dont les coûts sont optimaux et qu'il est possible de faire évoluer.

Par exemple, avec un même matériel d'E/S, il est possible de concevoir différents Instruments Virtuels. Outre les économies qui en découlent, c'est surtout un formidable exemple de ce que signifie "systèmes définis par l'utilisateur". Un jour, l'ensemble matériel-logiciel peut être utilisé comme un analyseur de spectre alors que le lendemain, il se comportera comme un enregistreur de données.

La réciproque est vraie aussi. En effet, une application peut et doit être utilisée avec différents types de matériel. Par exemple, une première version d'un Instrument Virtuel met en œuvre un boîtier externe USB d'acquisition de données alors que la seconde utilise un module d'acquisition PCI ou tout un système de conditionnement du signal.

L'autre notion importante qui est présente dans la définition est le fait que l'Instrumentation Virtuelle résulte d'une "association" entre le logiciel et le matériel. Fondamentalement, cela signifie que dès leur conception, les logiciels et les matériels utilisés dans un Instrument Virtuels doivent avoir été pensés pour travailler ensemble.

I.1.3. Analyse des coûts

D'une manière générale, l'instrumentation virtuelle réduit le coût des mesures. En effet, la conception et la mise en œuvre de systèmes de test et mesure engendrent des coûts fixes et des coûts liés au temps de travail.

a) Les coûts fixes

Les coûts fixes comprennent le prix du package logiciel et du matériel d'acquisition de données. Dans notre cas, ce prix est de $250 soit environ Ar 600 000. À l'heure actuelle, grâce à la convivialité des achats en ligne, nous pouvons rapidement et facilement déterminer ces coûts dès que nous savons ce dont nous avons besoin.

b) Les coûts liés au temps de travail

Les coûts liés au temps de travail, en revanche, sont plus difficiles à calculer. En conséquence, ils sont fréquemment oubliés du coût total du système. L'instrumentation virtuelle combine des logiciels et du matériel modulaire avec des technologies informatiques standards afin de fournir plusieurs solutions capables de minimiser les coûts cachés liés au temps de travail qui interviennent au cours du développement.

En se référant aux données fournies par la société NI (*National Instruments*) qui est un leader mondial de l'instrumentation virtuelle, on peut savoir comment les ingénieurs et scientifiques emploient leur temps et, de ce fait, mieux comprendre leurs besoins, NI a enquêté auprès des experts de différents secteurs industriels et de différents pays afin de déterminer la répartition des coûts de leur système de test ou de mesure. Cette enquête a permis de distinguer cinq types de coûts représentés dans la **Figure1.1** ci-après pour le développement d'une application de mesure. Le coût le plus important est le prix des matériels et logiciels, qui représente 36% du total ; autrement dit, c'est le coût fixe d'après ce que nous avons vu précédemment **[1].** Toutefois, les quatre autres peuvent être regroupés pour former les coûts cachés liés au temps de travail et représentent alors 64% du total. Les dernières innovations de l'instrumentation virtuelle concernent tout particulièrement ces derniers (développement du logiciel, installation, spécifications du système, validation du système et étalonnage du matériel) et réduisent ainsi les risques d'échec ou les dépassements de budget.

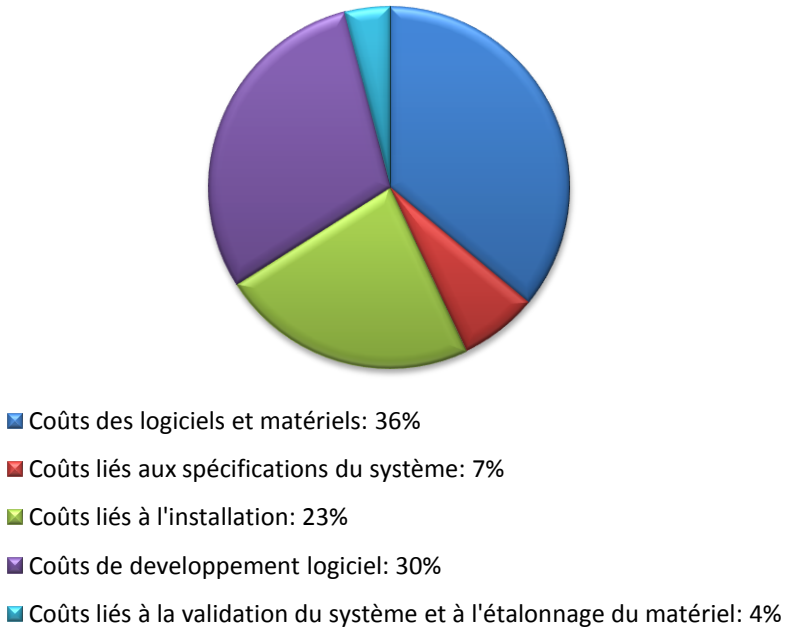


Figure 1.1 : Les différents types de coûts (données fournies en 2010) **[2]**

I.2. Conception d'un instrument virtuel

I.2.1. Architecture d'un système Instrument Virtuel Reconfigurable

Un instrument reconfigurable (Reconfigurable Instrument) est un périphérique matériel polyvalent qui peut être configuré en un instrument électronique en utilisant un outil logiciel. Une instrumentation virtuelle reconfigurable (RVI pour Reconfigurable Virtual Instrumentation) est un système matériel et logiciel qui sert à émuler l'instrument reconfigurable à l'aide d'une console virtuelle et d'une interface graphique. Cette interface graphique permet les réglages et les commandes du matériel à travers la console logicielle comme pour un instrument réel.

Une application logicielle de haut niveau fournit une interface graphique à l'utilisateur qui peut choisir un instrument virtuel comme un oscilloscope numérique, générateur de forme d'onde, depuis une librairie d'instruments et configure le système RVI en l'instrument sélectionné avec sa console.

Un approche modulaire permet les évolutions indépendantes des architectures matérielles et logicielles, par le biais de la réutilisation de la plupart des composants développés (codes sources logiciels et cores matériels).

Le système RVI de haut niveau comprend les sous-systèmes matériel et logiciel. Le système matériel est formé d'un PC et de l'instrument reconfigurable connecté physiquement au PC via un port (Port série, Port parallèle, USB, Ethernet…). Cette connexion permet la reconfiguration de l'instrument reconfigurable aussi bien que l'échange d'information entre le PC et l'instrument reconfigurable. Ces informations peuvent être des données, commandes, messages d'erreurs ou accusés de réception.

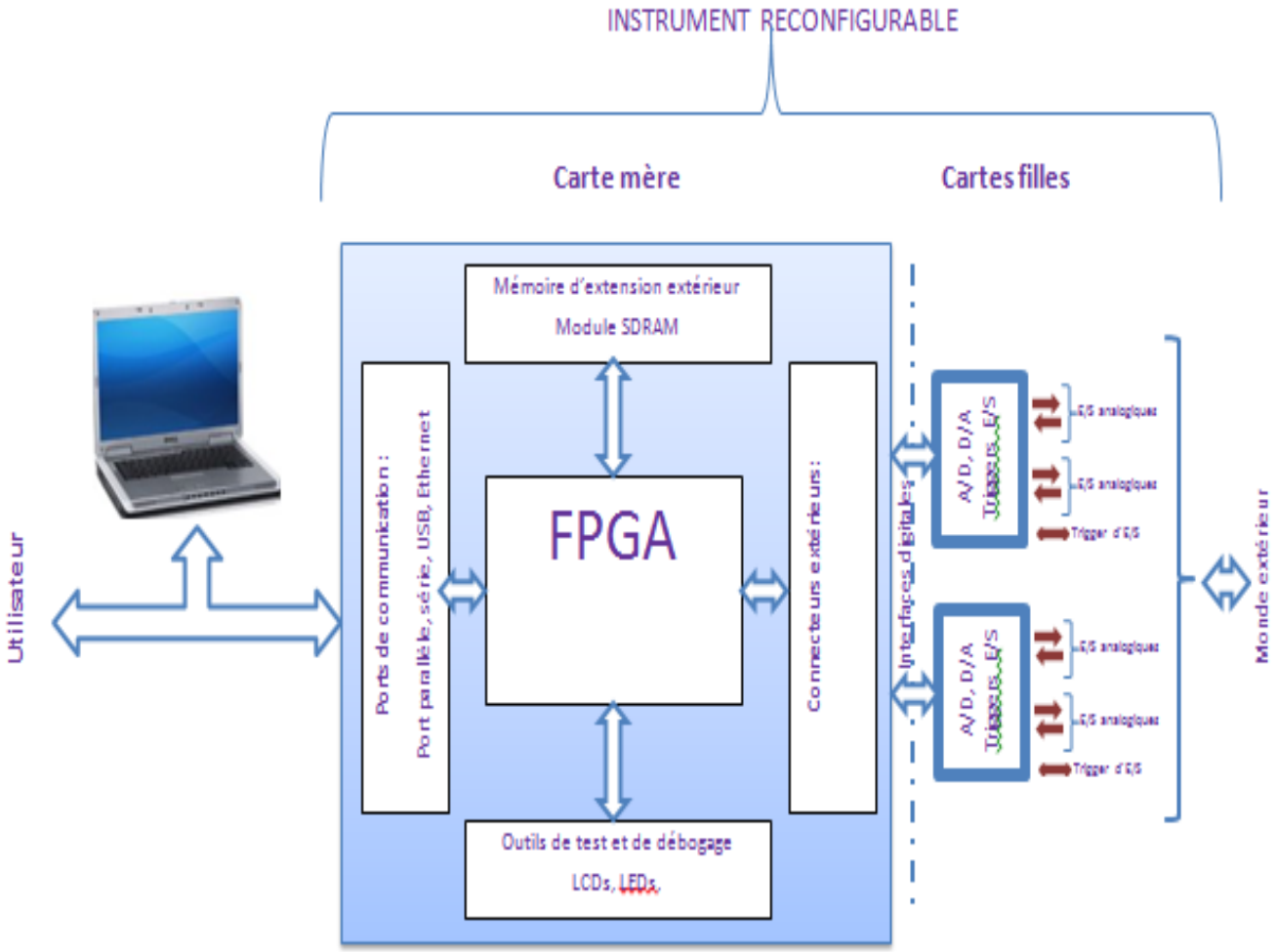


Figure 1.2 : Architecture d'un système RVI **[3]**

I.2.2. L'instrument reconfigurable

L'instrument reconfigurable a une carte mère (la principale carte mère du RVI) et deux cartes filles. La carte mère est faite d'un FPGA à capacité moyenne ou grande et quatre blocs : le bloc de communication de port, extension mémoire, outils de débogage, et d'autres outils variés.

Le logiciel est une collection des modules indépendants organisés hiérarchiquement. Chaque module opère sur des fichiers indépendants contenant les informations relevées : données, commandes, messages d'erreur. Dans le système RVI, les opérations gourmandes en temps d'exécution sont à la charge de l'instrument reconfigurable tandis que les autres opérations sont exécutées par le logiciel.

Généralement, le logiciel fournit :

- Une interface graphique du RVI
- Une librairie des instruments virtuels avec ses propres interfaces.
- Des possibilités de stockage des données
- L'établissement de la communication automatique avec l'instrument virtuelle à chaque fois que l'utilisateur choisit un instrument (driver).

Le code de la description matérielle est écrit en langage VHDL. Une description matérielle est divisée en quatre blocs de base :

- Un module de communication ;
- Une mémoire et des registres pour communiquer avec les cores ;
- Les cores de l'instrument;
- Une description des utilisations des ports.

Optionnellement, le système peut disposer des modules SDRAM extérieurs, des circuits logiques analyseurs et générateur stimuli pour déboguer l'exécution.

Le module de communication par port supporte les protocoles de base de connexion à un protocole haut niveau RVI spécifique pour manier l'instrument virtuel. La communication entre le PC et le core de l'instrument s'effectue de deux manières :

- La première est par lecture et écriture des données dans la mémoire dual port. Le PC accède la mémoire par un port et le core de l'instrument par l'autre port.

- La deuxième manière est par lecture et écriture dans les registres. Certains registres sont en lecture seule pour le core de l'instrument et certains autres en lecture seule pour le PC. Quelques registres sont réservés pour contrôler le protocole de communication de l'RVI, donc ne sont pas disponibles pour des usages généraux.

Suivant la nature des ports et du core de l'instrument, l'instrument reconfigurable peut envoyer une demande d'interruption au PC pour déclencher une action particulière. Aussi le PC peut vérifier des registres spécifiques pour savoir s'il y a une requête pour une certaine action. Le PC peut aussi utiliser des registres pour passer des commandes et des paramètres aux cores de l'instrument. L'échange des données par registres peut être utilisé pour implémenter un mécanisme permettant à des accès simultanés de la mémoire dual port. Cet échange peut aussi être utilisé pour contrôler l'accès à la mémoire externe.

Afin d'intégrer un core d'un instrument reconfigurable dans un système RVI, ce core doit se conformer à l'interface standardisée du bloc de communication PC-FPGA et le bloc d'interfaçage des matériels extérieurs. Il doit aussi se conformer à un mécanisme commun d'interaction. Quand les trois blocs principaux (le bloc de communication PC-FPGA, le core de l'instrument, l'interface des matériels extérieurs) respectent ces conditions, alors chaque bloc pourra être modifié ou mis à jour indépendamment et pourra être réutilisé dans des contextes différents. Les interfaces et le mécanisme d'interaction dépendent de la complexité et la performance du système entier.

Afin de définir un standard pour un système RVI basé sur le FPGA, plusieurs facteurs importants doivent être considérés :

- Les évolutions estimées des circuits FPGA en termes et capacités, performance et des ressources spéciales.

- Les matériels extérieurs comme les cartes filles, CAN, CNA, câbles, connecteurs

- Les communications possible avec le PC (port parallèle, port série, USB, Ethernet.)

- Les outils logiciels : synthétiseurs, outils de programmation des FPGA, les outils de simulation et débogage, des outils Ad Hoc.

- La quantité et qualité des instruments cibles.
- La portabilité, la maintenabilité du système et/ou tous ces sous-systèmes (logiciel, système d'exploitation, port des PC, familles d'FPGA, des matériels externes)

Les interactions entre les différents composants du système RVI sont présentées à la **Figure 1.3**.

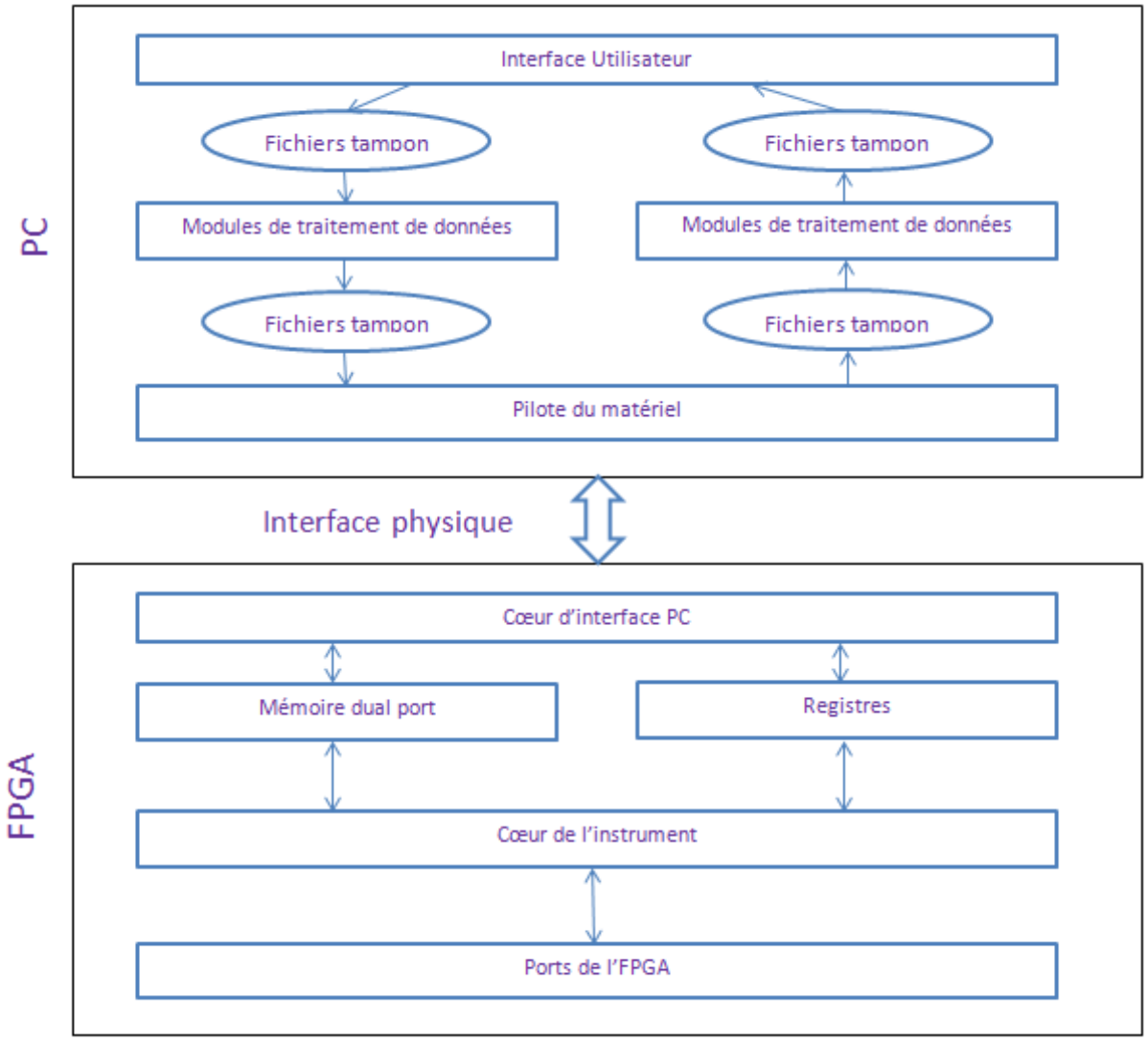


Figure 1.3 : Architecture générale de la communication de l'ensemble du système RVI **[3]**

La possibilité de reconfiguration de le FPGA pour le système RVI pour exécuter une action particulière et des algorithmes en cas de demande à de haute performance ouvre la possibilité de computation reconfigurable. L'instrument reconfigurable peut être vu comme

un coprocesseur parallèle du PC ce qui accélère l'exécution des tâches gourmandes en temps d'exécution et les contrôles matériels en temps réel. Certaines applications d'acquisition de signaux nécessitent de traiter les échantillons en temps réel, c'est-à-dire en cours d'acquisition, afin de réduire immédiatement après conversion analogique-numérique le flot de données à transmettre. Il s'agit de soulager la charge de calcul du processeur central en lui communiquant une information déjà prétraitée, beaucoup moins volumineuse.

I.3. Introduction aux circuits FPGA

On peut distinguer deux types de FPGA :

a) Les FPGA de type RAM

Lancé sur le marché en 1984 par la firme XILINX, le FPGA est un circuit prédiffusé programmable. Le concept du FPGA est basé sur l'utilisation d'un multiplexeur comme élément combinatoire de la cellule de base. La **Figure 1.4** suivante représente la cellule type de base d'un FPGA. Elle comprend un multiplexeur 8 vers 1 permettant de réaliser n'importe quelle fonction logique combinatoire de 4 variables (appelé LUT : Look Up Table ou encore générateur de fonction). La bascule D permet la réalisation de fonctions logiques séquentielles. La configuration du multiplexeur 2 vers 1 de sortie autorise la sélection des deux types de fonction **[4]**.

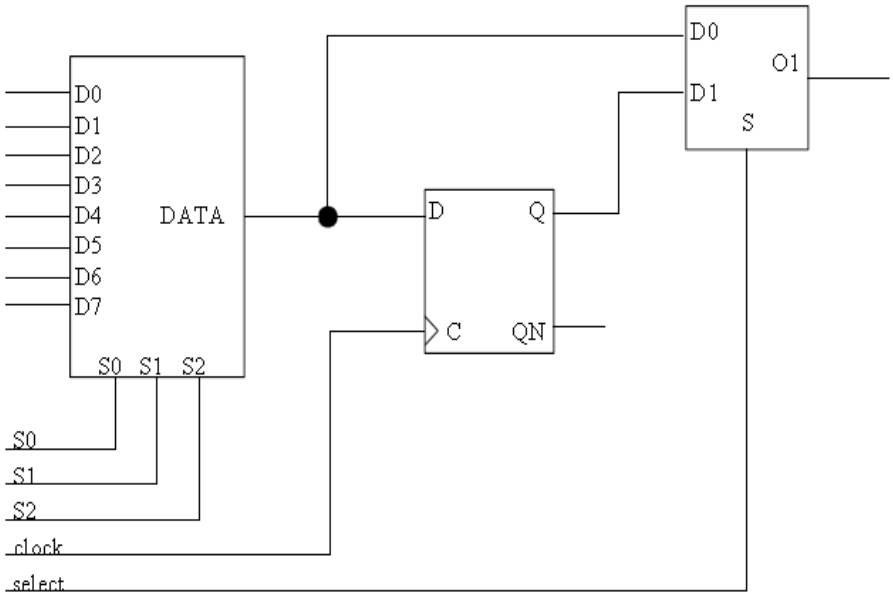


Figure 1.4 : cellule type de base d'un FPGA

Les cellules de base d'un FPGA sont disposées en rangées et en colonnes. Des lignes d'interconnexions programmables traversent le circuit, horizontalement et verticalement, entre les diverses cellules. Ces lignes d'interconnexions permettent de relier les cellules entre elles, et avec les plots d'entrées/sorties. Les connexions programmables sur ces lignes sont réalisées par des transistors MOS dont l'état est contrôlé par des cellules mémoires SRAM. Ainsi, toute la configuration d'un FPGA est contenue dans des cellules SRAM.

Tous les FPGA sont fabriqués en technologie CMOS, les plus gros d'entre eux intègrent des millions de portes logiques utilisables. Il faut noter que la surface de silicium d'un FPGA est utilisée au 2/3 pour les interconnexions et au 1/3 pour les fonctions logiques. Le taux d'utilisation global des ressources ne dépasse pas 80 %.

b) Les FPGA à anti-fusibles

Commercialisés à partir de 1990, ce FPGA, programmable une seule fois, est basé sur la technologie des interconnexions à anti-fusibles. Sa structure dispose de cellules élémentaires organisées en rangées et en colonnes. Les lignes d'interconnexions programmables traversent le circuit, horizontalement et verticalement, entre les diverses cellules. La technologie à anti-fusibles permet de réduire considérablement la surface prise par les interconnexions programmables, par rapport aux interconnexions à base de SRAM. La cellule élémentaire diffère d'un fabricant à un autre, mais elle est généralement composée de quelques portes logiques. Le nombre de ces cellules est généralement très important.

Alors que le FPGA SRAM est utilisé pour des prototypes ou des petites séries, le FPGA à anti-fusibles est destiné pour des plus grandes séries, en raison de son coût de fabrication moins élevé. Il est généralement conçu avec des outils de synthèse de type VHDL que nous allons voir dans le chapitre suivant.

c) Historique

Les avances récentes dans les FPGA a rendu possible la conception de plusieurs systèmes d'émulation de matériels complexes. Cette technologie promet des nouveaux niveaux d'intégration du système sur un seul FPGA, mais elle présente aussi des défis considérables aux « designers » ou concepteurs. La toute dernière avance technologique sur les FPGA étant le « Reconfigurable Virtual Instrumentation (RVI) » ou instrumentation virtuelle reconfigurable que nous avons déjà mentionnée précédemment. Son but est de fournir une plate-forme de matériel/logiciel réutilisable et à bas-prix pour l'émulation de multiples systèmes de l'instrumentation électroniques et scientifiques. Avec ses évolutions

architecturales récentes, les FPGA représentent aujourd'hui le segment croissant le plus rapide sur le marché des circuits logiques programmables. Un des attributs fondamentaux des FPGA est sa flexibilité. Il existe aussi des avantages supplémentaires tel que le fait que la reprogrammation dans le circuit est quasiment illimitée, le cycle du design est rapide, les outils du design sont presque libres, et l'ingénieur peut faire des systèmes FPGA exceptionnels mais à bas prix basés sur une solution de plus en plus attirante pour évaluer et rendre effectif des architectures de design alternatives; donc accélérer le processus du design et le temps pour vendre de nouveaux produits. De nos jours, la technologie FPGA est arrivée à un niveau de maturité et de popularité que ce travail est en partie supporté par « Actel Corp » en prévoyant et en donnant naissance à une plate-forme générale pour la création d'une bibliothèque « open core, open source » de composants modulaires pour l'instrumentation virtuelle reconfigurable.

La plate-forme est basée essentiellement sur les appareils FPGA et les ordinateurs personnels standards. Excepté l'autorisation d'une évaluation rapide de nouvelle méthode d'instrumentation, une telle plate-forme est un intérêt particulier à la communauté académique et scientifique, comme il représente une solution pédagogique à bas-prix pour les universités et les institutions de recherche.

I.4. Le logiciel

Etant donné qu'instrument virtuel est une combinaison d'un architecture matériel et logiciel, une conception logicielle doit être mise en œuvre en parallèle avec la conception matérielle. Les rôles du logiciel sont essentiellement les suivants :

- Présenter une interface utilisateur informant l'utilisateur de l'état de l'instrument
- Traiter des données relatives à l'instrument reconfigurable
- Interchanger des données entre l'utilisateur et l'instrument reconfigurable. Interfacer l'instrument reconfigurable par un moyen d'entrée (USB, Ethernet, PCI)

Interface utilisateur

Le rôle de l'interface utilisateur est d'afficher diverses informations selon la spécificité de l'instrument, mais aussi de prendre compte des commandes et des réglages de l'utilisateur lui permettant ainsi de piloter l'instrument.

Traitement des données

Afin d'afficher correctement les informations (nombres, courbes) selon l'état de l'instrument et les paramètres ajustés par l'utilisateur, le logiciel doit intégrer des traitements de ces données en tenant compte des divers paramètres. Un autre rôle du traitement de donnée est d'interpréter les commandes de l'utilisateur et les communiquer à l'instrument reconfigurable.

Interfaçage de l'instrument reconfigurable

Afin d'interchanger les données entre l'application logicielle et l'instrument reconfigurable, un module doit assurer les échanges des données via un port d'entrée/sortie

L'interaction entre ces différentes couches de traitement du logiciel est représentée à la **Figure 1.5**

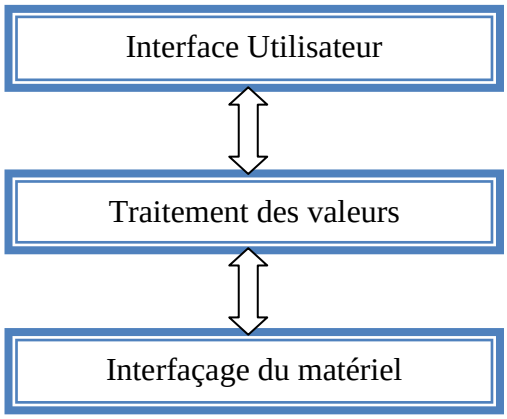


Figure 1.5 : Vue globale des différents modules du logiciel

CHAPITRE II. LA PROGRAMMATION MATERIELLE

D'après ce qui précède, nous connaissons déjà la technologie des circuits FPGA et leurs performances en matière d'instrumentation virtuelle. Maintenant, nous allons présenter les outils et moyens permettant le développement et la réalisation de différentes implémentations sur les systèmes reconfigurables type FPGA. Pour cela, commençons par apprendre la conception sur la base d'un langage spécifique à des fins de programmation de composants FPGA, puis en se basant sur des outils de conception graphique et textuel convenables aux circuits FPGA que nous avons sur les mains.

II.1. Les langages de description matérielle

II.1.1. Généralités

Auparavant pour décrire le fonctionnement d'un circuit électronique programmable les techniciens et les ingénieurs utilisaient des langages de bas niveau (**ABEL**, **PALASM**, **ORCAD/PLD**) ou plus simplement un outil de saisie de schémas.

Actuellement la densité de fonctions logiques (portes et bascules) intégrée dans les FPGAs est telle (plusieurs milliers de portes voire millions de portes) qu'il n'est plus possible d'utiliser les outils d'hier pour développer les circuits d'aujourd'hui.

Les sociétés de développement et les ingénieurs ont voulu s'affranchir des contraintes technologiques des circuits. Ils ont donc créé des langages dits de haut niveau à savoir **VHDL**« **V**ery high speed integrated circuits **H**ardware **D**escription **L**angage » et **VERILOG**. Ces deux langages font abstraction des contraintes de technologies des circuits FPGAs. Ils permettent au code écrit d'être portable, c'est à dire qu'une description écrite pour un circuit peut être facilement utilisée pour un autre circuit. Ces langages dits de haut niveau permettent de matérialiser les structures électroniques d'un circuit. En effet les instructions écrites dans ces langages se traduisent par une configuration logique de portes et de bascules qui est intégrée à l'intérieur des circuits FPGAs. Dans ce travail de mémoire, nous nous intéressons seulement à VHDL et aux fonctionnalités de base de celui-ci lors des phases de conception ou synthèse des FPGAs.

II.1.2. VHDL et normalisation

Développé dans les années 80 aux États-Unis, le langage de description VHDL est ensuite devenu une norme **IEEE** (**I**nstitut of **E**lectrical and **E**lectronics **E**ngineers) numéro 1076 en 1987. Révisée en 1993 pour supprimer quelques ambiguïtés et améliorer la portabilité

du langage, le langage VHDL est un standard IEEE (IEEE 1076-1993) pour la modélisation, la simulation et la synthèse de systèmes matériels logiques **[5]**. La norme qui définit la syntaxe et les possibilités offertes par le langage de description VHDL est très ouverte. Il est par exemple possible de spécifier les temps de propagations et de transitions des signaux d'une fonction logique, c'est à dire créer une description VHDL du système que l'on souhaite obtenir en imposant des temps précis de propagation et de transition. Or les outils actuels de synthèses logiques sont incapables de réaliser une fonction avec de telles contraintes. Seuls des modèles théoriques de simulations peuvent être créés en utilisant toutes les possibilités du langage.La situation peut donc se résumer de la façon présentée dans les **Figures .2.1 et 2.2**.

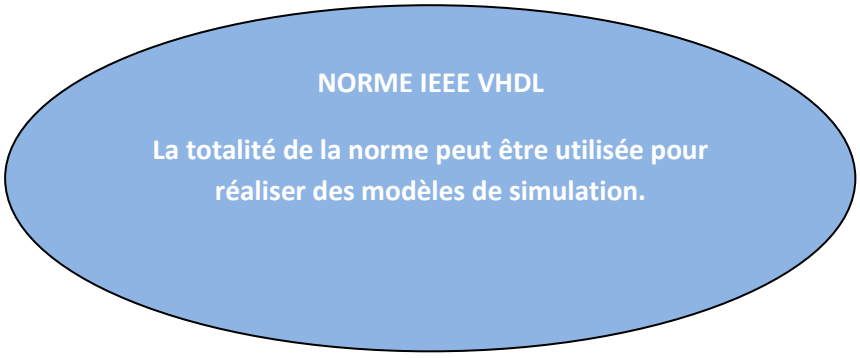


Figure 2.1 : création des modèles de simulation

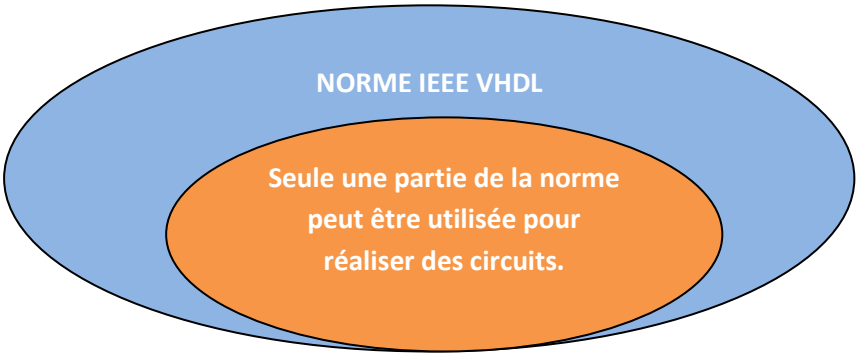


Figure 2.2 : création des circuits programmables

II.1.3. Relation entre une description VHDL et les circuits logiques programmables comme les circuits FPGAs

L'implantation d'une ou de plusieurs descriptions **VHDL** dans un FPGA va dépendre de l'affectation que l'on fera des broches d'entrées / sorties et des structures de base du circuit logique programmable. Cette affectation peut se faire de plusieurs manières:

– *L'affectation automatique*

On laisse le synthétiseur propre au fondeur du circuit implanter la structure correspondante à la description **VHDL**. Les numéros de broches seront choisis de façon automatique.

– *L'affectation manuelle*

On définit les numéros de broches dans la description **VHDL** ou sur un schéma bloc définissant les liaisons entre les différents blocs **VHDL** ou dans un fichier texte propre au fondeur. Les numéros de broches seront affectés suivant les consignes données.

II.1.4. Utilité du VHDL

Le VHDL est un langage de spécification, de simulation et également de conception. Contrairement à d'autres langages comme ABEL qui se trouvaient être en premier lieu des langages de conception, VHDL est d'abord un langage de spécification. La normalisation a d'abord eu lieu pour la spécification et la simulation (1987) et ensuite pour la synthèse (1993). Cette notion est relativement importante pour comprendre le fonctionnement du langage et son évolution. Grâce à la normalisation, on peut être certain qu'un système décrit en VHDL standard est lisible quel que soit le fabricant de circuits.

a) Spécification

Le VHDL est un langage de spécification. C'est dans ce domaine que la norme est actuellement la mieux établie. Il est tout à fait possible de décrire un circuit en un VHDL standard pour qu'il soit lisible de tous. Certains fabricants (de circuits ou de CAO : Conception Assistée par Ordinateur) adaptent ce langage pour donner à l'utilisateur quelques facilités supplémentaires, au détriment de la portabilité du code **[6]**. Heureusement, il y a une nette tendance de la part des fabricants à revoir leurs positions et à uniformiser le VHDL. Il est donc probable que l'on s'approche d'un vrai standard VHDL et non plus d'un standard théorique. Il y aura toujours des ajouts de la part des fabricants, mais il ne s'agira plus d'une modification du langage, mais de macros offertes à l'utilisateur pour optimiser le code VHDL en fonction du circuit cible (en vue de la synthèse).

b) Simulation

Le VHDL est également un langage de simulation. Pour ce faire, la notion de temps sous différentes formes y a été introduite. Des modules, destinés uniquement à la simulation, peuvent ainsi être créés et utilisés pour valider un fonctionnement logique ou temporel du code VHDL. La possibilité de simuler avec des programmes VHDL devrait considérablement faciliter l'écriture de tests avant la programmation du circuit et éviter ainsi de nombreux essais sur un prototype qui sont beaucoup plus coûteux et dont les erreurs sont plus difficiles à trouver. Bien que la simulation offre de grandes facilités de test, il est toujours nécessaire de concevoir les circuits en vue des tests de fabrication, c'est-à-dire en permettant l'accès à certains signaux internes.

c) Conception

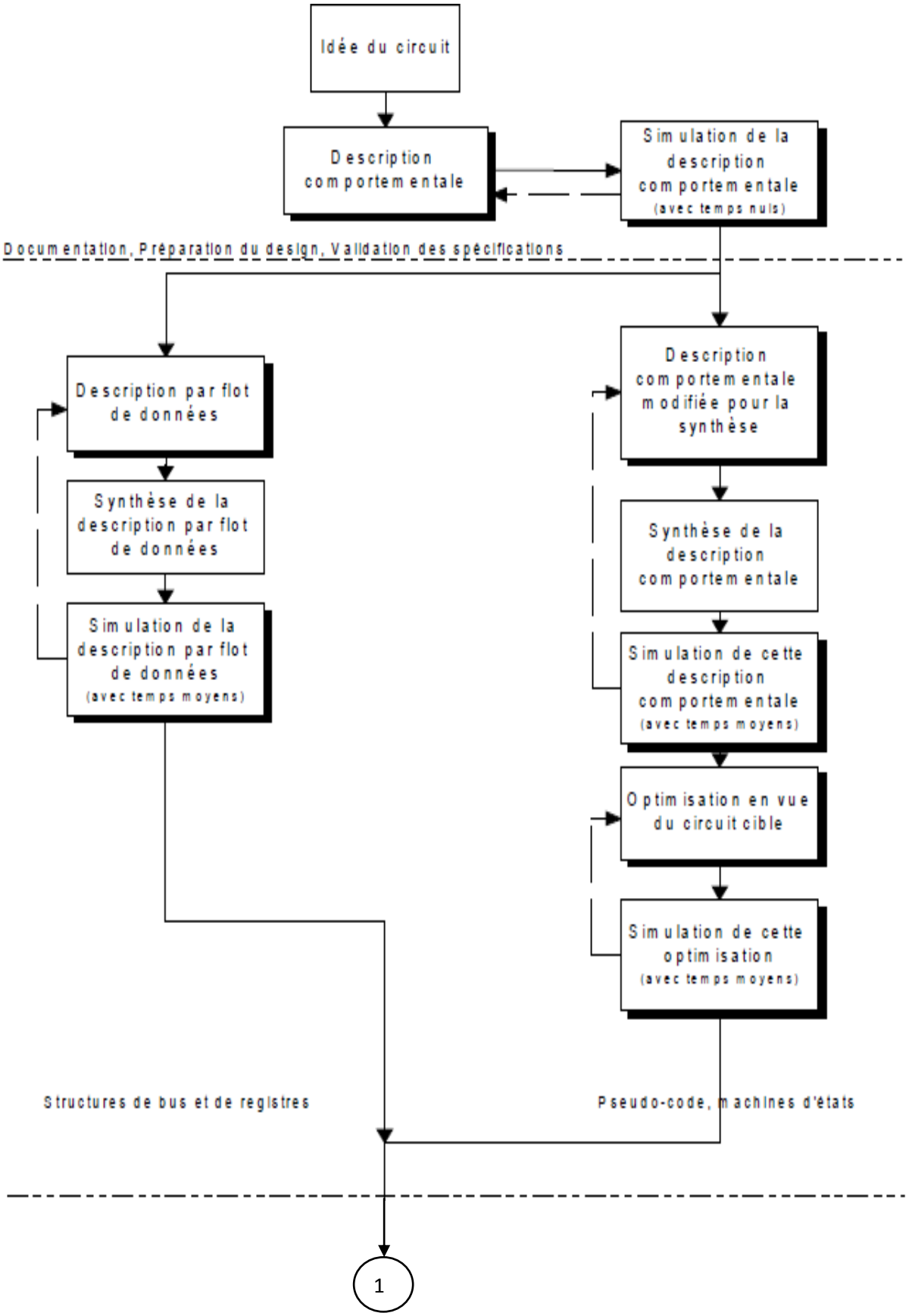
Le VHDL permet la conception de circuits avec une grande quantité de portes. C'est cette utilité de VHDL qui nous concerne le plus dans ce projet de mémoire. Il est à noter que les circuits FPGA actuels comprennent entre 500 et 1 000 000 portes et que ce nombre augmente très rapidement, comme dans notre conception d'instruments virtuels, nous utilisons un circuit FPGA d'Actel comprenant 1 500 000 portes. L'avantage d'un langage tel que celui-ci par rapport aux langages précédents de conception matérielle est comparable à l'avantage d'un langage informatique de haut niveau (C, Pascal, Ada) vis-à-vis de l'assembleur. Ce qui veut dire que malgré l'évolution fulgurante de la taille des circuits, la longueur du code VHDL n'a pas suivi la même courbe. Le VHDL bien que facilement accessible dans ses bases, peut devenir extrêmement compliqué s'il s'agit d'optimiser le code pour une architecture de circuit. C'est pour cette raison que de plus en plus de fabricants offrent des macros, gratuites pour les fonctions sans grandes difficultés et payantes pour les autres. Donc avant de concevoir un processeur RISC (Reduced Instruction Set Computer), une ALU (Arithmetic Logic Unit), une interface PCI (Peripheral Component Interconnect) ou d'autres éléments de cette complexité, il peut être judicieux de choisir un circuit cible en fonction des besoins et d'acheter la macro offerte par le constructeur. Dans notre cas d'instrumentation virtuelle, nous avons choisi le circuit FPGA d'Actel (déjà mentionné précédemment) mais nous n'avons pas acheté chez eux les macros qui nous conviennent puisqu'il y a aussi des versions gratuites destinées aux recherches pédagogiques. Il est donc bien évident qu'il faudra évaluer les besoins (performance du code nécessaire, quantité d'instruments à produire) et le coût d'une telle macro.

II.2. Méthodologie de développement matériel

II.2.1. Analyse des besoins

Lorsqu'on commence un développement matériel, il est utile de se demander de quelle manière celui-ci sera réalisé. Faut-il attaquer les problèmes les uns après les autres ou en parallèle ? Quel type de description faut-il utiliser pour telle ou telle partie du circuit à concevoir ? Et il y en a d'autres encore. Ce chapitre tente d'y répondre par une méthode de travail. Celle-ci minimisera les erreurs de développement en VHDL et facilitera la compréhension du code VHDL écrit par quelqu'un d'autre (pour autant que ce quelqu'un ait suivi cette méthode).

Ensuite, la prochaine étape de développement matériel sera la spécification et la synthèse. Mais avant d'entamer ce paragraphe, l'organigramme représenté par la **Figure 2.3** nous donne trois informations nécessaires à notre phase de conception matérielle : les diverses étapes du projet, puis l'ordre qu'on va les traiter et enfin le type de structure VHDL qu'il faut employer pour les différentes parties du développement.


Idée du circuit
Description comportementale
Simulation de la description comportementale (avec temps nuls)
Documentation, Préparation du design, Validation des spécifications
Description par flot de données
Synthèse de la description par flot de données
Simulation de la description par flot de données (avec temps moyens)
Description comportementale modifiée pour la synthèse
Synthèse de la description comportementale
Simulation de cette description comportementale (avec temps moyens)
Optimisation en vue du circuit cible
Simulation de cette optimisation (avec temps moyens)
Structures de bus et de registres
Pseudo-code, machines d'états
1

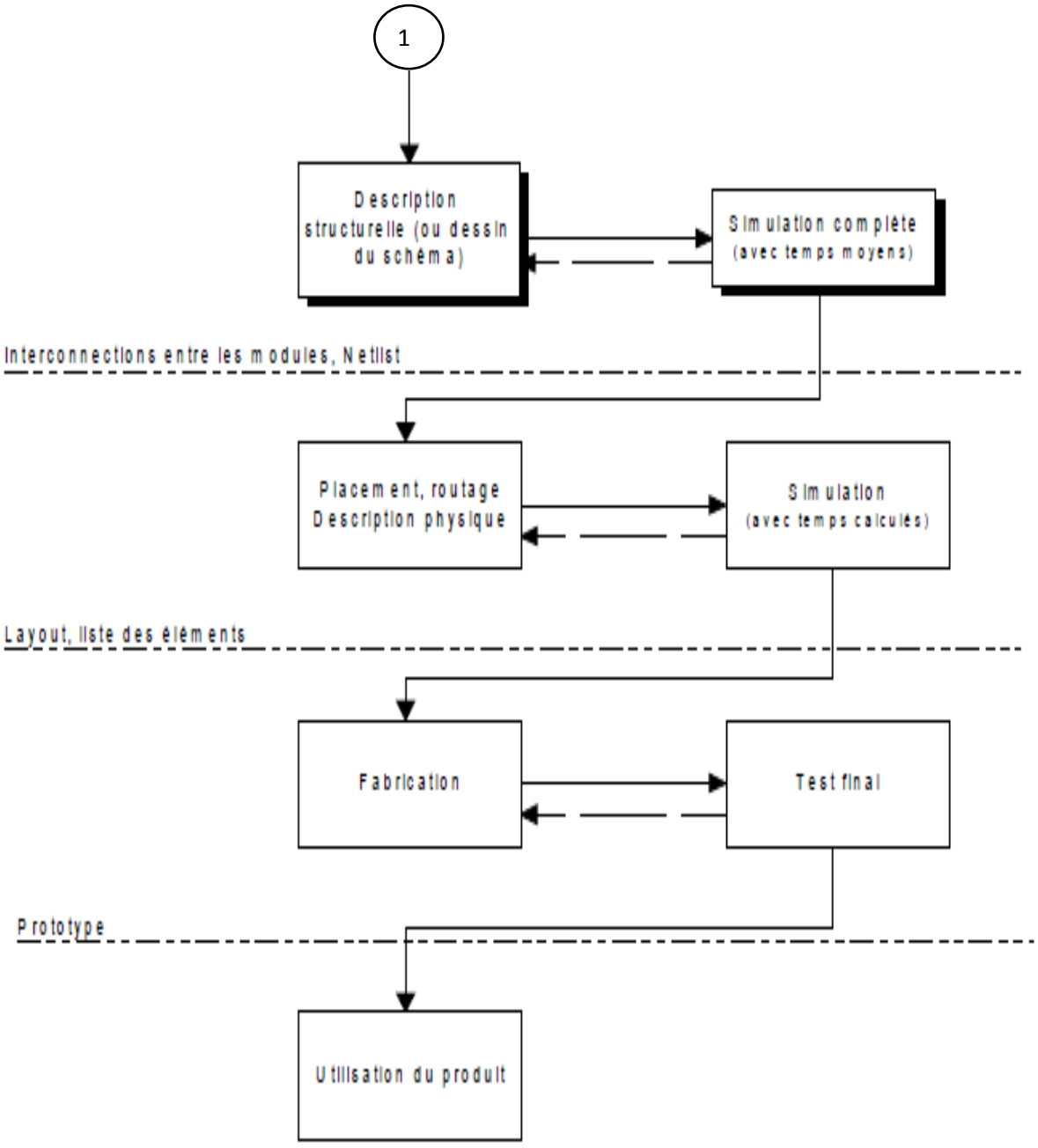


Figure 2.3 : Organigramme représentant le développement matériel

Pour faciliter la compréhension de cet organigramme, tout ce qui s'écrit en VHDL se trouve dans des cases ombrées.

Un développement matériel se décompose donc de la manière suivante :

- Une première étape consiste à préparer la documentation. Celle-ci, écrite en VHDL, permet aussi de valider les spécifications avant de poursuivre le développement.
- L'étape suivante consiste à développer conjointement deux parties du système. Il s'agit de la description des bus et registres (généralement pour une description du type flot de données) et de la description de pseudo-code ou des machines d'état (en général à l'aide d'une description comportementale). La simulation valide ces descriptions.
- Un assemblage du tout est alors réalisé par une description structurelle et l'ensemble est aussi simulé.
- Les dernières étapes sont pour le placement-routage. La simulation de celui-ci et la fabrication du produit ou de l'instrument.

II.2.2. Spécification et synthèse

Deux étapes particulières apparaissent dans la **Figure 2.3**, celle de la spécification et de la synthèse. En VHDL, il convient de bien distinguer ces deux étapes, car le code écrit pour l'une ou l'autre n'est pour l'instant pas complètement identique.

La spécification est la partie d'un développement qui consiste à valider par la simulation ce qui a été demandé par le mandataire. Elle permet de corriger un cahier des charges incorrect ou de compléter celui-ci. La spécification en VHDL permet de créer une sorte de maquette qui a, vu de l'extérieur, le comportement du système désiré.

La synthèse permet de gérer automatiquement à partir du code VHDL un schéma de câblage permettant la programmation du circuit cible. La description du code pour une synthèse est, en théorie, indépendante de l'architecture du circuit. En pratique, le style utilisé aura une influence sur le résultat de la synthèse, influence liée au type de circuit et au synthétiseur utilisé.

II.2.3. Interface VHDL et codage

Avant d'écrire un programme en VHDL, il faut définir l'interface entre l'intérieur du circuit et le monde extérieur comme présenté à la **Figure 2.4**. Il ne suffit pas d'avoir un programme qui décrit un certain comportement. Ce comportement réagit selon des entrées et

agit sur des sorties (ou des entrées-sorties) et tous ces ports seront connectés à une carte externe au circuit. Toutes ces connexions correspondent à cette interface.

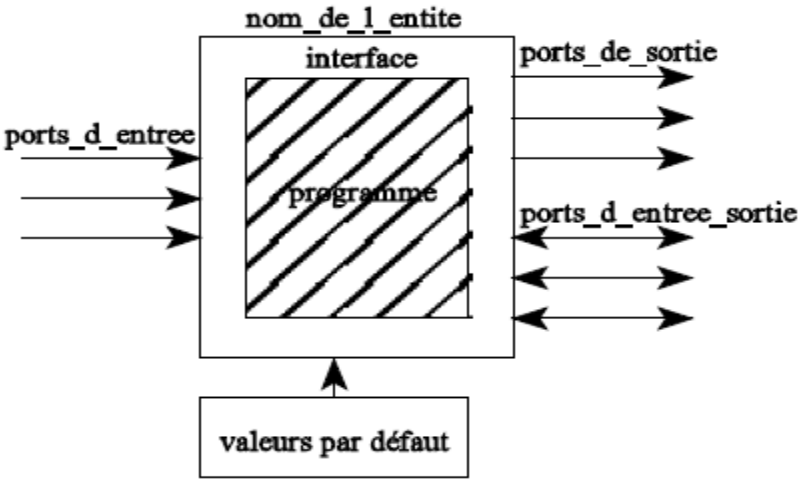


Figure 2.4 : Interface VHDL

La **Figure 2.4** nous montre qu'il y a un certain nombre d'entrées, de sorties et d'entrées-sorties, de plus quelques valeurs par défaut qui sont déjà connues comme par exemple le délai typique d'une porte logique simple. A ce moment de la conception, nous ne connaissons pas encore ce que nous allons programmer. Nous possédons juste ces informations et le nom du module que nous devons créer (**nom_de_l_entite**). La structure générale d'un programme VHDL est présentée en **ANNEXE II**.

II.2.4. Simulation

Pour être efficace, une simulation doit être la plus complète possible, sans être redondante. Une bonne simulation épurée de tous les tests doublés permettra une bonne économie tant pour le temps consacré à cette simulation que sur les tests sur circuits. De plus, le fichier écrit pour la simulation peut être avantageusement utilisé pour tous les tests ultérieurs.

Un système de test, fournissant les stimulis et vérifiant les réponses, peut aussi être écrit en VHDL. Un module de simulation (test bench) sera créé et à l'intérieur de celui-ci nous intégrons le module à tester.

II.3. Les outils de développement matériel

II.3.1. Les principaux outils sur le marché

Avant de savoir les outils que nous avons utilisés dans cette phase de développement matériel de la conception d'instruments virtuels, décrivons successivement les principaux outils présents sur le marché. Notons que les noms en caractères gras qui suivent représentent des noms d'entreprises de semi-conducteurs les plus connus à travers le monde, et que les noms en italique correspondent à leurs propres outils de développement :

- **Actel:***Libero* (environnement de conception intégré), *SmartTime* (analyse des délais), *SmartPower* (analyse de la consommation), *SmartGen*,.
- **Altera:***Quartus II* (environnement de conception intégré), *SOPC Builder* (développement de systèmes), *DSP Builder*, *TimeQuest* (analyse des délais), *PowerPlay* (analyse de la consommation),
- **Lattice:***ispLever* (environnement de conception intégré), *ispLeverDSP*,
- **Mentor Graphics:***Precision RTL* (synthèse logique), *Precision Physical* (synthèse physique), *ModelSim* (simulation numérique), *CatapultC* (synthèse de haut niveau),.
- **Synplicity:***Synplify Pro* (synthèse logique), *Synplify Premier* (synthèse physique), *Synplify DSP*, *Certify* (prototypage des circuits spécifiques), *Identify* (instrumentation embarquée et débogage),
- **Xilinx:***ISE* (environnement de conception intégré), *System Generator for DSP et AccelDSP*, *ChipScope Pro* (débogage embarqué), *PlanHead*,

En effet, nous remarquons que chaque entreprise de semi-conducteurs a ses propres outils. Pourtant, la plupart de ces entreprises se coopèrent afin de répondre aux besoins de leurs clients.

II.3.2. L'environnement intégré de conception Actel Libero

Actel est une entreprise américaine de semi-conducteurs, elle fait partie des plus grandes entreprises spécialisées dans le développement et la commercialisation de composants logiques programmables, et des services associés tels que les logiciels de CAO électroniques, les blocs de propriété intellectuelle réutilisables ou « IP Cores » (Intellectual Property Cores) **[6]**. Le logiciel Libero IDE (*Integrated Design Environment*) est un logiciel complet et puissant en matière de développement des FPGAs, développée par Actel. Dans ce projet de mémoire, nous avons utilisés l'environnement intégré de développement «Actel

Libero IDE» fonctionnant sous Windows et une carte FPGA produite par cette société dont la description sera détaillée ci-après. En fait, Libero IDE nous permet de programmer ce FPGA, en d'autres termes de concevoir, simuler, synthétiser, placer-router et valider facilement des blocs décrits en VHDL (ou autre langage) dans un seul environnement.

La **Figure 2.5** suivante représente un extrait de l'interface du flot de conception d'un projet dans Actel Libero appelé *« Project Manager »*:

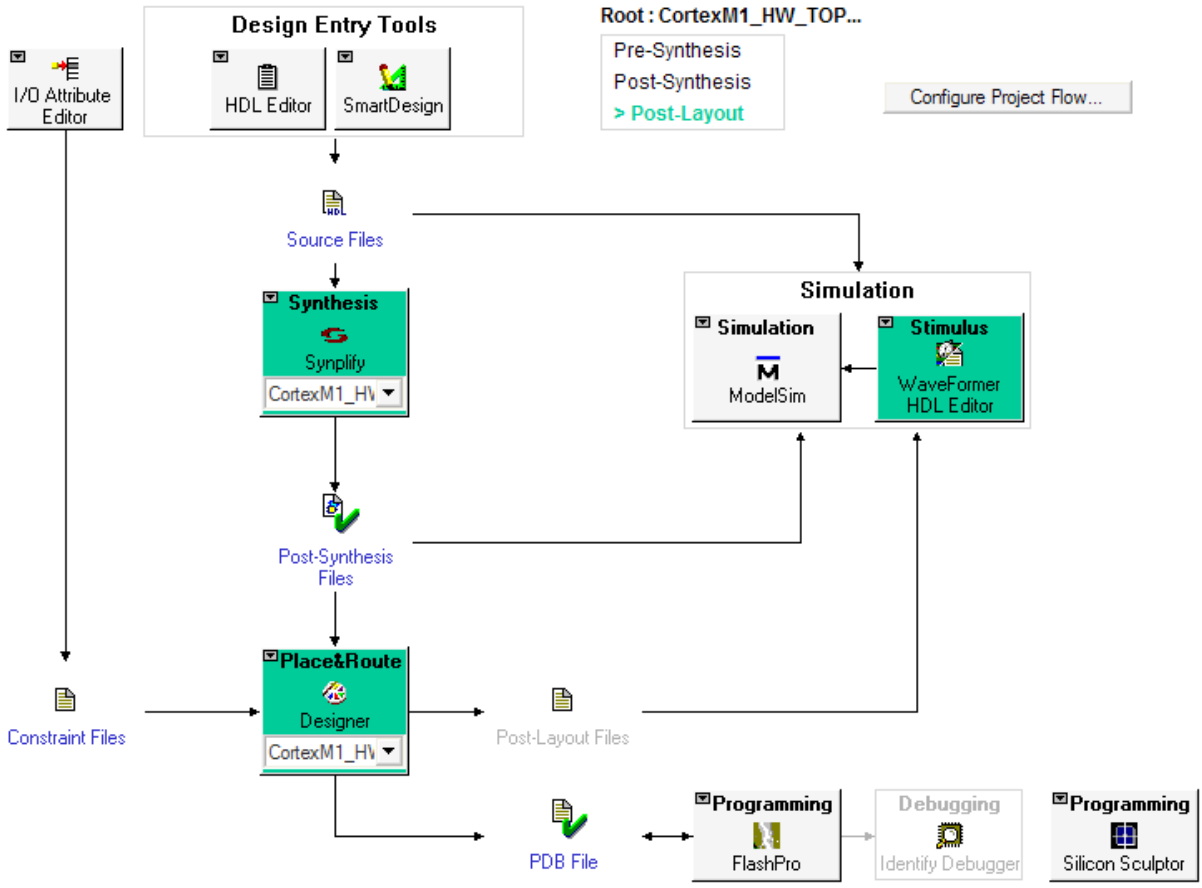


Figure 2.5 : Extrait de l'interface du flot de conception d'un projet sous Libero IDE **[7]**

a) Le flot de conception de Libero IDE

Le flot de conception de Libero IDE comprend six étapes:

– **Première étape :** « ***Design Creation*** **»** ou Création du Design

Ici, nous pouvons avoir deux modes de création de design : soit en utilisant un éditeur de texte HDL avec l'outil *« HDL Editor »* pour le langage VHDL que nous avons choisi, c'est l'approche de conception HDL, soit en utilisant un éditeur graphique comme

*« SmartDesign »*qui nous donne un canevas permettant d'instancier des *« IP cores »,* c'est l'approche de conception par schéma. Cette étape concerne la partie *« Design Entry Tools »* présentée dans la Figure précédente.

- **Deuxème étape :** ***« Design Verification - Functional Simulation »*** ouVérification du Design - Simulation fonctionnelle

Après que nous ayons défini notre design, nous pouvons créer un *« testbench »* ou banc d'essai en utilisant *« WaveFormer »*. Un banc d'essai est un fichier VHDL non synthétisable utilisé comme niveau hiérarchique supérieur. Il permet d'instancier les modules VHDL à tester et de leur assigner des valeurs de signaux d'entrée. Le banc d'essai peut aussi déclarer et instancier d'autres composantes qui ne seront pas synthétisées à l'intérieur du FPGA. Il suffit de simuler le banc d'essai avec *« ModelSim »* pour tester le module : c'est la simulation fonctionnelle. L'outil *« ModelSim »* permet alors de vérifier une description matérielle VHDL avant et après la synthèse.

- **Troisième étape :** « ***Synthesis »*** ou synthèse

Cette étape correspond à la génération de la liste des interconnexions ou « *netlist* » à partir de la description matérielle à l'aide du synthétiseur logique *« Synplify »* de Synplicity.

- **Quatrième étape :** ***« Design Implementation »*** ou implémentation du design

Après que nous ayons vérifié fonctionnellement que notre design fonctionne, la prochaine étape est l'implémentation du design en utilisant le logiciel *«Actel Designer »*, c'est le placement et le routage ou ***« Place&Route »***. L'outil de placement et de routage « *Actel Designer »* génère le fichier de programmation ainsi que les fichiers nécessaires aux simulations avec délais.

- **Cinquième étape :** « ***Timing Simulation »*** ou simulation avec délais

Après que nous ayons faits l'implémentation du design, nous pouvons vérifier que notre design rencontre des spécifications de réglage. Après avoir créer un banc d'essai avec *« WaveFormer »*, utilisons *« ModelSim »* pour exécuter la simulation avec délais.

- **Sixième étape :** ***« Device Programming »*** ou programmation de la carte FPGA

Une fois que nous avons complété notre design, et que nous sommes satisfaits avec la simulation avec délais, créons maintenant notre fichier de programmation. Selon la famille de

notre carte FPGA, nous avons besoin de générer un fichier de programmation appelé *« Bitstream »*.

Résumons ces différentes étapes à l'aide de la **Figure2.6** suivante :

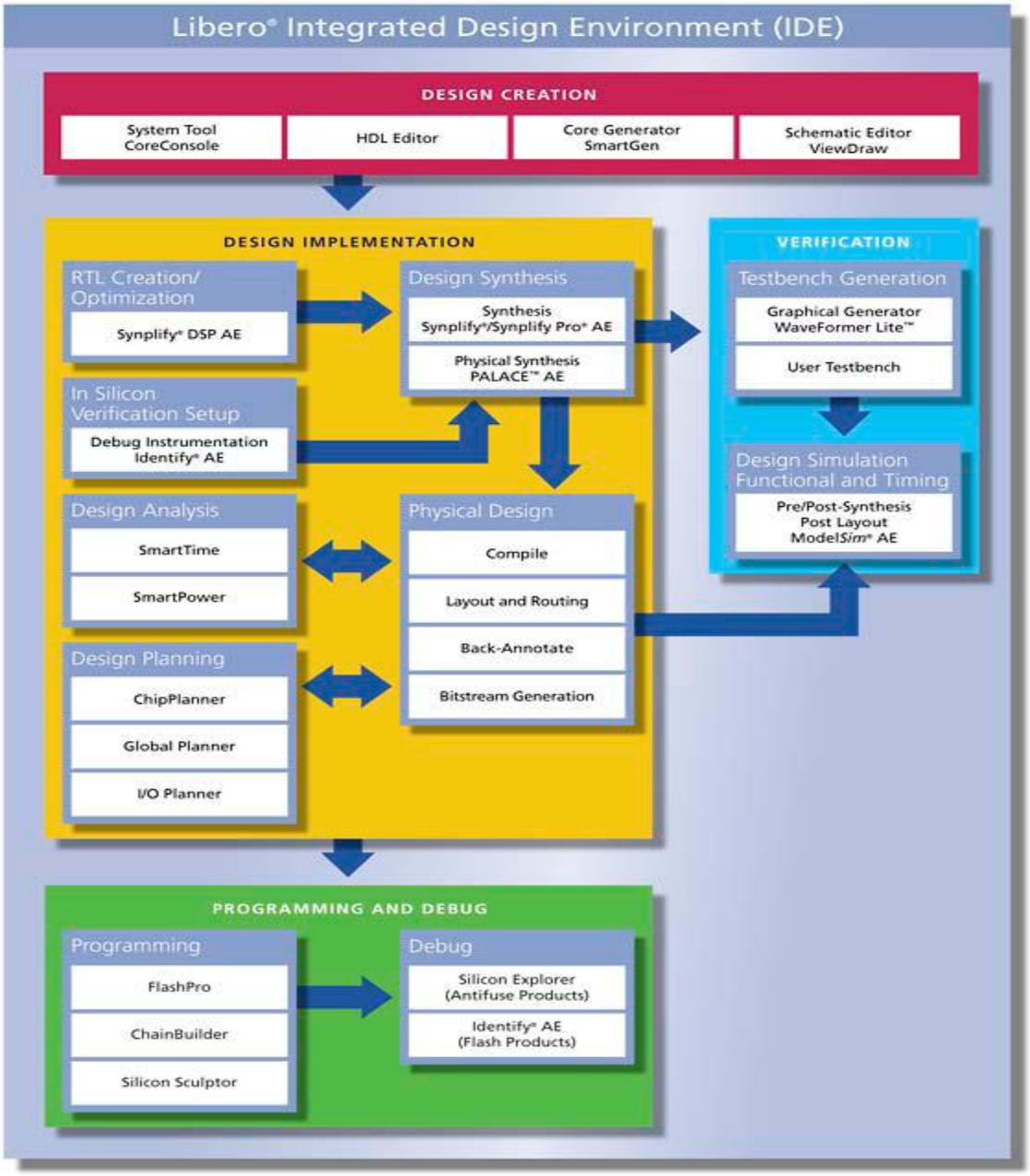


Figure 2.6: Flot de conception de Libero IDE **[7]**

b) Les types de fichiers rencontrés dans Libero IDE

Quand nous créons un projet dans le flot de conception de projet *« Project Manager »* de Libero IDE, il crée automatiquement de nouveaux répertoires et des fichiers de projet.

Selon nos préférences du projet et la version de Libero IDE que nous avons installé, « Project Manager » crée alors des répertoires pour notre projet. Le répertoire de niveau supérieur contient notre fichier projet *.prj qui est seulement le fichier *.prj permis pour chaque projet Libero IDE.

Maintenant, voyons ci-après dans le **tableau I** suivant les types de fichiers se trouvant dans les répertoires qui peuvent exister lors d'un projet sous Libero IDE :

Tableau I : Listes des fichiers rencontrés ou générés dans un projet sous Libero IDE**[8]**

Répertoire	Fichiers
component	*.sdb et *.cxf pour les composants de *« SmartDesign »*
constraint	tous les fichiers de contraintes *.sdc, *.pdc, *.gcf, *.dcf,
designer	*.adb, *.stp, *.prb, *.tcl, -_ba.sdf, _ba.v(hd), impl.prj_des, designer.log
hdl	tous les fichiers sources hdl *.vhd si VHDL, *.v et *.h si VERILOG
phy_synthesis	_palace.edn, _palace.gcf, palace_top.rpt
simulation	meminit.dat, modelsim.ini
smartgen	fichiers pour les « cores » générés *.gen et *.log
stimulus	fichiers stimulus *.btim et *.vhd
synthesis	*.edn, *_syn.prj, *.psp, *.srr, precision.log, exemplar.log, *.tcl
viewdraw	viewdraw.ini

II.3.3. La suite Soft Console IDE

L'environnement de conception intégré Libero IDE que nous avons vu précédemment comprend aussi un environnement de développement logiciel embarqué appelé « SoftConsole » qui est un logiciel basé sur le logiciel Eclipse. Puisqu'on a besoin d'intégrer un processeur dans le projet, alors pour cela, on utilise aussi la suite SoftConsole IDE d'Actel.

Nous utilisons ce logiciel dans ce projet d'instrumentation virtuelle pour charger un fichier exécutable écrit en langage C dans le programme mémoire d'un système processeur convenable à notre carte FPGA pour pouvoir déboguer ou exécuter ce programme dans ce système **[9].** Nous pouvons trouver ce code dans l'**ANNEXE III**. En effet, il permet de rajouter la composante conception conjointe matérielle/logicielle en intégrant la configuration, la programmation d'un processeur (Cortex M1). Cette suite logicielle permet de réaliser des systèmes à base de microprocesseur pouvant utiliser tous les périphériques disponibles sur la carte cible et / ou de piloter ses propres blocs matériels. SoftConsole IDE possède toute la chaîne de programmation et de compilation des processeurs cibles. Nous allons voir ce processeur dans le prochain paragraphe.

II.4. Les circuits utilisés

D'après ce qui précède, nous savons réaliser la conception matérielle d'instruments virtuels puis de l'implémenter sur un circuit FPGA en le commandant par une partie logicielle exécutée sur un processeur. Maintenant, nous allons voir les circuits ou bien les cartes (cartes mère et cartes filles) servant d'interface pour l'instrumentation virtuelle ainsi que le processeur utilisé.

II.4.1. La carte d'acquisition d'Actel « Fusion Embedded Development Kit »

L'entreprise Actel propose plusieurs cartes d'acquisition ou cartes d'expérimentation sur le marché avec différents circuits FPGA dont elle les classe chacun à une certaine famille. Notons que notre support de réalisation est la carte Actel Fusion Embedded Development Kit. Cette carte d'Actel « Fusion Embedded Development Kit » est représentée sur la**Figure2.7**ci-après. Elle est construite avec huit couches de cartes à circuits imprimés (cf. **ANNEXE I**). Elle embarque un FPGA nommé « M1AFS1500-FGG484 » appartenant à la famille « Actel Fusion ». Le M1AFS1500-FGG484 possède 1500000 portes logiques assemblables à souhait à l'aide du VHDL (cf. **ANNEXE II**) et supporte l'intégration d'un processeur ARM Cortex M1.

II.4.2. Caractéristiques de Fusion Embedded Development Kit

Les différentes caractéristiques du M1AFS1500-FGG484 sont résumées dans le **Tableau II**.

Tableau II : Caractéristiques de M1AFS1500 **[10]**

portes logiques	1500000
bascules (D flip-flop)	1024
sûr(AES) ISP	oui
PLLs	2
Globals	18
blocs de mémoire flash (2Mbits)	4
mémoire flash totale	8Mbits
FlashROM	1kbits
blocs RAM (4,608 bits)	60
RAM	270kbits
Analog Quads	10
canaux d'entrée analogique	30
pilotes des sorties d'une bascule	10
E/S Banks (+JTAG)	5
E/S numériques maximum	252
E/S analogiques	40

L'aspect physique du Fusion Embedded Development Kit est représenté à la **Figure 2.7**.

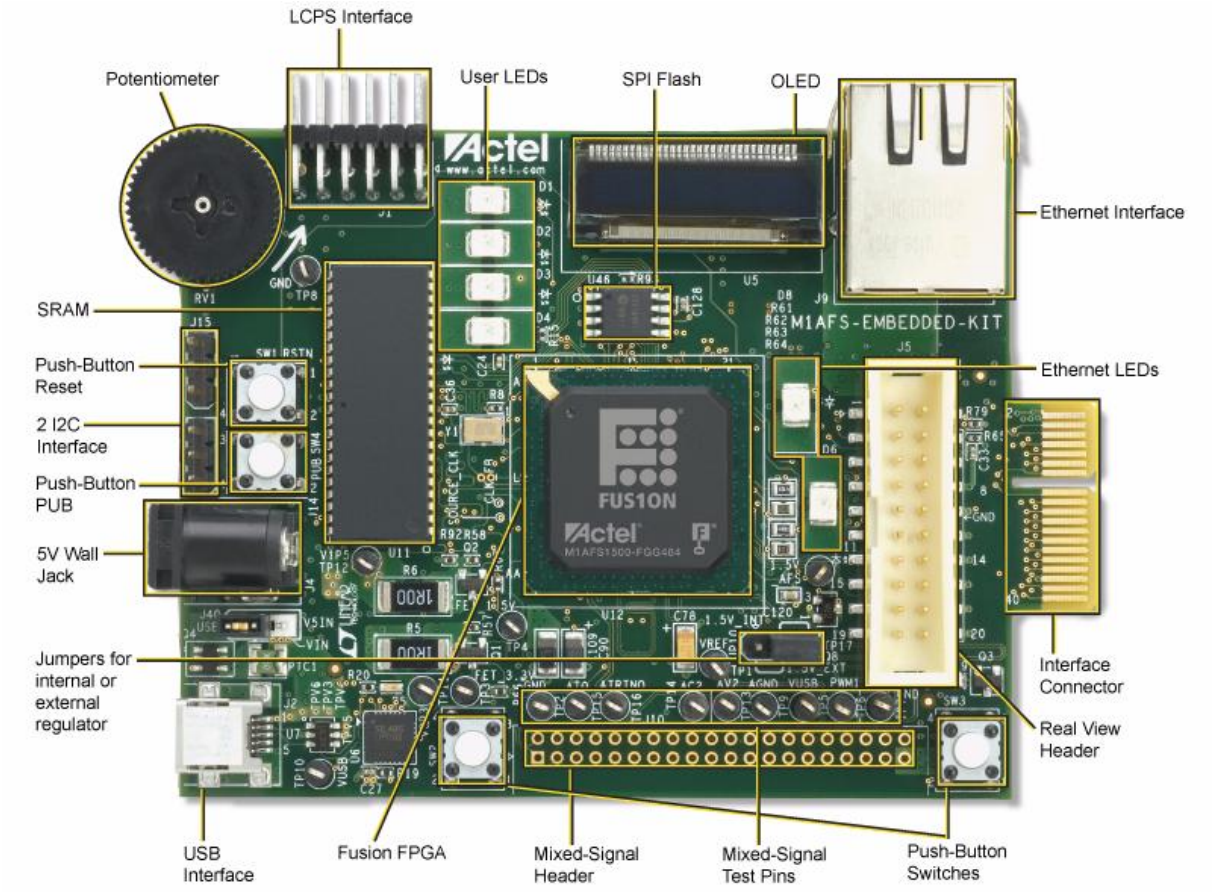


Figure 2.7: Fusion Embedded Development Kit **[10]**

Les paramètres par défaut du jumper présent dans la carte Fusion Embedded Development Kit sont décrits dans le **Tableau III** ci-dessous :

Tableau III : Paramètres par défaut du jumper présent dans la carteFusion Embedded Development Kit **[10]**

Jumper	Paramètres par défaut	Commentaires
JP10	Pin 3-2	Jumper pour sélectionner soit le régulateur externe 1,5 V soit le régulateur interne 1,5 V
J40	Pin 1-2	Jumper pour sélectionner la source d'alimentation Pin 3-2 = 5 V Adapteur Pin 1-2 = USB

Ce kit dispose d'un grand nombre d'E/S et d'un convertisseur analogique numérique intégré et il supporte un signal mixte et soutient une série de processeurs, y compris *l'ARM Cortex*-M1qui est le processeur que nous avons mentionné auparavant et le *Core8051s*. Cela nous donne des avantages pour notre conception d'instruments virtuels.

Nous avons choisi d'utiliser ARM Cortex-M1 grâce à sa haute performance avec une petite dimension dans les FPGAs. En effet, c'est un microprocesseur 32 bits, Cortex - M1 exécute un sous-ensemble de « *Thumb-2 instruction set* » (ARMv6-M) cela inclut toutes les bases d'instructions 16 bits et quelques instructions « *Thumb-2* » 32 bits (BL, MRS, MSR, ISB, DSB, et DMB). Cela permet l'écriture très serré et efficace du code du processeur qui est idéal pour une mémoire limitée typiquement trouvée dans des applications profondément intégrées. Les blocs principaux dans Cortex - M1 sont montrés dans la **Figure2.8** ci-après incluant le cœur du processeur, le *Nested Vectored Interrupt Controller* (NVIC) (Contrôleur Emboîté d'Interruption Vectorisée), les interfaces AHB, et l'unité de débogage.

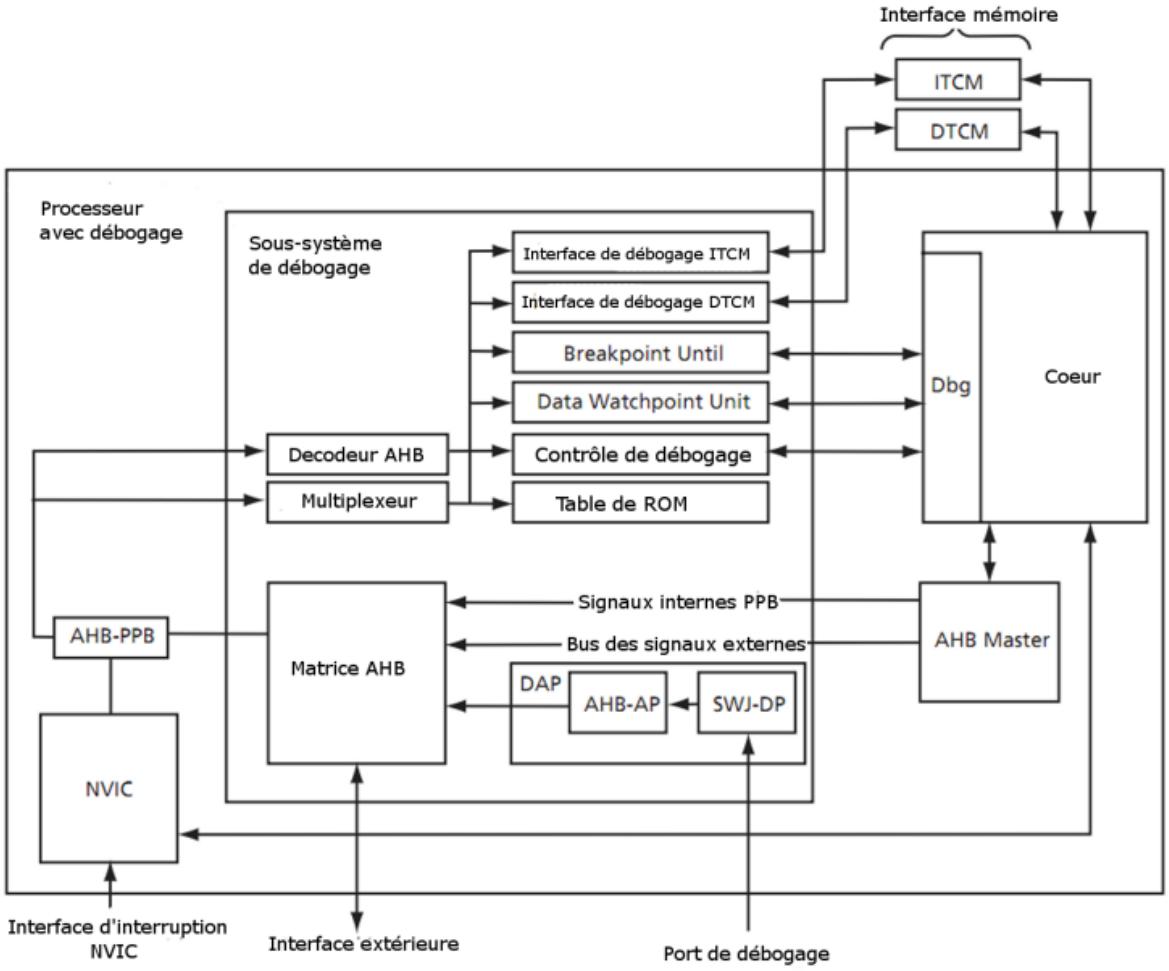


Figure 2.8: Les blocs principaux dans Cortex-M1 **[11]**

Le cœur du processeur supporte 13 registres 32 bits à usage général, y compris le *Link Register* (LR) (Registre de Lien), le *Program Counter* (PC) (registre d'instructions), le Program Status Register (xPSR) (Registre d'état du Programme), et deux registres de pile *Stack Pointers* (SP). En plus de l'interface AHB-Lite, il y a une interface mémoire privée pour l'accès à l'*Instruction and Data Tightly Coupled Memories* (ITCM et DTCM) (Instruction et les Données Associées Hermétiquement en mémoires).

II.4.3. Le low-cost programming stick (LCPS) »

Le LCPS est une carte fille qui sert à programmer le FPGA M1AFS1500 via le port USB d'un PC et à l'aide d'un logiciel appelé *« FlashPro »*. Il suffit de la connecter à *« LCPS connector »* du *« Fusion Embedded Development Kit »*.

La **Figure 2.9** montre le LCPS connecté à la carte mère du M1AFS1500-FGG484.

Figure 2.9 : Le LCPS connecté à la carte mère du M1AFS1500-FGG484 **[10]**

CHAPITRE III. LA PROGRAMMATION LOGICIELLE

III.1. Les langages et compilateurs

L'instrument virtuel, pour être utilisable, doit fournir une interface visuelle permettant à l'utilisateur d'interagir avec lui, ainsi qu'un programme traitant les informations relatives aux mesures provenant de l'instrument physique ou à envoyer vers celui-ci. Toutes ces tâches se résument en une conception d'une application logicielle.

De nos jours, différents langages de programmation sont disponibles pour créer une application logicielle. Pour chacun de ces langages, différents compilateurs sont également disponibles. Ces langages et ces compilateurs ont chacun leur particularité ce qui fait que tel ou tel langage convient mieux pour tel ou tel genre d'application. Ainsi le choix du langage et du compilateur doit être fait en rapport avec le type de l'application à concevoir.

Dans l'application que nous allons créer, notre choix a été porté sur le langage C++ et le compilateur Visual C++ en utilisant la bibliothèque MFC (Microsoft Fondation Classes). Voyons maintenant les avantages et les particularités de ces outils.

III.1.1. Le langage C++

Le langage utilisé, C++, est un des langages les mieux structurés et sans doute le plus performant. En effet, la performance est vraiment requise pour des applications faisant appel à des traitements de données en temps réel comme l'acquisition ou l'envoi des données pour les instruments virtuels. De même, la disposition des fonctionnalités avancées peut s'avérer indispensable pour des gros programmes traitant un volume important de données. En fait, pendant longtemps, la principale préoccupation des programmeurs était de concevoir des applications très courtes pouvant s'exécuter rapidement, car la mémoire et le temps de calcul coûtaient cher.

Le langage C++ est un langage orienté objet, impératif et compilé : un **langage orienté obje**t fournit les techniques permettant de traiter des applications très complexes, exploite des composants logiciels réutilisables et associe les données aux tâches qui les manipulent. La caractéristique essentielle de la programmation orientée objet est de modéliser des "objets" (c'est-à-dire des concepts) plutôt que des "données". Ces objets peuvent être des éléments graphiques affichables, comme des boutons ou des zones de liste, ou des objets réels, comme des stylos, des ordinateurs, ou des chats, ou encore des objets abstraits comme la sécurité sociale . Les objets possèdent des caractéristiques, également

appelées propriétés ou attributs, comme âge, couleur, volume. Ils ont aussi des fonctionnalités, appelées opérations ou fonctions, comme changer, écrire, voler. Le rôle de la programmation orientée objet est de représenter ces objets dans le langage de programmation.

Un **langage impératif ou procédural** permet de faire des programmes constitués de suites d'instructions permettant de modifier l'état de l'ordinateur stocké dans sa mémoire

Un langage compilé est un langage dont le code source d'un programme est traduit en langage machine une bonne fois pour toute par le compilateur. Celui-ci donne un fichier objet puis le lie à un fichier exécutable, lors de la compilation, qui peut s'exécuter indépendamment plus tard. Les langages compilés sont nettement plus rapides comparés aux langages interprétés dont le code (généralement appelé script) est interprété par un logiciel appelé interpréteur qui traduit et exécute une à une les instructions du programme et les transforme directement en actions chaque fois qu'on exécute le programme. L'autre avantage des langages compilés comme C++ tient à la diffusion des programmes puisque l'on peut distribuer un fichier exécutable à des personnes qui ne disposent pas du compilateur. Avec un langage interprété, par contre, l'utilisateur doit nécessairement posséder l'interpréteur pour pouvoir exécuter le programme.

Mais ce qui est à l'origine de la puissance du langage C++ est surtout le fait que ses instructions sont à bas niveau, censées permettre au programmeur d'utiliser au mieux les ressources de son ordinateur, contrairement au langage plus haut niveau tels que Java ou WinDev dont l'exécution du code est nettement plus lente.

Par ailleurs, le langage C++ reste en constante évolution pour intégrer les technologies les plus récentes.

Tout de même, tous ces avantages ne nous laissent pas passer sous silence quelques inconvénients :

Une des principales limites du C++ résulte du fait que ses instructions sont trop bas niveau et les fonctionnalités avancés comme l'utilisation des pointeurs, l'héritage multiple,… peuvent conduire à des programmes trop compliqués pour être maintenables. Tout comme les langages impératifs, le langage C++ est sujet à des bogues provenant d'incohérences dans la gestion de l'état du programme. Surtout pour les gros programmes, le programmeur est souvent confronté à des gestions des conflits ou à des analyses des exceptions qui peuvent être une tâche ardue. De plus, le langage C++ est réputé très rébarbatif et compliqué. Sa syntaxe

est peu lisible et fort contraignante. La mise au point d'un gros logiciel écrit en C/C++ est longue et pénible. Les statistiques montrent qu'on rencontre des bogues mineurs tous les 10 lignes de code en C++ et des bogues majeurs tous les 100 lignes. De bonnes techniques de programmation et des mécanismes implémentés dans le langage peuvent toutefois limiter ces erreurs.**[12] [13] [14]**

III.1.2. Visual C++ et MFC

Visual C++ est un outil puissant et complexe pour créer des applications fonctionnant sous Windows. MFC est une bibliothèque des classes C++ fournie par Microsoft comme une interface orientée objet encapsulant l'API Windows. Elle encapsule les différentes fonctions propres à Windows. Elle est devenue le standard industriel pour les développements logiciels pour de nombreux compilateur C++.

Les avantages de MFC sont :

- La prédisposition des classes : la bibliothèque MFC est une collection hiérarchisée très vaste de différentes classes qu'on peut utiliser directement ou qu'on se sert comme classe de base dont dérivent les classes spécialisées, ou encore qu'on implémente comme des donnés membres des classes. Il reste au programmeur de définir les attributs et méthodes supplémentaires de ses propres classes descendantes des classes standards MFC.

- La facilité de la création d'interface graphique avec la disponibilité des contrôles standards encapsulés dans le Framework MFC. Les différentes fenêtres, les menus, les barres d'outils, ou les contrôles d'une application sont vus comme des objets instanciés à partir des classes MFC ou de classes dérivées des classes MFC. L'écriture d'un programme Windows va donc consister à créer des classes et à gérer des instances de celles-ci.

- La génération des codes à l'aide de l'outil AppWizard, un assistant visuel qui se charge de l'écriture d'une partie du code en recevant les paramètres dans une interface graphique. Cela facilite grandement la tâche du programmeur et évite les erreurs de frappe, de redondance ou de contradiction.

- Un Cadre de travail (Workspace) conviviale et très intelligible. On peut visualiser graphiquement les propriétés et les méthodes des classes utilisées. Cela dispense le programmeur de se perdre dans des centaines de lignes de code.

III.2. Méthodologie de développement logiciel

Le langage de programmation, et le compilateur étant choisi, ce n'est pas tout. En effet développer une application logicielle efficace ne se résume pas à l'écriture des codes. Pour que l'application soit bien conçue (c'est-à-dire satisfaisant aux fonctionnalités attendues) et maintenable (c'est-à-dire qu'on peut faire évoluer facilement), il faut suivre certaines étapes. Ces étapes constituent le cycle de développement d'un logiciel et sont les suivantes : l'analyse des besoins, la spécification et conception, le codage, test du logiciel, la maintenance.

III.2.1. L'analyse des besoins

La première étape à suivre lorsqu'on souhaite écrire un programme, ou plus généralement développer un logiciel, consiste à se poser clairement la question suivante: Que doit faire mon programme ? Il est donc nécessaire de formaliser son besoin, d'écrire un cahier des charges qui énonce ce que va pouvoir faire l'utilisateur du programme, ce que pourra être l'interface graphique, s'il existe des contraintes de sécurité ou de fiabilité. Cette étape est essentiellement un travail de réflexion et d'analyse qui évite bien des déconvenues en pratique lorsqu'on aura fini de programmer et de tester le logiciel.

Dans notre application alors qui consiste à concevoir des instruments virtuels (oscilloscope et générateur de forme d'ondes dans le cas présent), nous allons détailler dans les premiers paragraphes des chapitres V et IV les fonctionnements de ces instruments.

III.2.2. Spécification et conception

Une fois l'analyse des besoins réalisée, commence la phase de spécification et conception du logiciel, c'est-à-dire le moment où on écrit sur le papier sa structure : Quelles sont les classes à écrire ? Quels sont les méthodes et les attributs ? Et le détail de son fonctionnement: Quels sont les algorithmes des méthodes ? Ce travail permet au bout du compte d'avoir un logiciel "sur le papier". Il est généralement réalisé en utilisant des méthodes orientées objet comme UML par exemple. UML est dédié à la conception de systèmes logiciels à haut niveau.

Il a pour objectif, cinq activités principales du processus de conception

- Une description graphique du système selon plusieurs points de vue,
- La spécification des besoins et de la mise en œuvre,

- La visualisation pour faciliter la compréhension et la communication parmi les partenaires de la conception avant la réalisation du système,
- La représentation graphique de systèmes complexes,
- La documentation de la totalité du projet, dès les spécifications jusqu'aux tests de fonctionnement.

Dans ce paragraphe nous allons utiliser la méthode UML (Unified Modeling Language : langage de modélisation unifié) pour modéliser les conceptions en tenant compte bien sûr des besoins spécifiés dans l'étape précédente. **[15]**

Les différents éléments de l'interface qu'on a décrits dans l'analyse de besoin (premiers paragraphes des chapitres 5 et 6) sont modélisables par des classes qui seront les briques de base du logiciel. Leurs structures ainsi que leurs interactions vont être décrites dans le diagramme de classe de la **Figure 3.1**.

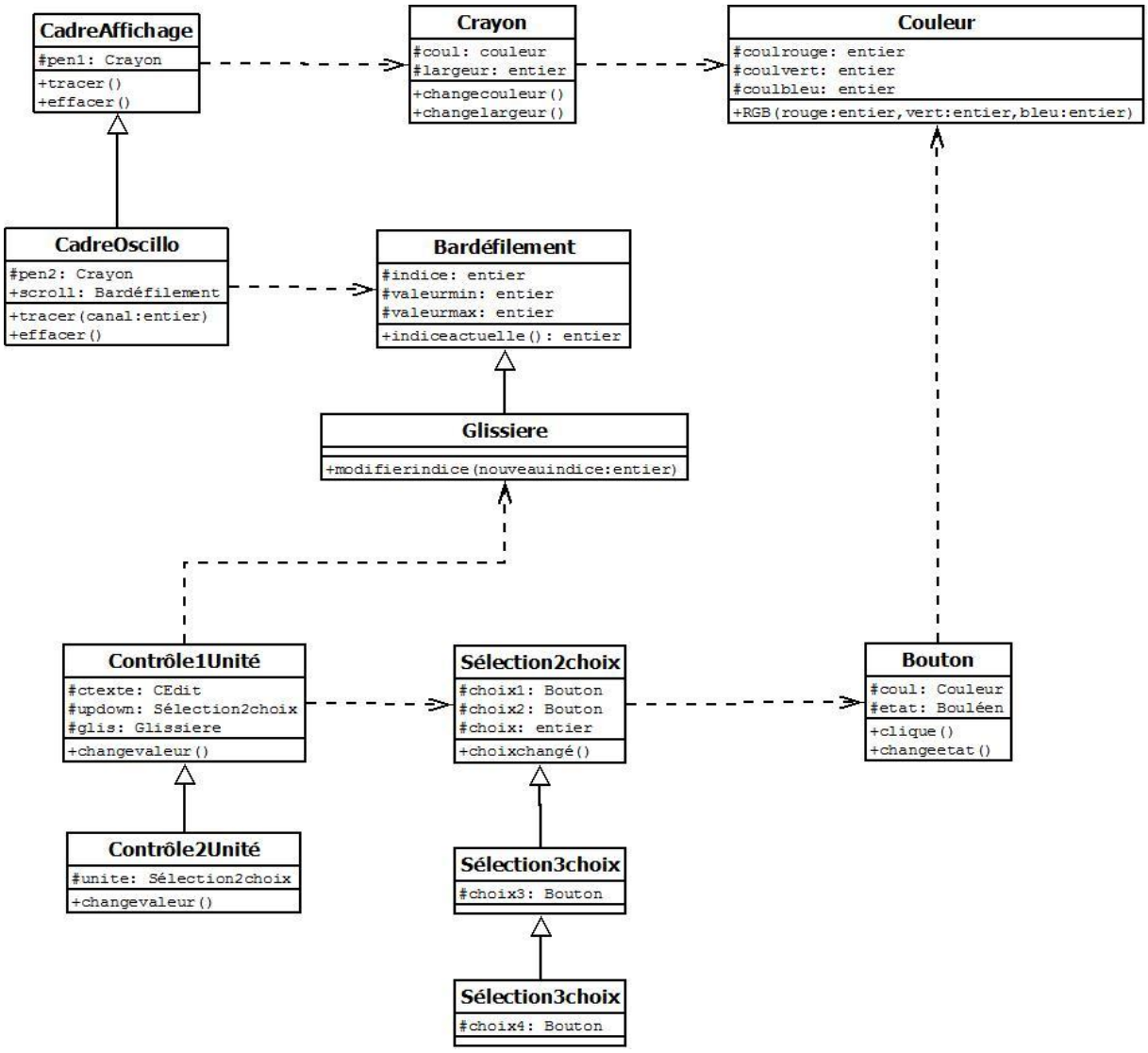


Figure 3.1 : Diagramme des classes

Remarque :

Les classes pour l'oscilloscope et le générateur de forme d'onde sont en même temps représentées étant donné leurs similitudes et leurs relations hiérarchiques.

III.2.3. Codage

Le codage est en quelque sorte la traduction en un langage de programmation des spécifications décrite dans la phase précédente pour donner naissance à la première mouture du logiciel. C'est dans cette étape qu'on implémente dans le langage les différentes classes définies dans la phase de spécification et qu'on traduit les algorithmes des méthodes de chaque classe en code du langage.

III.2.4. Test du logiciel

Cette phase consiste à tester le logiciel à l'aide de méthodes scientifiques bien définies comme les tests fonctionnels, les tests structurels. Ceci est nécessaire pour s'assurer que les fonctionnalités implémentées correspondent bien aux besoins spécifiés.

III.2.5. Maintenance

Une fois que la conception de l'application est terminée, la phase de maintenance peut s'avérer utile dans le futur. La maintenance consiste à faire évoluer l'application, si nécessaire. Ces évolutions peuvent être de la correction de bogues, de l'amélioration de performances, de l'ajout de nouvelles fonctionnalités. En effet, généralement, une application logicielle n'est pas conçue pour être figée. Les environnements de développement et le système d'exploitation évoluent, des nouvelles normes pourront avoir lieu, des nouveaux besoins pourront voir le jour, ainsi la conception doit anticiper les évolutions futurs du logiciel, c'est-à-dire que la conception doit avoir un maximum de souplesse pour des possibles évolutions. Ceci est particulièrement vrai pour l'instrumentation virtuelle où les fonctionnalités et les précisions sont en constante croissance du fait de l'évolution sans cesse des technologies de l'électronique et de l'informatique**.[16]**

III.3. Interfaçage matériel de l'Instrument Reconfigurable avec le PC

Afin de pouvoir acquérir les données résultant de la mesure venant de l'instrument reconfigurable (dans le cas de l'oscilloscope) ou bien envoyer les données vers celui-ci (dans le cas du générateur de forme d'onde), le logiciel doit établir une communication via un port. L'ordinateur dispose de nombreux port pour communiquer avec les périphériques externes : le port parallèle, le port série, le port Ethernet, l'USB, le FireWire disponible seulement pour les ordinateurs portables,

Chacun de ces ports a ses particularités et ses domaines d'utilisations. Les ports parallèles sont destinés pour les périphériques lents. Ils ne supportent pas de débits élevés car cela créerait des interférences électromagnétiques pour les données circulant en parallèles, ce qui peut entraîner des erreurs. C'est pour surmonter cette limite en débit qu'arrivent les ports séries dans lesquels les données circulent en série et par conséquent le problème d'interférence n'a plus raison d'être et le débit maximum est plus élevé. Ces deux ports sont devenus de plus en plus rares sur les ordinateurs récents.

III.3.1. Le port USB

Le port USB (Universal Serial Bus) est considéré actuellement comme le standard. Il est disponible sur les ordinateurs de bureau et les ordinateurs portables. Les données y circulent en série et le débit peut atteindre 480 Mbps (Mégabits par seconde) (ce n'est qu'un débit théorique) pour la norme USB 2.0, ce qui permet une acquisition en temps réel pour des fréquences assez élevées. De plus, le port USB peut alimenter le périphérique branché. Un autre apport important de l'USB est la possibilité de branchement à chaud (hot plug in), c'est à dire qu'on peut brancher ou débrancher une connectique USB sans avoir à redémarrer l'ordinateur. Le système d'exploitation le reconnait immédiatement. Les protocoles utilisés par l'USB sont assez complexes mais très souples, ce qui lui permet d'être utilisé pour de nombreux périphériques très disparates, voire tous les périphériques. C'est pour cela que l'USB est devenu le standard et tous les périphériques ont actuellement tendance à fournir une interface USB. A côté de l'USB, il existe des interfaces considérablement plus rapides comme le PCI mais dont la mise au point la programmation est compliquée, l'accès physique nécessite d'ouvrir l'unité centrale du PC, donc beaucoup moins pratique.

Le développement d'une application communiquant avec un bus USB nécessite la programmation d'un driver gérant toutes les protocoles de communication entre le programme appelant et le système d'exploitation. Cette tâche nécessite la connaissance au bout des doigts de tous les protocoles USB et ses interactions avec le Système d'Exploitation. Heureusement, des librairies C avec ses drivers précompilés sont disponibles pour contourner ce grand défi. Pour ce faire, il faut installer le driver préconçu fourni avec la bibliothèque et le programme utilise l'API de la bibliothèque pour accéder au port USB. Les fonctionnements du driver sont alors totalement transparents pour le programmeur. Dans notre application, nous allons nous servir d'une de ces bibliothèques, une des plus utilisés, c'est la librairie *libusb* qui est de plus open source (libre). Néanmoins, il est nécessaire de comprendre les modes de communications entre l'application développée, le système d'exploitation, le driver, et le bus USB.

III.3.2. Principe de communication avec le port USB

Au branchement d'un périphérique USB, le système détecte le nouveau périphérique connecté. En effet, le système d'exploitation scanne régulièrement tous les ports USB et une différence de potentiel lui met au courant du branchement d'un nouveau périphérique. Après, le système d'exploitation identifie et configure le périphérique en lui donnant une adresse unique. C'est la gestion dynamique de la connexion et de la déconnexion des périphériques

reliés à un bus USB. Le périphérique fournit à l'hôte une suite de descripteurs qui permettent son identification complète. Lors de cette phase d'énumération, on assigne une adresse unique (Unique ID) au périphérique, on charge le driver correspondant et on positionne le composant dans la configuration qui lui à été donnée par les descripteurs. Il n'est pas indispensable de connaître parfaitement le processus d'énumération et le système de descripteur pour pourvoir faire fonctionner un composant USB mais il est bon d'en connaître les grandes lignes pour pouvoir, au besoin, changer les descripteurs. Notons également que cette phase d'énumération est automatique et totalement transparente pour l'utilisateur. L'ensemble de ces étapes constitue ce qu'on appelle l'énumération qui se résume comme la reconnaissance d'un périphérique USB par le système d'exploitation.

Maintenant que le périphérique est reconnu par le système d'exploitation et que son driver est chargé, le parti est alors joué par le driver qui se charge de la communication avec le matériel connecté au bus USB. Le programme s'interchange des données avec le périphérique alors via le driver. **[17] [18] [19]**

III.4. Les contraintes du logiciel

Il est nécessaire de bien réfléchir des besoins en tenant compte des possibilités technologiques et des différentes solutions possibles pour que le système fonctionne en parfaite harmonie. Ce point est particulièrement important pour le système RVI car les choix de la plateforme logicielle, de l'environnement de développement, de l'interface physique, du circuit programmable sont énormes. Ainsi, il est possible que certains besoins ne soient pas entièrement satisfaits pour les solutions retenues. C'est le cas des fréquences maximales supportées par l'oscilloscope pour une interface USB. En effet, la limitation en débit devient un problème lorsque l'on parle d'instruments qui nécessitent une large bande passante. C'est le cas des instruments radiofréquence entre autres, mais aussi des oscilloscopes à haute fréquences, qui sont pourtant les premiers produits à avoir décliné sous une version modulaire. Familiers de l'univers Windows, les industriels sont naturellement tentés par l'USB… mais seuls les bus VXI ou PXI donnent accès à des cadences extrêmement rapides. Les modules USB restent réserver à des applications d'acquisition de données à des vitesses limitées. « Si l'utilisateur a besoin d'une mesure de signal en temps réel : l'instrumentation sur bus PCI, PXI ou PCI Express reste à privilégier. La bande passante actuelle de l'USB ne permet pas d'atteindre des vitesses de transfert en continu vers un PC de l'ordre du Géch/s de

données, elles se cantonnent à des vitesses de l'ordre de 100 Méch/s ». De ce fait, les modules oscilloscopes USB sont plutôt dédiés à des applications basse fréquence.

CHAPITRE IV. REALISATION D'UN OSCILLOSCOPE VIRTUEL

IV.1. Généralités

Un oscilloscope est basiquement un appareil pour affichage de courbes. Il dessine la courbe d'un signal électrique. Dans la plupart des applications, la courbe représente la variation d'une ou plusieurs tensions électriques en fonction du temps, mais elle peut aussi afficher la variation d'un signal électrique en fonction d'un autre permettant de comparer les deux signaux. En général les oscilloscopes peuvent capter des signaux électriques allant de quelques millivolts à quelques centaines de volts et ayant des fréquences jusqu'à quelque Mégahertz pour les oscilloscopes analogiques. Cet ordre de fréquence élevé est cependant difficilement atteint pour les oscilloscopes numériques du fait des volumes trop importants des données à traiter. En effet le taux d'échantillonnage doit être supérieur au double de la fréquence maximale d'après le théorème de Shannon or le débit des ports des ordinateurs n'est que limité. Comme nous allons interfacer le matériel avec l'USB dont le débit maximal est de 480 Mbps, d'après les calculs, l'oscilloscope pourra capter les signaux ayant des fréquences jusqu'à 100 KHz.

Spécifications

Un oscilloscope peut avoir un, deux ou même davantage de canaux, ce qui permet de capter et de visualiser simultanément plusieurs signaux électriques, ou bien de comparer ces signaux. Dans la mesure où le nombre des canaux implique encore plus de quantité de donnés à partager le même débit dans le bus de communication, notre oscilloscope aura deux canaux qui seront suffisant dans beaucoup d'applications.

Bien que principalement l'oscilloscope serve à afficher une courbe dessinant la variation d'une tension électrique en fonction du temps, cette courbe parle beaucoup de choses :

- le temps et le voltage du signal ;
- la fréquence d'un signal oscillant ;
- quelle proportion du signal est la composante continue ou alternative ;
- une distorsion affecte le signal ou pas ;
- on peut aller même jusqu'à l'analyse harmonique d'un signal périodique pour identifier les différents harmoniques constituant le signal.

Ainsi, l'oscilloscope possède avant tout un cadre afficheur qui affichera en plus des courbes des grilles qui servent des repères pour le temps et les valeurs mesurées. Dans la mesure où cet afficheur est limité, les échelles des mesures devront être réglables, c'est-à-dire que l'utilisateur pourra modifier manuellement les valeurs des échelles pour ajuster l'oscillogramme à l'intérieur du cadre d'affichage. Ainsi deux paramètres devront figurer : la Base de temps (en milliseconde par division ou en microseconde par division) qui est l'échelle horizontale, et la Base de Y (en volt par division ou en millivolts par division) qui est l'échelle verticale. Chaque canal devra avoir sa propre échelle horizontale.

En plus, on constate que même en variant ces deux paramètres, un signal alternatif possédant une composante continue relativement élevée (en valeur absolue) risque de ne pas se cadrer dans l'afficheur. Deux solutions sont alors possibles : la première c'est de varier la position de l'axe horizontale. La seconde est de faire en sorte que l'oscilloscope puisse éliminer optionnellement la composante continue et n'afficher que l'alternative. Ces deux solutions méritent d'être prises en compte.

Ainsi, l'utilisateur devra pouvoir varier la position de l'axe horizontale pour chaque canal. De même, pour chaque canal, il devra pouvoir faire le choix soit afficher la valeur du signal entier (composante continue + composante alternative) soit afficher seulement la composante alternative. C'est ainsi que les oscilloscopes disposent toujours d'un commutateur marqué AC / DC / 0. Voici leurs significations : AC pour la composante alternative seulement, DC pour le signal entier, et 0 pour faire référence à la masse.

Il peut aussi être nécessaire de faire en sorte que la position de l'axe vertical soit lui aussi paramétrable.

Puisqu'un oscilloscope peut disposer de plusieurs canaux (typiquement 2), il peut aussi comparer deus signaux en affichant la variation du signal d'un canal en fonction de celui de l'autre. C'est pour cela que beaucoup d'oscilloscope possède des fonctions permettant de faire les choix entre le mode d'affichage des courbes. Ainsi pour un oscilloscope à deux signaux, trois modes sont possibles : affichage des deux signaux en fonctions du temps, affichage du signal du premier canal en fonction de celui du second canal, et affichage du signal du second en fonction de celui du premier canal.

En plus de tout ça, des triggers sont disponibles sur la plupart des oscilloscopes. Littéralement déclencheur, ce paramètre sert à définir les conditions sous lesquelles

l'oscilloscope va afficher la courbe. Ces conditions s'appuient soit sur les signaux mesurés eux-mêmes soit sur une entrée spéciale appelée trigger. Lorsque le trigger est activé, on spécifie une valeur en volts ou en millivolts et un sens descendant ou ascendant ainsi que le signal auquel le trigger s'applique. L'affichage ne commence alors qu'après que le signal spécifié ait atteint la valeur du trigger dans le sens spécifié. **[20]**

IV.2. Conception et développement de la partie matérielle

IV.2.1. Synoptique du système à concevoir

La **Figure 4.1** représente le schéma synoptique de l'oscilloscope digital.

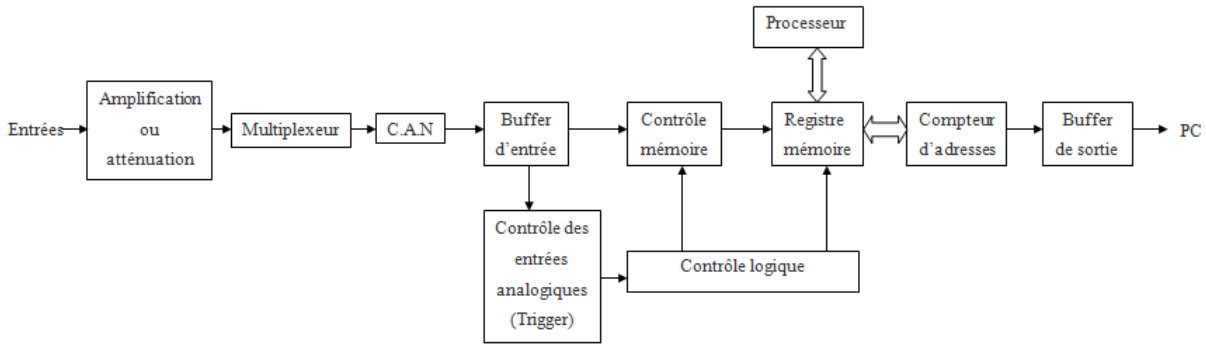


Figure 4.1 : Schéma synoptique du système matériel de l'oscilloscope

IV.2.2. Le flot de conception de l'oscilloscope (Hardware design flow)

Dans cette première partie du développement matériel de l'oscilloscope, nous allons réaliser les étapes de programmation du FPGA à l'aide de la suite logicielle Libero IDE d'Actel **[21]**.

a) *Approche de conception par schéma*

Cette partie porte sur la conception de l'oscilloscope numérique à l'aide de l'approche de conception FPGA par schéma. Dans cette partie, notre projet d'instrumentation virtuelle présente une structure hiérarchique reposant sur des modules schématiques. En effet, le fichier top level du projet contient une représentation schématique de l'ensemble du système qui fait appel à plusieurs types de macros de niveaux inférieurs. Ces macros utilisent plusieurs types de modules de natures variées dont des modules schématiques, des IP générés à l'aide du « Core Generator », des modules paramétrés générés à l'aide du « *SmartGen Core* » de Libero IDE et des modules VHDL. Dans cette section, voyons successivement les étapes suivantes : la création du schéma top level de l'oscilloscope, la simulation fonctionnelle,

l'implémentation du projet, la simulation avec timing et la configuration du FPGA avec le système conçu. La **Figure 4.2** suivante représente le schéma bloc de l'oscilloscope.

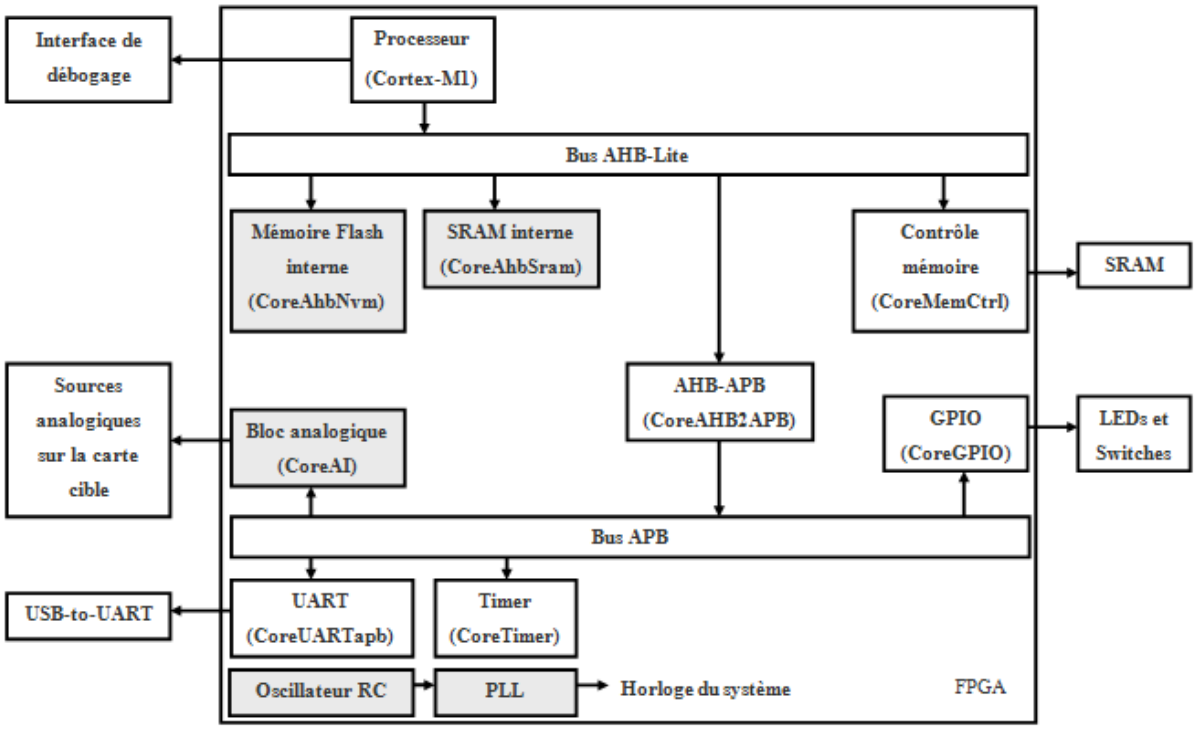


Figure 4.2 : Schéma bloc de l'oscilloscope

b) *Création d'un schéma top level*

Le schéma top level permettra d'instancier plusieurs modules de niveaux hiérarchiques inférieurs et d'interconnecter le tout pour réaliser le schéma complet de l'oscilloscope numérique (**Figure 4.3**). Les blocs : CoreAhvNvm, CoreAhbSram, CoreAI, Oscillateur RC, PLL sont des périphériques « Hard IP » tandis que les autres blocs représentent des périphériques « Soft IP ».

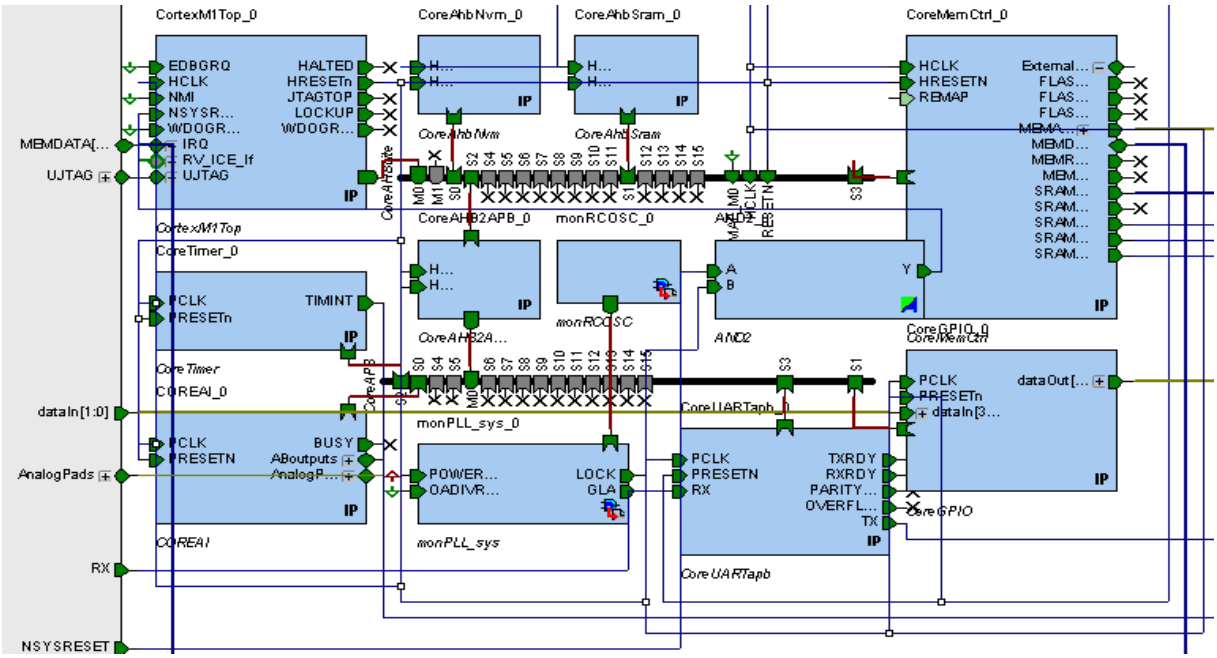


Figure 4.3 : Schéma top level de l'oscilloscope

Dans la configuration du processeur Cortex-M1, nous allons choisir FlashPro3 comme interface de débogage car cela autorise notre logiciel de debogger sur JTAG en utilisant l'outil de développement Actel SoftConsole.

Le bus AHB-Lite est un bus maître 32 bits (données et adresse) qui nous permet de relier au maximum 2 maîtres et 16 périphériques esclaves, où chaque périphérique possède un slot (numéroté de 0 à 15). Chaque slot correspond à 256 Mo dans l'espace mémoire du processeur (pour un total de 4 Go).

Le CoreAhbNvm fournit une interface AHB au processeur pour accéder au NVM interne (la mémoire non volatile, connue aussi comme mémoire flash intégré) d'Actel Fusion. Le M1AFS1500 a 1 Mo de NVM interne composé de 4 mémoires flash intégrées, chacun a une taille de 256 Ko.

Le CoreMemCtrl crée une interface pour la mémoire SRAM et/ou la mémoire flash avec des données partagées et des bus d'adresse. Le slot AHB est divisé en deux avec la mémoire flash à la partie inférieure et la mémoire SRAM à la partie supérieure du slot. En mettant l'entrée « *Remap* » à '1', nous pouvons échanger les emplacements de la mémoire flash et de la SRAM. Pour la conception de notre oscilloscope, nous utiliserons seulement l'interface SRAM. L'horloge du système dans ce design est de 20 MHz qui a une période de 50 ns. Le SRAM asynchrone a un temps d'accès 10 ns. Par conséquent, nous pouvons mettre

les états latents à leurs valeurs minimums: 0 pour les états latents de lecture et 1 pour les états latents de l'écriture. Les options d'adressage de la mémoire flash et du SRAM déterminent quel bit du bus AHB-Lite est relié au bit 0 de la mémoire externe.

Le CoreAhbSram fournit une interface AHB au processeur pour accéder à la mémoire SRAM interne de la Fusion. La Fusion M1AFS1500 a 30 Ko de mémoire SRAM interne qui consiste en 60 blocs mémoires RAM dont 0.5 Ko chacun. Le Cortex-M1 utilise 2 Ko et le CoreUARTApb utilise 1 Ko en mode FIFO. Par conséquent, le CoreAHBSram ne doit pas utiliser plus de 27 Ko.

Le CoreAHB2APB est un pont du bus AHB au bus APB. Les transferts d'écriture et de lecture sur le bus AHB est converti aux transferts APB au bus APB.

Le bus CoreAPB est un bus 32 bits (données et adresse) lequel nous permet de relier à 16 périphériques esclaves où chaque périphérique possède un slot (numéroté de 0 à 15). Chaque slot représente 16 Mo dans l'espace mémoire du processeur pour un total de 256 Mo. Dû à la performance, typiquement plus lent, les périphériques de faible priorité sont placés au bus APB.

Le CoreAI permet auprocesseur Cortex-M1 d'accéder et contrôler le Bloc Analogique de la Fusion. La fréquence d'horloge ACM dans le Bloc Analogique doit être 10 MHz ou moins. L'horloge du bus APB (PCLK) est divisée pour créer l'horloge ACM.

Le CoreUARTapb est un périphérique UART configurable avec une interface APB esclave.

Le CoreGPIO nous permet d'accéder jusqu'à 32 entrées à usage général et 32 sorties à usage général.

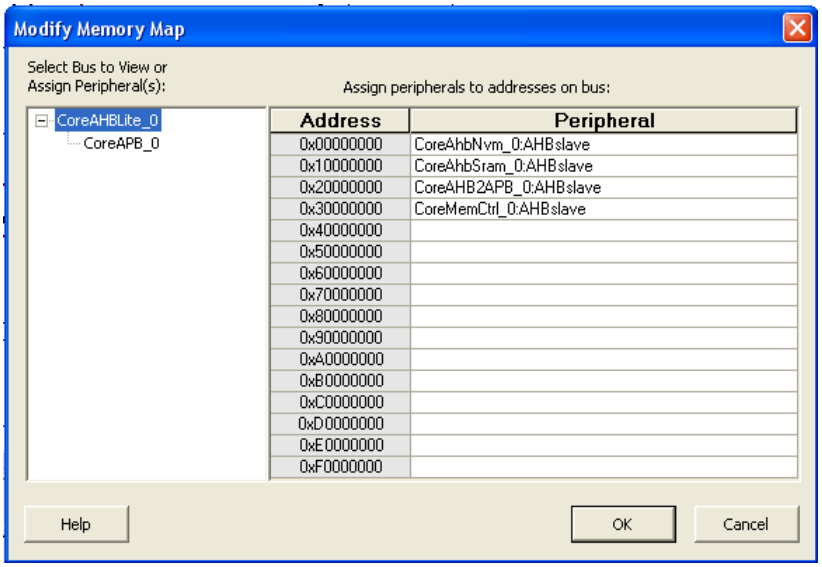
Le module CoreTimer est un APB esclave qui fournit l'accès au générateur d'interruption et au compteur de décrémentation programmable. La taille du compteur de décrémentation dans le module CoreTimer peut être configurée sur 16 ou 32 bits.

Les circuits d'Actel Fusion ont un oscillateur RC interne de 100 MHz qui n'exige pas de composants supplémentaires. Nous mettons ceci comme une entrée au PLL à l'intérieur du FPGA.

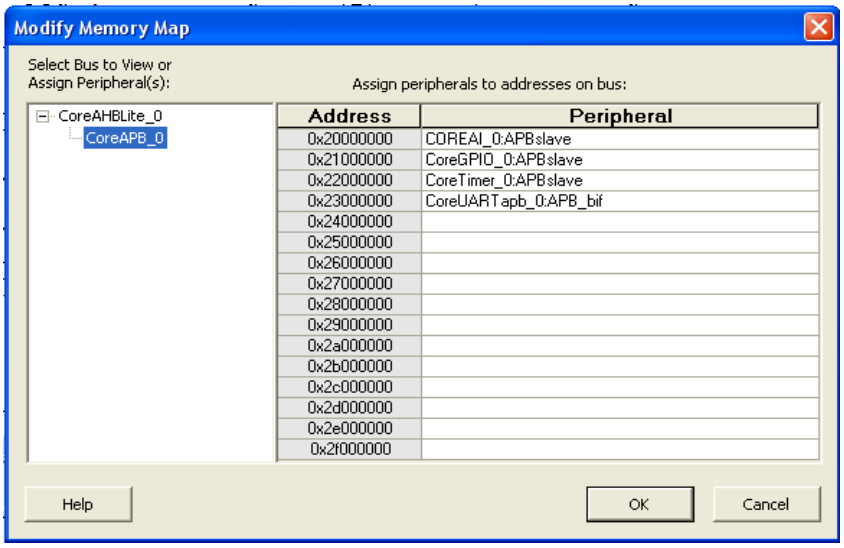
Le PLL dans le FPGA peut être utilisé pour produire de nouveaux signaux d'horloge à partir d'une source d'horloge. Pour ce design, nous configurerons le PLL pour utiliser l'Oscillateur RC (de l'étape précédente) comme horloge d'entrée et pour générer une horloge de sortie de 20 MHz (comme l'horloge du système).

ACTEL recommande que nous tenons le système dans la réinitialisation jusqu'à ce que le PLL ait fermé. Nous utilisons cette porte AND2 pour accomplir ceci.

Retrouvons maintenant dans la **Figure 4.4** la définition du plan d'adressage.



(a)



(b)

Figure 4.4 : définition du plan d'adressage : (**a)** partie1, (**b)** partie2

c) *Simulation fonctionnelle avec Modelsim*

Une fois l'étape de conception terminée, nous allons effectuer une simulation fonctionnelle du projet à l'aide d'un testbench et du simulateur Modelsim. Pour ce faire, nous allons d'abord générer un fichier testbench VHDL à l'aide du Navigateur de projet Libero IDE. Ensuite, nous configurerons et lancerons notre simulation Modelsim à partir du Navigateur Libero IDE. Une fois appelé par Libero IDE, Modelsim prendra en charge la simulation et la présentation des résultats.

d) *Implémentation du projet*

Dans cette étape, nous effectuerons la synthèse et l'implémentation du projet oscilloscope virtuel. Nous créons d'abord des contraintes de timing et les incorporer au projet. Nous spécifierons ensuite la configuration des broches d'entrées/sorties à adopter et nous générerons enfin le fichier de programmation du FPGA M1AFS1500-FGG484. L'implémentation du projet consiste à effectuer les étapes d'interprétation (translation), de mapping, de placement & routage et de génération du fichier de programmation. Tous les outils d'implémentation du projet sont intégrés au Navigateur Libero IDE.

e) *Simulation avec timing*

Une fois l'étape d'implémentation du projet terminée, nous allons effectuer une simulation avec timing. Nous réaliserons cette simulation à l'aide du testbench que nous avons créé à la section Simulation fonctionnelle avec Modelsim. La simulation avec timing (ou post placement & routage) est une étape essentielle du flot de conception pour s'assurer que le fonctionnement du projet est conforme avant de le réaliser physiquement. La simulation avec timings utilise les informations sur les délais dans les blocs FPGA et dans le routage afin de reproduire le comportement réel du projet avec grande précision. La simulation avec timing utilise l'information détaillée sur le placement & routage générée lors de l'étape d'implémentation du projet. Une fois appelé par Libero IDE, Modelsim intégrera l'information sur les timings et prendra en charge la simulation et la présentation des résultats.

f) *Configuration de la carte Fusion Embedded Development Kit avec FlashPro*

Après avoir fait la synthèse, le mapping et le placement & routage du projet d'oscilloscope numérique lors de l'étape d'implémentation du projet, nous allons générer le fichier de programmation pour configurer la carte fusion embedded development kit dans **l'ANNEXE III**.

IV.2.3. Conception de la carte fille pour l'oscilloscope

Bien que l'oscilloscope soit numérique, un circuit analogique est indispensable à l'entrée pour réaliser les couplages AC/DC ainsi que pour atténuer les tensions élevées à une valeur adéquate avant de les numériser.

Le premier étage est le couplage AC/DC montré à la **Figure 4.5**. Le couplage AC est assuré par un condensateur de 47 nF qui se comporte comme un filtre passe haut. Ce condensateur peut être passé en fermant le relais et on a alors le couplage DC.

Quand le relais est fermé, le condensateur est déchargé à travers le relais. Pour respecter le courant maximal supporté par le relais, une résistance est placée en série avec le condensateur de couplage.

La résistance de la bobine du relais (environ 91 Ω) est en série avec la résistance de 20 Ω. On a ainsi un courant de 30 mA à l'état fermé. La diode Schottky est une protection contre des piques de tension pouvant avoir lieu lorsqu'on débranche la tension d'alimentation du relais.

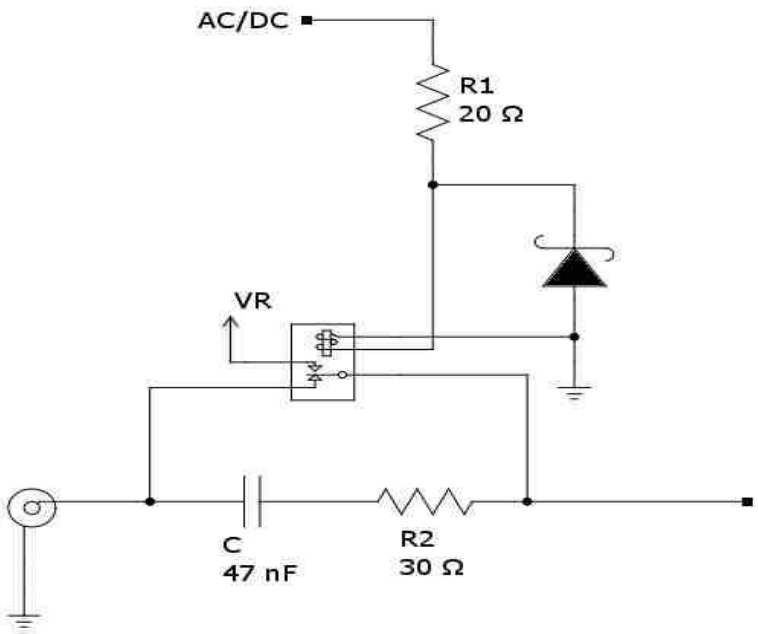


Figure 4.5 : Couplage AC/DC de l'oscilloscope

Après être filtré, le signal entre dans l'étage suivant qui est un circuit atténuateur. En effet, la tension maximale supportée par le circuit FPGA est de 16 V. Ainsi, en prenant comme tension maximale admissible 300 V, le circuit doit atténuer le signal d'entrée à un

rapport de 1/20. Pour ce faire, un pont diviseur de tension à résistances convient pour les fréquences basses et un pont diviseur de tension à condensateurs pour les fréquences élevées, comme le montre la **Figure 4.6**.

$$A_v = \frac{R_2}{R_1 + R_2} = 20 => \frac{R_1}{R_2} = 19$$

$$A_v = \frac{\frac{1}{C_2}}{\frac{1}{C_1} + \frac{1}{C_2}} = 20 => \frac{C_2}{C_1} = 19$$

En outre, les valeurs des résistances et condensateurs sont choisis pour que l'impédance d'entrée soit environ 1 MΩ et 25 pF, valeurs standard pour les oscilloscopes.

$$R_1 + R_2 = 1\ \text{M}\Omega$$

$$\frac{C_1 C_2}{C_1 + C_2} = 24\ \text{pF}$$

Ainsi nous avons à résoudre les systèmes d'équations

$$\frac{R_1}{R_2} = 19$$

$$R_1 + R_2 = 1\ \text{M}\Omega$$

$$\frac{C_2}{C_1} = 19$$

$$\frac{C_1 C_2}{C_1 + C_2} = 24\ \text{pF}$$

$$R_1 = 950\ \text{k}\Omega$$

$$R_2 = 50\ \text{k}\Omega$$

$$C_1 = 25{,}26 \text{ pF}$$

$$C_2 = 480 \text{ pF}$$

En les ajustant aux valeurs standards, on aura

$R_1 = 953 \text{ k}\Omega$ (série E48)

$R_2 = 51 \text{ k}\Omega$ (série E24)

$$C_1 = 24{,}7 \text{ pF}$$

$$C_2 = 470 \text{ pF}$$

Pour plus de précision (et pour surmonter les tolérances des condensateurs) on peut utiliser un condensateur variable pour C1 en l'ajustant à environ 24,7 pF de manière à avoir une atténuation de 1/20 pour les fréquences élevées.

Pour mesure de précaution, l'étage est protégé des tensions au-delà de ± 15,6 V, sinon le circuit FPGA risque d'être endommagé par une surtension. Ceci est fait en écrêtant les signaux par deux diodes Zener de 15V et deux diodes ordinaires qui sont à l'état ON lorsque la tension de sortie est inférieure à -15,6 V ou supérieure à 15,6 V, l'une des deux diodes Zener, montée en inverse alimentée par une tension de 15,4 V stabilise alors la tension.**[22]**

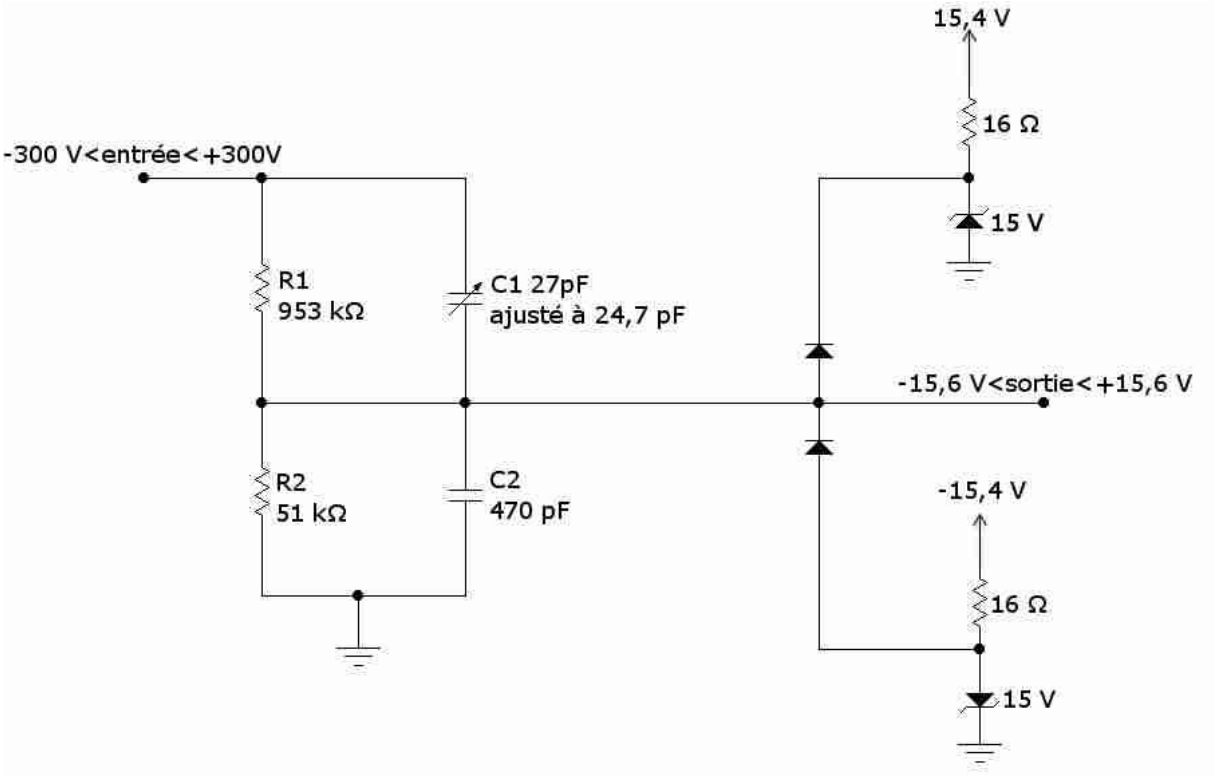


Figure 4.6 : Circuit atténuateur à l'entrée de l'oscilloscope

Notons que notre carte Fusion Embedded Development Kit intègre déjà un CAN (cf. **ANNEXE I**).

IV.3. Conception et développement de la partie logicielle

IV.3.1. Synoptique du système à concevoir

La **Figure 4.7** montre le schéma synoptique de l'architecture logiciel de l'oscilloscope. Le système logiciel est divisé en plusieurs modules faisant chacun un traitement spécifique :

- La visualisation des courbes sert à visualiser la courbe correspondant aux signaux mesurés. Cette forme dépend à la fois des signaux et des paramètres d'affichage.
- Les paramètres d'affichage servent à varier les échelles des axes, les décalages sur le cadre d'affichage. Ces paramètres sont traités par le logiciel pour modifier les échelles et les positions de l'affichage. Ils n'ont pas besoin d'être envoyés à l'instrument reconfigurable.
- Les contrôles de l'instrument sont l'ensemble des commandes pour le contrôle des mesures et sont traités par l'instrument reconfigurable. Ainsi ces paramètres sont envoyés à l'instrument reconfigurable et celui-ci réagit instantanément.

- Le driver gère les flux d'entrée/sortie des données entre le PC et l'instrument reconfigurable via le port USB.

  L'algorithme général de ces traitements est présenté dans l'**ANNEXE IV**. On peut voir dans la **Figure 4.7** le synoptique fonctionnel du système logiciel.

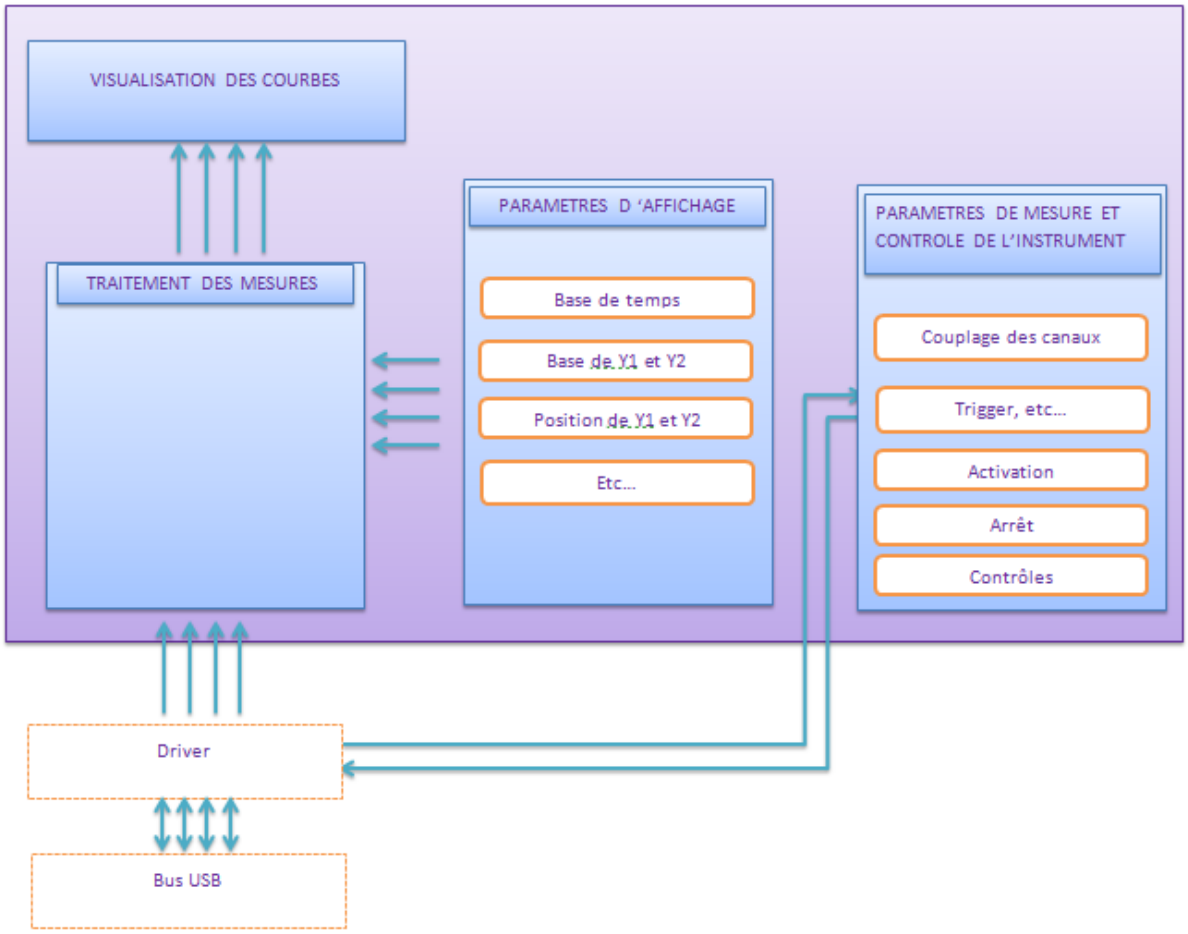
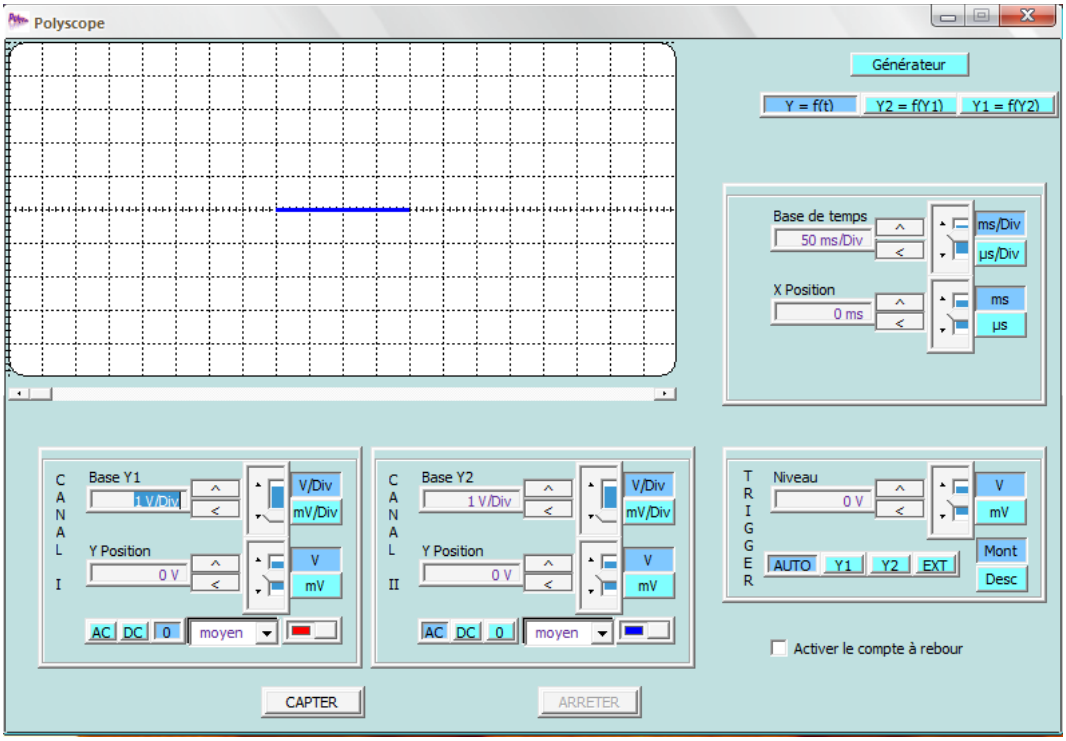


Figure 4.7 : Schéma synoptique du système logiciel de l'oscilloscope

IV.3.2. Présentation de l'interface utilisateur de l'oscilloscope virtuel

L'interface utilisateur de l'oscilloscope se présente comme suit



Ces contrôles et leurs fonctionnalités sont détaillés ci – dessous :

- **La zone d'affichage**

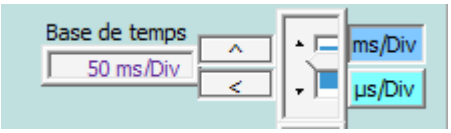
La zone d'affichage sert à afficher la ou les courbes résultant de la mesure. Elle est un rectangle de 20 x 10 divisions

On y trouve deux axes principaux: l'axe horizontal X qui sert de repère de temps pour le mode Y = f(t) ou bien un des deux signaux pour chacun des modes Y2 = f(Y1) ou Y1 = f(Y2). L'axe vertical Y sert toujours de repère des tensions mesurées.

L'axe X a 20 divisions. On utilise le défilement horizontal au dessous de la zone d'affichage pour voir au delà du cadre. On peut aussi régler le paramètre base de temps pour mieux cadrer l'oscillogramme dans le cadre d'affichage.

L'axe Y a 10 divisions. Pour ajuster les valeurs dans le cadre, on règle les paramètres des canaux expliqués ci - dessous.

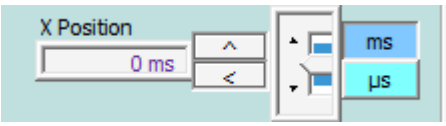
- **Base de temps**



La base de temps (en milliseconde par division ou microseconde par division) contrôle l'échelle de l'axe horizontale de l'oscilloscope si le mode Y = f(t) est activé.

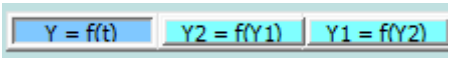
Pour obtenir un bon affichage, on ajuste base de temps en proportion inverse de la fréquence du signal capté. Plus la fréquence est élevée, plus la base de temps doit être petite. Par exemple si on veut voir un cycle d'un signal de 50Hz, une base de temps égale à 1 milliseconde conviendra.

- **X Position**



Ce paramètre (en milliseconde ou microseconde) configure le point de départ du signal dans l'axe X. Quand cette valeur est 0, le signal débute à l'extrémité gauche de l'axe X. Une valeur positive décale le point de départ à droite, une valeur négative décale le point de départ à gauche.

- **Y = f(t) / Y1 = f(Y2) / Y2 = f(Y1)**



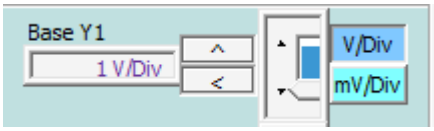
Les axes de l'oscilloscope peuvent afficher soit les valeurs des tensions mesurées en fonction du temps soit l'entrée d'un canal en fonction de celle de l'autre. Dans le cas où Y1 =f(Y2) est sélectionné, l'échelle de l'axe X est évalué en Volt par Division pour le canal 2 (et vice versa).

- **Y = f(t)** : Affichage des deux signaux en fonction du temps. C'est le mode par défaut.
- **Y2 = f(Y1)** : Affichage de la courbe représentant la variation du signal du premier canal en fonction de celui du second canal.
- **Y2 = f(Y2)** : Affichage de la courbe représentant la variation du second canal en fonction du premier.

- **Paramètres des canaux I et II**

Comme notre oscilloscope admet deux canaux, on peut mesurer et afficher en même temps deux signaux.

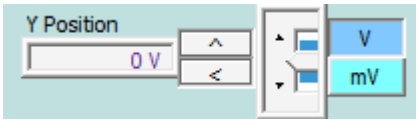
- **Base de Y**



Ce paramètre (en Volt par division ou millivolt par division) détermine l'échelle de l'axe Y. Il contrôle aussi l'échelle de l'axe X si le mode Y1 = f(Y2) ou Y1 = f(Y1) est sélectionné.

Pour avoir un bon affichage, on ajuste les échelles en tenant compte des valeurs estimées de la mesure. Par exemple, un signal alternative d'amplitude 5 Volts remplit l'affichage de l'oscilloscope quand l'échelle de l'axe Y est réglé à 1V/Div. Si l'échelle augmente, l'oscillogramme devient plus petit. Si elle diminue, l'oscillogramme s'agrandit et risque d'être coupé si l'échelle est trop petite.

- **Y position**



Ce paramètre (en Volt ou millivolt) contrôle le point d'origine de l'axe Y. La valeur par défaut est 0. Augmenter Y position déplace le point d'origine (0V) en dessus de l'axe X. La diminuer déplace le point d'origine (0V) en dessous de l'axe X.

En outre, attribuer des valeurs différentes aux Y position des canaux 1 et 2 permet de mieux distinguer les deux oscillogrammes.

- **AC / 0 / DC**

Ces paramètres déterminent l'impédance d'entrée de l'oscilloscope.

- Avec le couplage AC, seul le composant alternatif du signal est affiché. Le couplage AC a l'effet de placer un condensateur en série avec la sonde de l'oscilloscope. Comme un oscilloscope physique, en utilisant le couplage AC, le premier cycle affiché peut ne pas être très précis. Une fois le composant continu du

signal est calculé et éliminé durant le premier cycle, l'oscillogramme devient précis.

Le couplage AC est utile quand on veut mesurer une petite variation alternative d'un signal autour d'une valeur constante, par exemple le bruit d'un générateur de tension continue.

- Avec le couplage DC, la somme des composants continu et alternatif du signal est affichée.
- Sélectionner 0 affiche une ligne horizontale référence du point d'origine défini par Y position.

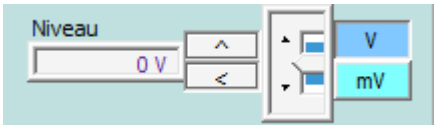
- **Sélection de l'épaisseur et de la couleur de l'oscillogramme**

Ces contrôles permettent le choix de l'épaisseur de la courbe : fine, moyenne, ou grasse ainsi que sa couleur, ceci afin de fournir plus d'accessibilité. En outre, pour mieux discerner les deux courbes, il est pratique de les colorer différemment.

- **Trigger (Déclenchement)**

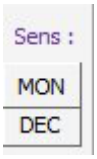
Ces paramètres déterminent les conditions sous lesquelles le premier oscillogramme est affiché sur l'oscilloscope. On utilise surtout cette fonction pour étudier les signaux périodiques.

- **Niveau du trigger**



Le niveau du trigger est le point de l'axe Y de l'oscilloscope devant être atteint par le signal avant que celui-ci soit affiché.

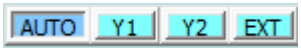
- **Sens du trigger**



- sens montant : Pour commencer à afficher l'oscillogramme lorsque la valeur est atteinte dans le sens croissant.

- sens descendant : Pour commencer à afficher l'oscillogramme lorsque la valeur est atteinte dans son sens décroissant.

- **AUTO / Y1 / Y2 / EXT**



- AUTO : l'affichage est automatique sans tenir compte du trigger. En d'autre terme, le trigger est désactivé. C'est le mode par défaut.

- Y1 : Le trigger s'applique au signal du canal 1.

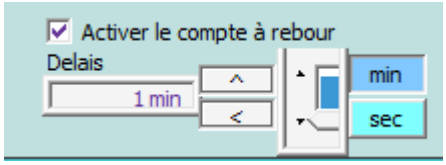
- Y2 : Le trigger s'applique au signal du canal 2.

- EXT : Cette option est pour le trigger extérieure, une entrée reliée à l'oscilloscope physique. Le trigger s'applique alors à ce signal extérieur.

- **CAPTER / ARRETER**

En cliquant sur capter l'instrument acquiert les mesures et affiche en même temps les oscillogrammes correspondant ainsi que les statistiques sur les mesures. Il attend qu'on appui sur le bouton 'Arrêter' pour interrompre l'acquisition.

On peut faire l'acquisition d'une autre manière si besoin est : En cochant la case 'Activer le compte à rebours', on peut régler la durée au cours de laquelle l'acquisition va s'effectuer. Le compte à rebours va alors démarrer la mesure s'arrête quand sera épuisée la durée définie.



CHAPITRE V. REALISATION D'UN GENERATEUR DE FORME D'ONDE

V.1. Généralité

Un générateur de signaux classiques, appelé aussi générateur de fonctions, ne délivre que des formes d'ondes pour lesquels il a été conçu. Si le test d'un matériel réclame un signal de forme particulière, le recours à un générateur de signaux arbitraires est nécessaire. Celui-ci délivrera le signal que l'utilisateur aura préalablement défini et stocké dans sa mémoire. Le choix de l'appareil sera dicté par les caractéristiques du signal à mettre en œuvre mais aussi par la souplesse et les performances des outils de création et de génération.

Tout scientifique le sait. Il y a souvent un gap entre la théorie et la pratique. Sur le papier un équipement électronique réagit par une réponse attendue à l'excitation de signaux minutieusement étudiés. Mais dans la vraie vie, ces signaux ne sont pas toujours si fidèlement reproduits, ils peuvent être bruités ou subir des perturbations. Parfois, ils ont des formes d'ondes si particulières que les générateurs de fonctions traditionnels sont incapables de les générer. On fait alors appel à des générateurs de signaux arbitraires. A y regarder de plus près, ces signaux sont loin d'être arbitraires puisqu'ils ont été définis par l'utilisateur pour que leurs formes d'ondes répondent exactement aux particularités de son application.

Pour tester un matériel électronique, quoi de plus naturel que d'appliquer, sur ces entrées, des signaux auxquels il sera confronté dans son environnement d'exploitation futur. Et d'y ajouter quelques parasites afin de s'assurer de sa résistance aux perturbations.

Délivrer des signaux de formes spécifiques voire complexes qui sortent des gabarits traditionnellement fournis par les générateurs de fonctions (sinus, carré, triangle notamment), tel est le rôle des générateurs de signaux arbitraires. On ne les trouve pas seulement dans les laboratoires de conception pour éprouver un composant mécanique aux vibrations et aux chocs ou pour valider des calculateurs embarqués dans les véhicules. En production, ils peuvent être intégrés dans des bancs de tests de matériel pour l'automobile, le secteur médical, l'aéronautique, les télécommunications On les rencontre également dans les centres d'essais et les laboratoires pour tester les prototypes, exciter l'électronique des capteurs, mettre en marche des actionneurs, alimenter des dispositifs électromécaniques, rejouer des séquences d'acquisition

Ces instruments évoluent donc avec les exigences technologiques des équipements qu'ils doivent contrôler. Ainsi, les générateurs arbitraires doivent délivrer des signaux de plus

en plus biscornus mettant en œuvre des modulations particulières. Ils doivent offrir une résolution entre deux points de plus en plus fine et des fréquences d'échantillonnage de plus en plus élevées,

Pour ce genre d'appareil, on parle effectivement, comme pour les oscilloscopes numériques, de résolution et de fréquences d'échantillonnage. Il faut dire que l'architecture d'un générateur arbitraire s'approche de l'image d'un oscilloscope vue dans un miroir. Les deux appareils exploitent un mécanisme similaire mais opérant en sens inverse. L'oscilloscope acquiert les signaux analogiques, les convertit en valeurs numériques, les stocke en mémoire et les affiche sur son écran. Le générateur réalise les mêmes opérations mais en sens inverse. On commence par créer la forme d'onde que l'on souhaite générer. Ensuite, les valeurs numériques de cette forme d'onde sont enregistrées dans la mémoire de l'appareil. Le convertisseur numérique/analogique transforme ensuite, à la fréquence d'échantillonnage appropriée, les données numériques en un signal analogique. En toute logique, la profondeur mémoire est liée à la fréquence d'échantillonnage. Plus on va vite plus la taille mémoire doit être importante.

Spécifications

Un générateur de forme d'onde est donc un appareil dont le rôle est de fournir des tensions périodiques dont l'allure, l'amplitude, et la fréquence sont configurables.

Pour l'allure des signaux à générer, il en existe une infinité de sorte qu'il est impossible de tous les générer. On constate cependant que trois allures sont principalement utiles pour les expériences en électronique : les tensions sinusoïdales, triangulaires et carrées. Qui plus est ces deux dernières sont encore paramétrables pour donner une variété de formes d'ondes.

L'onde sinusoïdale est considérée comme l'onde fondamentale dans la nature : toutes les ondes électromagnétiques naturelles et artificielles sont sinusoïdales, la tension du secteur est sinusoïdale.

Une onde triangulaire monte linéairement d'une valeur basse en une valeur haute puis redescend linéairement jusqu'à la valeur basse en une période. La proportion du temps de monté et le temps de descente en une période est configurable par le rapport de cycle (en % allant de 1% à 99%). Dans le cas extrême où ce rapport est égal à 1% ou 99% on obtient des signaux en dent de scie.

Les ondes carrées prennent deux valeurs haute et basse à intervalle régulier. On les appelle aussi signaux à créneaux. En variant le rapport de cycle, comme pour les ondes triangulaire, on spécifie le rapport entre le temps du créneau haut et le temps du créneau bas en une période.

La plage des fréquences varie largement pour les générateurs de forme d'onde. Il y a les générateurs basse fréquence (GBF) fournissant des signaux jusqu'à 200 Hz. Certains autres peuvent générer des signaux à des fréquences élevés jusqu'à 1 MHz. En fait, comme les oscilloscopes, les générateurs numériques de forme d'onde ont du mal à produire des ondes ayant des fréquences élevées.

L'amplitude est la différence entre le niveau de tension maximum et le niveau du composant continu défini par l'Offset. Si les sorties sont connectées au broche commun et l'une des sorties positive ou négative, la valeur crête à crête du signal généré sera le double de l'amplitude. Si le signal est prélevé des broches de sortie positive et négative, la valeur crête à crête du signal généré sera le quadruple de l'amplitude. L'amplitude peut aller de quelques millivolts à quelques centaines de volts.

L'Offset détermine la composante continue du signal autour duquel va osciller la composante alternative. C'est en quelque sorte comme le signal alternatif est décalé de cette valeur.

Pour permettre à l'utilisateur de prévisualiser la forme d'onde qu'il paramètre, l'interface devra être munie d'un cadre d'affichage de courbe analogue à celui de l'oscilloscope.

V.2. Conception et développement de la partie matérielle

V.2.1. Synoptique du système à concevoir

La **Figure 5.1** représente le schéma synoptique du générateur de forme d'onde.

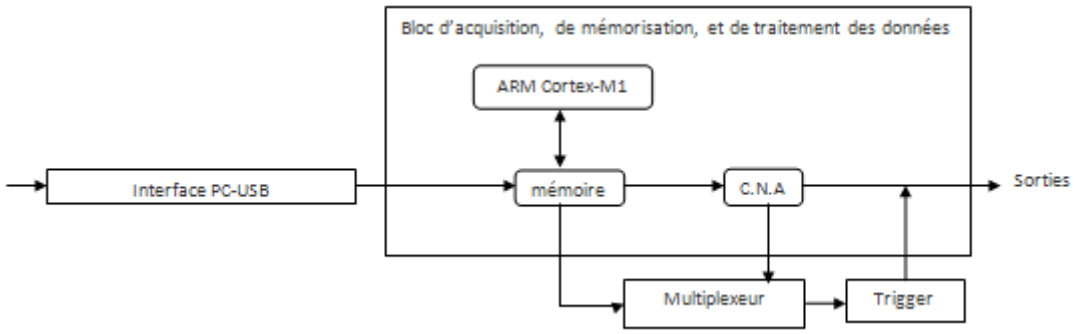


Figure 5.1 : Schéma synoptique du système matériel du générateur de forme d'onde

V.2.2. Le flot de conception du générateur de forme d'onde (Hardware design flow)

La réalisation du générateur de forme d'onde suit la même démarche que celle de l'oscilloscope. En effet, on utilise aussi l'approche de conception FPGA par schéma. La **Figure 5.2** suivante montre le schéma bloc du générateur de forme d'onde.

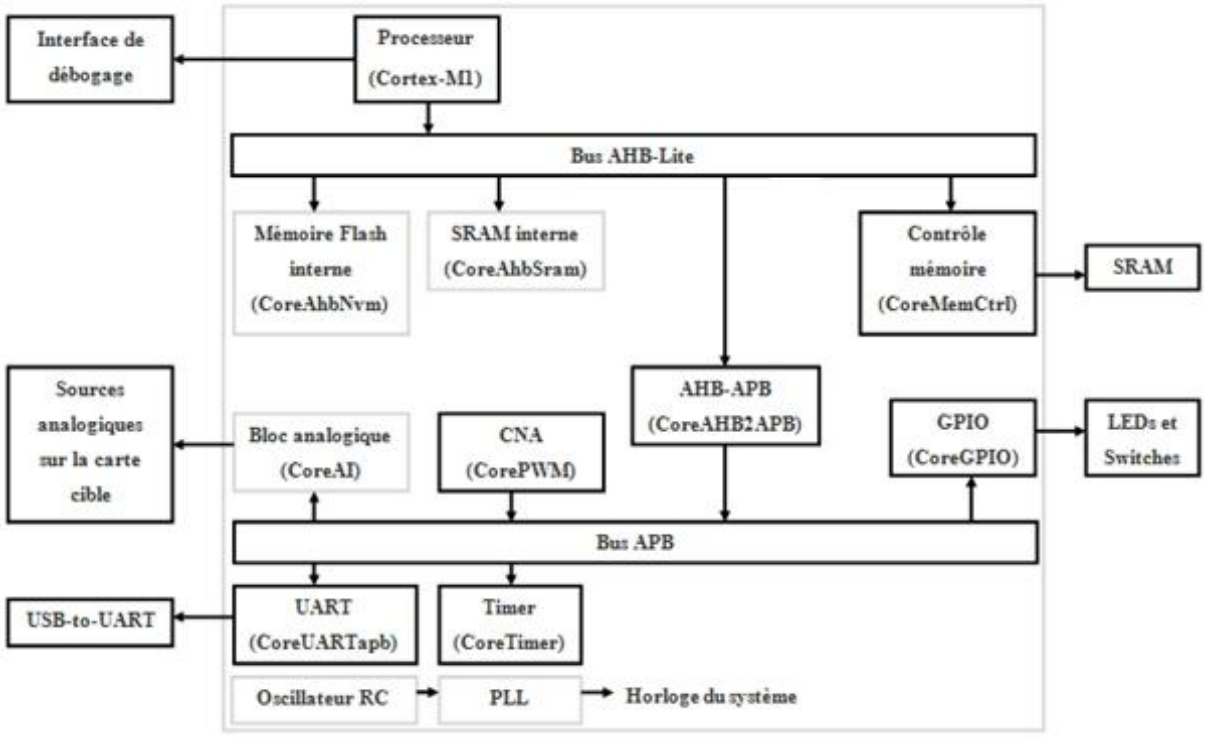


Figure 5.2 : Schéma Bloc du générateur de forme d'onde

Notons que le CorePWM représente un CNA (Convertisseur Numérique Analogique). Le schéma top level du générateur de forme d'onde dans l'**ANNEXE III** nous montre ceci plus clairement.

De plus, les blocs : CoreAhvNvm, CoreAhbSram, CoreAI, Oscillateur RC, PLL sont aussi des périphériques « Hard IP » tandis que les autres blocs représentent des périphériques « Soft IP ».

V.2.3. Conception de la carte Fille pour le générateur de forme d'onde

Etant donné que la valeur maximale de tension que le circuit FPGA peut fournir est de 15 V, il faut amplifier cette tension pour pouvoir générer des tensions plus élevées. Ainsi nous allons concevoir un circuit analogique amplificateur élevant cette valeur à 40V. Le circuit de la **Figure 5.3** est à base d'un amplificateur opérationnel PR2201 qui est un modèle d'un amplificateur opérationnel pour des tensions élevées. La valeur maximale de la tension de sortie du circuit amplificateur est donc de 40 V. Donc on souhaite avoir un gain de 40/15 soit 2,67.

Le gain du circuit de la **Figure 5.3** est donné par $1 + \frac{R3}{R2} = 2{,}67$

Soit $\frac{R3}{R2} = 1{,}67$

Ainsi une résistance de 3 kΩ conviendra pour R2 et une de 5 kΩ pour R3

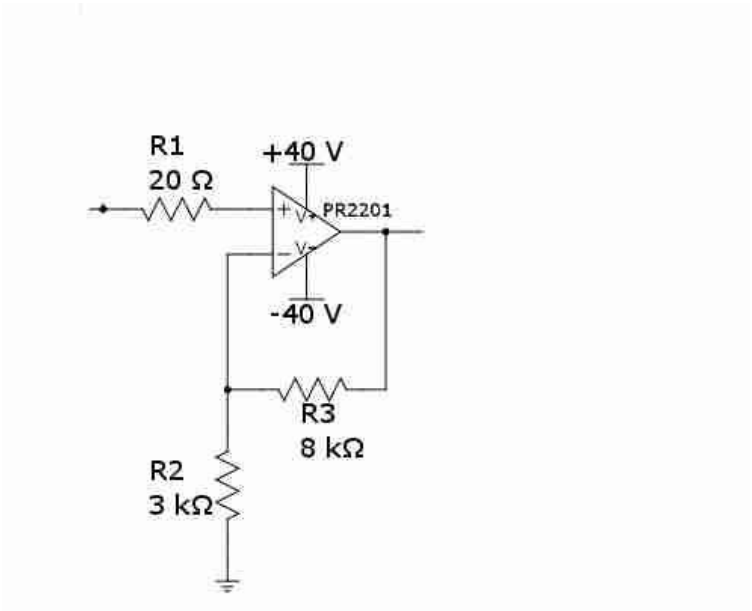


Figure 5.3 : Amplificateur à la sortie du générateur de forme d'onde

V.3. Conception et développement de l'interface utilisateur de la partie logicielle

V.3.1. Synoptique du système à concevoir

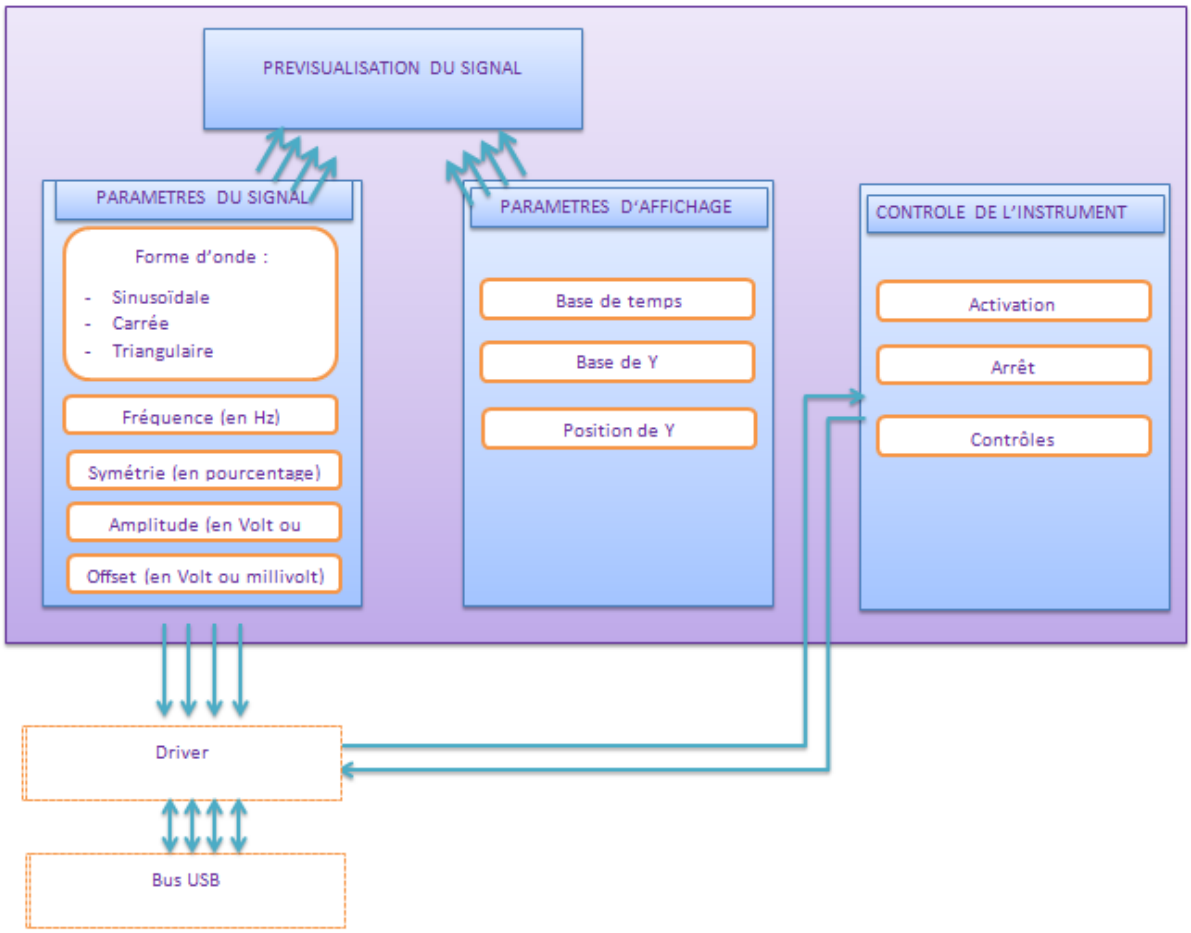


Figure 5.4 : Schéma synoptique du système logiciel du générateur de forme d'onde

La **Figure 5.4** montre le schéma synoptique de l'architecture logiciel du générateur de forme d'onde. Le système logiciel est divisé en plusieurs modules analogues à ceux de l'oscilloscope :

- La visualisation des courbes se sert de prévisualiser la forme d'onde paramétrée. Cette forme dépend à la fois des paramètres du signal à générer et des paramètres d'affichage pour mieux cadrer la prévisualisation.
- Les paramètres du signal : Ce module comprend l'allure du signal, sa fréquence, son amplitude et son offset. Seuls ces paramètres sont -envoyés à l'instrument reconfigurable pour générer l'onde. Les paramètres d'affichage ne servent qu'à ajuster la prévisualisation de la courbe pour mieux la présenter.

- Les paramètres d'affichage, comme expliqué plus haut, servent à varier les échelles sur le cadre d'affichage et non les paramètres du signal. Ces paramètres affectent seulement la prévisualisation du signal et non le signal lui-même.

- Les contrôles de l'instrument sont l'ensemble des commandes que l'utilisateur peut envoyer à l'instrument pour le manier. Ces commandes sont prises en charge par l'instrument reconfigurable pour démarrer ou arrêter la génération du signal par exemple.

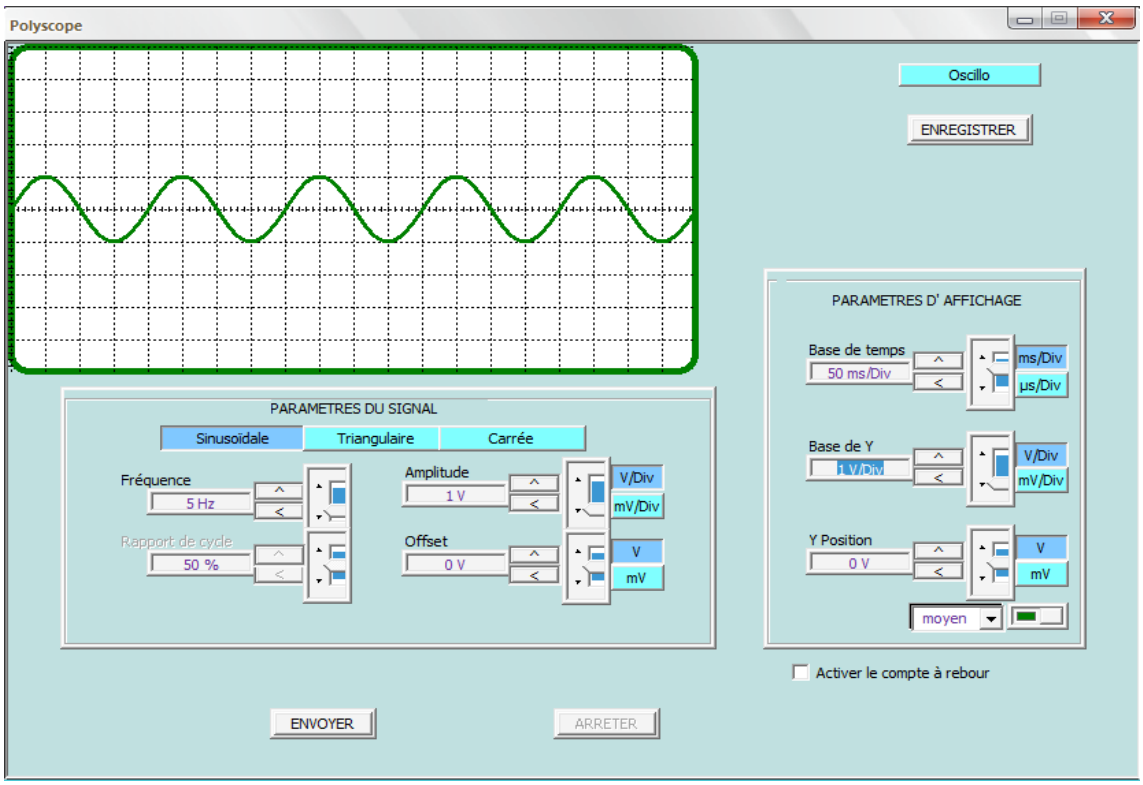
- Le driver gère les flux d'entrée/sortie des données entre le PC et l'instrument reconfigurable via le port USB.

  Pour l'algorithme des traitements voir l'**ANNEXE IV**.

V.3.2. Présentation de l'interface utilisateur du générateur de forme d'onde

Pour accéder au générateur de forme d'onde on appuie sur l'onglet générateur en haut à droite de la fenêtre.

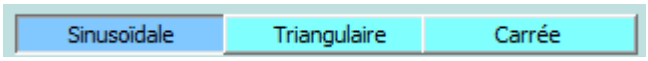
Voici alors ce à quoi ressemble l'interface obtenue :



Cet instrument sert à générer un signal vers l'instrument physique connecté à l'ordinateur.

Il permet également de visualiser le signal paramétré avant de l'envoyer. C'est pourquoi les contrôles d'affichage du premier canal de l'oscilloscope et les contrôles du temps sont encore présents. En plus les nouveaux réglages sont les suivants :

- **Sélection de la forme d'onde**



L'allure du signal peut être sinusoïdale, triangulaire ou carrée. Ce choix s'effectue sur clic su l'un de ces trois boutons.

- **Sinusoïdale** : L'onde sinusoïdale est considéré comme l'onde fondamentale pour quelques raisons. Il a les propriétés harmoniques mathématiques. Les tensions du secteur sont sinusoïdales. Les ondes électromagnétiques sont sinusoïdales.
- **Triangulaire** : Un signal triangulaire monte linéairement d'une valeur basse en une valeur haute puis redescend linéairement jusqu'à la valeur basse en une période.
- **Carrée** : Un signal carré est basiquement un signal qui prend deux valeurs haute et basse à intervalle régulier. Un exemple d'application d'un tel signal est pour le test d'un amplificateur – Un bon amplificateur augmente l'amplitude d'une onde carrée avec un minimum de distorsion.

On note que, pour les cas des ondes carrée et triangulaire, l'allure varie notablement en fonction du rapport de cycle (voir ci-après)

- **Rapport du cycle** (1% - 99%)

C'est le rapport des demi-périodes pour les ondes triangulaire et carrées.

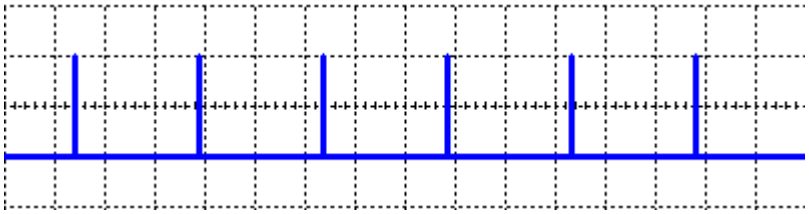
Pour les ondes carrées, il s'agit du rapport entre la durée du créneau haut et celle du créneau bas. Quand le rapport du cycle est de 50 %, on obtient un signal parfaitement carrée; pour une valeur différente, le signal sera rectangulaire.

Pour les ondes triangulaires, il s'agit du rapport entre du monté du signal et sa descente.

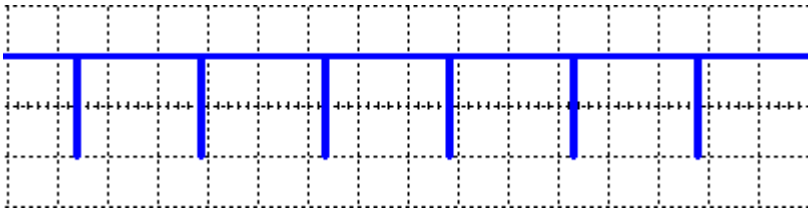
Cette option est désactivée pour les ondes sinusoïdales.

Ainsi, en variant ce paramètre, on obtient des signaux d'allures variés. Quelques cas extrêmes mérites d'être mentionnés :

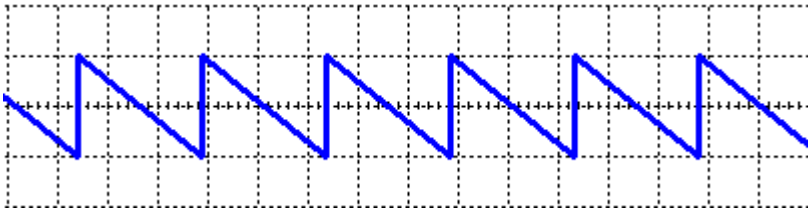
- Pour un signal carré avec un rapport de cycle minimal (1 %), on obtient un signal impulsionnel périodique positif:



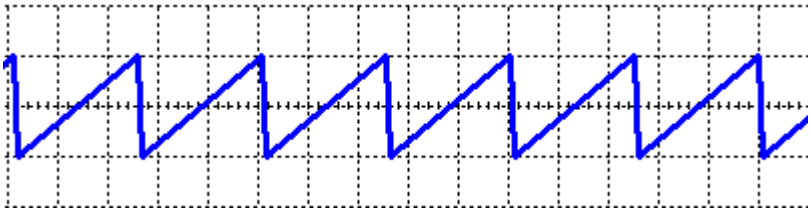
- Pour un signal carré avec un rapport de cycle maximal (99 %), on obtient également un signal impulsionnel périodique mais cette fois-ci négatif.



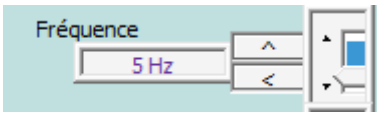
- Pour un signal triangulaire avec rapport un de cycle minimal (1 %), on a un signal dit en dent de scie :



- Pour un signal triangulaire avec un rapport de cycle maximal (99 %), on a de nouveau un signal en dent de scie mais de sens montant.

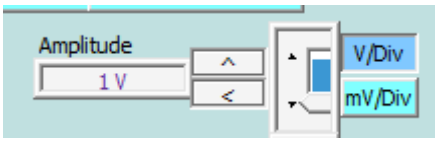


- **Fréquence** (1Hz – 100Hz)



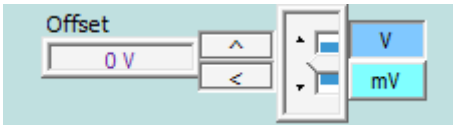
C’est le nombre de cycles par seconde du signal généré.

- **Amplitude** (1mV – 40V) :



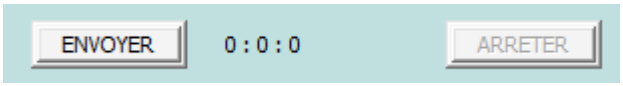
Ce paramètre règle le voltage du signal mesuré comme la différence entre le niveau de tension maximum et le niveau du composant continu défini par l’Offset. Si les sorties sont connectées au broche Commun et l’une des sorties positive ou négative, la valeur crête à crête du signal généré sera le double de l’amplitude. Si le signal est prélevé des broches de sortie positive et négative, la valeur crête à crête du signal généré sera le quadruple de l’amplitude.

- **Offset** (-20V - +20V) :

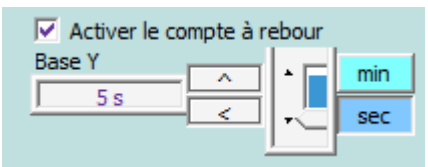


Le niveau du composant continu autour duquel va varier le signal alternatif généré. Une valeur 0 (qui est la valeur par défaut) positionne le signal le long de l’axe X. Une valeur positive le décale vers le haut tandis qu’une valeur négative le décale vers le bas.

- **Envoyer**



On clique sur le bouton ‘Envoyer’ après avoir paramétré l’allure de la forme d’onde pour l’envoyer à l’instrument physique connecté à l’ordinateur. Comme dans le cas de l’oscilloscope, on peut l’envoyer immédiatement ou bien régler le temps durant lequel le signal va être activé puis il s’arrête automatiquement



CONCLUSION

La technologie des circuits FPGA ouvre de nouvelles opportunités dans le domaine des instrumentations scientifiques. Bien que le concept des instrumentations virtuelles reconfigurables ne soit pas nouveau, ce n'est que récemment que nous avons atteint la possibilité d'implémenter un système RVI basé sur la technologie FPGA à un rapport cout/performance raisonnable. Toutefois le reste du prix à payer pour un tel système est l'effort requis pour développer le système logiciel/matériel pour un nouvel instrument.

Nous avons montré à travers la description de l'architecture d'un RVI comment les FPGA peuvent être utilisés pour émuler les instrumentations électroniques et scientifiques. L'oscilloscope et le générateur de forme sont des exemples que nous avons choisis étant donné que ces instruments sont parmi les plus utilisés et sont indispensables dans divers domaines d'application. Pour interfacer l'instrument reconfigurable, nous avons choisi la solution USB. Le débit offert par ce bus est suffisant pour les applications courantes mais pour les fréquences élevées, on aura recours au bus PCI dont le débit est nettement plus élevé.

Grâce à une plateforme RVI, il est possible d'émuler beaucoup d'instruments standards mais il est aussi possible d'implémenter des instrumentations sophistiquées pour des usages spécifiques qui ne peuvent être accomplis par les instruments standards. Une plateforme RVI représente une solution à bas prix pour ceux qui ne peuvent pas se procurer des instrumentations électroniques et scientifiques à prix élevé, ce qui arrive souvent dans les universités et les instituts de recherche dans des pays en voie de développement.

Enfin, dans le cadre spécifique de l'enseignement des sciences de l'ingénieur, l'Instrumentation Virtuelle permet de faire évoluer la démarche pédagogique avec des outils et des méthodes largement diffusés dans l'industrie. Enfin, pour l'étudiant, c'est l'assurance de posséder des connaissances et un savoir-faire utilisables dans une très large gamme de domaines d'application.

ANNEXE I: PRESENTATION DU FUSION EMBEDDED DEVELOPMENT KIT

Les figures : **Figure A.1.1** et **Figure A.1.2** ci-après représentent les couches extrêmes des huit couches du circuit de la carte Fusion Embedded Development Kit.

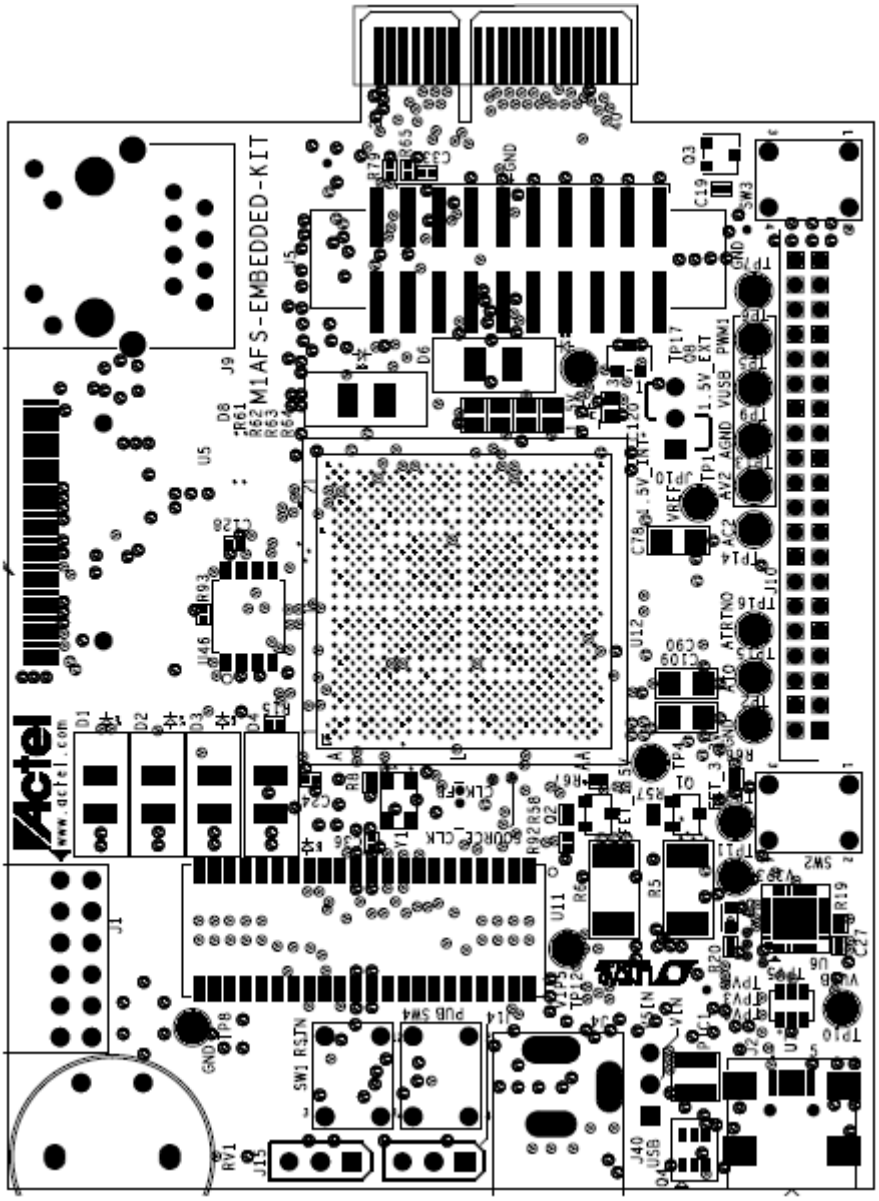
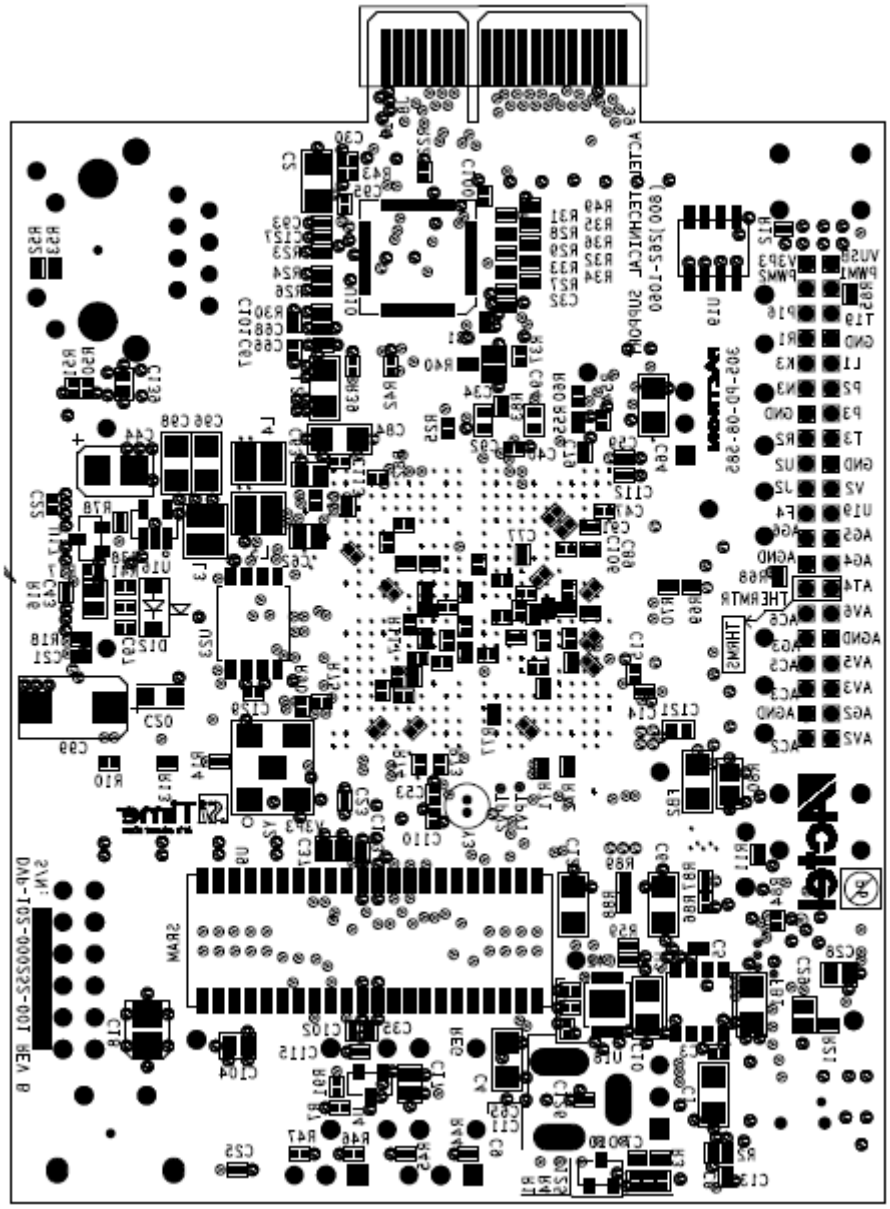


Figure A.1.1 : couche extrême supérieure du circuit de la carte Fusion Embedded Development Kit



<u>Figure A.1.2</u> : couche extrême inférieure du circuit de la carte Fusion Embedded Development Kit

Les figures Figure A.1.3 à Figure A.1.9 nous montrent les différentes broches d'entrées, de sorties, d'entrées/sorties nécessaires à la réalisation de nos circuits de conception.

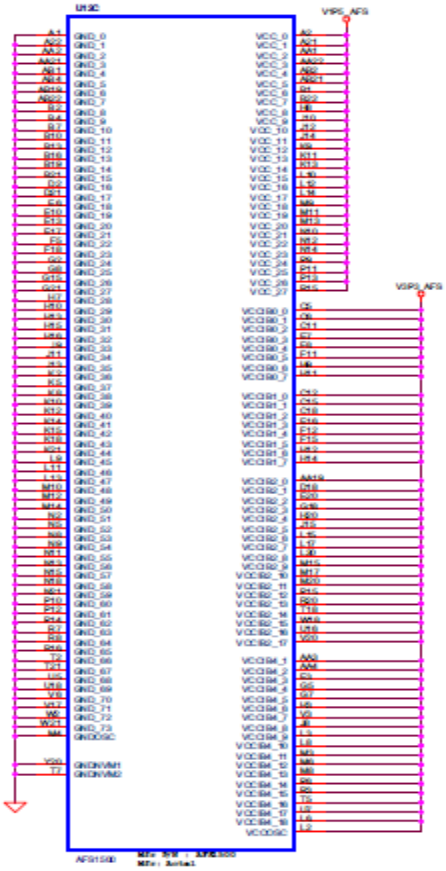


Figure A.1.3: Digital Power Supply Pins Schematic

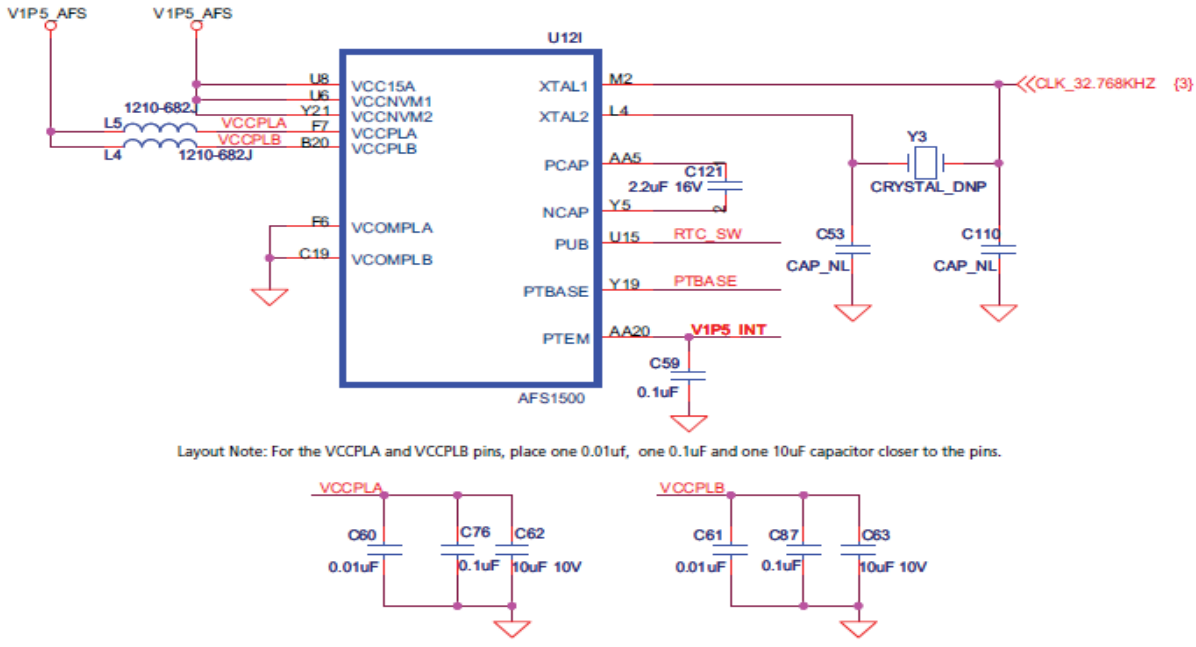


Figure A.1.4: Digital Signal Pins Schematic

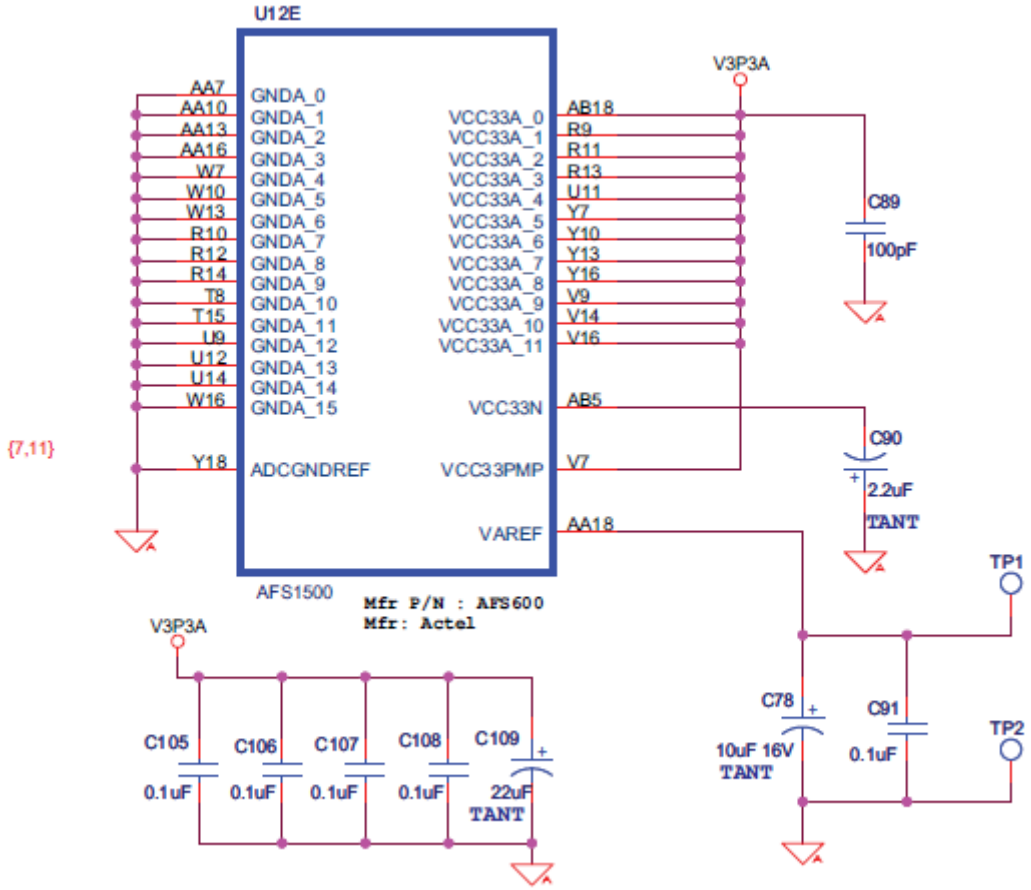


Figure A.1.5: Analog Power Supply Pins Schematic

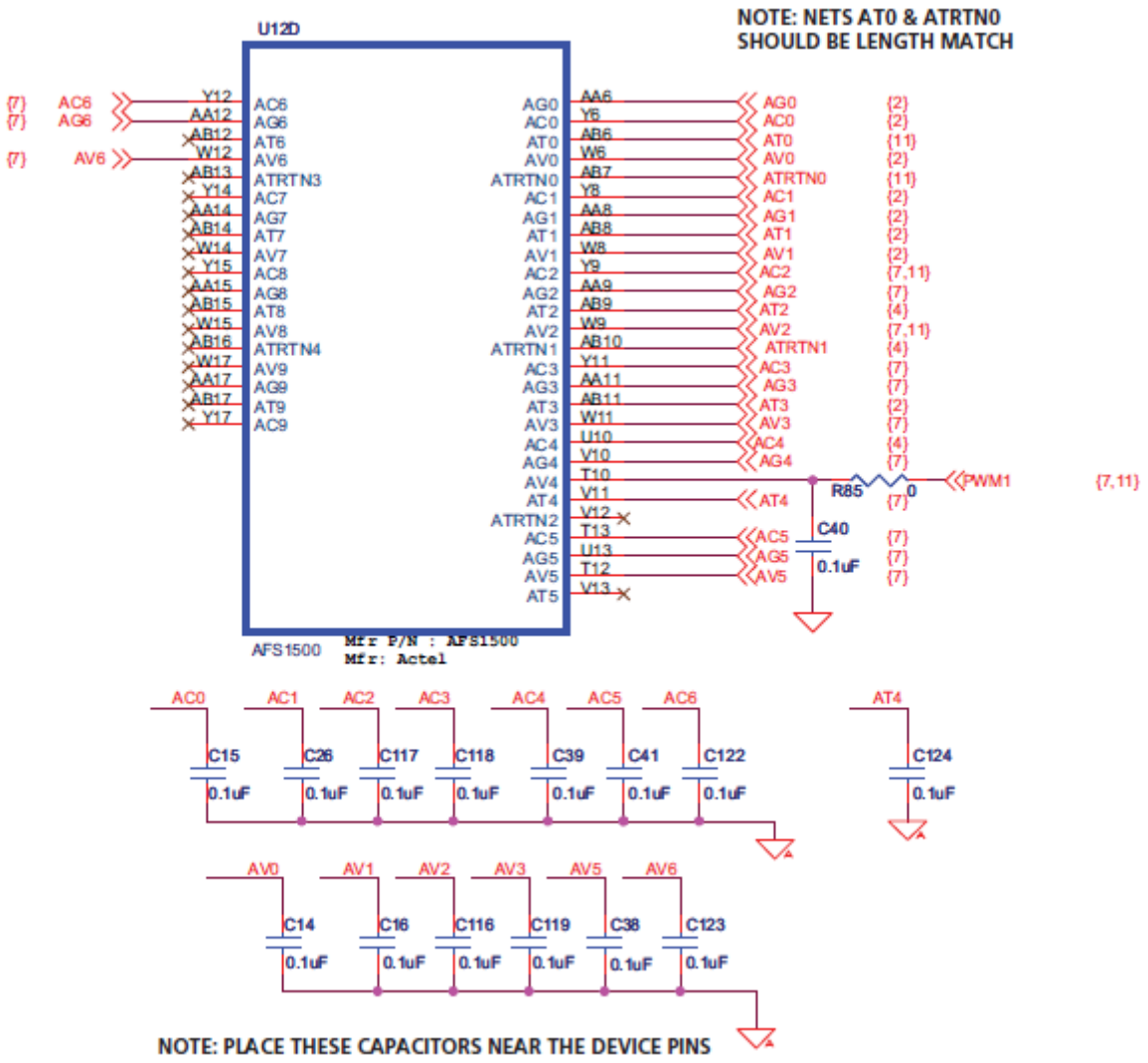


Figure A.1.6: Analog Supply Pins Schematic

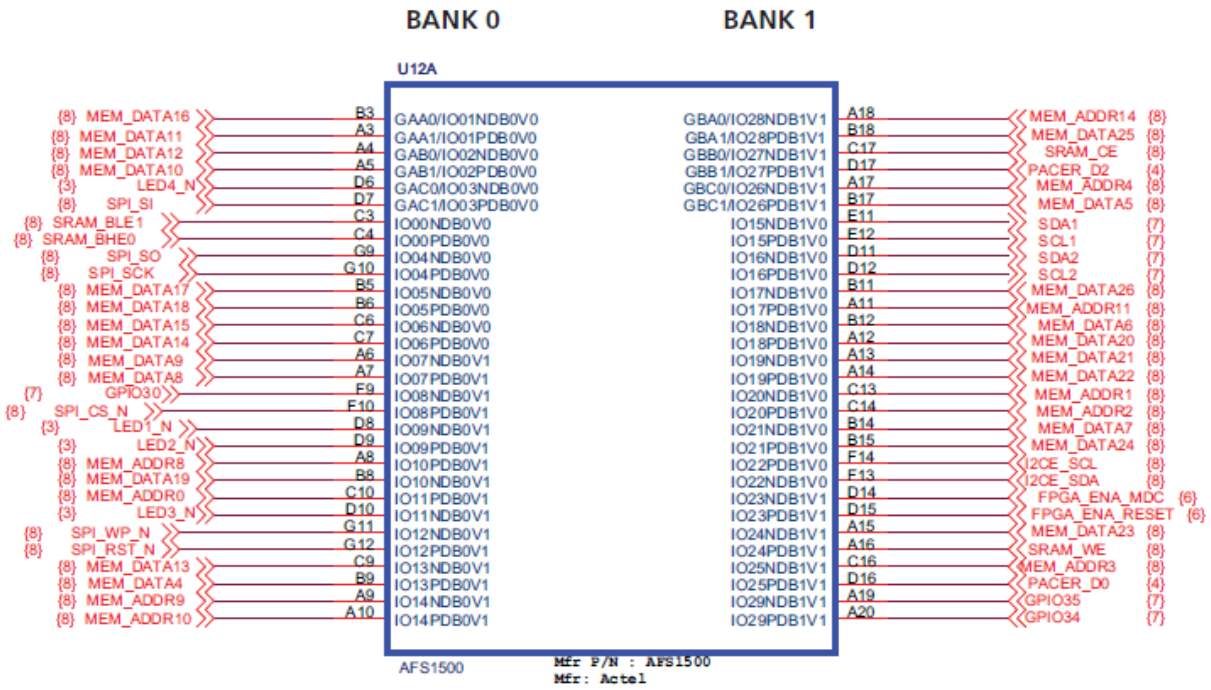


Figure A.1.7: I/O Pins Schematic for Banks 0 and 1

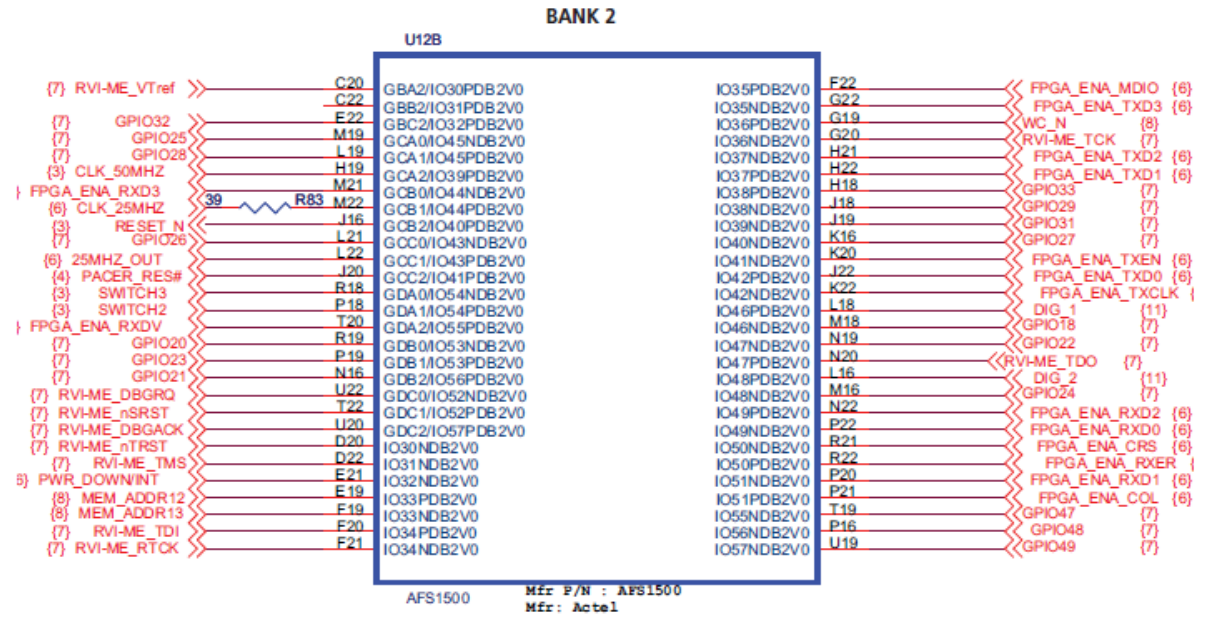


Figure A.1.8: I/O Pins Schematic for Banks 2

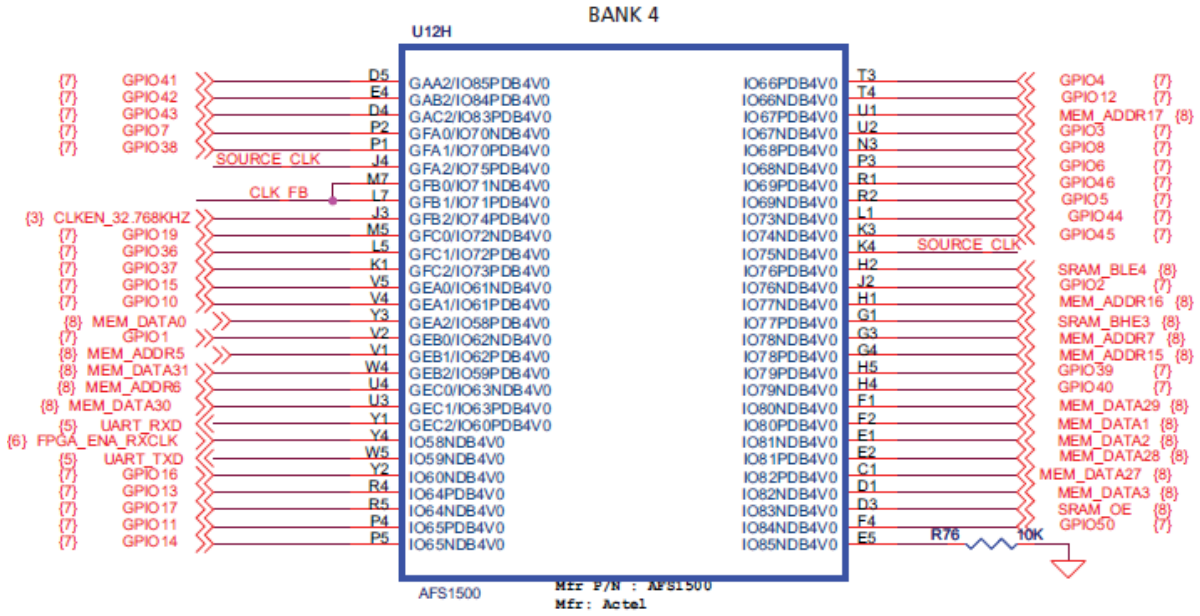


Figure A.1.9: I/O Pins Schematic for Banks 4

Le CAN intégré dans la carte a 12 bits de sortie et il est précédé d'un multiplexeur 32 vers 1 comme le montre la **figure A.1.10** :

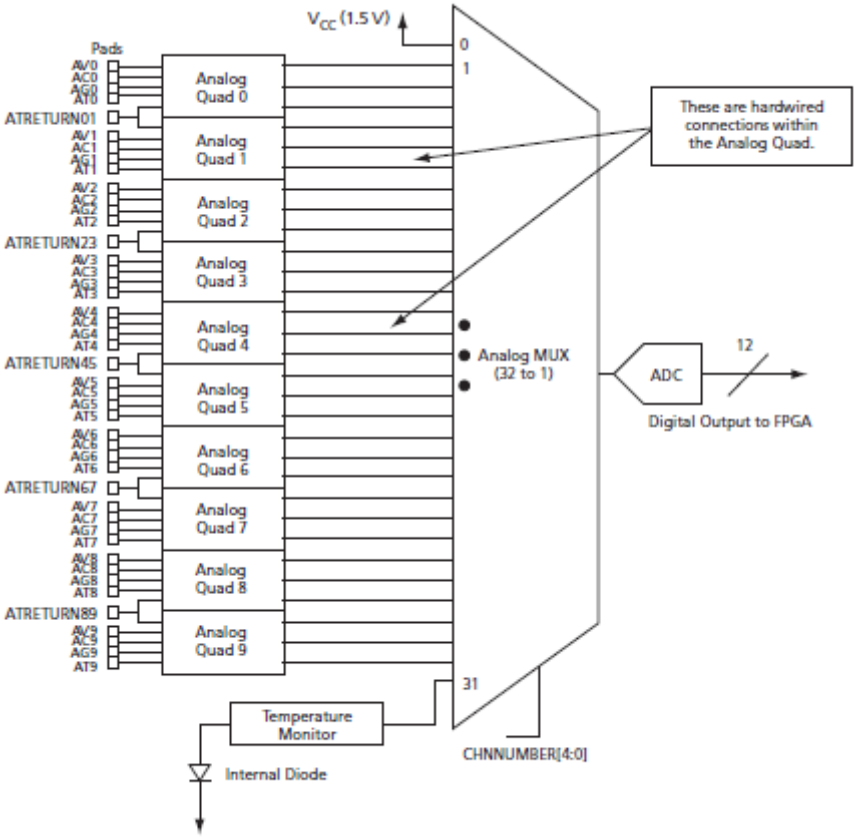


Figure A.1.10 : Architecture du CAN intégré et du multiplexeur

ANNEXE II : STRUCTURE GENERALE D'UN PROGRAMME VHDL

Un programme VHDL se structure de la manière suivante :

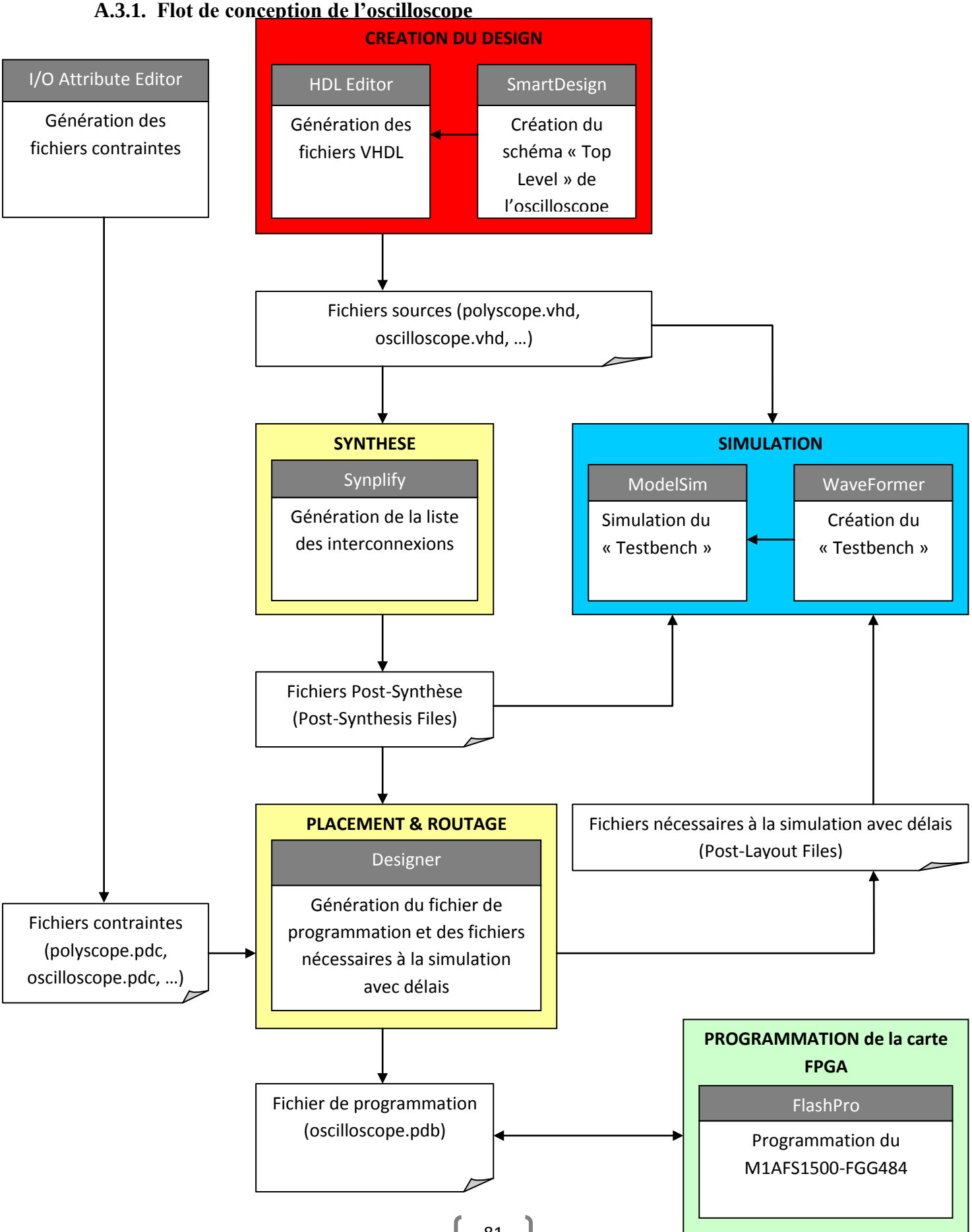
```
-- declaration et utilisation des bibliotheques necessaires
USE work.bibliotheques_necessaires.ALL;

-- declaration de l'entite et de ses ports d'entree-sortie
ENTITY nom_de_l_entite IS
   GENERIC (parametres_generiques: TYPE := Valeur_par_defaut);
   PORT (ports_d_entree: IN type_ports_d_entree;
         ports_de_sortie: OUT type_ports_de_sortie;
         ports_d_entree_sortie: INOUT type_entree_sortie);
END nom_de_l_entite;

-- architecture du programme, structure interne
ARCHITECTURE type_de_description OF nom_de_l_entite IS
BEGIN
   -- programme interne
END type_de_description;
```

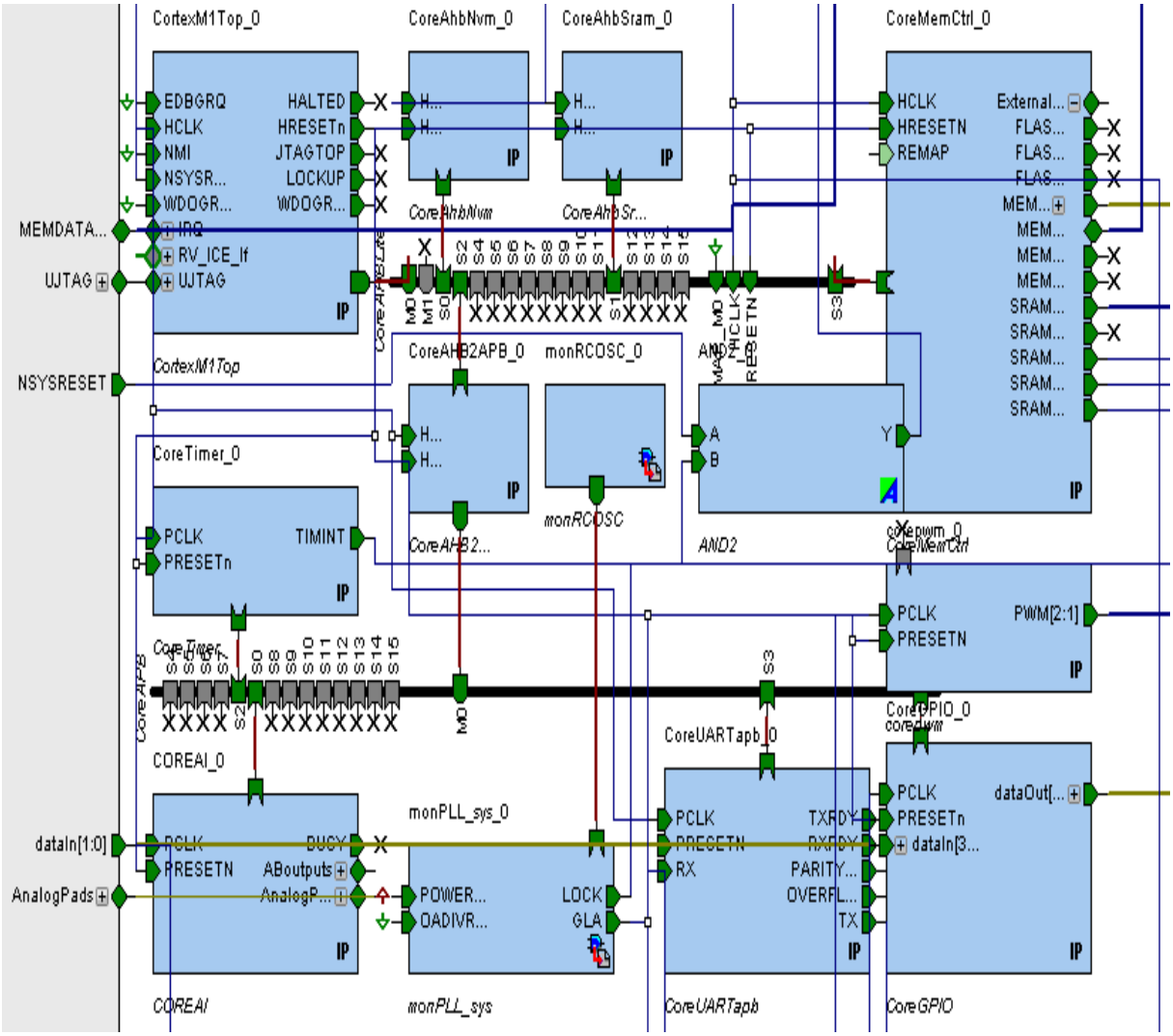
ANNEXE III : FLOT DE CONCEPTION MATERIELLE

A.3.1. Flot de conception de l'oscilloscope



A.3.2. Flot de conception du générateur de forme d'onde

La démarche est identique à celle du flot de conception de l'oscilloscope. Pourtant, le générateur de forme d'onde a comme schéma « top level » :



A.3.3. Présentation du programme exécutable via l'outil SoftConsole IDE

Voyons successivement le fichier d'en-tête (**polyscope.h**) ainsi que le fichier principal (**main.c**) de notre polyscope.

Voici le contenu du fichier **polyscope.h** :

```
#ifndef __POLYSCOPE_H_
#define __POLYSCOPE_H_
#define COREAHBNVM_BASE_ADDR      0x00000000UL
#define COREAHBSRAM_BASE_ADDR     0x10000000UL
#define EXT_SRAM_BASE_ADDR        0x30000000UL
#define COREAI_BASE_ADDR          0x20000000UL
#define COREGPIO_BASE_ADDR        0x21000000UL
#define CORETIMER_BASE_ADDR       0x22000000UL
#define COREUARTAPB_BASE_ADDR     0x23000000UL
#endif
```

Et celui du fichier **main.c** :

```
#include "hal.h"
#include "polyscope.h"
#include "coreai.h"
#include "core_uart_apb.h"
#include "core_gpio.h"
#include "core_timer.h"
#define MAX_TEXT_BUFFER_SIZE    64
#define CHAN_NAME_MAX_SIZE      16
#define ADC_RESOLUTION  0xFFF
/* CoreUARTapb_0 Configuration
    BAUD_VALUE = (CLOCK    / (16 * BAUD_RATE)) - 1
    BAUD_VALUE = (40000000 / (16 * 19200))     - 1 = 129 */
#define BAUD_VALUE_19200    129
/* CoreTimer_0 Configuration
    40 MHz / 512 = 78125 Hz    (effective timer frequency using prescalar)
    78125 Hz * 1 sec = 78125   (timer load value for 1 second interval)
    78125 = 0x1312D            (convert to hex) */
#define TMR0_PRESCALE         PRESCALER_DIV_512
#define TMR0_LOAD_VALUE       0x1312D
#define GPOUT_INIT_STATE                0x00000000
```

```
const uint8_t channel_name_lut[ADC_NB_OF_CHANNELS][CHAN_NAME_MAX_SIZE] =
{
  {"Internal Vcc:  "},
  {"   AV0    :  "},
  {"   AC0    :  "},
  {"   AT0    :  "},
  {"   AV1    :  "},
  {"   AC1    :  "},
  {"   AT1    :  "},
  {"   AV2    :  "},
  {"   AC2    :  "},
  {"   AT2    :  "},
  {"   AV3    :  "},
  {"   AC3    :  "},
  {"   AT3    :  "},
  {"   AV4    :  "},
  {"   AC4    :  "},
  {"   AT4    :  "},
  {"   AV5    :  "},
  {"   AC5    :  "},
  {"   AT5    :  "},
  {"   AV6    :  "},
  {"   AC6    :  "},
  {"   AT6    :  "},
  {"   AV7    :  "},
  {"   AC7    :  "},
  {"   AT7    :  "},
  {"   AV8    :  "},
  {"   AC8    :  "},
  {"   AT8    :  "},
  {"   AV9    :  "},
  {"   AC9    :  "},
  {"   AT9    :  "},
  {"Intern. Temp:  "}
};
const uint8_t line_return[2] = "\n\r";
```

```
const uint8_t sep[] = "================================================================================";
uint8_t key_pressed( void );
void process_samples( uint16_t *adc_samples );
void display_current
(
   uint8_t channel_nb,
   uint16_t raw_value
);
void display_temperature
(
   uint8_t channel_nb,
   uint16_t raw_value
);
void display_voltage
(
   uint8_t channel_nb,
   uint16_t raw_value
);
void display_start( void );
void display_update_sep( void );
void display_input_sep( void );
UART_instance_t g_the_uart;
int g_signal_count = 0;
gpio_instance_t g_gpio;
int main( void )
{
   static uint16_t adc_samples[ADC_NB_OF_CHANNELS] =
   {
      UNUSED_CHANNEL,    /* ADC_CHAN_VCC */
      SAMPLED_CHANNEL,    /* ADC_CHAN_AV0 */
      SAMPLED_CHANNEL,    /* ADC_CHAN_AC0 */
      UNUSED_CHANNEL,    /* ADC_CHAN_AT0 */
      SAMPLED_CHANNEL,    /* ADC_CHAN_AV1 */
      SAMPLED_CHANNEL,    /* ADC_CHAN_AC1 */
      UNUSED_CHANNEL,    /* ADC_CHAN_AT1 */
      UNUSED_CHANNEL,    /* ADC_CHAN_AV2 */
```

```
    UNUSED_CHANNEL,    /* ADC_CHAN_AC2 */
    UNUSED_CHANNEL,     /* ADC_CHAN_AT2 */
    UNUSED_CHANNEL,    /* ADC_CHAN_AV3 */
    UNUSED_CHANNEL,    /* ADC_CHAN_AC3 */
    UNUSED_CHANNEL,     /* ADC_CHAN_AT3 */
    UNUSED_CHANNEL,    /* ADC_CHAN_AV4 */
    UNUSED_CHANNEL,    /* ADC_CHAN_AC4 */
    UNUSED_CHANNEL,    /* ADC_CHAN_AT4 */
    UNUSED_CHANNEL,    /* ADC_CHAN_AV5 */
    UNUSED_CHANNEL,    /* ADC_CHAN_AC5 */
    UNUSED_CHANNEL,    /* ADC_CHAN_AT5 */
    UNUSED_CHANNEL,    /* ADC_CHAN_AV6 */
    UNUSED_CHANNEL,    /* ADC_CHAN_AC6 */
    UNUSED_CHANNEL,    /* ADC_CHAN_AT6 */
    UNUSED_CHANNEL,    /* ADC_CHAN_AV7 */
    UNUSED_CHANNEL,    /* ADC_CHAN_AC7 */
    UNUSED_CHANNEL,    /* ADC_CHAN_AT7 */
    UNUSED_CHANNEL,    /* ADC_CHAN_AV8 */
    UNUSED_CHANNEL,    /* ADC_CHAN_AC8 */
    UNUSED_CHANNEL,    /* ADC_CHAN_AT8 */
    UNUSED_CHANNEL,    /* ADC_CHAN_AV9 */
    UNUSED_CHANNEL,    /* ADC_CHAN_AC9 */
    UNUSED_CHANNEL,    /* ADC_CHAN_AT9 */
    UNUSED_CHANNEL,    /* ADC_CHAN_INT_TEMP */
};
uint32_t gpio_out = GPOUT_INIT_STATE;
uint32_t count = 0;
GPIO_init( &g_gpio, COREGPIO_BASE_ADDR, GPOUT_INIT_STATE );
CAI_init( COREAI_BASE_ADDR );
UART_init( &g_the_uart, COREUARTAPB_BASE_ADDR, BAUD_VALUE_57600, (DATA_8_BITS | NO_PARITY) );
display_start();
while( 1 )
{
    if ( (count++ % 0x08FF) == 0 )
            GPIO_set_output( &g_gpio, gpio_out++ );
            CAI_round_robin( adc_samples );
```

```
        if ( key_pressed() )
        {
            process_samples( adc_samples );
        }
    }
}
uint8_t key_pressed( void )
{
    uint8_t ret_val = 0;
    size_t rx_size;
    uint8_t rx_buffer[16];
    rx_size = UART_get_rx( &g_the_uart, rx_buffer, sizeof(rx_buffer) );
    if ( rx_size > 0 )
    {
        ret_val = 1;
    }
    return ret_val;
}
void process_samples( uint16_t *adc_samples )
{
    uint8_t channel_nb;
    display_update_sep();
    for ( channel_nb = ADC_CHAN_VCC; channel_nb < ADC_NB_OF_CHANNELS; channel_nb++ )
    {
        if ( (adc_samples[channel_nb] != UNUSED_CHANNEL) && (adc_samples[channel_nb] != SAMPLED_CHANNEL) )
        {
            uint16_t raw_value;
            raw_value = adc_samples[channel_nb];
            if ( CAI_is_current( channel_nb ) )
            {
                display_current( channel_nb, raw_value );
            }
            else if ( CAI_is_temperature( channel_nb ) )
            {
                display_temperature( channel_nb, raw_value );
            }
```

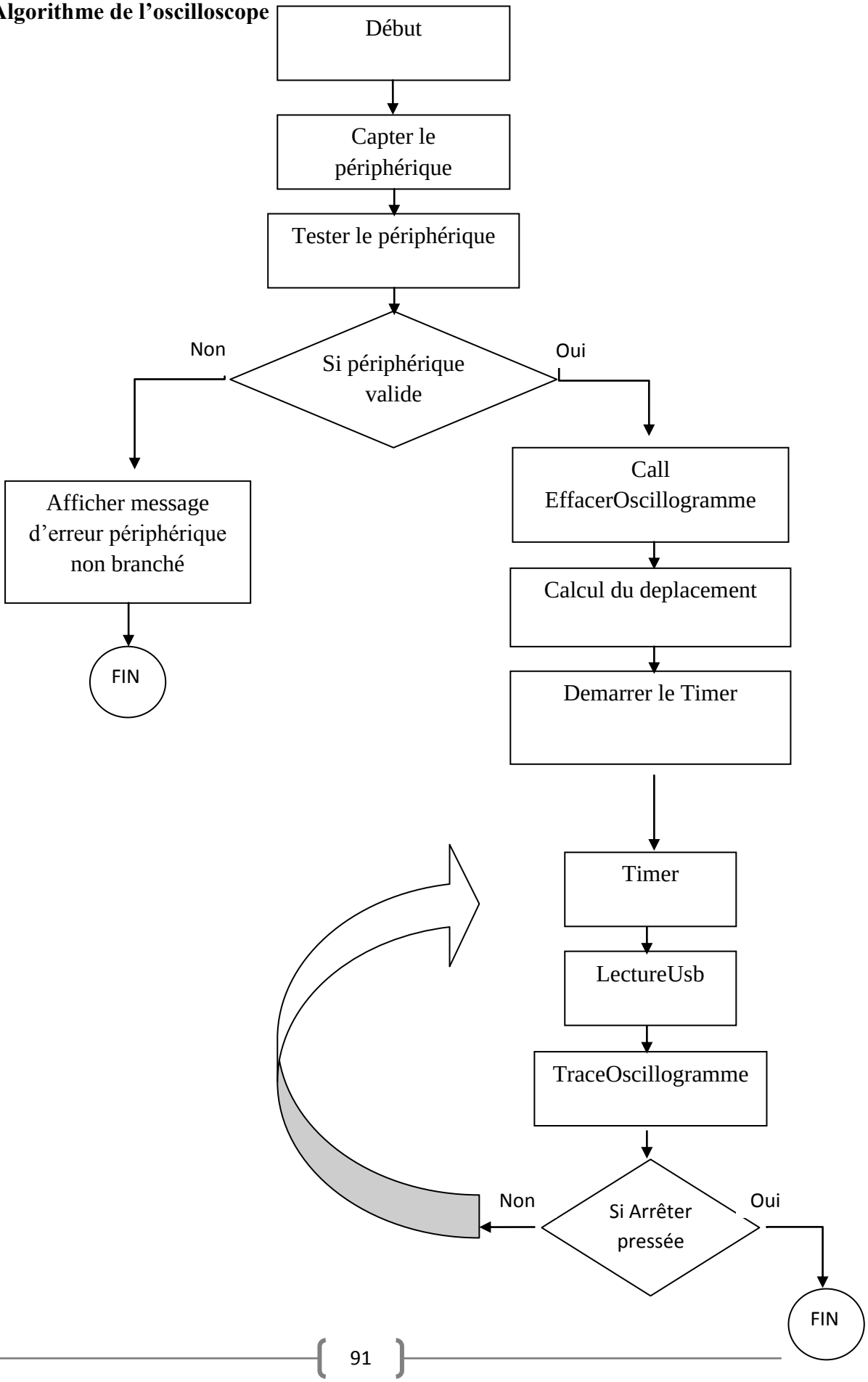
```
            else
            {
                display_voltage( channel_nb, raw_value );
            }
            display_input_sep();
        }
    }
}
void display_current
(
    uint8_t channel_nb,
    uint16_t raw_value
)
{
    uint8_t value_text_reverse[MAX_TEXT_BUFFER_SIZE];
    uint8_t idx = 0;
    uint16_t voltage;
    /* Translate from raw ADC value to mV. */
    voltage = (2560 * raw_value) / ADC_RESOLUTION;
    /* display channel name.*/
    UART_send( &g_the_uart, &channel_name_lut[channel_nb][0], CHAN_NAME_MAX_SIZE );
    /* generate text. */
    idx = MAX_TEXT_BUFFER_SIZE - 1;
    value_text_reverse[idx--] = 0x00;
    value_text_reverse[idx--] = 'A';
    value_text_reverse[idx--] = 'm';
    value_text_reverse[idx] = ' ';
    do {
        idx--;
        value_text_reverse[idx] = (voltage % 10) + '0';
        voltage = voltage / 10;
    } while( ( voltage > 0 ) && (idx > 0) );
    /* pad with spaces for alignment. */
    while( idx > (MAX_TEXT_BUFFER_SIZE - 8) )
    {
        idx--;
```

```
        value_text_reverse[idx] = ' ';
    }
    UART_send( &g_the_uart, &value_text_reverse[idx], MAX_TEXT_BUFFER_SIZE - idx );
}
void display_voltage
(
    uint8_t channel_nb,
    uint16_t raw_value
)
{
    uint8_t value_text_reverse[MAX_TEXT_BUFFER_SIZE];
    uint8_t unit[] = " mV";
    uint8_t idx = 0;
    uint32_t full_scale;
    uint16_t voltage;
    /* Translate from raw ADC value to mV. */
    full_scale = CAI_input_full_range( channel_nb );
    voltage = (full_scale * raw_value) / ADC_RESOLUTION;
    /* display channel name.*/
    UART_send( &g_the_uart, &channel_name_lut[channel_nb][0], CHAN_NAME_MAX_SIZE );
    idx = MAX_TEXT_BUFFER_SIZE;
    do {
        idx--;
        value_text_reverse[idx] = (voltage % 10) + '0';
        voltage = voltage / 10;
    } while( ( voltage > 0 ) && (idx > 0) );
    /* pad with spaces for alignment. */
    while( idx > (MAX_TEXT_BUFFER_SIZE - 4) )
    {
        idx--;
        value_text_reverse[idx] = ' ';
    }
    /* Display voltage value. */
    UART_send( &g_the_uart, &value_text_reverse[idx], MAX_TEXT_BUFFER_SIZE - idx );
    UART_send( &g_the_uart, unit, sizeof(unit) );
}
```

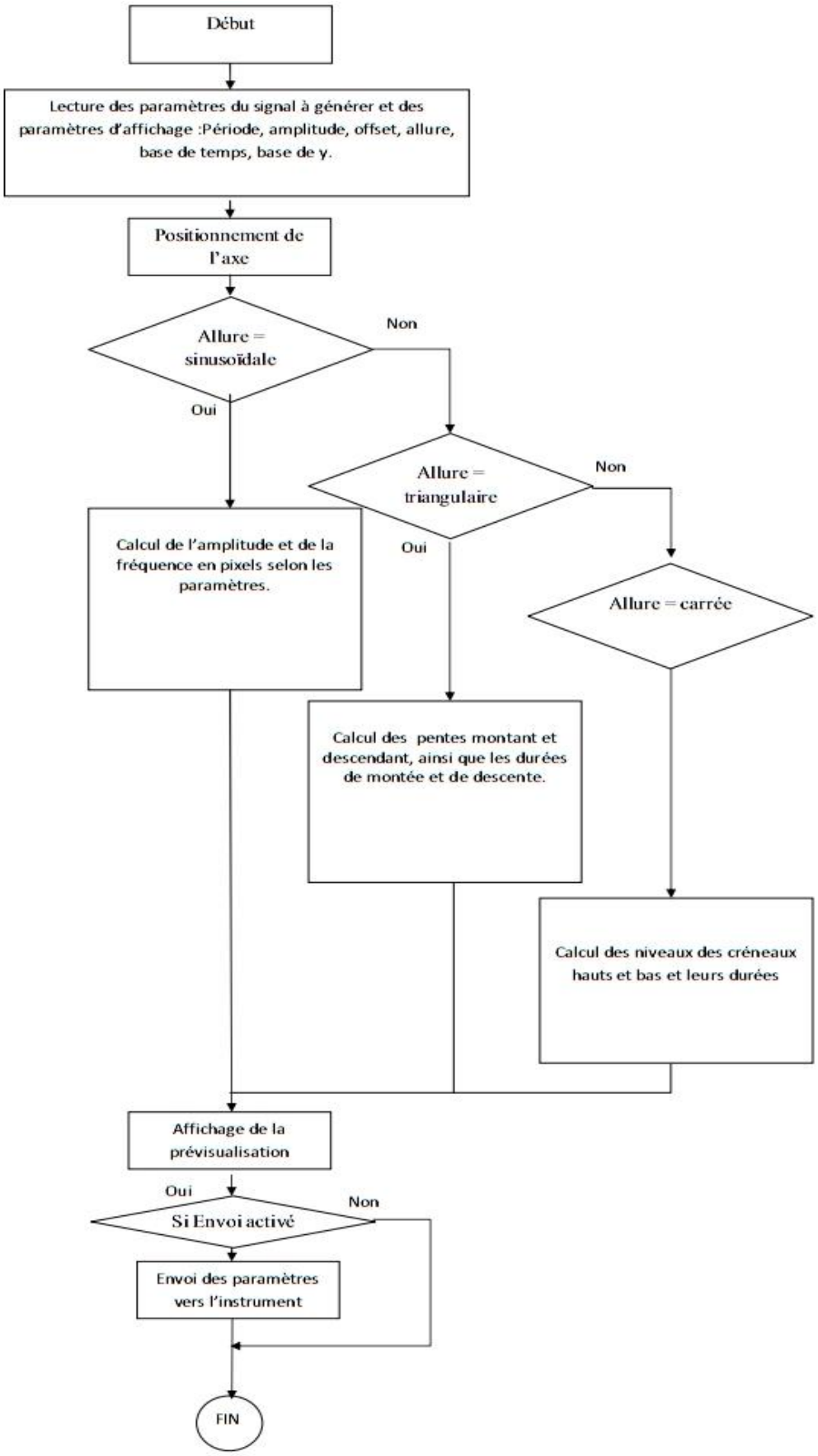
```
void display_start( void )
{
   uint8_t mesg[] = "Press any key to display current analog inputs values.";
   UART_send( &g_the_uart, line_return, sizeof(line_return) );
   UART_send( &g_the_uart, sep, sizeof(sep) );
   UART_send( &g_the_uart, line_return, sizeof(line_return) );
   UART_send( &g_the_uart, mesg, sizeof(mesg) );
   UART_send( &g_the_uart, line_return, sizeof(line_return) );
}
void display_update_sep( void )
{
   UART_send( &g_the_uart, line_return, sizeof(line_return) );
   UART_send( &g_the_uart, sep, sizeof(sep) );
   UART_send( &g_the_uart, line_return, sizeof(line_return) );
   g_signal_count = 0;
}
void display_input_sep( void )
{
   uint8_t inp_sep[] = "     |      ";
   g_signal_count++;
   if ( (g_signal_count % 2) == 1 )
   {
      UART_send( &g_the_uart, inp_sep, sizeof(inp_sep) );
   }
   else
   {
      UART_send( &g_the_uart, line_return, sizeof(line_return) );
   }
}
```

ANNEXE IV : ALGORITHME GENERAL DE LA PARTIE LOGICIELLE

A.4.1. Algorithme de l'oscilloscope



A.4.2. Algorithme pour le générateur de forme d'onde



:

Vue d'ensemble du projet Polyscope

Voici le résumé du contenu de chacun des fichiers qui constituent l'application Polyscope.

- Fichiers standard d'un projet Visual C++

  Le nom de chacun de ces fichiers est indentique à celui du projet, dans notre cas c'est Polyscope

  - Polyscope.vcxproj

Il s'agit du fichier projet principal pour les projets VC++ générés par Visual C++. Il contient les informations sur la version de Visual C++ qui a généré le fichier et des informations sur les plates-formes, configurations et fonctionnalités du projet sélectionnées.

- Polyscope.vcxproj.filters

Il s'agit du fichier de filtres pour les projets VC++ générés à l'aide d'un Assistant Application.

Il contient des informations sur l'association entre les fichiers du projet et les filtres. Cette association est utilisée dans l'IDE pour afficher le regroupement des fichiers qui ont des extensions similares sous un nœud spécifique (par exemple, les fichiers ".cpp" sont associés au filtre "Fichiers sources").

- Polyscope.h

Il s'agit du fichier d'en-tête principal de l'application. Il contient d'autres en-têtes de projet spécifiques (y compris Resource.h) et déclare la classe d'application CPolyscopeApp.

- Polyscope.cpp

Il s'agit du fichier source principal de l'application qui contient la classe d'application CPolyscopeApp.

- Polyscope.rc

Il s'agit de la liste de toutes les ressources Microsoft Windows que le programme utilise. Elle comprend les icônes, les bitmaps et les curseurs qui sont stockés dans le sous-répertoire RES. Ce fichier peut être modifié directement dans Microsoft Visual C++. Les ressources de projet sont dans res\

- Polyscope.ico

Il s'agit d'un fichier icône, qui est utilisé comme icône de l'application.

Cette icône est incluse par le fichier de ressource principal Polyscope.rc.

- res\Polyscope.rc2

Ce fichier contient les ressources qui ne sont pas modifiées par Microsoft

Visual C++. on doit placer toutes les ressources non modifiables par l'éditeur de ressources dans ce fichier.

- PolyscopeDlg.h, PolyscopeDlg.cpp – la boîte de dialogue

Ces fichiers contiennent la classe CPolyscopeDlg. Cette classe définit le comportement de la boîte de dialogue principale de l'application. Le modèle de boîte de dialogue se trouve dans Polyscope.rc et peut être modifié dans Microsof Visual C++.

- Prise en charge de l'aide :
  - hlp\Polyscope.hhp

Ce fichier est un fichier projet d'aide. Il contient les données requises pour compiler les fichiers d'aide dans un fichier .chm.

- hlp\Polyscope.hhc

Ce fichier répertorie le contenu du projet d'aide.

- hlp\Polyscope.hhk

Ce fichier contient un index des rubriques d'aide.

- hlp\afxcore.htm

Ce fichier contient les rubriques d'aide standard relatives aux objets de l'écran et aux commandes MFC standard. On doit ajouter à ce fichier ses propres rubriques d'aide.

- hlp\afxprint.htm

Ce fichier contient les rubriques d'aide relatives aux commandes d'impression.

- makehtmlhelp.bat

Ce fichier est utilisé par le système de génération pour compiler les fichiers d'aide.

- hlp\Images*.gif

Il s'agit de fichiers bitmap requis par le fichier des rubriques d'aide standard pour les commandes standard de la bibliothèque Microsoft Foundation Class.

- Autres fichiers standard :
  - StdAfx.h, StdAfx.cpp

Ces fichiers sont utilisés pour générer un fichier d'en-tête précompilé (PCH) nommé Polyscope.pch et un fichier de type précompilé nommé Stdafx.obj.

- Resource.h

Il s'agit du ficher d'en-tête standard, qui définit les nouveaux ID de ressources. Microsoft Visual C++ lit et met à jour ce fichier.

- Polyscope.manifest

Les fichiers manifestes d'application sont utilisés par Windows XP pour décrire les dépendances des applications sur des versions spécifiques des assemblys côte à côte. Le chargeur utilise ces informations pour charger l'assembly approprié à partir du cache de l'assembly ou directement à partir de l'application. Le manifeste de l'application peut être inclus pour redistribution comme fichier .manifest externe installé dans le même dossier que l'exécutable de l'application ou être inclus dans l'exécutable sous la forme d'une ressource.

Si l'application utilise les MFC dans une DLL partagée on doit redistribuer les DLL MFC. Si la langue de l'application n'est pas celle du système d'exploitation, on doit également redistribuer le fichier des ressources localisées MFC100XXX.DLL.

- Fichiers personnalisés:

Enfin, le programmeur peut ajouter des fichiers pour ses propres classes et fonctions

Dans notre cas il s'agit des fichiers suivants:

- WaveForm.h, WaveForme.cpp

Ces fichiers contiennent la classe CWaveForm. Cette classe définit le comportement de la boîte de dialogue du générateur de forme d'onde.

- Variable.h,

  Fichier contenant les principaux variables communes des autres fichiers.

- DisplayCrid.h,

  Fichier contenant la fonction AfficherGrid()

- Oscillogramme.h,

Fichier contenant la déclaration des variables correspondant au comportement de la courbe.

- accesusb.h

Fichier contenant les définitions des fonctions de gestion de l'accès usb.

REFERENCES

[1] E557EA Instrumentation, cours sur le principe de l'instrumentation 5ème année ELECTRONIQUE ESPA 2009-2010

[2] Initiation aux fondamentaux de l'instrumentation virtuelle en Labview et ses applications, l'instrumentation virtuelle comparée à l'instrumentation traditionnelle **http://sine.ni.com/nipdfgenerator/nipdfgenerator?pageURL=http://zone.ni.com/devzone/cda/tut/p/id/5526&clientAppName=dz&dotsPerPixel=&dotsPerPoint=**

[3] Reconfigurable Virtual Instrumentation,**http://mlab.ictp.it/uploads/En/zz/EnzzDYlGc1u0QI-NNI4PHQ/RVI_paper.pdf**

[4] E301 Circuits séquentiels, cours sur les compteurs3ème année ELECTRONIQUE ESPA 2007-2008

[5] E558EA Modélisation Comportementale des Systèmes, cours de VHDL 5ème année ESPA 2009-2010

[6] Conception Assistée par Ordinateur : « Analog/Mixed-Signal Design (page 20)» **http://iccad.com/files/files/ICCAD_2011_Final_Program_web.pdf**

[7] Flot de conception d'Actel Libero, **http://www.actel.com/techdocs/manuals/default.aspx**

[8] Actel technical support, **http://www.actel.com/support/**

[9] E331 Programmation en C, cours de programmation 3ème année ELECTRONIQUE ESPA 2007-2008

[10] Fusion Embedded Developement Kit, **http://www.actel.com/documents/M1AFS_EMBEDDED_KIT_QS.pdf**

[11] Cortex-M1 v3.1 Handbook, **http://infocenter.arm.com/help/topic/com.arm.doc.dui0255l/DUI0255L_getting_started.pdf**

[12] E410 Programmation en Langage Orientée Objet, cours 4ème année ELECTRONIQUE ESPA 2008-2009

[13] Cours de C/C++ Christian Casteyde, **http://casteyde.christian.free.fr/cpp/cours/online/book1.html**

[14] Thinking in C plus plus,**http://www.EckelObjects.com/ThinkingInCPP2e.html**

[15] E550IA Génie Logiciel II, cours - La modélisation UML 5ème année ELECTRONIQUE ESPA 2009-2010

[16] E411 Génie Logiciel I, cours - Cycle de vie des logiciels 4ème année ELECTRONIQUE ESPA 2008-2009

[17] L'USB ET SA NORME, **http://u.s.b.free.fr/pdf/L_USB_et_sa_norme_v1.pdf**

[18] libusb Developers Guide Johannes Erdfelt,**http://www.libusb.org/wiki/libusb-win32.html**

[19] USB GUIDE Fourth Edition Jan Axelson, **http://www.fileboar.com/process/2332190/9791041/USB%20Complete%20The%20Developer%E2%80%99s%20Guide,%20Fourth%20Edition.pdf**

[20] XYZs of Oscilloscopes, **http://socrates.berkeley.edu/~phylabs/bsc/Supplementary/Lab1/xyz_scopes.pdf**

[21] ARM® Cortex™-M1 Embedded Processor Hardware Development Tutorial for Fusion Mixed-Signal FPGAs, **http://www.actel.com/documents/CortexM1_Proc_HW_Tutorial_UG.pdf**

[22] E 303 Circuits Analogiques, cours sur le comparateur de tension 3ème année ELECTRONIQUE ESPA 2007-2008

Auteur : RAKOTOARITINA Tojoarisoa

Titre : CONCEPTION MATERIELLE D'INSTRUMENTS VIRTUELS

Nombre de pages : 72

Nombre de tableaux : 3

Nombre de Figures : 26

RESUME

Ce travail de mémoire présente la réalisation des Instrumentations Virtuelles Reconfigurables à faible coût en utilisant un système FPGA-PC. Cette réalisation invoque deux démarches en parallèle : un développement logiciel et un développement matériel. On l'a démontrée avec la réalisation d'un oscilloscope et d'un générateur de forme d'onde. En effet avec le développement de la technologie des circuits numériques reconfigurables, il est dorénavant possible de concevoir des Instruments Virtuels aux formes multiples. Les avantages offerts par l'Instrumentation Virtuelle sont essentiellement la diminution des temps de développement, l'optimisation des moyens et la réduction des coûts.

Mots clés : C, C++, VHDL, Instrumentation Virtuelle, RVI, FPGA

Rapporteur : Madame RAMANANTSIHOARANA Harisoa Nathalie

Adresse de l'auteur :

RAKOTOARITINA Tojoarisoa

Lot 0512-C-195 Tsarasaotra ANTSIRABE 110 ANTANANARIVO MADAGASCAR

Tél: 034 01 259 55