الجمهورية الجزائرية الديمقراطية الشعبية

République Algérienne démocratique et populaire
Ministère de l'Enseignement Supérieur & de la Recherche Scientifique
Université Docteur Yahia Farés de MEDEA

Faculté des sciences et de la technologie
Département du Génie Electrique
Et de L'informatique

Projet de Fin d'études

Présenté pour l'obtention du diplôme

Master enGénie Électrique

Spécialité : *Systèmes Intelligents et Communicants*

Thème :

Développement d'un système de communication série à base d'un circuit FPGA

Proposé et dirigé par :

Mr. BENGHERBIA Billel

Mr. OUILD ZMIRLI Mohamed

Réalisé Par :

BOUDISSA Radhia

MAZARI Fatma

Année universitaire2014-2015


بسم الله الرحمن الرحيم
السلام عليكم و رحمة الله و بركاته

REMERCIMENT

Nous tenons à remercie premièrement Dieu le tout puissant pour la santé,la volonté, la patience, l'amour et la générosité qu'il nous a donné durant toutes ces longues années. Merci car sans toi ce travail n'existerait pas

Un immense remerciement à Monsieur BENGHERBIA Billel (promoteur) et Dr. OUILD ZMIRLI Mohamed, pour l'effort fourni pour l'encadrement de ce travail et de nous avoir aidé à mener ce rapport sur les bonne voies. Et la pertinence de ses remarques nous la remerciée également pour sa disponibilitée quotidienne et pour sa degré d'implication dans ce travail, merci aussi pour sa sympathiee, sa simplicitée, et sa convivialité.

Nos sincères remerciements vont également à tous les enseignants du département deGenie Electrique et de l'informatique de l'université de Médéa, et à tous les enseignants qui ont participé à notre formation.

Nous tenons aussi à adresser nos remerciements aux membres de jury Pour nous avoir fait l'honneur et le plaisir d'accepter et juger notre travail.

Nous remercierTous ceux qui m'ont aidé à réaliser ce travail du près ou de loin.

Radhia et Fatma

Dédicace

Je dédie ce modeste travail.
Tout d'abord je tiens à remercier Dieu le tout puissant de m'avoir donné la santé, la volonté et le courage.
Ensuite, à mon père et ma mère pour leurs soutien et leurs encouragements et qu'aucune dédicace ne pourra compenser leur sacrifice pour mon bien être et mon bonheur.
A mes chers frères Ayoub, Othmane et Rayane pour lequel j'ai beaucoup d'estimes.
A mon sœur Asmaa et son mari abde lhamid que je lui souhaite une longue vie pleine de joie et de réussite.
A mes oncles, mes tantes et toute ma famille BOUDISSA ET KCHIDA.
A mes amis, en particulier AMINE qui m'ont aidé et pour m'avoir supportée.
A mon collègue du présent projet Mazari Fatma qui n'a jamais manqué de travailler et ceux de la promotion de la 2ème année master systèmes intelligent et communicant
A notre promoteur M. BENGHARBIA Billel et Dr. OUILD ZMIRLI Mohamed.
Et à tous ceux qui nous ont soutenus pour l'achèvement de ce projet avec réussite.

BoudissaRadhia

Dédicaces

Je dédie ce modeste travail.
Tout d'abord je tiens à remercier Dieu le tout puissant de m'avoir donné la santé, la volonté et le courage.
Ensuite, à mon père et ma mère pour leurs soutien et leurs encouragements et qu'aucune dédicace ne pourra compenser leur sacrifice pour mon bien être et mon bonheur.
A mes chers frères Ridha, Abdelhake et Azzdinepour lequel j'ai beaucoup d'estimes.
A mes oncles, mes tantes et toute ma famille.
A mes amis Amina, Salma,Hanane, Imane ,Sihem et Ibtissem et Mohamedet Nasima en particulier mon collègue du présent projet radhiaboudissa qui n'a jamais manqué de travailler et ceux de la promotion de la 2ème année master systèmes intelligent et communicant
A notre promoteur M. BENGHARBIA Billel
et ouildzmirli.
Et à tous ceux qui nous ont soutenus pour l'achèvement de ce projet avec réussite.

MAZARI FATMA

الملخص:

الهدف من هذا المشروع هو تطوير وتمثيل واجهة اتصال على التسلسل من النوع SPIالقائم على رقاقة من النوع FPGA أيضا تم برمجت معالج من نوعMicroBlazeلتمثيل إدارة الاتصال التسلسلي كلا نمطين الاستقبال والإرسال تم دراستهما واختبارهما من خلال وصلة اتصال FPGA\أرد وينو

الكلمات المفتاحية: أرد وينو ,SOC , منصات الرقاقات FPGA,USB للربط

Résumé :

L'objectif de ce projet est le développement et la réalisation d'une interface de communication Série de type SPI à base d'FPGA. Un processeur softcore de type Microblaze est implémenter afin de permettre la gestion de la communication série. Les deux modes de communication Maitre et Esclave de l'SPI sont étudies et testé à travers une liaison FPGA/Arduino.

Les mots clés : FPGA, Arduino,SOC, SPI.

Abstract:

The objective of this project is the development and the implementation of a Serial Peripheral Interface SPI-based FPGA. A MicroBlaze softcore processor is implemented to enable the management of the serial communication. Both Master and Slave modes of the SPI communication are studying and are tested through a FPGA / Arduino link.

Key words: FPGA, Arduino, SOC, SPI.

TABLE DES MATIERES

LISTE DES FIGURES I

LISTE DES TABLEAUX III

LISTE DES ACRONYMES IV

Introduction générale 1

Chapitre I : Généralité du système embarqué

1. Introduction 3

2. Conception des systèmes numériques : 3

2.1. Architectures à base d'un processeur 3

2.1.1. Les processeurs d'usage général (GPP) 4

2.1.2. Les processeurs de traitement du signal 4

2.1.3. Les processeurs reconfigurables 5

2.2. Les circuits logiques programmables 5

2.2.1. Les FPGAs 6

2.2.2. Les ASICs 8

3. Les systèmes numériques embarqués 9

3.1. Codesign : Conception conjointe logicielle/matérielle 9

3.1.1. Flot de la conception conjointe logicielle/matérielle 10

3.1.2. Les composants virtuels (IPs) 12

3.2. Les systèmes sur puce (SoC) 14

3.2.1. Définitions d'un SoC 14

3.2.2. Les SoC et les SoPC : 15

3.3. Communications sur puce 17

3.3.1. Les différents types des communications sur puce 17

3.3.2. Exemple : Le Bus CoreConnect 19

4. Les processeurs embarqués 21

4.1. Les processeurs hardcore 20

4.2. Les processeurs softcore 22

5. Conclusion 28

ChapitreII : Interface de communication

1. Introduction 28
2. Méthodes de transfert des données 28
2.1.Transmission sérielle et parallèle 28
2.1.1. liaison parallèle 28
2.1.2. liaison en série 29
3. Protocole de communication série 31
3.1.Définition et principe générale 31
3.1.1. UART (Universal Asynchronous Receiver Transmitter) 31
3.1.2. Protocol I2C (Inter-Integrated Circuit) 32
3.1.2.1. Présentation 32
3.1.2.2. Prise de contrôle du bus 33
3.1.3. Interface et le protocole de communication SPI 34
3.1.3.1. La liaison SPI (Serial Peripheral Interface) 34
3.1.3.2. Configuration du bus 34
3.1.3.2.1. SCK (Horloge) 34
3.1.3.2.2. SS (Slave Select) 35
3.1.3.2.3. MOSI (Master Output - Slave Input) 35
3.1.3.2.4. MISO (Master Input - Slave Output) 35
3.1.3.3. Protocole du bus SPI 35
3.1.3.3.1. Transmission de données 35
3.1.3.3.2. Mode de service 36
3.1.3.3.3. Format du transfert de données 37
4. Conclusion 39

Chapitre III : Réalisation et mise en machine

1. Introduction 40

2. Description du matériel utilisé..... 40
2.1. La carte Arduino Uno : 40
2.1.2. Caractéristiques de la carte Arduino uno : 41
2.2. La carte de développement Xilinx Xtreme DSP Starter Kit-Spartan-3A DSP 1800A Edition 42
3. Les Outils de développement de Xilinx 44
3.1. Flots de conception Xilinx 44
3.2. Xilinx EDK..... 46
3.2.1. Etude pratique..... 55
3.2.1.1.Partie émission..... 55
3.2.1.2.Partie réception..... 56
4.Conclusion 60
Conclusion générale 61

LISTE DES FIGURES

Figure I.1. *Les différents circuits programmables* 6
Figure I.2. *Architecture conceptuelle d'un FPGA (Architecture, Interconnexions et CLB).* 8
Figure I.3. *Flot de conception standard* 11
Figure I.4 : *Objectif du SoC* 14
Figure I.5. *Système embarqué (SoC) typique*. 15
Figure I.6. *Flot de conception Xilinx* *17*
Figure I.7. *Le bus PLB dans un système à base de MicroBlaze* *20*
Figure 1.8. *Architecture interne du PowerPC405* 22
Figure I.9.*montre un schéma-bloc fonctionnel du noyau MicroBlaze* 24
Figure I.10 : *MicroBlaze Micro Controller System, MCS* 24
Figure I.11. *Les processeurs NIOS et NIOS-II de Altera* 27
Figure II.1. *Liaison parallèle* 29
Figure II.2. *Liaison série* 30
Figure II.3. *La transformation parallèle* 31
Figure II.4. *La transformation série –parallèle* 31
Figure II.5. *La communication d'un UART* 33
Figure II.6. *Une trame série d'un UART* 33
FigureII.7. *Architecture d'un bus I2C* 34
Figure II.8. *La transmission des données sur le bus I2C* 35
Figure II .9. *Exemple de configuration de bus SPI avec un maitre et plusieurs esclaves* 36
Figure II.10.*Transmission de données en maitre et esclave par registre à décalage*…. 37
Figure II.11. *Exemple d'une transmission de données SPI de 1 octet pour CPHA=0* 39
FigureII.12. *Exemple d'une transmission de données SPI de 1 octet pour CPHA=1* 40

Figure III.1. *Différents éléments de la carte Arduino Uno.* 42
Figure III.2 : *Vue externe de la plateforme XtremeDSP Starter Kit - Spartan-3A DSP 1800A Edition* 44
Figure III.3. *Flot de conception Xilinx générique* 46
Figure III.4.*création nouveau projet* 48
Figure III.5.*Système de base Builder* 49
Figure III.6.*Configurer les options de MicroBlaze* 50
Figure III.7.*Fenêtre Projet XPS* 51
Figure III.8.*Fenêtre d'IP Core* 52
Figure III.9*Les bus interfaces* 52
Figure III.10.*Les ports* 53
Figure III.11*Les adresses* 53
Figure III.12.*Fenêtre d'UCF.* 53
Figure III.13.*Fenêtre Projet SDK* 54
Figure III.14.*création nouveau proje* 55
Figure III.15. *Transfert en mode émission* 56
Figure III.16. *Transfert en mode réception* 57
Figure III.17. *Fenêtre de Generate Linker Script* 58
Figure III.18.*Charger le programme dans la carte cible* 59
Figure III.19. *L'affichage de mode maitre / esclave* 60
Figure III.20. *L'affichage de mode maitre / esclave* 61

LISTE DES TABLEAUX

Tableau 2.1. Modes de transmission SPI 37

LISTE DES ACRONYMES :

ARM	Acorn Risc Machine
ASIC	Application Specific Integrated Circuit
CISC	Complex Instruction Set Computer
CLB	Configurable Logic Block
CPHA	Clock Phase
CPLD	Complex Logic Programmable Device
CPOL	Clock Polarity
CLB	Configurable Logic Block
DCR	Device Processors
EDK	Embedded Development Kit
FPGA	Field Programmable Gâte Array
FPLA	Field Programmable Logic Array
GPP	General Purpose Processor
HDL	Hardware Design Language
ISE	Integrated Software Environnent
I/O	Input Output
I2C	Inter-Integrated Circuit
LUT	Look Up Table
MISO	Master Input, Slave Output
MOSI	Master Output, Slave Input
NoC	Network-On-Chip
OPB	On-chip Peripheral Bus
PIC	Peripheral Interface Controller
PLB	Processor Local Bus
PLD	Programmable Logic Device
PROM	Programmable Read Only Memor
RISC	Reduced Instruction Set Computer
SCL	Serial Clock Line

SCK	Serial Clock
SDK	Le Kit de développement logiciel
SIMD	Single-Instruction Multi-Data
SOC	System on Chip
SoPC	System on Programmable ChipF
SPI	Serial Peripheral Interface
SS	Slave Select
UART	Universal Asynchronous Receiver Transmitter
USB	Universal Serial Bus
VHSIC-HDL	Very High Speed Integrated Circuit Hardware Description Language
XPS	Xilinx Platform Studio

Introduction générale :

La communication série est utilisée dans toutes les communications notamment en réseaux informatiques, dont le coût de câblage et de synchronisation des données rendent la communication parallèle impraticable. Une connexion en série nécessite moins des câbles d'interconnexion (par exemple, des fils / fibres) et donc elle occupe moins d'espace. Le gain d'espace permet une meilleure isolation. Les liaisons série peuvent être cadencés considérablement et, plus rapidement que des liaisons parallèles en vue d'atteindre un plus haut débit de données.

Dans de nombreux cas, la communication en série est une meilleure option car elle est moins chère à mettre en œuvre. De nombreux circuits intégrés ont des interfaces série, par opposition à ceux parallèles, de sorte qu'ils puissent avoir moins de broches et sont par conséquent moins coûteux. L'interface série synchrone ou SSP (Synchronous Serial Port) dont disposent de nombreux PIC peut fonctionner également en mode SPI, appelée encore bus SPI (Serial Port Interface) ou bus série 3 fils **[1],** permettant d'envoyer et de recevoir des données sous forme série à l'aide de 4 lignes :

- Une ligne d'entrée de données série soit MISO.
- Une ligne de sortie de données série autrement MOSI.
- Une ligne d'horloge soit SCK.
- Une ligne de validation des transferts autrement dit SS.

Une des particularités de cette interface est de pouvoir fonctionner dans 2 modes : le mode maître et le mode esclave. Le SPI est utilisé dans de nombreuses applications telles que : Robot télé opération, est le contrôleur hôte et le satellite, les contrôleurs sont mis en réseau via le protocole SPI, avec une vitesse maximale de 5 Mbps. Dans ce travail notre objective est le développement d'une interface Série de Type SPI qui sera implémenter sur FPGA est gérée par une softcore de type Microblaze. Le choix de ce type de circuit n'est pas au hasard mais vu leur avantages. On peut citer par exemple : mise en marché rapide, court temps de conception et de développement et les faibles frais non récurrents **[2].**

Ce mémoire est structuré en trois chapitres : dans le premier chapitre est divisé on deux partie la première est consacré à l'étude de la conception des systèmes numériques embarqués, et les circuits logiques programmables. Dans la deuxième partie nous présentons les systèmes sur puce (SOC) ainsi que le flot de conception conjointe logicielle/matérielle. Le chapitre deux présentes les différents méthodes de transfert des données ,ainsi les modes d'interface série :

SPI, ou I2C, et UART. Finalement un troisième chapitre qui présente l'environnement de développement, la description de l'architecture matérielle de notre conception, développé et implémenté sur un circuit FPGA, on va détailler ainsi les différentes étapes de l'implémentation et les différents algorithmes de transmission/réception.

CHAPITRE I : GENERALITE DU SYSTEME EMBARQUE

1. Introduction :

Dans ce chapitre on présente une étude sur la conception des systèmes numériques embarqués. Nous introduisons dans un premier lieu l'évolution des systèmes numériques et les circuits logiques programmables. Ensuite, nous présentons les systèmes sur puce (SOC) ainsi que le flot de conception conjointe logicielle/matérielle. Nous introduisons à la fin de ce chapitre les plateformes multi-composants reconfigurables.

2. Conception des systèmes numériques :

Un calcul numérique est un ensemble de calculs réalisés sur un système informatique, encore appelé système numérique (ou ordinateur). Pendant longtemps, les calculs sont effectués par les ordinateurs et entièrement pris en charge par le processeur central (CPU). On ce processeur s'avère insuffisant dans un certain nombre de domaines. Des circuits plus efficaces ont été créés et travaillant en parallèle avec le processeur, spécialement pour des tâches bien définies.

En plus, les évolutions technologiques acquises dans le monde industriel actuel ont permis l'intégration de plus en plus de fonctions sur le même circuit intégré numérique.

Aujourd'hui, on arrive à embarquer plusieurs microprocesseurs, des accélérateurs matériels, des systèmes de communications diverses, des systèmes d'exploitation, etc.
Ceci permet donc de réaliser toutes les fonctions nécessaires pour effectuer des traitements informatiques complexes sur la même puce, d'où la naissance de la notion des systèmes sur puce (SOC pour System on Chip), successeurs des circuits spécialisés ASIC (Application Specific Integrated Circuit). Dans cette première partie, nous définissons les différents processeurs et les différents circuits programmables.

2.1. Architectures à base d'un processeur

Les GPP (General Purpose Processor) sont des processeurs dont l'image est souvent associée à celle de l'ordinateur personnel (PC). Leur popularité est liée à leur flexibilité, leur simplicité d'utilisation et leur puissance de calcul. L'association processeur-PC n'est toujours vraie, puisque les processeurs destinés aux systèmes embraqués occupent la plus grande partie du marché des processeurs. Par contre, les processeurs destinés aux systèmes embarqués sont différents des processeurs destinés aux ordinateurs grand public. D'abord le concept CISC (Complex Instruction Set Computer) a subi au cours des années une « cure d'amincissement » afin d'arriver à des architectures RISC (ReducedInstruction Set Computer) dans lesquelles les

instructions et les modes d'adressage complexes étaient bannis. Des principes de simplification ont été mis en œuvre (comme par exemple le codage uniforme des instructions) afin de permettre notamment d'avoir des compilateurs fournissant de bonnes performances.

2.1.1. Les processeurs d'usage général (GPP) :

La puissance de calcul et la flexibilité sont les objectifs premiers d'un GPP. Ils sont atteints grâce à une fréquence de fonctionnement élevée et des architectures matérielles et logicielles adaptées : super scalaire, fort pipeline, prédiction de branchement, caches multi-niveaux, instructions spécialisés (par exemple l'instruction multimédia : MMX pour le Pentium et AltiVec pour le PowerPC) et extension SIMD (Single-Instruction Multi-Data). Cependant, un GPP n'est pas bien adapté aux applications embarquées du traitement du signal et de l'image, et cela à cause de deux principales raisons :

- Dissipation d'énergie généralement élevée dans les GPP.
- Les GPP ne sont pas adaptés au temps réel pour les applications complexes.

2.1.2. Les processeurs de traitement du signal :

Certains systèmes numériques intègrent également des processeurs à usages spécifiques pour des applications de traitement du signal et des images (DSP pour Digital Signal Processors). Ces processeurs spécialisés sont surtout destinés aux traitements d'images de type «bas niveau» pour lesquels les calculs sont réguliers (les traitements sont identiques pour toutes les données d'entrée) et doivent être réalisés à hautes fréquences.

Les DSP sont des processeurs simplifiés relativement semblables aux GPP. Leur particularité essentielle est qu'ils sont conçus pour effectuer des calculs en temps réel et intègrent donc de nombreux opérateurs. En effet leur architecture interne est dimensionnée pour le calcul intensif.

2.1.3. Les processeurs reconfigurables :

En plus des GPP et des DSP, on retrouve également des processeurs reconfigurables qui peuvent intégrer des unités fonctionnelles selon le besoin. Ces processeurs peuvent être taillés sur mesure selon les besoins de l'application et de gagner par conséquent énormément de ressources. En effet, un processeur softcore (synthétisable) est un processeur implémenté sur un système reprogrammable comme un FPGA (Field Programmable Gâte Array). Si on

intègre plusieurs processeurs sur la même puce, on parle alors de système sur puce programmable (System on Programmable Chip ou SoPC).

Architecture très flexible de par sa nature, une implémentation softcore d'un processeur peut être reconfigurée en tout temps contrairement à un processeur dit hardcore dont le cœur dispose de sa propre puce qui ne peut être modifiée. Un processeur softcore s'adapte donc aux besoins de ses développeurs et aux contraintes matérielles (périphériques, performances, consommation, etc.).

Toutefois, ces performances sont inférieures à celles d'un processeur hardcore. Un softcore est en général programmé dans un langage de description matérielle comme le VHDL (acronyme de VHSIC-HDL : Very High Speed Integrated Circuit Hardware Description Language) ou le Verilog. Parmi les processeurs softcore les plus connus, on peut citer : le Microblaze de Xilinx1, le NIOS de la société Altera2, Openrisc d'Opencore3 et le LEON de chez Gaisler Research4. **[3]**

2.2. Les circuits logiques programmables :

Un circuit logique programmable, ou réseau logique programmable, est un circuit logique intégré qui peut être reprogrammé après sa fabrication. Il est composé de nombreuses cellules logiques élémentaires librement assemblables. Un circuit logique programmable permet la réalisation de circuits logiques à grand nombre de variables d'entrée et de sortie, selon la spécification de l'utilisateur et en utilisant un même support matériel.

Les premiers circuits programmables ont été les PROM (Programmable Read Only Memory). Dans ces circuits, le bus d'adresses est utilisé comme entrée et le bus de données comme sortie du circuit logique. Les limitations de ce type de composants pour la réalisation d'architectures logiques ont vite trouvé leurs limites et nous avons vu apparaître les FPLA (Field Programmable Logic Array).

Actuellement, on trouve différentes familles de circuits programmables tels que les PLD (Programmable Logic Device), les CPLDs (Complex Logic Programmable Device) et les FPGAs.

La différence entre ces composants est structurelle. Les CPLDs sont des composants pour la plupart reprogrammables électriquement ou à fusibles, peu chers et très rapides (fréquence de fonctionnement élevée) mais avec une capacité fonctionnelle moindre que les FPGA. La figure suivante montre les différentes familles des circuits programmables :

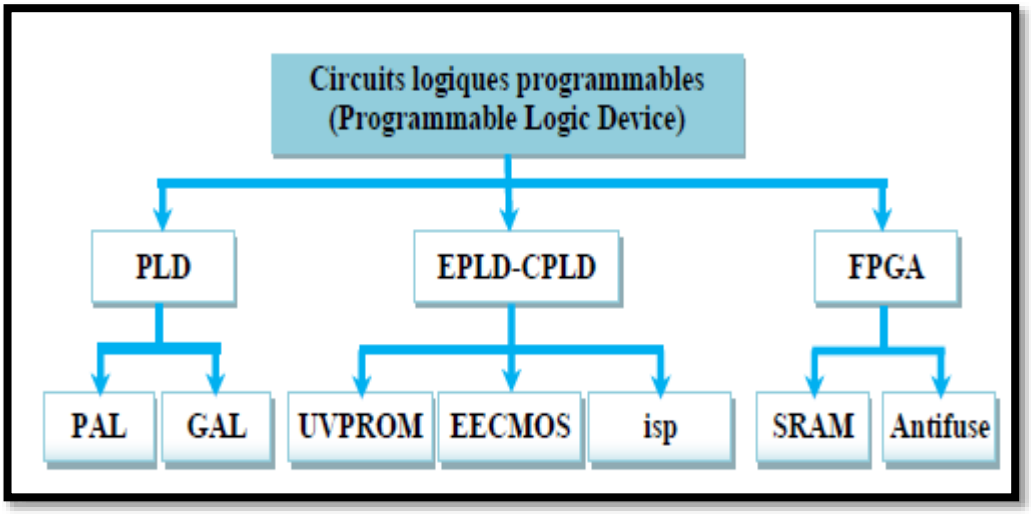


Figure I.1. Les différents circuits programmables.

2.2.1. Les FPGAs :

Les FPGAs, ou bien les réseaux logiques programmables, sont des composants électroniques programmables de la famille des PLDs. Un FPGA est un ensemble de blocs logiques élémentaires que l'utilisateur peut interconnecter pour réaliser les fonctions logiques de son choix. La densité des portes est importante et sans cesse en évolution. L'avantage d'un FPGA est leur grande souplesse dans leur technologie permettant une réutilisation à volonté et en un temps très court (quelques millisecondes) dans des algorithmes différents. Le progrès technologique permet de faire des composants toujours plus rapides et à plus haute intégration, autorisant la programmation d'applications importantes.

Grâce à l'évolution des procédés de fabrication, ces composants peuvent actuellement supporter des applications complexes. Ils sont constitués d'un réseau de blocs logiques, de blocs mémoires, de blocs dédiés et d'entrées/sorties. L'ensemble est relié par un réseau d'interconnexions programmable. Les blocs logiques permettent de réaliser des opérations avec quelques variables à travers une LUT (Look Up Table). Le résultat peut être éventuellement stocké dans un registre, les blocs RAM permettent d'implanter des mémoires adressables et des FIFO (First In, First Out), les blocs dédiés permettent de réaliser facilement de nombreuses opérations de traitement (blocs DSP), de gérer l'horloge, ou des interfaces de communication (RocketIO, Ethernet, PCI Express). Les nombreux ports permettent également de connecter des périphériques de la plateforme matérielle à base de FPGA.

Les FPGA se programment grâce à leurs LUT et leur réseau d'interconnexion. La programmation se fait avec un langage de programmation hardware tel que le VHDL ou bien le Verilog. L'outil de développement transforme cette description en un fichier de

configuration du FPGA en plusieurs étapes : le HDL (Hardware Design Language) doit d'abord être synthétisé (transformé en éléments logiques de base), puis les éléments doivent être placés sur le composant (placement) et enfin interconnectés (routage).

La Figure I.2 montre la structure interne d'un FPGA de type matrice symétrique. Il s'agit de l'architecture que l'on retrouve dans les FPGA actuels. L'utilisateur peut programmer la fonction réalisée par chaque cellule (CLB : Configurable Logic Block). On programme aussi les interconnexions entre les cellules. Les FPGA les plus récents sont configurables en une centaine de millisecondes. Les FPGA sont utilisés pour un développement rapide et bon marché des ASIC.

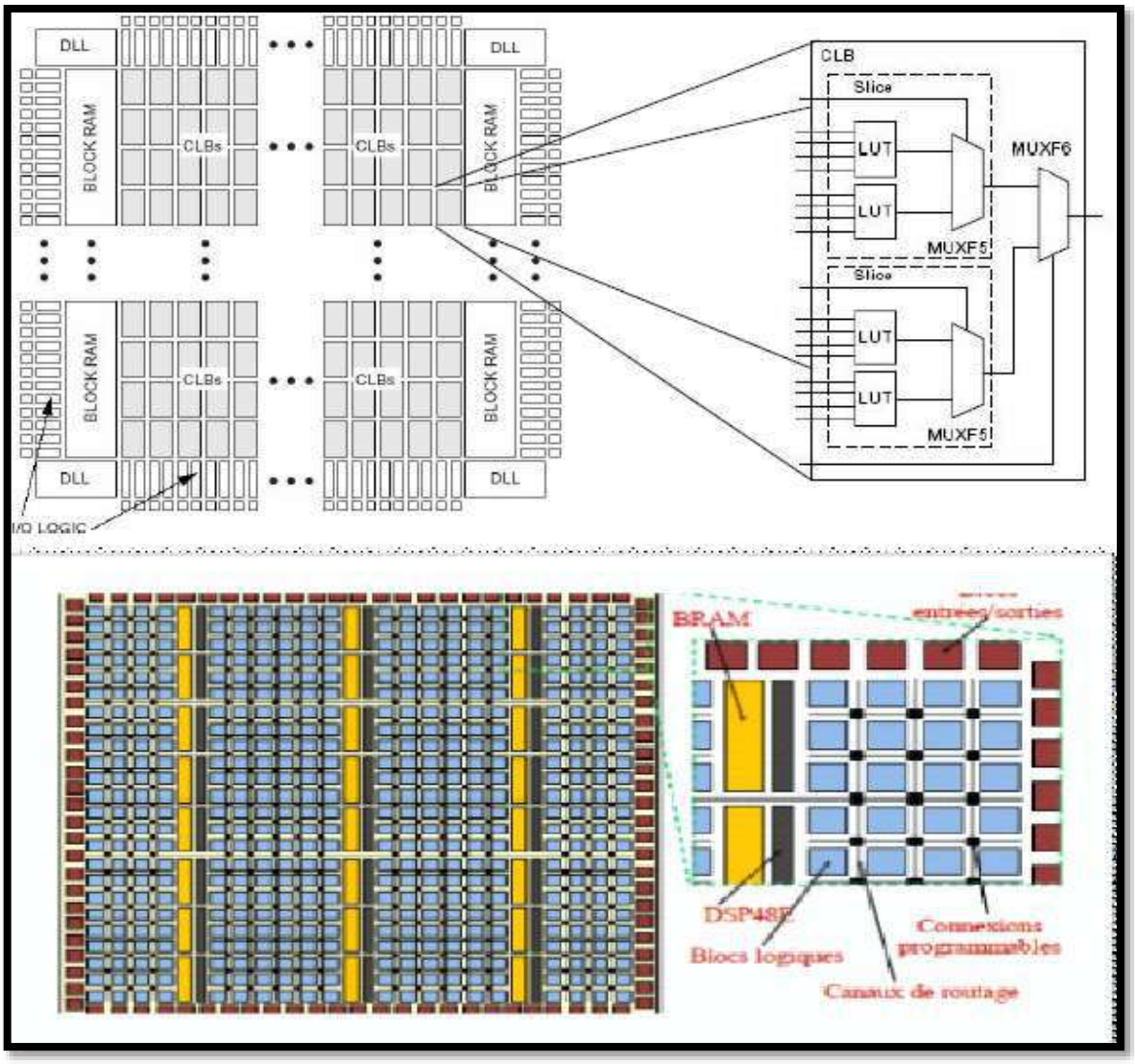


Figure I.2. Architecture conceptuelle d'un FPGA (Architecture, Interconnexions et CLB).

2.2.2. Les ASICs :

Un ASIC est un circuit électronique dédié. Il peut regrouper sur une même puce tous les éléments actifs nécessaires à la réalisation d'une fonction ou d'un ensemble électronique.

Ce composant n'est pas modifiable. Comparé à un FPGA, il offre une capacité d'intégration plus élevée, une vitesse plus rapide et un coût plus faible. En effet, les ASIC permettent l'intégration d'un nombre très important d'opérateurs de calculs et de mémoire. Cependant, le développement d'un ASIC est très long. Il s'agit d'un circuit intégré conçu exclusivement pour le projet ou l'application qui l'utilise. Grâce aux ASIC les concepteurs peuvent désormais regrouper dans un seul boîtier toutes les fonctions nécessaires à leur application en occupant une surface comparable à celle d'un simple composant. Dans un produit, le circuit intégré est un élément porteur d'avenir, fiable, performant et peu encombrant, de plus il assure la confidentialité du savoir-faire.

3. Les systèmes numériques embarqués :

De nos jours, les systèmes électroniques embarqués ont été introduits dans de nombreux domaines d'applications tels que l'automobile, l'avionique, les systèmes multimédia, les appareils électroménagers, les terminaux de communication sans fil, etc.

Les concepteurs doivent donc gérer les développements de systèmes électroniques embarqués multidisciplinaires et multiprocesseurs.

L'architecture générale d'un système sur puce intègre trois principaux types de composants qui sont : les unités de traitements, les mémoires et les systèmes de communication.

On peut retrouver également des adaptateurs (wrapper ou bien bridge) qui servent à traduire les protocoles de communication pour connecter des composants ayant des entrées/sorties incompatibles.

3.1. Codesign : Conception conjointe logicielle/matérielle :

La conception des systèmes contenant des composants hardware et software n'est pas un problème nouveau. Les techniques classiques de conception de systèmes imposent de séparer le hardware du software dans le cycle de conception. En outre, le consensus grandissant pour un modèle de systèmes informatiques «en couches» semble éviter aux développeurs software de telles préoccupations. Ainsi, les concepteurs de hardware spécifient leurs produits sans totalement apprécier les besoins, en ressources matérielles, des logiciels qui s'exécuteront dessus. De leur côté, les concepteurs de logiciels ne tirent pas suffisamment profit des ressources matérielles de la plate-forme hardware sur laquelle seront implantés leurs produits (Le software et le hardware sont deux activités qui ont toujours été menées de manières relativement indépendantes). Finalement, la phase d'intégration des deux parties nécessite

souvent des modifications parfois majeures du logiciel et/ou du matériel. Cette approche présente peu de flexibilité dans l'évaluation des différentes options de conception et de répartition du hardware et du software.La tendance actuelle, principalement lorsqu'il s'agit de systèmes temps réel complexes (systèmes critiques embarqués par exemple), est de tenir compte dès la phase de spécification des interactions entre les deux parties. Ainsi, la séparation est effectuée le plus tard possible dans le cycle de conception tendant à unifier la spécification pour qu'elle englobe le hardware et le software. Cette approche est nommée "hardware/software codesign", ou "codesign". Le système est conçu à partir d'une spécification unique décrivant son architecture et/ou son comportement. Après la phasede spécification survient une phase de partitionnement du système ayant pour but de décomposer ce dernier en partitions comportant trois parties :

- Une partie matérielle implantée sous forme de circuits hardware. Cette partie peut être un IP récupéré à partir d'une bibliothèque ou bien accélérateur matériel réalisé spécialement pour une tâche précise.
- Une partie logicielle implantée sous forme d'un programme exécutable sur un processeur. Le processeur peut être un GPP ou bien un processeur reconfigurable (configuré à l'avance selon les besoins de l'application).
- Une interface de communication entre les deux parties.

Les partitions obtenues doivent ensuite être vérifiées et validées avant de passer à la phase de synthèse et de mise en œuvre. Un feed-back permet de revenir à la phase de partitionnement en affinant différemment les poids associés aux contraintes si les partitions obtenues ne s'avèrent pas satisfaisantes.

3.1.1. Flot de la conception conjointe logicielle/matérielle :

Le développement de systèmes complexes fait de plus en plus appel à des composants logiciels et spécifiques. La conception conjointe logicielle/matérielle apporte des solutions à ce type de développement. Elle repose sur un ensemble d'étapes qui permettent de synthétiser, à partir d'une description du système au niveau fonctionnel, un système sur puce intégrant des composants logiciels et matériels qui respectent les contraintes de conception (temps et surface) imposées au départ. Un flot de conception standard est généralement constitué de trois étapes principales : la spécification, le partitionnement, et la vérification conjointe HW/SW (Figure I.3).

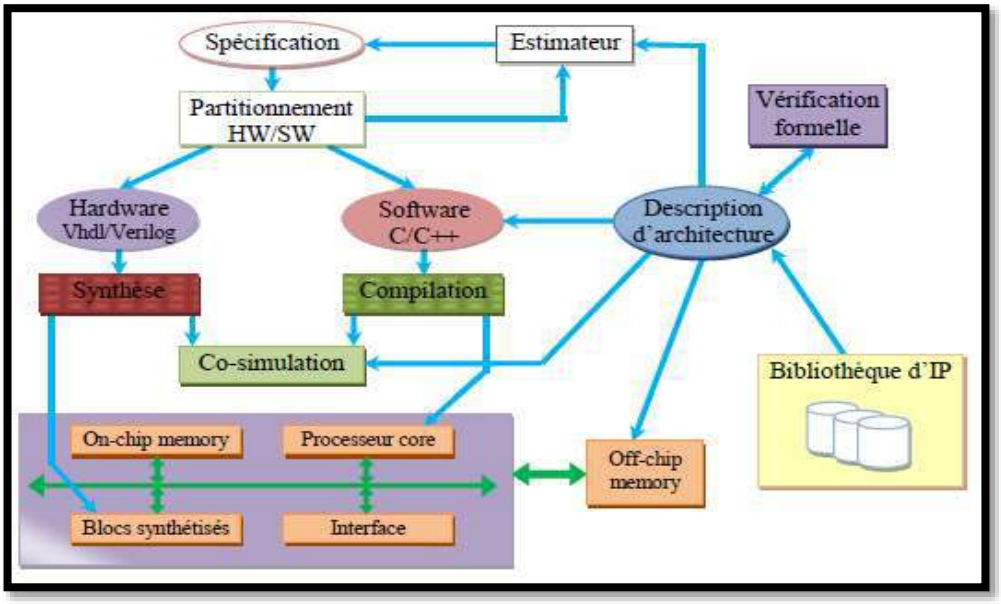


Figure I.3. Flot de conception standard**.**

D'après avoir figure, les étapes principales de la conception conjointe logicielle/matérielle peut être résumées comme suit :

1. Le point de départ est une spécification au niveau du système de l'application à réaliser. Cette spécification permet de décrire le comportement fonctionnel du système sans tenir compte de l'aspect architectural de celui-ci. A ce niveau de la conception, on ne pourra en aucun cas obtenir des résultats pertinents au niveau des performances et timing.
2. L'étape qui suit la vérification fonctionnelle du modèle au niveau système, sera donc le partitionnement logiciel/matériel. A ce stade de conception, on intègre les détails architecturaux de communication et d'ordonnancement des opérations. Plusieurs tentatives de simulation seront effectuées pour choisir la meilleure répartition entre les parties logicielles et les parties matérielles.
3. Après le découpage logiciel/matériel de la spécification d'entrée, l'étape de synthèse appelée aussi d'implémentation aura lieu. Dans cette étape, nous obtenons une description au niveau RTL pour la partie matérielle et un code source (e.g. C ou C++) pour la partie logicielle du système. Bien évidemment, la vérification et la validation de la fonctionnalité du modèle ainsi généré devrait être faite. A ce stade de conception, nous pouvons effectuer des analyses de performances de l'architecture au niveau cycle et au niveau bit à travers des Co-simulations.
4. La dernière étape du flot de conception consiste en premier lieu à la synthèse logique de la partie RTL du système. Ensuite les fonctions logiques qui ont été synthétisées seront

placées et routées sur le plan de la puce. Ce processus est réalisé à l'aide des outils de synthèses commerciaux comme par exemple : Synplify , XST, Leonardo Spectrum, etc. Quant à la partie logicielle du système, elle sera compilée pour générer une image hexadécimale. Enfin, une Co_simulation sera établie après la synthèse logique afin d'obtenir les performances en surface et en consommation d'énergie. Une fois l'architecture est validée, nous passons aux tests réels sur les cartes reconfigurables de type FPGA.

Plusieurs outils et compagnies ont adopté cette approche de codesign par le fait de sa simplicité de synthèse et d'intégration de nouveaux composants. Parmi ces outils de développement notons celui proposé par Xilinx (EDK pour Embedded Development Kit).

3.1.2. Les composants virtuels (IPs) :

Les outils de développement et de conception, des circuits spécialisés, sont de plus en plus performants malgré l'augmentation considérable de la capacité des composants matériels. Cependant il est indispensable de réutiliser des conceptions déjà réalisées dans ces outils de développement. En effet, afin de réduire le temps de conception et de prototypage des systèmes sur puce, une approche de réutilisation des composants a été introduite. Cette approche consiste à reprendre des modèles de composants au niveau RTL qui sont considérés comme étant des propriétés intellectuelles (IP). LesIPs peuvent être définis comme des composants virtuels, ce sont des circuits génériques ainsi que des outils associés que l'on peut trouver sur le marché ou bien sous une forme téléchargeable (OpenSource).

Un IP peut être une fonction de base ou bien un bloc plus complexe. L'intégration de cœurs de processeurs est également possible, tel que le MicroBlaze ou bien le NIOS. Ces processeurs sont relativement simples, et personnalisables en fonction de l'application. L'assemblage des IPs permet de décrire rapidement des applications complexes. On peut utiliser des IP sans les avoir conçus, en ayant uniquement une description haut-niveau (vue externe etfonctionnelle) de leur comportement. Un composant IP peut apparaître sous différentes formes :

- IP Logiciel (softcore) : Le composant est livré sous sa forme HDL synthétisable. Son principal avantage est sa portabilité. La propriété du fichier source est en soi la meilleure documentation, on peut ainsi maintenir le produit pendant des années et éventuellement modifier à volonté la source et même changer de technologie cible. L'inconvénient majeur est qu'il ne peut être prédictif en termes de superficie,

puissance et temps. Le travail d'optimisation du circuit final est à la charge de l'intégrateur du système.

- IP Matériel (hardcore) : Dans ce cas, la description du bloc IP est au niveau porte logique, optimisée pour une technologie particulière (optimisationgarantie). Cela englobe la netlist entière, le routage et l'optimisation pour une librairie technologique spécifique et un lay-out personnalisé.L'inconvénient est qu'il est moins flexible car le processus est dépendant de la technologie, par contre il a l'avantage d'être prédictif.
- IP Firm (firmcore) : Le bloc IP firmcore offre un compromis entre les deux autres IPs, plus flexible que le hardcore, plus prédictif en termes de performance et de surface que le softcore (parexemple une interface USB où la vitesse de la norme ne pourra être assurée que dans unetechnologie bien définie). En général, le travail de synthèse HDL est déjà réalisé pour unetechnologie cible donnant lieu à une description par netlist (format EDIF par exemple).

Dans un système complexe, les IPs communicants à travers un réseau de communication sont aussi susceptibles d'être réutilisables après reconfiguration. La conception se résume à l'assemblage de l'ensemble des unités de calculs qui ont été choisies au préalable et les faire communiquer à travers unréseau de communication. L'inconvénient majeur de cette approche réside dans le fait que le concepteur est confronté très souvent aux problèmes d'adaptation des entrées/sorties des composants réutilisés avec le réseau d'interconnexions. Ce problème d'adaptation amène le concepteur à intégrer des adaptateurs de bus autrement dit, des interfaces de bus entre le réseau de communication et les composants du système.

3.2. Les systèmes sur puce (SOC) :

Un système mono puce, appelé encore SOC ou système sur puce, désigne l'intégration d'un système complet, avec de nombreux composants, sur une seule puce. Un Soc peut intégrer, aujourd'hui, plusieurs millions de transistors sur une surface de silicium de quelques millimètres carré.

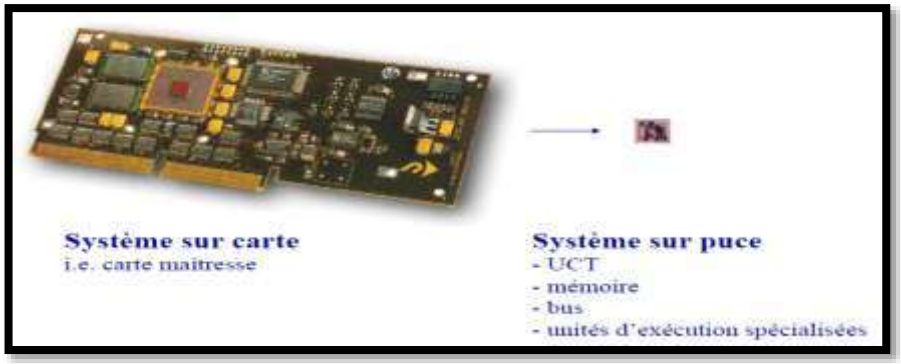
3.2.1. Définitions d'un SOC :

Le terme SOC est devenu très populaire dans le milieu industriel malgré l'absence d'une définition standard. Certains considèrent qu'un circuit complexe en fait automatiquement un SOC, mais cela inclurait probablement chaque circuit existant aujourd'hui. Une autre définition considère le système embarqué comme un système électronique et informatique

autonome ne possédant pas des entrées/sorties standards comme un clavier ou un écran d'ordinateur. Une définition plus appropriée de systèmemono puce serait : un système complet sur une seule pièce de silicium, résultant de la cohabitation de nombreuses fonctions déjà complexes en elles-mêmes : processeurs, DSP, mémoires, bus, convertisseurs, blocs analogiques, etc. Il doit comporter au minimum une unité de traitement de logiciel (un processeur qui exécute un logiciel dédié pour la réalisation d'une fonctionnalité précise) et doit dépendre d'aucun (ou de très peu) des composants externes pour exécuter sa tâche, en conséquence, il nécessite à la fois du matériel et du logiciel. En plus un système embarqué n'exécute pas une application scientifique ou commerciale.

L'objectif est diminué au minimum le nombre de composants sur une carte pour mettre tout sur une seule puce. **[5]**

Figure I.4 : Objectif du SOC



La figure suivant présente les caractéristiques principales d'un système embarqué typique. Il se compose d'un cœur de processeur , d'un processeur de signal numérique (DSP), de la mémoire embarquée, et de quelques périphériques tels qu'un DMA et un contrôleur d'E/S. Le CPU peut exécuter plusieurs tâches via l'intégration d'un OS (système d'exploitation). Le DSP est habituellement responsable de décharger le CPU en faisant le calcul numérique sur les signaux de provenance du convertisseur A/N**. [3]**

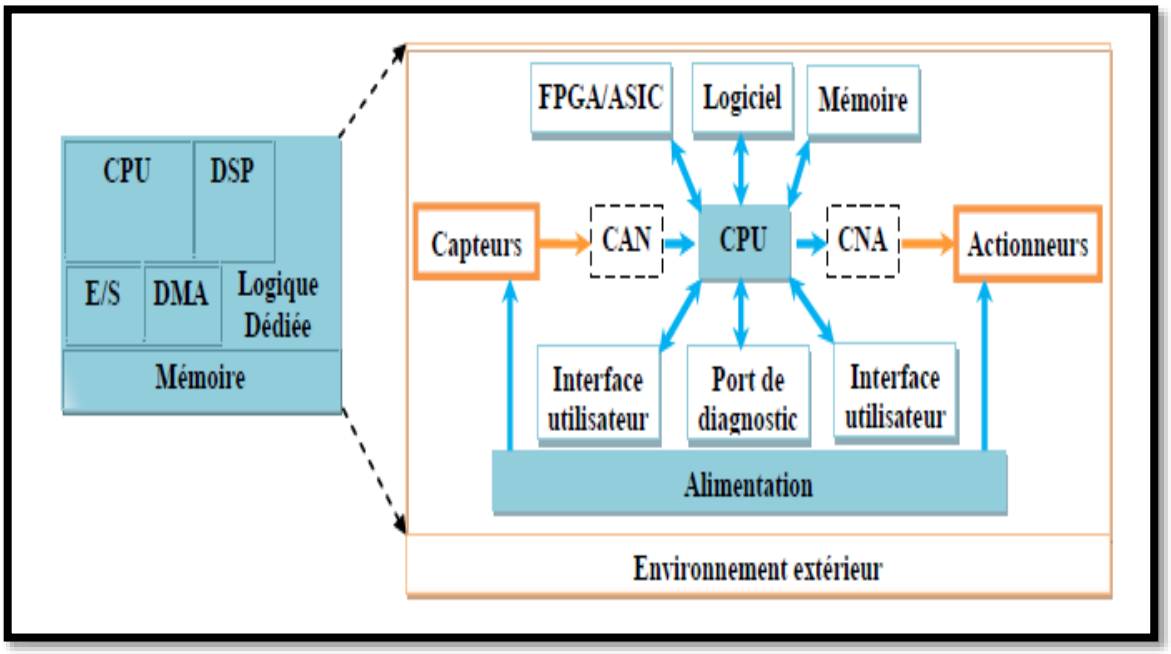


Figure I.5. Système embarqué (SOC) typique.

3.2.2. Les SOC et les SoPC :

L'approche SoC a été créée dans un premier temps pour le développement d'ASIC mais a été étendue pour le développement de FPGA. On parle alors de SoPC pour System on Programmable Chip. Le SoPC (acronyme a été inventé par Synopsys) correspond à l'intégration d'un ou plusieurs cœurs de processeurs et de ses périphériques sur une même puce programmable de type FPGA. Le logiciel est alors situé soit dans une mémoire du circuit FPGA si l'empreinte mémoire le permet, soit dans une mémoire externe le plus souvent **[4]**. Les deux approches peuvent être résumées à :

- ✓ *L'approche SOC (généralement en technologie ASIC) :* répond aux besoins de performances et d'intégration mais elle est peu adaptée à l'évolutivité des systèmes et reste réservée aux grands volumes de production. En effet, un SoC peut être retenu

pour les applications destinées au grand public. Il permet de meilleures performances en termes de consommation, de vitesse et de surface, mais, la fabrication et le test sont des étapes longues et coûteuses. De plus, un SoC est figé et n'est donc pas réutilisable pour une autre application.

- ✓ *L'approche SoPC (technologie FPGA) :* Spécialement pour résoudre les problèmes de développement et les prototypages rapides, ainsi que les composants reconfigurables en quelques « ms » et à volonté. Il permet donc un développement et prototypage rapide du système. En contrepartie, la densité d'intégration est généralement moindre (~10 Millions de portes) et la consommation d'énergie est plus grande avec une performance plus faible que Les composants FPGA permettent d'implanter deux types de SoPC. Le premier type de SoPC utilise uniquement des IP soft de processeur tels que le MicroBlaze et le NIOS. Le nombre de processeurs intégrés sur un même composant dépend des ressources du composant utilisé. Le deuxième type de composants en plus de la logique reconfigurable, implémente en dur un ou plusieurs cœurs de processeurs (PowerPC pour Xilinx, ARM pour Altera). Ceci n'empêche en rien l'utilisation de processeurs Soft (netlist). Ces deux types de SoPC utilisent les mêmes flots de conception. Chez Xilinx, tout le flot de conception est regroupé sous la forme d'un seul environnement dit Xilinx Platform Studio (XPS).

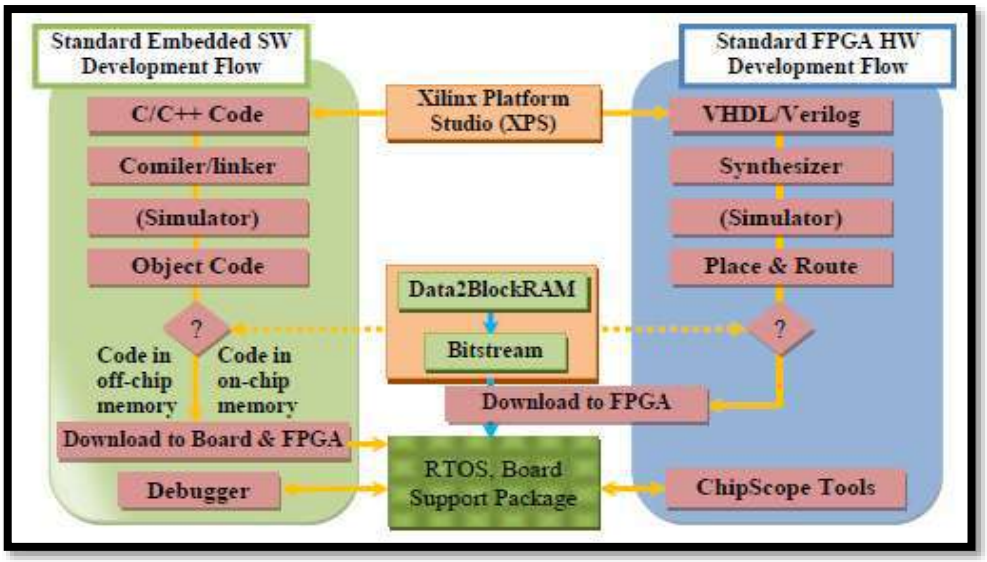


Figure I.6. Flot de conception Xilinx**.**

3.3. Communications sur puce :

Les systèmes logiques sur puce sont souvent conçus de manière à être dimensionnés pour une seule application particulière. Chaque application a sa propre architecture, de ce fait l'extension ou la réutilisation de telle architecture est difficile et nécessite un effort de réadaptation et pour cette raison plusieurs types de communication sur puce ont été proposés. Un système de communication permet d'établir des connexions entre tous les composants dans un SoC. Il se présente sous forme de bus ou bien de réseau de communication. La plupart des systèmes embarqués sont basés sur des architectures de bus. Quant aux réseaux, on les retrouve dans certaines architectures complexes, qui nécessitent une bande passante considérable.

3.3.1. Les différents types des communications sur puce :

a. Les connexions point-à-point :

Permettent d'établir des liaisons directes entre lescomposants du SOC.Il existe deux types de connexions point à point avec ou sans mémoire partagée. Généralement dans le domaine des SOC, on utilise des FIFOs pour la réalisation de ce type de connexion, la profondeur de laFIFO est déterminée par l'utilisateur ou bien par l'outil de conception lors de la phase d'exploration d'architecture.

b. Les bus :

Un bus représente un canal unique de communication. Ce canalsupporte le transfert detoutes les données entre les différents composants du SOC. Une architecture de bus intègre deuxprincipaux éléments qui sont le canal et l'arbitre. Il existe deux typesd'architectures de bus, des architectures dites partagées et des architectures matricielles.

- Les bus partagés sont très répandus par leur souplesse et légèreté dans l'occupation de ressources, mais l'ennui dans l'utilisation des bus partagés réside dans le fait qu'on est limité par la bande passante du système. Le bus partagé intègre un seul et unique arbitre qui sera responsable de la gestion des accès au bus. Dans ce type d'architectures, à chaque instant, un seul composant peut avoir accès au bus.
- Contrairement aux architectures partagées, les architectures matricielles (cross bar) augmentent davantage le parallélisme du système au détriment des ressources matérielles allouées. Les bus matriciels nécessitent des ressources matérielles considérables. Il existe deux topologies différentes dans la conception des bus matriciels : des architectures distribuées et des architectures centralisées. La différence entre les deux architectures réside dans la façon de gérer l'arbitrage du bus. Actuellement, les concepteurs de SOC ont plus tendance à s'orienter vers des conceptions de bus matriciels.

c. Les réseaux sur puce (NoC) :

Comme le problème majeur de la communication revient toujours à la limitation en bande passante du bus qui reste plus ou moins insuffisante pour la plupart des applications complexes, les réseaux sur puce (NoC : Network-On-Chip) ont été introduits afin d'y remédier à cette limitation et d'intégrer davantage de nouveaux processeurs et de composants matériels. Un réseau sur puce est constitué d'un ensemble de composants et de routeurs. Les routeurs sont reliés entre eux via des canaux de communication. Cependant, l'inconvénient majeur de ce typed'architecture revient à l'existence de blocages (dreadlocks), dus à des accès simultanés à des ressources partagées, et à la difficulté d'analyser les performances effectives du réseau (latence, dreadlocks, etc.).

3.3.2. Exemple : Le Bus CoreConnect :

Spécialement pour des systèmes sur puce, IBM a proposé l'architecture CoreConnect composée de trois bus qui permettent la connexion : de cœurs de processeur, de logique dédiée, et différents autres accélérateurs matériels. Les trois bus de CoreConnect sont :

- *Processor Local Bus (PLB) :* Généralement, le PLB fournit un chemin de données large bande.En effet il est utilisé pour connecter des composants tels que les cœurs de processeurs, les interfaces avec les mémoires externes, et les contrôleurs DMA (Direct Memory Accès).
- *On-chip Peripheral Bus (OPB) :* L'OPB est utilisé pour réduire la charge du bus PLB. Il est plus adapté aux composants tels que les ports série, les ports parallèles, UARTs, GPIO et tous les composants à faible largeur de bande. Un Maître sur le PLB peut accéder aux composants connectés à l'OPB via une macro passerelle (Bridge). Cette passerelle est vue en tant que maître par l'OPB et en tant qu'esclave par le PLB. Les registres d'état et de configuration de faible performance sont généralement lus et écrits via le bus DCR.
- *Device Control Register (DCR) :* Le DCR est principalement utilisé pour accéder aux registres de l'état et de contrôle au sein des différents bus PLB et OPB.

Xilinx a repris le comportement de tous les bus CoreConnect d'IBM et les a décrits en VHDL, ce qui permet d'avoir des bus paramétrables. En effet, pour simplifier la connexion d'un module logique utilisateur à un des bus CoreConnect, Xilinx a ajouté une interface de bus préconçue paramétrable appelée IPIF (IP InterFace). L'IPIF gère les signaux du bus, le protocole de communication et plus généralement, l'ensemble des caractéristiques du bus. L'IPIF intègre aussi une interface de gestion, du logique utilisateur, appelée IPIC (IP InterConnect). Lorsque le logique utilisateur est conçu avec un IPIC, elle peut être portable et facilement réutilisable sur d'autres bus en changeant seulement l'IPIF.

Des fichiers VHDL qui instancient l'IPIF et fournissent le code nécessaire à l'utilisateur pour ajouter ses modules, simplifient la tâche de connexion du logique utilisateur. Ces fichiers sont les User Coré Reference Design. Un système de traitement réalisé par EDK doit être connecté le bus OPB et/ou le bus PLB, le périphérique ajouté au système (accélérateur matériel) doit être connecté à OPB ou bien PLB. La figure suivante montre la connexion du bus PLB dans un système à base d'un MicroBlaze, d'une DDR2 et d'une mémoire Compact Flash.

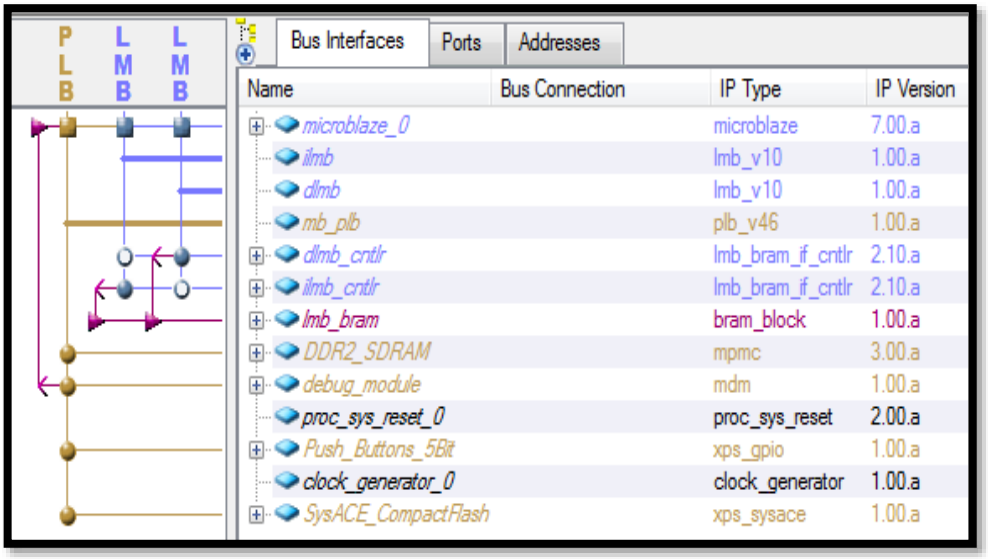


Figure I.7. Le bus PLB dans un système à base de MicroBlaze.

4. Les processeurs embarqués :

Un cœur de processeur est la partie matérielle qui traite les données en fonction du décodage des instructions contenues en mémoire. La grande quantité de données à prendre en compte ainsi que les contraintes en temps souhaitées, font du cœur de processeur un élément critique d'un système. Le cœur de processeur idéal doit donc fournir le maximum de performances sur une surface la plus réduite possible. Il doit être le plus flexible possible et ne pas subir de dégradation de performances en fonctions des différents environnements rencontrés (Outils de développement, taille de la cible ou électronique adjacente).

Les cœurs de processeurs peuvent être, soit des macro-cellules spécialisées, dites processeur Hardcore (PowerPC et ARM), soit des modules synthétisables, dits processeur softcore (MicroBlaze, NIOS-II, etc.). Ces soft corés sont configurables à travers plusieurs paramètres (profondeur du pipeline, taille du cache, présence ou non de multiplieurs, etc.). Un processeur softcore est réalisé à partir de la logique programmable présente sur la puce. Ainsi, un concepteur peut mettre autant de processeurs softcore que la puce le permet. A titre illustratif, un processeur MicroBlaze occupe seulement 4% de la surface d'un FPGA de la famille Virtex5-LX50 (composant à 28.800 cellules logiques), et 1,7% de la surface d'un FPGA de la famille Virtex5-LX110T (composant à 69.120 cellules logiques).

4.1. Les processeurs hardcore :

Le cœur d'un processeur hard est décrit au niveau physique (lay-out sur silicium). Dans ce cas, l'optimisation est maximale en termes de vitesse et de surface utilisée (technologie fixe).

Les performances du système sont parfaitement connues et ne varient pas d'un système à l'autre (surface, consommation, délais). Cependant, le cœur du processeur hard souffre d'une trop faible flexibilité ce qui le rend très spécifique à l'application pour laquelle il a été étudié.

a. Le processeur PowerPC405 de Xilinx :

Le PowerPC405 d'IBM est un processeur RISC 32 bits avec une architecture Harvard. Son architecture pipeline est de 5 étages avec une fréquence de fonctionnement de 400MHz. Il dispose de 32 registres de 32 bits, une mémoire cache d'instructions et de données de 16 KB chacune et d'une unité de gestion de mémoire (Memory Management Unit). Sur les circuits FPGA Xilinx supportant le processeur PowerPC, l'accès aux mémoires internes (Blocs RAM du FPGA) se fait à travers une unité "On-Chip memory Controllers".

Le cœur de processeur matériel PowerPC est enfoui dans quelques catégories de circuits FPGA (Virtex5-FXT par exemple), c'est à l'utilisateur de rajouter les périphériques qui sont nécessaires à son fonctionnement. Le processeur PowerPC 405 ainsi que le MicroBlaze sont exploitables grâce à l'outil EDK de Xilinx.

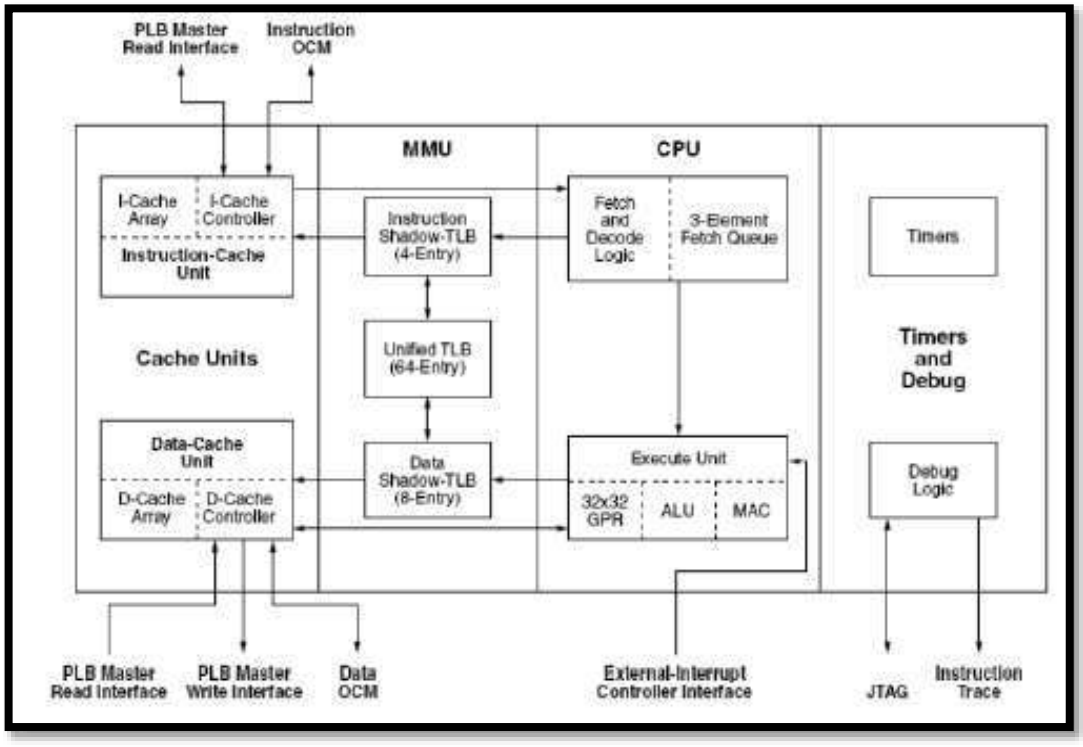


Figure I.8. Architecture interne du PowerPC405**.**

b. Le processeur ARM :

L'architecture ARM était originellement conçue pour un ordinateur de la société Ac0orn, puis elle a été complétée pour devenir une offre indépendante pour le marché de l'électronique embarquée. ARM est l'acronyme de Advanced Risc Machine (auparavant Acorn Risc Machine). Une particularité des processeurs ARM est leur mode de vente. En effet, ARM Ltd ne produit ni ne vend ses processeurs sous forme de circuits intégrés. La société vend les licences de ses processeurs de façon à ce qu'ils soient intégrés dans le silicium par d'autres fabricants. Actuellement, la majorité des grands fondeurs de puces proposent de l'architecture ARM. En plus, ARM Ltd a développé la norme AMBA qui est aujourd'hui un standard de bus informatique.

4.2. Les processeurs softcore :

La capacité d'intégration des FPGA permet d'y implémenter des processeurs IPs soft. Ce sont des processeurs décrits à l'aide d'un langage de description matériel (VHDL ou bien Verilog) pouvant être synthétisable. Ces processeurs reprennent exactement le fonctionnement et les performances des autres processeurs implémentés en hard, l'avantage ici réside dans leur capacité d'être paramétrée. En effet, un processeur softcore s'adapte aux besoins de ses développeurs ainsi qu'aux contraintes matérielles (périphériques, performances, ressources, consommation, etc.). Le fait de paramétrer un processeur permet d'avoir une flexibilité et une facilité à faire un compromis performance-coût accélérant ainsi le cycle de conception. Cependant, les performances d'un softcore sont, généralement, inférieures à celles d'un processeur hardcore**[3].** Parmi les produits existant en ce moment sur le marché on peut citer :

a. *Le processeur MicroBlaze de Xilinx* **:**

Le MicroBlaze est un microprocesseur virtuel qui est construit en combinant des blocs de code appelés noyaux à l'intérieur d'une matrice Xilinx Field Programmable Gâte (FPGA) (xilinx est une entreprise américaine de semi-conducteurs).

Un processeur RISC "soft IP" à 3 étages avec une architecture Harvard et avec 32 registres internes de 32 bits. Il dispose d'un bus d'instructions et de données internes et externes (ILMB, DLMB, IOPB et DOPB), une unité arithmétique et logique (ALU), une unité de changement de vitesse, et deux niveaux d'interrompre. Cette conception de base peut alors être configuré avec des fonctionnalités plus avancées pour répondre à tous les besoins exacts de l'application embarquée cible tels que: le baril shifter, diviseur, multiplicateur, simple précision unité à virgule flottante (FPU), des instructions et des caches de données, la gestion

des exceptions, logique de débogage, Fast Simplex Link (FLS) et d'autres interfaces. Cette flexibilité permet à l'utilisateur d'équilibrer les performances requises de l'application cible par rapport au coût du processeur programmable zone logique.

Le pipeline de MicroBlaze est un pipeline parallèle, divisé en trois étapes : Fetch, Décode, et exécution. En général, chaque étage a un cycle d'horloge pour terminer. Par conséquent, il faut trois cycles d'horloge (retards ou des stalles ignorant) pour l'instruction pour terminer. Chaque étape est active à chaque cycle d'horloge si trois instructions peuvent être exécutées simultanément, une dans chacun des trois étages de pipeline. MicroBlaze implémente une préfecture tampon qui réduit l'impact de plusieurs cycles mémoire d'instructions latence. Bien que le pipeline est bloqué par une instruction multi-cycles dans l'étage d'exécution d'instruction de la mémoire tampon de préfecture continue à charger instructions séquentielles. Une fois l'exécution des programmes vitae de pipeline stade fetch peut charger de nouvelles instructions directement de la préfecture tampon plutôt que d'avoir à attendre pour l'accès de mémoire d'instructions pour terminer. Le préfecture tampon fait partie de l'épine dorsale de l'architecture MicroBlaze et n'est pas la même chose que le cache d'instruction en option.

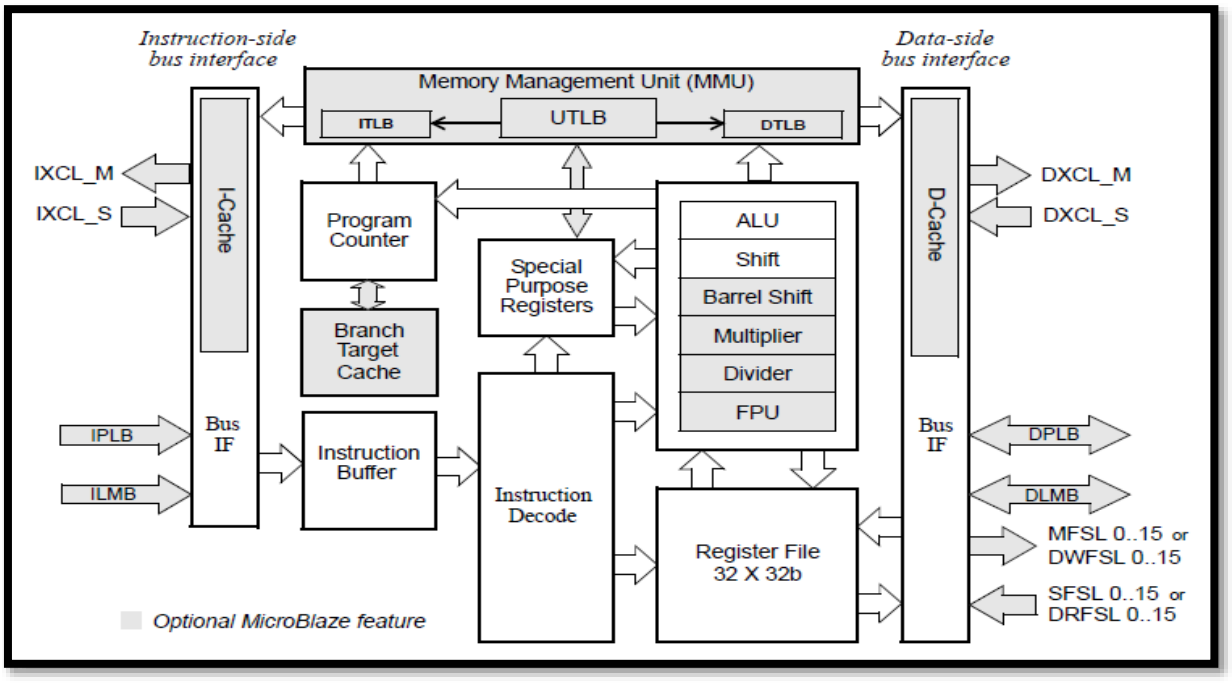


Figure I.9. schéma-bloc fonctionnel du noyau MicroBlaze**.**

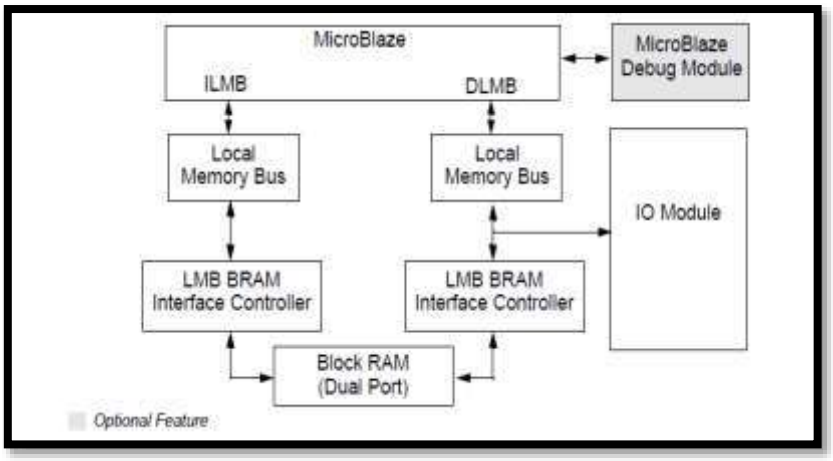


Figure I.10 : MicroBlaze Micro Controller System, MCS**.**

Toutes les instructions de Microblaze 32 bits de largeur et sont définis comme étant de type A ou B. instructions de type A ont jusqu'à deux opérandes de registre source et un opérande de registre de destination. Des instructions de type B ont un registre de source et un 16-bit opérande immédiat. Des instructions de type B avoir un seul opérande de registre de destination. Les instructions sont fournies dans les catégories suivantes fonctionnels : l'arithmétique, logique, branche, chargement / stockage, et spéciale.

MicroBlaze est un type de charge / magasin processeur ce qui signifie qu'il ne peut charger des données / des magasins de / vers la mémoire. Il ne peut pas faire des opérations sur les données directement en mémoire, place les données en mémoire doivent être portées à l'intérieur du MicroBlaze processeur et placé dans l'usage général registres de faire toutes les opérations. Les deux interfaces d'instructions et de données de MicroBlaze sont 32 bits de large et utilise Big-Endian, le format binaire inverse pour représenter des données (ordre des bits : Bit 0 Bit 1 ... Bit 30 Bit 31 bit 0 MSB et Bit 31 du LSB). MicroBlaze soutient mot (32 bits), demi-mot (16 bits), et l'octet accède à la mémoire de données. Accès de données doivent être alignés (c.-à-mot accède doivent être sur les limites de mot, demi-mot sur les limites demi-mot), à moins que le processeur est configuré pour supporter exceptions non alignés. Tous les accès d'instruction doivent être alignés mot.

La convention utilisée dans la pile MicroBlaze commence à partir d'un emplacement de mémoire supérieur et pousse vers le bas pour abaisser les emplacements de mémoire lorsque les éléments sont poussés sur la pile avec un appel de fonction. Les articles sont extraits de la pile de l'ordre inverse de leur mise en place ; élément à l'emplacement mémoire le plus bas de la pile va d'abord et etc.

MicroBlaze soutient également réinitialiser, interrompre, l'utilisateur exception, pause et matériels exceptions. Pour interruptions, MicroBlaze soutient une seule source d'interruption externe (connexion au port d'entrée d'interruption). Si plusieurs interruptions sont nécessaires, un contrôleur d'interruption doit être utilisé pour traiter plusieurs demandes d'interruption à MicroBlaze. Un contrôleur d'interruption est disponible pour une utilisation avec Xilinx Embedded Development Kit (EDK) outils logiciels

L'écriture de logiciels pour contrôler le processeur MicroBlaze doit être fait en C / C++ langue. Utilisation de C / C ++ est la méthode préférée par la plupart des gens et est le format que les outils du Xilinx Embedded Development Kit (EDK) de logiciels attendent. Les outils EDK ont construit dans les compilateurs C / C++ pour générer le code machine nécessaire pour le processeur MicroBlaze.

Le processeur MicroBlaze est inutile par lui-même sans un certain type de périphériques à connecter et EDK est livré avec un grand nombre de périphériques couramment utilisés. De nombreux types de systèmes peuvent être créés avec ces périphériques, mais il est probable que vous devrez créer votre propre périphérique personnalisé pour implémenter des fonctionnalités non disponibles dans les bibliothèques périphériques EDK et l'utiliser dans votre système de processeur.

Le bus utilisé avec le MicroBlaze est le bus OPB (On-chip Peripheral Bus). C'est un bus IBM Core-Connect utilisé aussi avec les processeurs PowerPC. Ce bus autorise un maximum de 16 maîtres et un nombre d'esclaves illimité. Il dispose d'une politique d'arbitrage (Bus multi-maîtres) paramétrable. **[4]**

Dans un système sur puce programmable à base de processeur MicroBlaze, le bus OPB est utilisé afin de connecter les périphériques dont les besoins en communication sont faibles. Autrement, Xilinx fournit le lien FSL (Fast Simplex Link) permettant des accès rapides (2 fronts d'horloge) des périphériques vers le MicroBlaze et vice versa (8 connexions FSL par MicroBlaze). Un autre type de bus LMB (Local Memory Bus) est utilisé pour accéder aux blocs RAM du FPGA. Dans le MicroBlaze, ce bus est utilisé pour les instructions et les données et il assure des accès rapides à la mémoire (1 front d'horloge).

En plus, de nombreux périphériques sont fournis avec le MicroBlaze, afin de constituer un microcontrôleur complet et personnalisable. Il y a, entre autres : contrôleur mémoire (SRAM, Flash), contrôleur mémoire SDRAM, UART lite, Timer/compteur avec fonction PWM,

interface SPI, contrôleur d'interruptions, GPIO (entrées-sorties génériques), convertisseurs A/N et N/A Delta-Sigma,

DMA, etc. MicroBlaze peut fonctionner en utilisant plusieurs systèmes d'exploitation (Xilinx MicroKernel, uClinux, FreeRTOS, etc.). Ces systèmes d'exploitation sont constitués généralement d'un ensemble de bibliothèques permettant d'obtenir des fonctions basiques telles que : pilotes de périphériques, séquencèrent de tâches, système de fichiers FAT, pile TCP/IP (avec le logiciel libre lwip), etc.

.

b. Les processeurs NIOS et NIOS-II de Altera

Le NIOS est un Processeur softcore propriétaire de la société Altera. Il est basé sur un cœur RISC 32 bits et doté du bus Avalon. Le NIOS-II est la deuxième génération de processeurs embarqués à cœur logiciel d'Altera, il annonce déjà une nouvelle aire pour l'électronique et l'informatique embarqué. NIOS-II incorpore de nombreuses améliorations sur l'architecture d'origine NIOS, le rendant plus adapté pour une large gamme d'applications d'électronique embarqué et systèmes de contrôle.

Contrairement au MicroBlaze, Synopsys DesignWare offre une licence de NIOS-II pour les standardcell ASICs, et grâce à cette licence, les concepteurs peuvent utiliser NIOS pour une production de masse d'ASIC. Pour bien gérer les applications à base de processeur NIOS, Altera a développé trois principaux outils de développement :

- L'outil Quartus pour l'implémentation du NIOS dans un FPGA ;
- Le développement du coeur et de ses composants ("Briques IP") s'effectue à l'aide du "SOPC Builder" ;
- Le développement du logiciel s'effectue en C sous NIOS-IDE (Windows) ou en utilisant nios2- toolchain sous Linux.

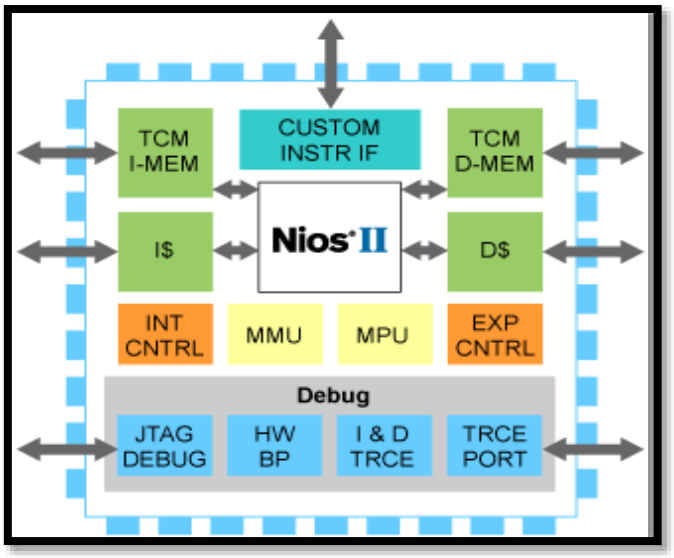


Figure I.11. Les processeurs NIOS et NIOS-II de Altéra

La nature embarquée de NIOS-II permet au concepteur de spécifier et générer un processeur sur mesure pour des besoins d'applications spécifiques. Les concepteurs peuvent étendre les fonctionnalités de base en y ajoutant une unité de gestion de mémoire préétablie pour définir des instructions personnalisées ou des périphériques personnalisés.

5. Conclusion :

Les systèmes sur puce (SOC) et les systèmes multiprocesseurs sur puce (MPSoC) sont deux aspects très utilisés de nos jours, ils sont motivés par l'augmentation continue de la capacité d'intégration (à cause des progrès des technologies CMOS) d'une part, et la complexité croissante des applications embarquées d'autre part, surtout en ce qui concerne les applications du multimédia. En effet, La conception de systèmes embarqués est généralement motivée par l'évolution technologique et la généralisation du multimédia

CHAPITRE II :
LES INTERFACES DE COMMUNICATION SERIE

1. Introduction :

Dans ce chapitre nous allons présenter les différentes méthodes de transfert des données en série, SPI, I2C, et UART ainsi que leur protocole de communication.

2. Méthodes de transfert des données : [6]

2.1.Transmission sérielle et parallèle :

Le mode de transmission désigne le nombre d'unités élémentaires d'informations (bits) pouvant être simultanément transmises par le canal de communication. En effet, un processeur (donc l'ordinateur en général) ne traite jamais (dans le cas des processeurs récents).

Un seul bit à la fois, il permet généralement d'en traiter plusieurs (la plupart du temps 8, soit un octet), c'est la raison pour laquelle la liaison de base sur un ordinateur est une liaison parallèle.

2.1.1. liaison parallèle :

On désigne par liaison parallèle la transmission simultanée de N bits. Ces bits sont envoyés simultanément sur N voies différentes (une voie étant par exemple un fil, un câble ou tout autre support physique). La liaison parallèle des ordinateurs de type PC nécessite généralement 10 fils.

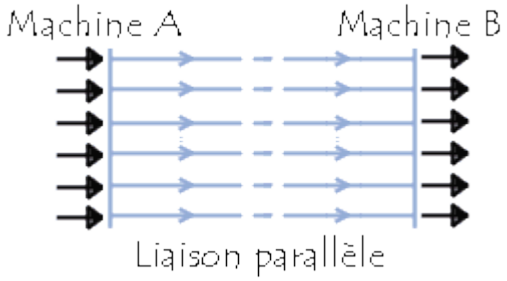


Figure II.1. Liaison parallèle

Ces voies peuvent être :

N lignes physiques : auquel cas chaque bit est envoyé sur une ligne physique (c'est la raison pour laquelle les câbles parallèles sont composés de plusieurs fils en nappe).

Une ligne physique divisée en plusieurs sous-canaux par division de la bande passante. Ainsi chaque bit est transmis sur une fréquence différente...

Etant donné que les fils conducteurs sont proches sur une nappe, il existe des perturbations (notamment à haut débit) dégradant la qualité du signal...

- ✓ ***Avantage* :**

❖ taux élevéde données (vitesse).

- ✓ ***Inconvénients*** *:*

❖ pas fiablessur des longues distances.

❖ les données peuvent arrivera des moments différents.

2.1.2. ***liaison en série :***

Dans une liaison en série, les données sont envoyées bit par bit sur la voie de transmission. Toutefois, étant donné que la plupart des processeurs traitent les informations de façon parallèle, il s'agit de transformer des données arrivant de façon parallèle en données en série au niveau de l'émetteur, et inversement au niveau du récepteur.

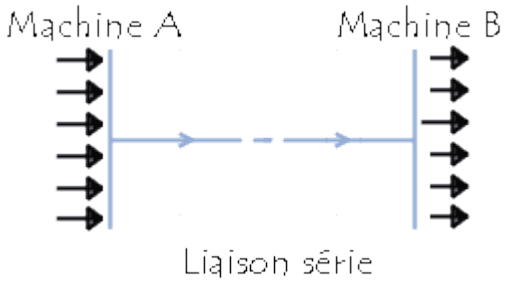


Figure II.2. Liaison série

- ***Avantage* :**
  - ✓ Utilisésur des longues distances.
  - ✓ L'exigenceréduitedecâblage.

- **Inconvénients :**
  - ✓ La transmission des données est plus lente.

Ces opérations sont réalisées grâce à un contrôleur de communication (la plupart du temps une puce UART, UniversalAsynchronousReceiverTransmitter). Le contrôleur de communication fonctionne de la façon suivante :

- *La transformation parallèle-série :*

se fait grâce à un registre Le registre de décalage permet, grâce à une horloge, de décaler le registre (l'ensemble des données présentes en parallèle) d'une position à gauche, puis d'émettre le bit de poids fort (celui le plus à gauche) et ainsi de suite :

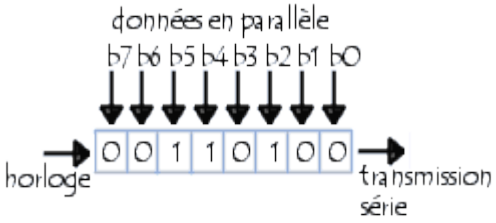
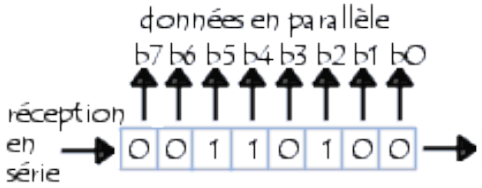


Figure II.3. La transformation parallèle

- *La transformation série-parallèle :*

se fait quasiment de la même façon grâce au registre de décalage. Le registre de décalage permet de décaler le registre d'une position à gauche à chaque réception d'un bit, puis d'émettre la totalité du registre en parallèle lorsque celui-ci est plein et ainsi de suite.



FigureII.4. La transformation série –parallèle

3. Protocole de communication série :

3.1.Définition et principe générale :

La communication série consiste à transmettre des informations après les avoir préalablement découpées en plusieurs morceaux de taille fixe (cette taille est le nombre de lignes de communication disponibles). Elle s'oppose à la communication parallèle où les informations à transmettre n'ont pas à être découpées avant d'être envoyées (car il y a au moins autant de lignes de communication disponibles que de bits pour transmettre l'information).

Les protocoles série sont :

- UART (Universal Asynchronous Receiver Transmitter).
- SPI (serial peripheral interface).
- I2C (Inter-Integrated Circuit) ou TWI (Tow Wire Interface).

3.1.1. UART (Universal Asynchronous Receiver Transmitter):

L'UART (Universal Asynchronous Receiver Transmitter) est un sous-système complet disponible en un seul circuit intégré LSI et conçu pour la transmission et la réception de données asynchrones en mode duplex ou semi-duplex. L'émetteur de l'UART accepte les caractères en parallèle et les convertit en une suite binaire sérielle. Le récepteur convertit les données sérielles en données parallèles. En plus de ses fonctions de conversion, l'UART remplit également des fonctions de contrôle et de commande.

L'UART ajoute aux données transmises le bit de départ (START) et les bits d'arrêt (STOP). En outre, c'est un circuit programmable qui permet de sélectionner :

- la longueur du mot binaire (5, 6, 7 ou 8 bits).
- le type de parité (paire, impaire ou sans parité).
- le nombre de bits d'arrêt (1, 1,5 ou 2).

L'UART permet aussi de détecter les erreurs de parité, de format et de débordement.

Enfin, un circuit UART peut fonctionner avec deux horloges séparées : une pour l'émetteur et une pour le récepteur. Cela permettra de transmettre et de recevoir à deux vitesses différentes, en cas de besoin. **[7]**

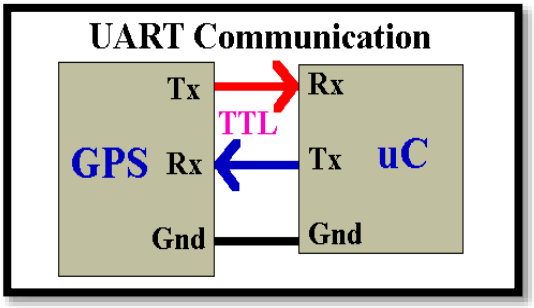


Figure II.5. La communication d'un UART.



Figure II.6. Une trame série d'un UART.

*3.1.2. **Protocol I2C (Inter-Integrated Circuit):***

3.1.2.1.Présentation :

Le bus I2C est caractérisé par une liaison en mode série réalisée à l'aide de 2 fils. C'est la société Philips qui en a créé le concept au début des années 80. Son succès est lié à sa simplicité. Voici l'architecture type d'un bus I2C est donné sur la figure II.7 les données transitent par les lignes :

- SDA : signal de donnée, généré par le Maître ou l'Esclave.
- SCL : signal d'horloge généré par le Maître.

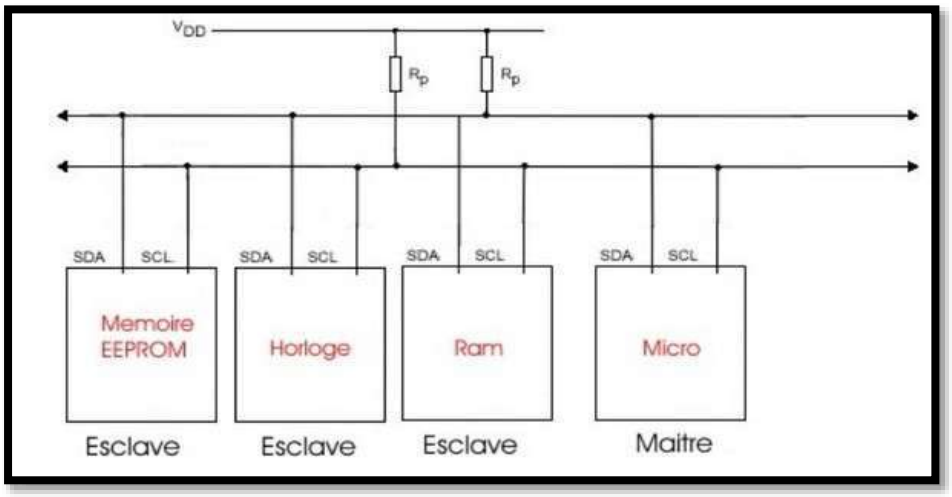


Figure II.7. Architecture d'un bus I2C.

La communication sur le bus est orchestrée de manière suivante :

- Le **Maître** envoie sur le bus l'adresse du composant avec qui il souhaite communiquer, chacun des esclaves ayant une adresse fixe.
- L'**esclave** qui reconnaît son adresse répond à son tour par un signal de confirmation, puis le **Maître** continue la procédure de communication... .(écriture/lecture).
- Dans tous les cas, les transactions seront confirmées par un **ACK**.

3.1.2.2.Prise de contrôle du bus :

- Le bus doit être au repos ; à savoir SDA et SCL à 1.
- Pour transmettre des données il faut surveiller :

- la condition de départ : SDA à 0 et SCL reste à 1.

- la condition d'arrêt : SDA passe à 1 et SCL reste à 1.

- Après avoir vérifié que le bus est libre puis pris le contrôle de celui-ci, le circuit en devient le maître ; c'est lui qui génère le signal d'horloge.

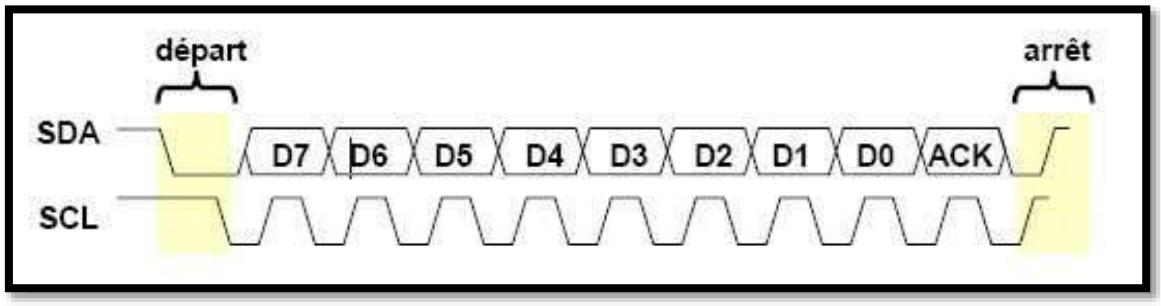


Figure II.8. La transmission des données sur le bus I2C

3.1.3. Interface et le protocole de communication SPI :

3.1.3.1. **La liaison SPI** (Serial Peripheral Interface) :

La liaison SPI (Serial Peripheral Interface) est un bus pour la transmission synchrone de données « série » entre un maître et un ou plusieurs esclaves (en théorie, il n'y a pas de limitations dans le nombre d'esclaves).

La transmission a lieu en Full Duplex, c'est-à-dire que les données circulent simultanément dans les deux sens. Le maître génère l'horloge et initialise une transmission de données en sélectionnant l'esclave avec qui il veut communiquer.

Chaque esclave est adressé par le maître par une ligne individuelle SS (Slave Select) et n'est actif que lorsqu'il est sélectionné. Les bus SPI est composé de deux lignes de signal et deux lignes de données :

- Slave Select (SS).
- Horologe (SCK).
- Master Output, Slave Input (MOSI).
- ·Master Input, Slave Output (MISO).

Toutes ces lignes sont unidirectionnelles.

3.1.3.2.Configuration du bus :

3.1.3.2.1. **SCK (Horloge) :**

L'horloge est générée par le maître et sert à la synchronisation de tous les appareils connectés.

3.1.3.2.2. **SS(Slave Select) :**

Chaque esclave est adressé par le maître par une ligne individuelle SS. Le maître

sélectionne un esclave donné en mettant sa ligne SS au niveau BAS. Le nombre possible d'esclaves au sein d'un tel système de bus n'est limité que par le nombre possible de raccordements SS du maître.

3.1.3.2.3. **MOSI (Master Output - Slave Input):**

Sur la ligne MOSI le maître transmet des données à l'esclave.

3.1.3.2.4. **MISO (Master Input - Slave Output):**

Sur la ligne MISO l'esclave transmet des données au maître.

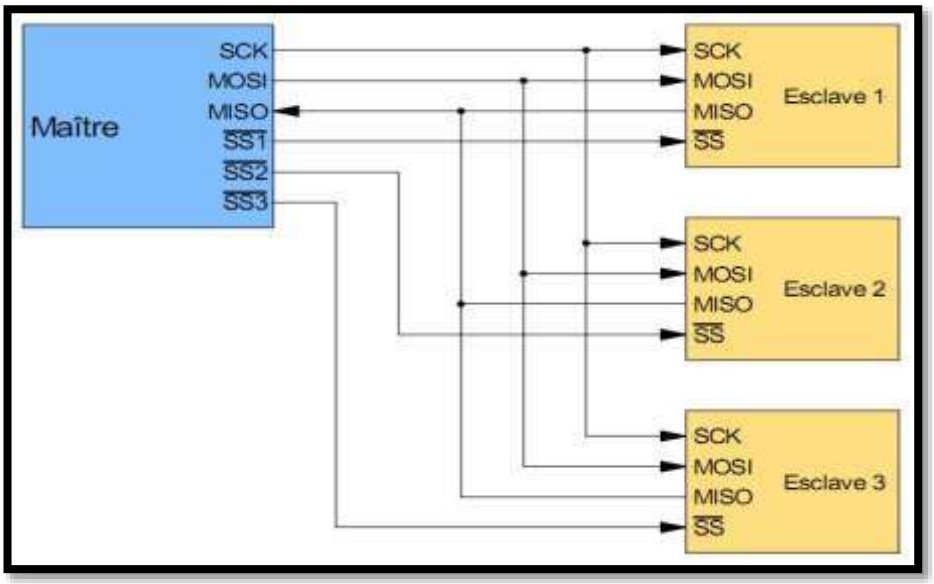


Figure II.9. Exemple de configuration de bus SPI avec un maitre et plusieurs esclaves

3.1.3.3.Protocole du bus SPI :

3.1.3.3.1. Transmission de données :

Pour démarrer une transmission de données, le maître sélectionne un esclave en mettant sa ligne SS respective au niveau BAS. Ensuite, le maître transfert ses données dans le registre de sortie et rapidement après il commence à générer le signal d'horloge.

La transmission de données a lieu en Full Duplex en utilisant des registres à décalage aussi bien par le maître que par l'esclave. Avec chaque coup d'horloge le maître envoie sur la ligne MOSI une donnée à l'esclave pendant que l'esclave envoie simultanément une donnée au maître sur la ligne MISO La transmission de données est terminée dès que la ligne SS est remise au niveau HAUT**. [9][1]**

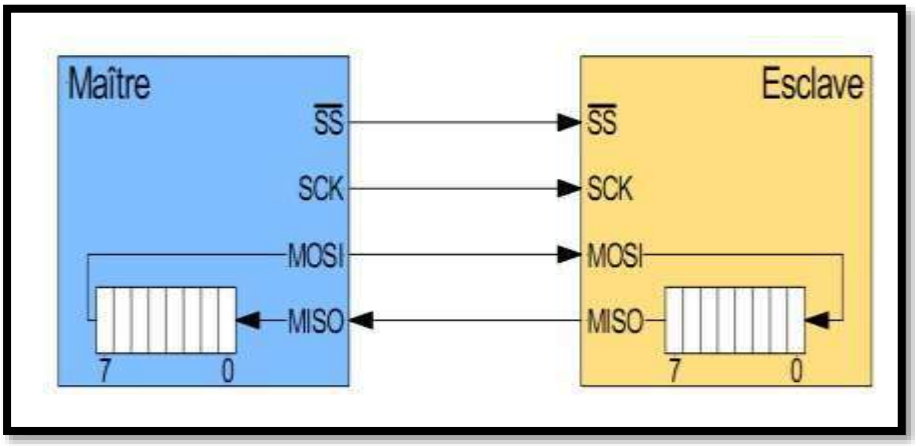


Figure II.10. Transmission de données en maitre et esclave par registre à décalage**.**

3.1.3.3.2. Mode de service :

Le protocole bus SPI détermine le type de signal d'horloge (SCK) grâce à deux paramètres : le bit de poids fort CPOL (ClockPolarity) et le bit de poids faible CPHA (Clock Phase). Ils en résultent quatre différents modes de transmission comme indiqués en TabII .1.

Ces paramètres de réglage déterminent sur quel front d'horloge les données sont transmises, et à quel moment ces données peuvent être modifiées. Pour une transmission de données correcte il est important que ces paramètres soient réglés de manière identique pour tous les appareils reliés au bus.

Mode SPI	**CPOL**	**CPHA**
0	0	0
1	0	1
2	1	0
3	1	1

Tableau II.1. Modes de transmission SPI

- CPOL (ClockPolarity) :

Le CPOL détermine si au repos l'horloge est sur le niveau BAS (CPOL=0) ou HAUT CPOL=1).

- CPHA (Clock Phase) :

Le CPHA détermine à quel front de l'horloge les données sont transmises. CPHA=0 signifie que les données sont valides au premier front d'horloge, CPHA=1 qu'elles sont valides au deuxième front.

3.1.3.3.3. Format du transfert de données :

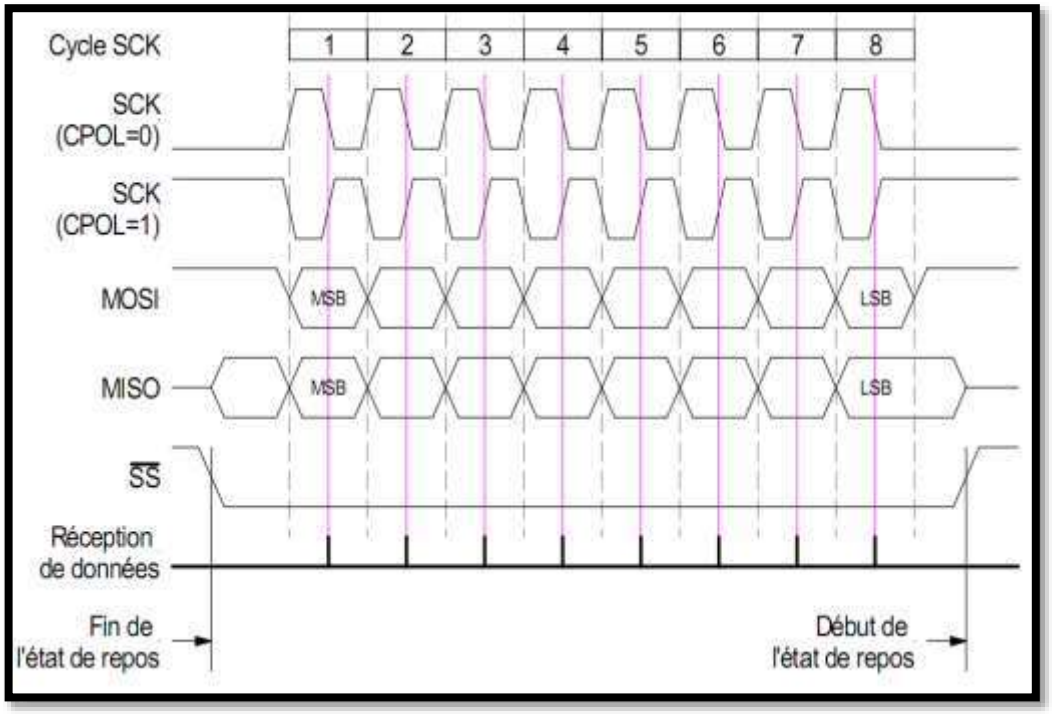
- CPHA = 0 :

Lorsque CPHA est mis à 0, les données sont valides au premier front du signal d'horloge. La polarité CPOL détermine s'il s'agit d'un front montant ou descendant. A CPOL=0, au repos, l'horloge est sur le niveau BAS ; le premier front est donc un front montant. A CPOL=1, au repos, l'horloge est sur le niveau HAUT ; le premier front est donc un front descendant. Cependant, la polarité de l'horloge n'a pas d'influence sur le moment où le premier bit de données est valide et par conséquent n'a pas d'effet sur le format du transfert de données, comme indiqué en figure II.11.

Figure II.11. Exemple d'une transmission de données SPI de 1 octet pour CPHA=0.

- CPHA = 1

Lorsque CPHA est mis à 1, les données sont réceptionnées avec le deuxième front du signal d'horloge. La polarité CPOL détermine s'il s'agit d'un front montant ou descendant. Avec CPOL=0, au repos, l'horloge est sur le niveau BAS et monte au niveau HAUT après le premier front ; le deuxième front est donc un front descendant. Avec CPOL=1, au repos, l'horloge est au niveau HAUT et descend au niveau BAS après le premier front ; le deuxième front est donc un front montant.



Cependant, la polarité de l'horloge n'a pas d'influence sur le moment où le premier bit de données est valide et par conséquent n'a pas d'effet sur le format du transfert de données, comme indiqué en Figure II.12. **[9][1]**

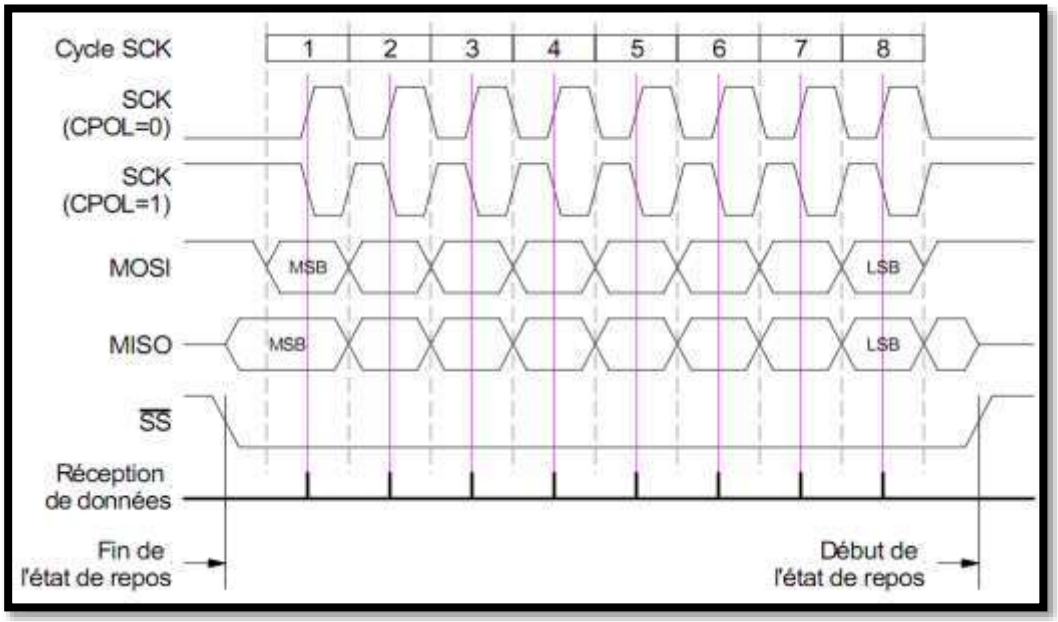


Figure II.12. Exemple d'une transmission de données SPI de 1 octet pour CPHA=1.

4. Conclusion

Les liaisons série peuvent être cadencés considérablement et, plus rapidement que des liaisons parallèles en vue d'atteindre un plus haut débit de données. La diaphonie est quasiment inexistante, car il y a moins de conducteurs à proximité. Dans de nombreux cas, la communication en série est une meilleure option car elle est moins chère à mettre en œuvre. De nombreux circuits intégrés ont des interfaces série, par opposition à ceux parallèles, de sorte qu'ils puissent avoir moins de broches et sont par conséquent moins coûteux

CHAPITRE III : REALISATION ET MISE EN MACHINE

1. Introduction :

L’objet de ce chapitre est le développement et la réalisation d’une interface série de type SPI, implémenté sur FPGA est gérée par un softcore de type Microblaze. Une description matériels est présenter en premier lieu, ensuite on va détailler les différentes étapes de l’implémentation ainsi les différents algorithmes de transmission/réception.

2. Description du matériel utilisé :

2.1.La carte Arduino Uno :

Il existe plusieurs types de cartes Arduino. La carte Arduino Uno est une des versions majeures des cartes Arduino.

Le modèle UNO de la société ARDUINO est une carte électronique dont le coeur est un microcontrôleur ATMEL de référence ATMega328. Le microcontrôleur ATMega328 est un microcontrôleur 8 bits de la famille AVR dont la programmation peut être réalisée en langage C. L'intérêt principal des cartes ARDUINO (d'autres modèles existent) est leur facilité de mise en œuvre. ARDUINO fournit un environnement de développement s'appuyant sur des outils open source. Le chargement du programme dans la mémoire du microcontrôleur se fait de façon très simple par port USB. En outre, des bibliothèques de fonctions "clé en main" sont également fournies pour l'exploitation d'entrées-sorties courantes : gestion des E/S TOR, gestion des convertisseurs ADC, génération de signaux PWM, exploitation de bus TWI/I2C, ...

Il s’agit d’une carte au format « standard » Arduino c'est-à-dire environ 52 mm sur 65 mm. Deux rangées de connecteurs, situées de part et d'autre de la carte, permettent la connexion des composants extérieurs ou des modules shields, au format Arduino, il est possible d'enficher directement ces derniers dans la carte de base, sur plusieurs niveaux si nécessaire.**[9]**

2.1.1. Différents éléments de la carte Arduino Uno :

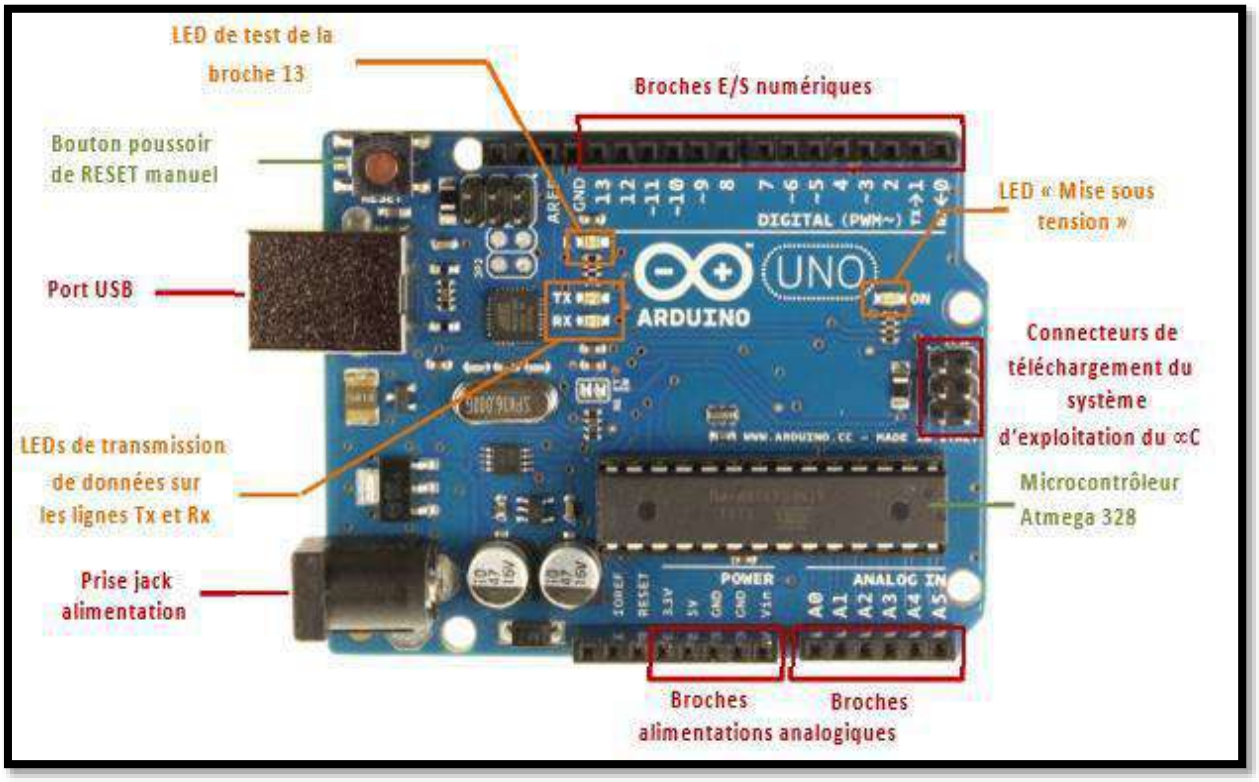


Figure III.1. *Différents éléments de la carte Arduino Uno.*

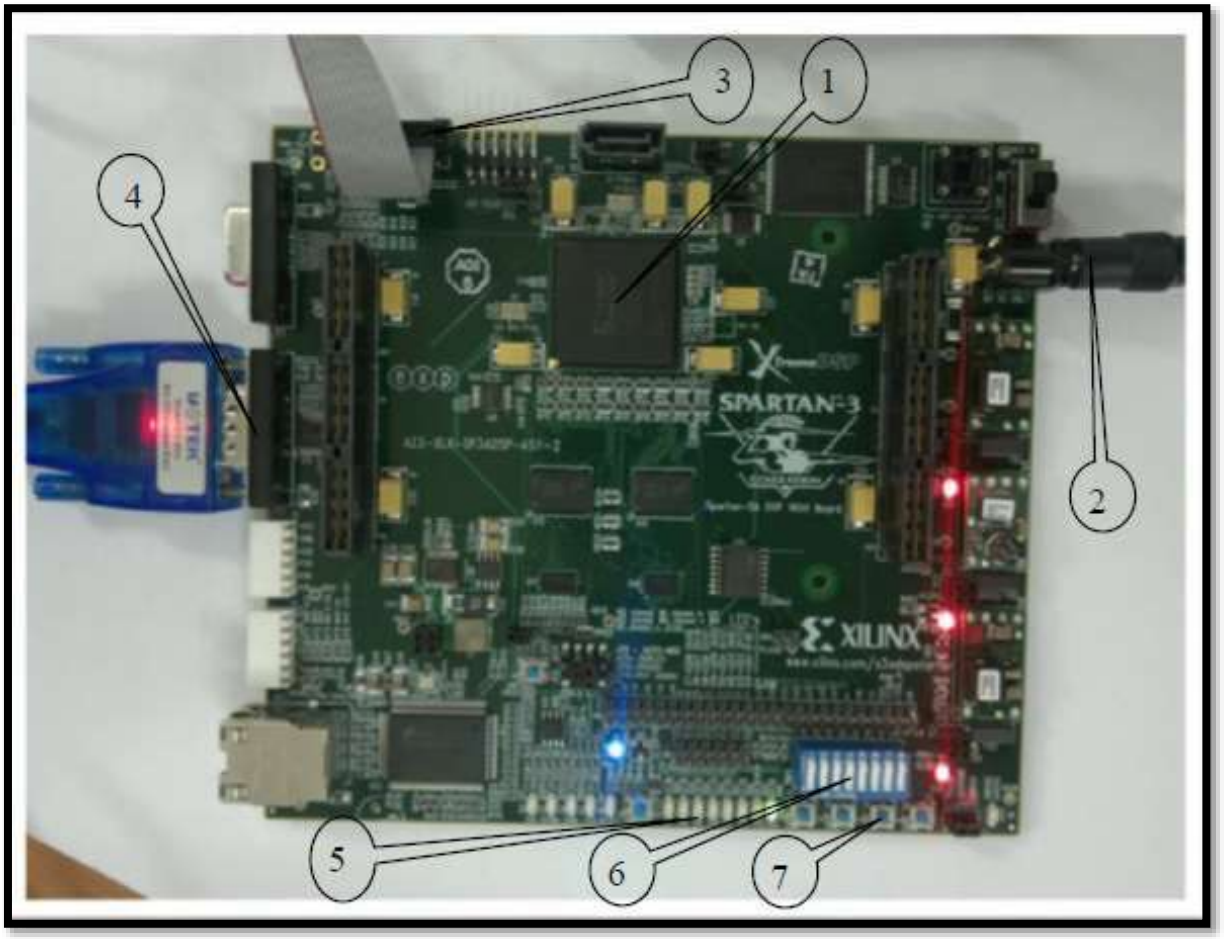
2.1.2. Caractéristiques de la carte Arduino Uno :

- Microcontrôleur : ATmega328
- Tension d'alimentation interne : 5V.
- Tension d'alimentation (recommandée) : 7 à 12 V.
- Entrées/sorties numériques : 14 dont 6 sorties PWM
- Entrées analogiques : 6
- Courant max par broches E/S : 40 Ma
- Courant max sur sortie 3,3V : 50mA
- Mémoire Flash 32 kb dont 0.5 kb utilisée par le bootloader
- Mémoire SRAM 2 kb
- mémoire EEPROM 1 kb
- Fréquence horloge : 16 MHz
- Dimensions : 68.6mm **[9]**

3. La carte de développement Xilinx Xtreme DSP Starter Kit-Spartan-3A DSP 1800A Edition :

La carte "Xilinx Xtreme DSP Starter Kit-Spartan-3A DSP 1800A Edition" (FigureIII.2), possède les caractéristiques suivantes **:**

- XC3SD1800A-4FGG676C Spartan-3A DSP FPGA
- Horloges 125 MHz LVTTL SMT Oscillateur
  - ❖ LVTTL Plot d'oscillateur
  - ❖ 25.175z LVTTL SMT Oscillateur (Horloge vidéo)
  - ❖ 25 MHz Horloge d'Ethernet (accessible à FPGA)
- Mémoire
  - ❖ 128 MB (32M x 32) DDR2 SDRAM
  - ❖ 16Mx8 Parallèle / BPI Configuration Flash
  - ❖ 64 Mb SPI Configuration / Mémoire Flash (avec 4 SPI supplémentaire choisit)
- Interfaces
  - ❖ 10/100/1000 PHY
  - ❖ JTAG Programmation/ Port de Configuration
  - ❖ Port RS232
  - ❖ Low-cost VGA
  - ❖ 4 SPI sélecteurs des lignes
- Boutons et Commutateurs
  - ❖ 8 LEDs
  - ❖ 8 Commutateurs
  - ❖ 4 Boutons poussoirs
  - ❖ Boutons poussoir de remise à zéro
- E/S
  - ❖ En-tête de 6 broches de Digilent (2)
  - ❖ Connecteur d'expansion EXP (2)
  - ❖ 30 broches GPIO Connecteur peut être utilisé pour System ACE™ Compact Flash carte de fille (non inclus)
- Configuration et Débogage
  - ❖ JTAG
  - ❖ Connecteur de module de l'ACE de système
  - ❖ Eridondebugconnector (SATA)**[2]**



1) Le Circuit FPGA XC3SD1800A-4FGG676C
2) Connecteur d'alimentation de la carte
3) Port de programmation/configuration
4) Port série RS-232
5) 8 LEDS d'utilisateur
6) 8 interrupteurs d'utilisateur
7) 4 boutons poussoir d'utilisateur

Figure III.2 : *Vue externe de la plateforme Xtreme DSP Starter Kit - Spartan-3A DSP 1800A*

3. Les Outils de développement de Xilinx :

Comme mentionnée dans le premier chapitre, Xilinx est une compagnie de composants électroniques programmables. En plus du matériel, Xilinx fourni des outils de conception des systèmes nous permettant de mettre au point des projets sur leurs différentes plateformes reconfigurables. En effet, Xilinx fournit dans son pack d'outils différents soft permettant la création de systèmes embarqués sur puce, parmi ces softs on dénombre ISE (Integrated Software Environnent) et EDK (Embedded Development Kit). L'outil ISE est spécialement pour la réalisation des projets d'IPs matériels à partir d'un code HDL. L'outil EDK, englobant SYSTEM GENERATOR FOR PROCESSOR et Xilinx Platform Studio (XPS), qui va nous permettre d'établir un lien direct entre la partie matérielle et logicielle d'un

système. Tous les deux nous offrent la possibilité d'avoir un bitstream pour la programmation des FPGA suivant l'application ciblée.

3.1. Flots de conception Xilinx :

Le flot de conception standard de Xilinx est composé de deux étapes principales : la conception et la synthèse du design et puis l'implémentation et la vérification de ce design. Une vue générale du flot de conception est présentée par la figureIII.3. La conception d'un système embarqué inclut typiquement quatre phases (la création et la vérification de la plateforme matérielle et la plateforme logicielle).

Dans l'outil EDK, la plateforme matérielle est définie par le fichier MHS (Microprocessor Hardware Specification). La plateforme de vérification permet à l'utilisateur de définir le modèle de simulation pour chaque composant du système (processeur et périphériques). Si l'application logicielle est disponible sous sa forme exécutable, elle peut être utilisée pour initialiser les mémoires. La plateforme logicielle est définie par le fichier MSS (Microprocessor Software Specification). La création et la vérification d'une application logicielle passe par différentes étapes : d'abord l'écriture du code en C, C++ ou assembleur qui va être exécuté sur les plateformes logicielles et matérielles. Ensuite ce code est compilé et linké à l'aide de l'outil GNU (d'autres outils peuvent aussi être utilisés) pour générer le fichier exécutable sous le format ELF (Executable and Link Format). Finalement, XMD et le débogueur de GNU (GDB) sont utilisés pour déboguer l'application.

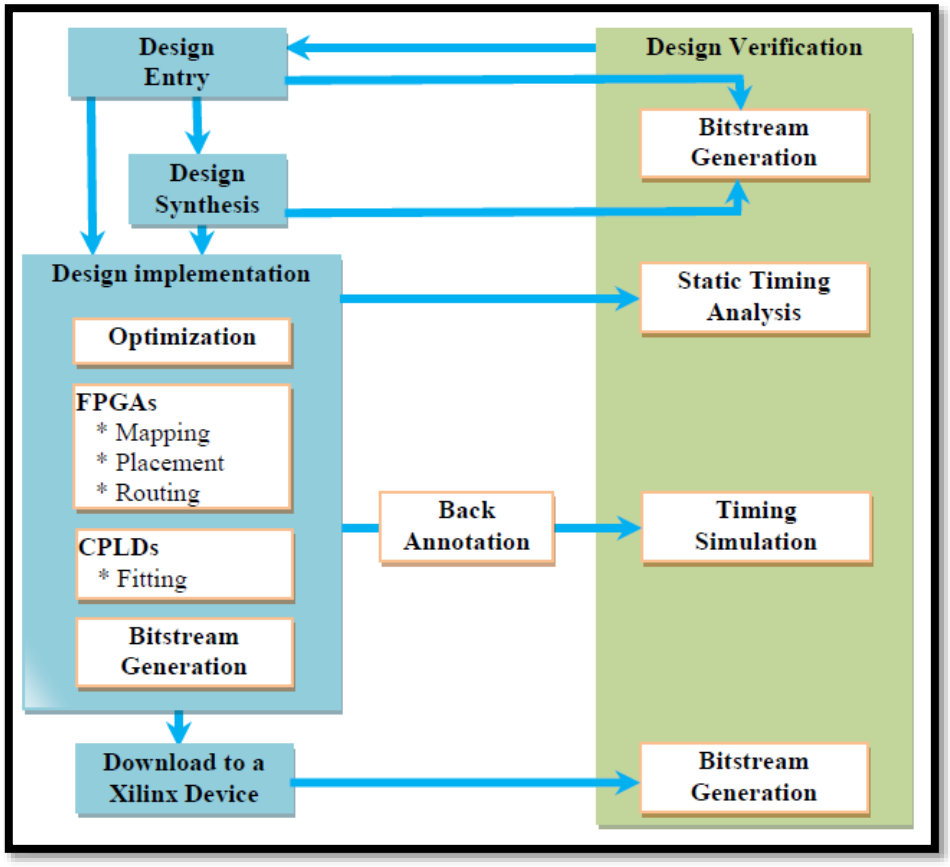


Figure III.3. *Flot de conception Xilinx générique.*

La conception et la simulation sont les deux étapes principales dans le flot de conception Xilinx. Les tâches de conception permettent de passer d'une description à une autre pour arriver au ficher bitstream de configuration. En effet, une synthèse logique permet de passer d'une description RTL de l'architecture à une description au niveau porte logique (Netlist). La description d'éléments logiques est optimisée suivant les contraintes de vitesse, de surface ou de consommations imposées par le concepteur. L'outil de synthèse remplace les éléments logiques génériques par ceux spécifiques au

FPGA ciblé. Le placement et le routage permettent de convertir la description matérielle en un fichier de configuration. L'outil de synthèse génère ce fichier qui sert à la configuration des matrices d'interconnexion du circuit FPGA. A chaque étape de la conception, l'environnement de CAO permet d'effectuer des simulations afin de valider chaque étape de l'implantation :

- Simulation fonctionnelle (niveau RTL = Register Transfer Level).
- Simulation post-synthèse (niveau porte logique).
- Simulation post-layout (niveau physique).

L'environnement ISE, de la société Xilinx, est utilisée à la fois pour la conception et pour l'implantation sur puce. Les simulations comportementales VHDL sont, généralement, réalisées avec l'outil ModelSim.

3.2. Xilinx EDK :

L'environnement EDK permet de développer une application complète à base de processeur embarqué et de l'intégrer à un FPGA. EDK donne accès à tous les réglages nécessaires pour l'application embarquée que l'on souhaite créer. Il permet la programmation d'un ou plusieurs MicroBlazes et de leurs périphériques. Il va s'occuper de placer les programmes des MicroBlazes en mémoire lors de la programmation du FPGA. EDK permet d'une part la création des accélérateurs matériels au niveau RTL à partir d'une description en VHDL (il est possible d'ajouter directement le code synthétisé avec une extension .ngc), et d'autre part de générer l'architecture globale du système en incluant les processeurs logiciels (CPU) ainsi que les contrôleurs d'entrée/sortie nécessaires. L'ensemble de ces composants vont être interconnectés via un bus, qui est lui aussi généré en fonction des besoins architecturaux de l'application. EDK permet aussi :

- Génération du logiciel : permet l'extraction des parties logicielles de l'application et de générer par la suite le code machine correspondant à chaque processeur de la plateforme matérielle.
- Génération des communications : permet de générer le système de communication de l'architecture en fonction du nombre de composants constituants la plateforme matérielle.

EDK est proposé en plusieurs versions : MDK 1.9 (fin 2001), … EDK 3.1 (2003) jusqu'aux versions 8.2 et 9.2 supportant les FPGA Virtex5. Nous utilisons cette dernière version dans nos implémentations malgré l'apparition d'autres versions (EDK10.1, EDK11.1, EDK12.1, EDK12.2, EDK12.4). En effet, selon plusieurs pratiquants de ce domaine, la version 9.2 est la plus stable, elle contient tous les outils nécessaires pour la conception des systèmes embraqués temps réel. D'autres pratiquants préfèrent toujours la plus récente version, à cause de la présence des fonctionnalités bien précises. Exemple des scientifiques travaillant sur la reconfiguration dynamique, préfèrent travailler avec la version 14.1 qui offre une rubrique spécialement pour ce type d'applications. [3]

Xilinx Platform Studio (XPS) est l'environnement de développement utilisé pour la conception de la partie matérielle de système de processeur embarqué. [10]

Étape 01 : création nouveau projet

1. Ouvrir Xilinx Platform Studio
2. Click sure 'Create New Project Using Base System Builder'
3. Sélectionnez le répertoire de conception pour le nouveau projet :
   - ➢ Cliquez sur le bouton Browse et indiquer l'emplacement où vous voulez enregistrer le projet. Dans notre cas c'est : C:\Users\rq\Desktop\spi\system.xmp
   - ➢ Cliquez sur Enregistrer
   - Sélectionnez PLB System
   - Cliquez sur OK



Figure III.4.*création nouveau projet*

4. Dans la fenêtre qui s'ouvre sélectionnez 'I would like to create à new design'
   - Click 'Next'
5. Sélectionnez la carte de développement cible :
   - Boardvendor : 'Xilinx'
   - Board Name : " Spartan-3A DSP 1800A Starter Board "
   - Cliquez sur 'Next'

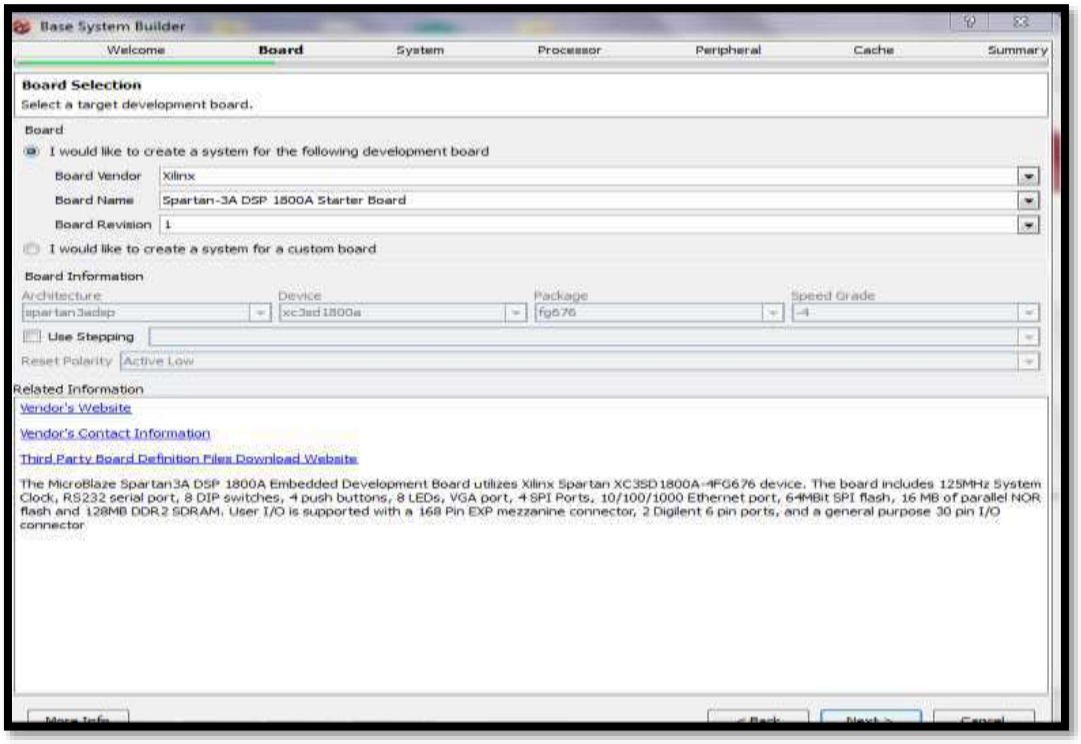


Figure III.5.*Système de baseBuilder*

6. Sélectionnez le processeur ‘Signle- processor System’
    - Cliquez sur ‘Next’
7. Configurer les options de processeur :
    - Processor Type : MicroBlaze
    - System ClockFrequency : 62,50
    - Local Memory : 64 KB
    - Cliquez sur ‘Next’
8. Configurer les options de périphérique :
    - Supprimer tout périphérique sauf ‘dlmb_cntlr’ et ‘ilmb_cntlr’ et ‘RS232_Uart_1’
    - Cliquez sur ‘Next’
9. Configurer les options de (MicroBlaze) cache :
    - Cliquez sur ‘Next’

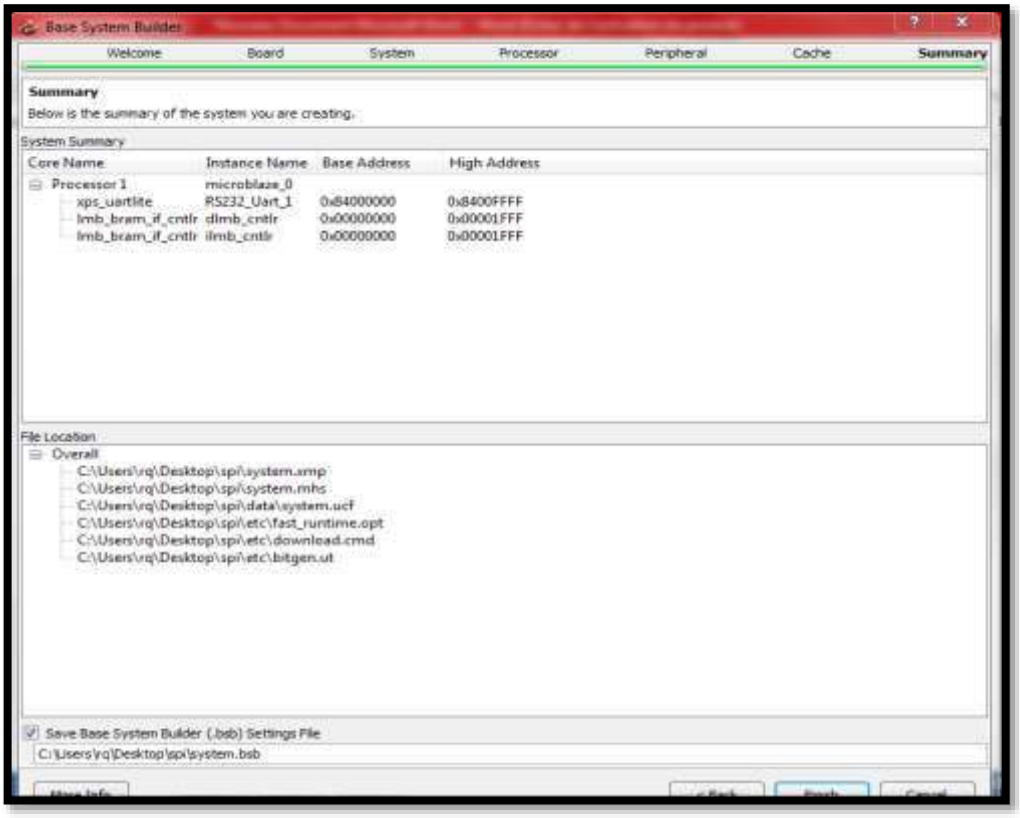


Figure III.6.*Configurer les options de MicroBlaze*

10. Cliquez sur ‘Finish’

La configuration du Microblaze est terminer, on passe à la configuration des différents IP core nécessaire pour notre travail qui sont considérée comme périphériques additionnels au Microblaze.

Étape 02 : Configuration des IP core et la génération du bitsream :

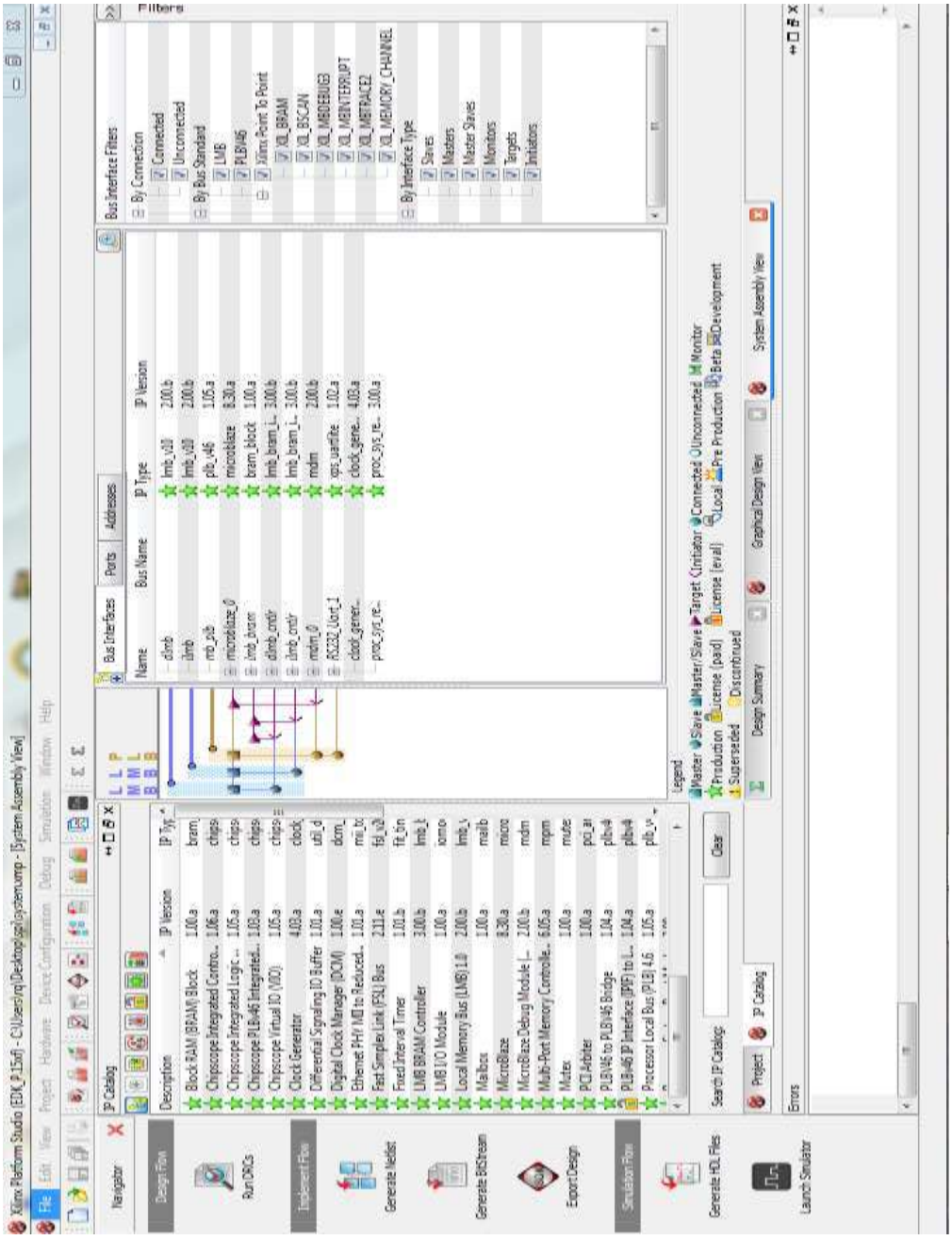


Figure III.7.*Fenêtre Projet XPS*

1. Cliquez sur ‘GenerateNetlist’

NOTE : Cette étape prendra près de 30 minutes dans notre carte et varier selon la carte FPGA ainsi que le type de PC.

2. Ajouté un IP core
   - Sélectionnez l’IP Catalog
   - Double clic sur l’IP core ‘XPS SPI Interface’
   - Cliquez sur ‘OK’

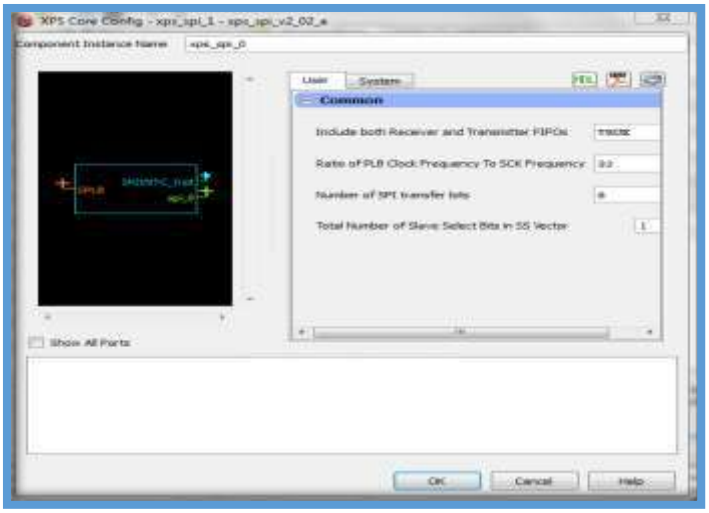


Figure III.8.*Fenêtre d’IP Core*

3. Il apparaît dans ‘Bus interfaces ‘
   - Sélectionnez le type de connexion on choisit ‘mb_plb’

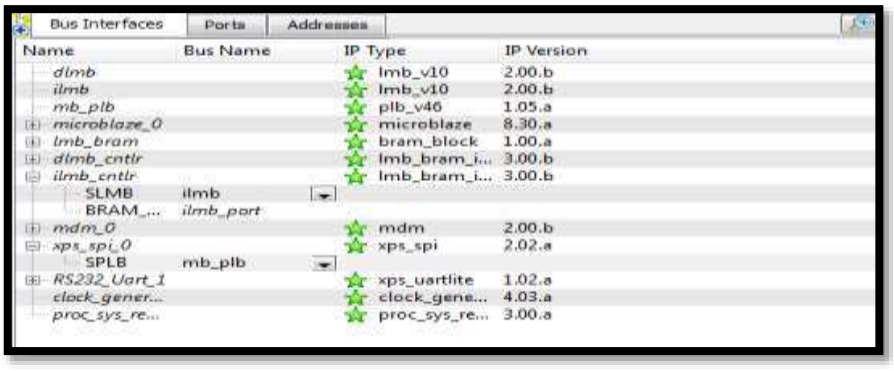


Figure III.9. *Les bus interfaces*

4. Sélectionnez ‘Ports’
    - Mettez tous MISO, MOSI, SS et SCK : ‘External Ports’

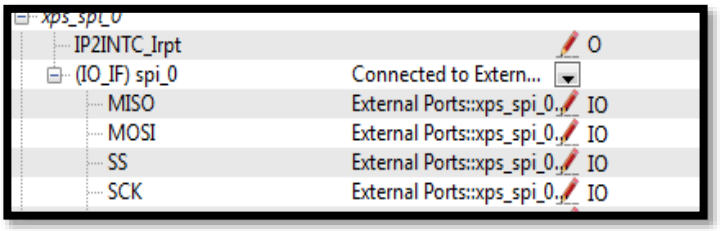
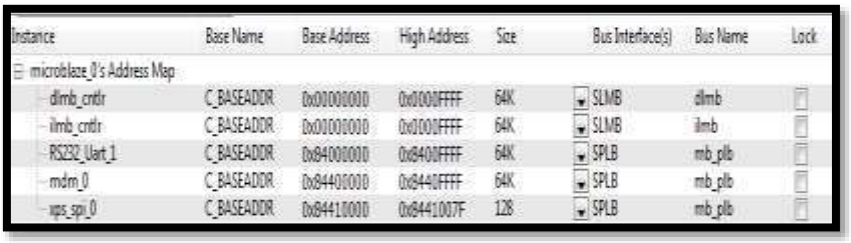


Figure III.10.*Les ports*

5. Sélectionnez ‘Addresse’
    - Donner adresse ‘XPS_SPI_0’



Instance	Base Name	Base Address	High Address	Size	Bus Interface(s)	Bus Name	Lock
microblaze_0's Address Map							
dlmb_cntlr	C_BASEADDR	0x00000000	0x0001FFFF	64K	SLMB	dlmb	
ilmb_cntlr	C_BASEADDR	0x00000000	0x0001FFFF	64K	SLMB	ilmb	
RS232_Uart_1	C_BASEADDR	0x84000000	0x8400FFFF	64K	SPLB	mb_plb	
mdm_0	C_BASEADDR	0x84400000	0x8440FFFF	64K	SPLB	mb_plb	
xps_spi_0	C_BASEADDR	0x84410000	0x8441007F	128	SPLB	mb_plb	

Figure III.11*Les adresses*

6. Cliquez sur projet puis UCF :

   Ajouter le différent pin de l’interface SPI «MISO, MOSI, SS et SCK » est précisé leur pin de sortie dans la carte FPGA

```
#  Spartan-3A DSP 1800A Starter Board
Net fpga_0_RS232_Uart_1_RX_pin LOC = N21  |  IOSTANDARD=LVTTL;
Net fpga_0_RS232_Uart_1_TX_pin LOC = P22  |  IOSTANDARD=LVTTL;
Net fpga_0_clk_1_sys_clk_pin TNM_NET = sys_clk_pin;
TIMESPEC TS_sys_clk_pin = PERIOD sys_clk_pin 125000 kHz;
Net fpga_0_clk_1_sys_clk_pin LOC = F13  |  IOSTANDARD=LVCMOS33;
Net fpga_0_rst_1_sys_rst_pin TIG;
Net fpga_0_rst_1_sys_rst_pin LOC = Y16  |  IOSTANDARD=LVTTL  |  PULLUP;
Net xps_spi_0_MISO_pin LOC = L18 |  IOSTANDARD=LVCMOS33;
Net xps_spi_0_MOSI_pin LOC = L17 |  IOSTANDARD=LVCMOS33;
Net xps_spi_0_SCK_pin LOC = E24 |  IOSTANDARD=LVCMOS33;
Net xps_spi_0_SS_pin LOC = F23 |  IOSTANDARD=LVCMOS33;
Net xps_spi_0_SPISEL_pin LOC = K19 |  IOSTANDARD=LVCMOS33;
```

Figure III.12.*Fenêtre d’UCF.*

7. Cliquez sur ‘GenerateBitStream ‘
8. Cliquez sur ‘Export Design’
    - Export &Lanch SDK.

Étape 3 : Développement de l’application

Dans cette étape on doit basculer de l’XPS vers le SDK.

Le Kit de développement logiciel (SDK) est un environnement de développement intégré, complémentaire à XPS, qui est utilisé le C / C ++ ainsi que le langage assembleur pour le développement logiciel des applications ainsi que le débogage et la vérification. **[10]**

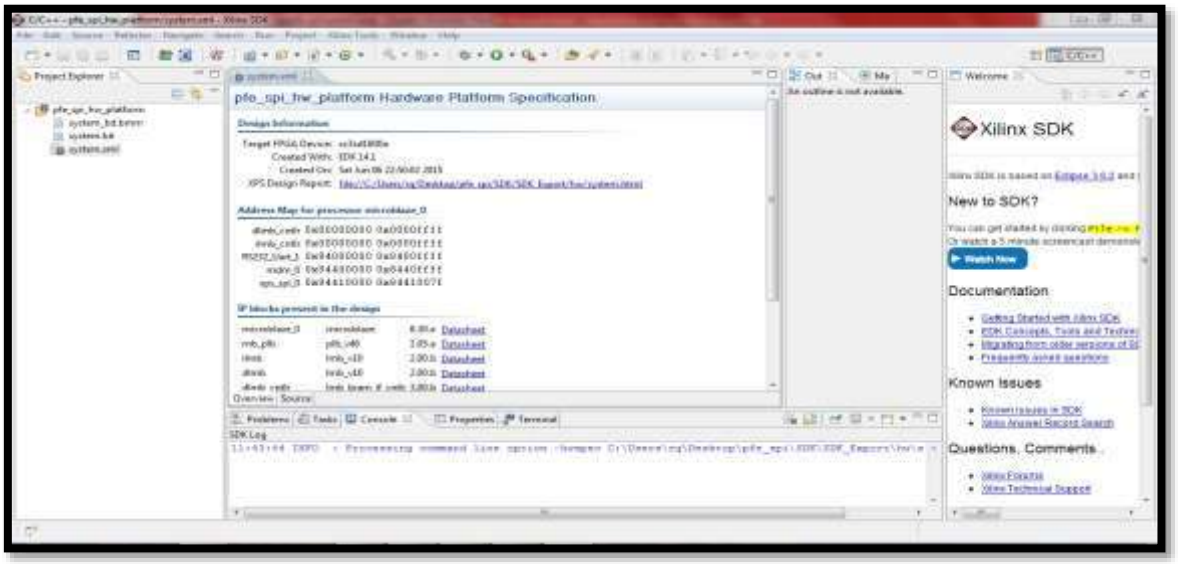


Figure III.13.*Fenêtre Projet SDK*

On commence par la création d’un projet vierge :

1. Sélectionnez ‘ File’ puis ‘new’ puis ‘ Xilinx C Project ‘
2. En choisir hello world puis ‘ next ‘ puis ‘ finich’

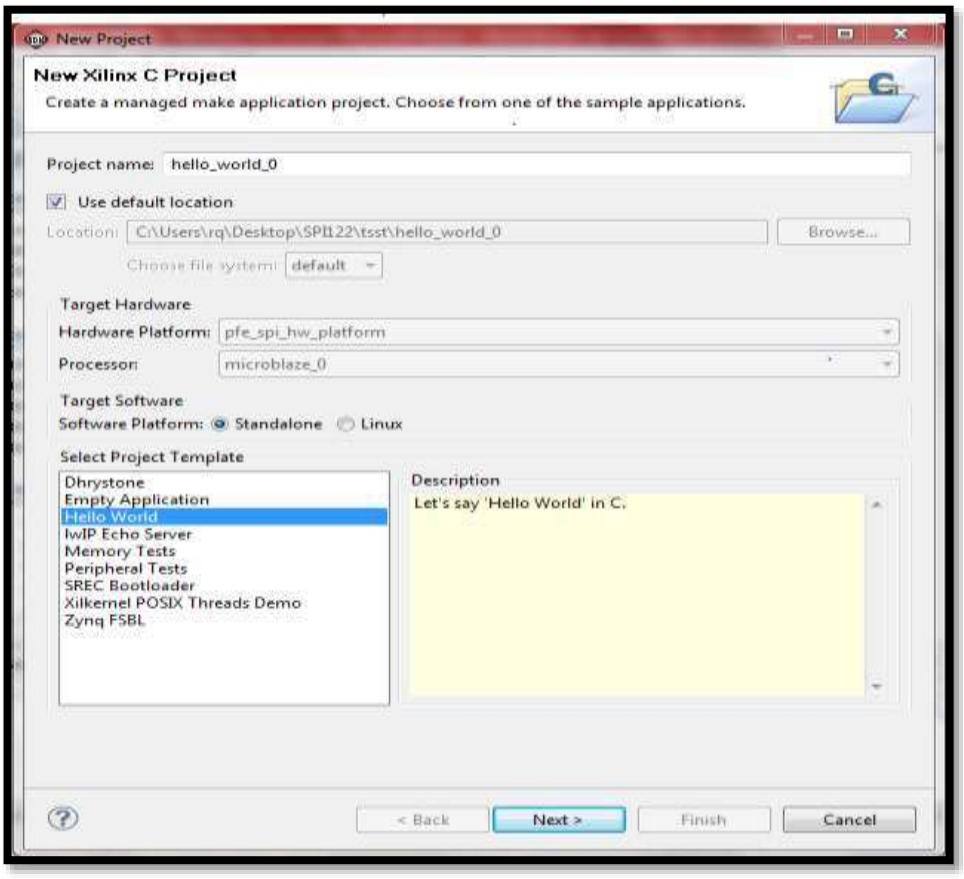


Figure III.14*.création nouveau projet*

3. Clique sur Hello_world_0 puis ‘src’ puis ‘ helloworld.c’

3.2.1. Etude pratique :

3.2.1.1.Partie émission :

✓ *L'organigramme de l'émission*

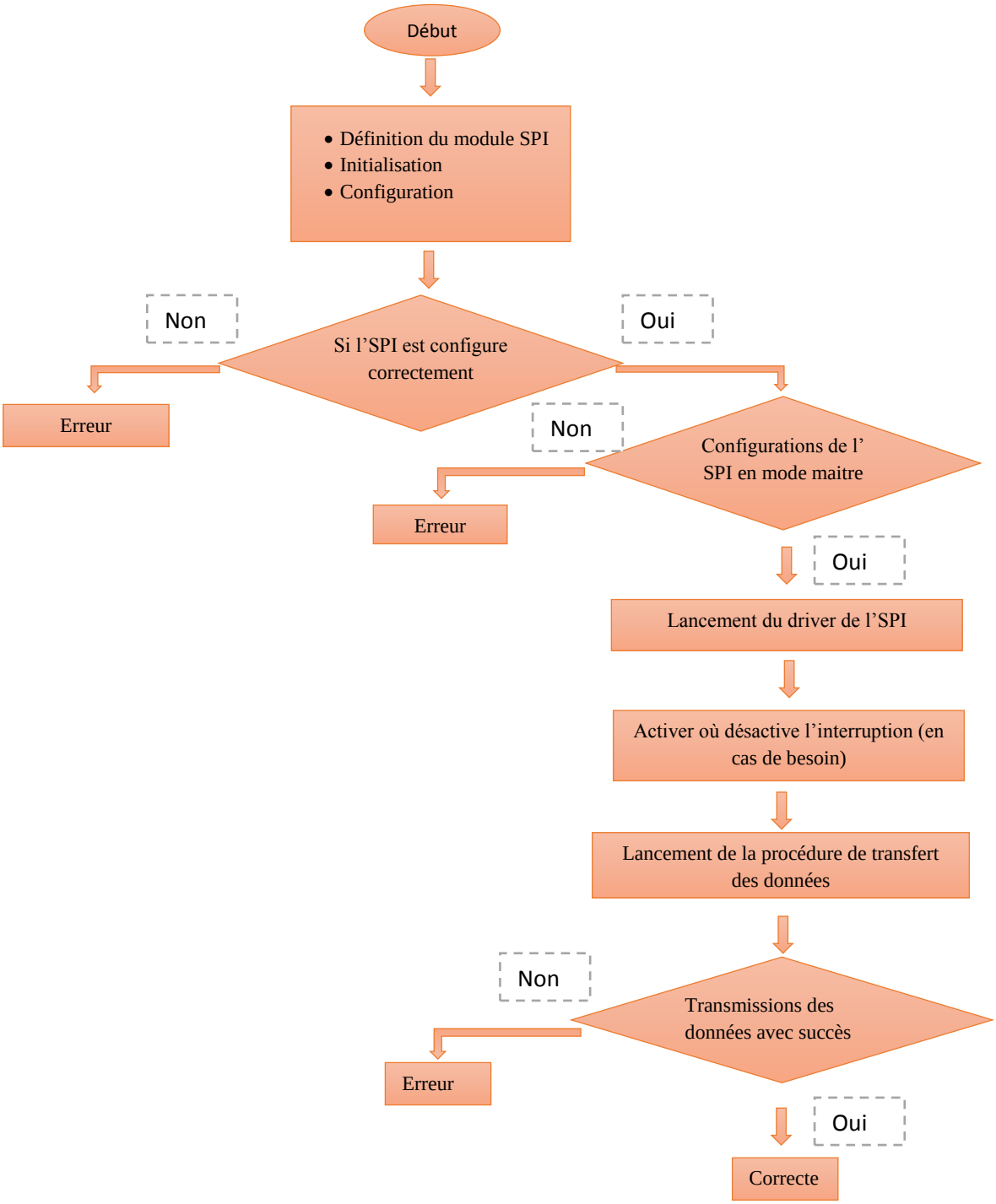


Figure III.15. *Transfert en mode émission*

✓ ***Description de partie émission :***

Comme première étape dans la programmation de l'SPI, on est besoin d'initialiser correctement le module IP de l'ISP avec un pré test du driver de l'IP core pour assurer que ce denier est bien connecté avec le Microblize. Cette étape est nécessaire pour la plupart des IP core. Ensuite on va procéder à la configuration du mode de transfert de l'SPI par la fonction XSpi_SetOptions (SpiInstancePtr, XSP_MASTER_OPTION XSP_LOOPBACK_OPTION), dans un premier cas le mode master est testé. Ont initialise le buffer et Transfer est le la transmission des données est commencé. Pour cette dernière étape en utilise la fonction XSpi_Transfer.

3.2.1.2.Partie réception :

- ✓ *L'organigramme de reception*

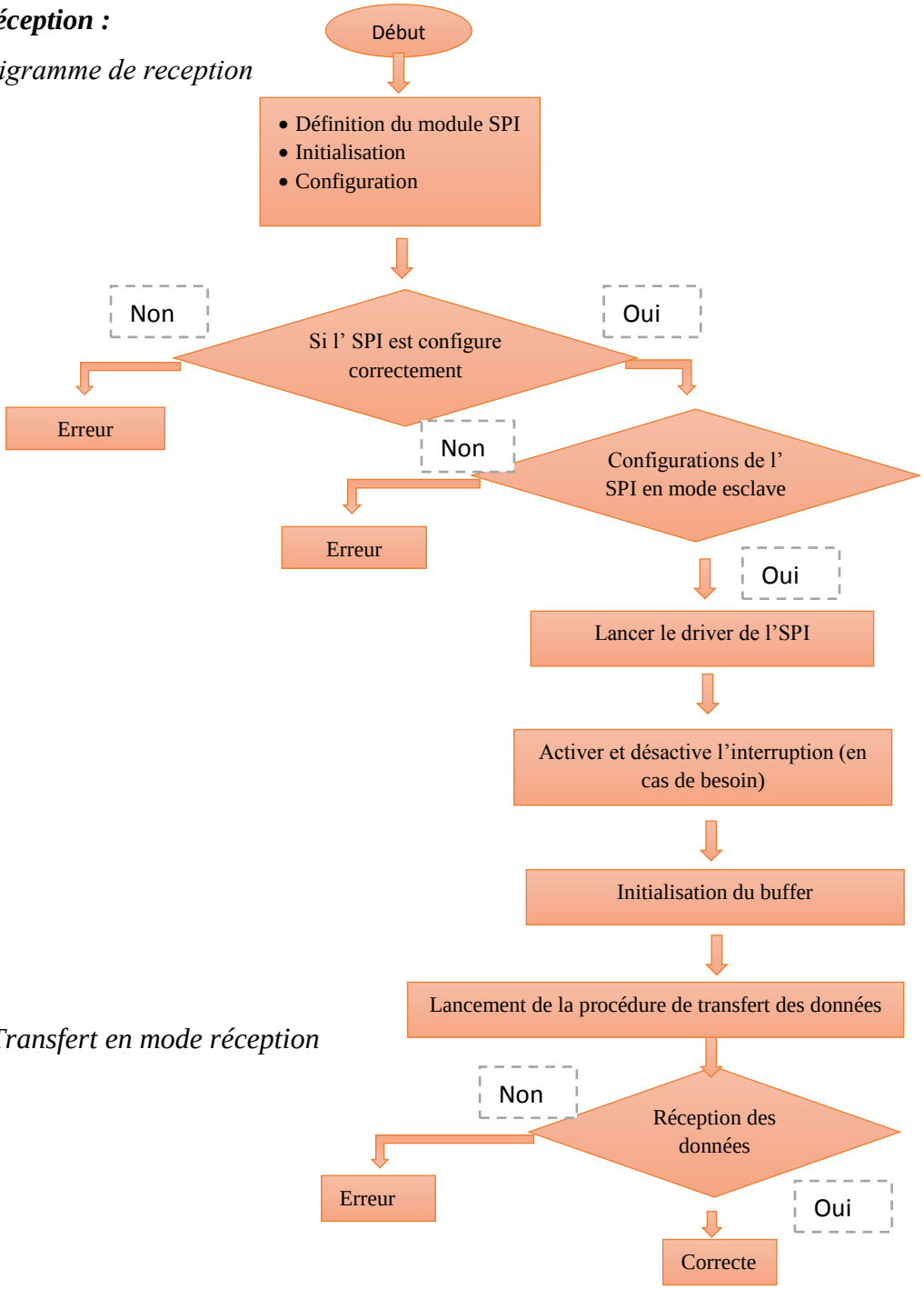


Figure III.16. *Transfert en mode réception*

✓ ***Description de partie reception :***

Mémé cas cité en partie émission .On va procéder à la configuration du mode de réception de l'SPI par la fonction XSpi_SetOptions (SpiInstancePtr, XSP_CLK_PHASE_1_OPTION|XSP_CLK_ACTIVE_LOW_OPTION), dans le cas le mode esclave est testé. En fois le buffer est chargé l'esclave soit du mode réception et initialise le spi en utilise la fonction XSpi_Transfer.

Une fois la programmation des différents modes de l'SPI (Mode master ou esclave) on terminer par configurer l'FPGA afin d'implémenter notre système selon l'application désirer.

4. Cliquer à droite sur Hello_world_0 puis 'Generate Linker Script' puis 'generate' puis 'yes'.

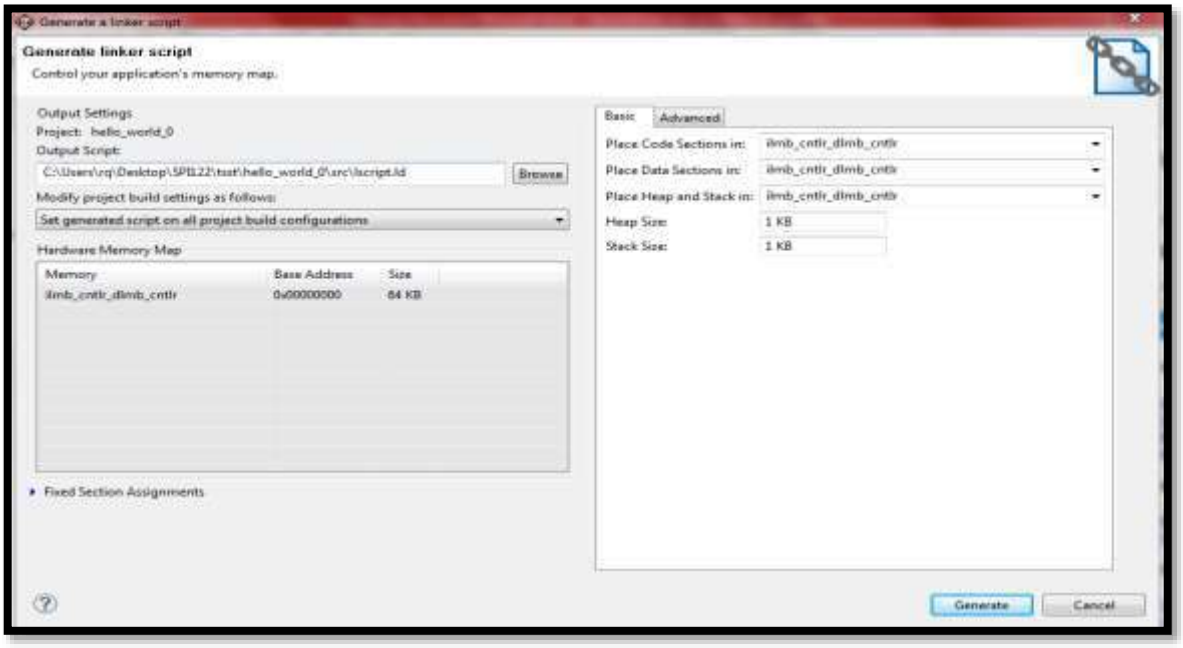


Figure III.17. *Fenêtre de Generate Linker Script*

5. la dernier étape consiste à charger le programme dans la carte cible en cliquant sur l'icône 'Program FPGA' 
6. Sélectionner le fichier de type 'elf', qui correspond a l'application et clic 'Programme'

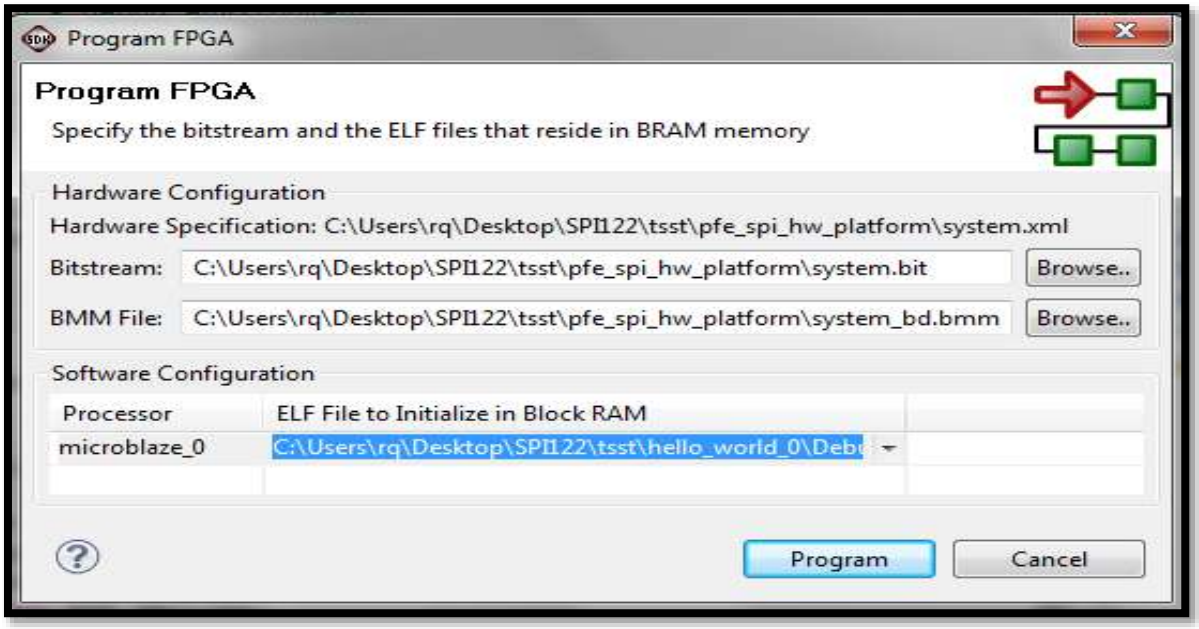


Figure III.18.*Charger le programme dans la carte cible*

Étape 4 : Visualisation des résultats :

La dernière étape consister a testé notre programme pour assurer le bon fonctionnement du système. Pour cela nous avons utilisé une carte ArduinoUno configurer aussi pour une communication série de type SPI, et nous essayons d'établir un système de communication. Deux cas son possible :

- Une FPGA en mode Maitre est l'Arduino en mode esclave.
- Un Arduino Uno en mode Maitre et l'FPGA en mode esclave.

Dans les deux cas les différents résultats sont affichés sur l'hyper Terminale du Maitre.

a. Une FPGA en mode Maitre est l'Arduino en mode esclave.

La figure III.9 représente le résultat obtenu pour ce mode. Dans ce cas l'FPGA envoyai me donnée égale a 43 cette donnée est reçu par l'Arduino et affiché sur l'hyper terminal de ce dernier. Dans le cas d'une male synchronisation un faux lecteur est effectue (donnée 41 enlier de 43).

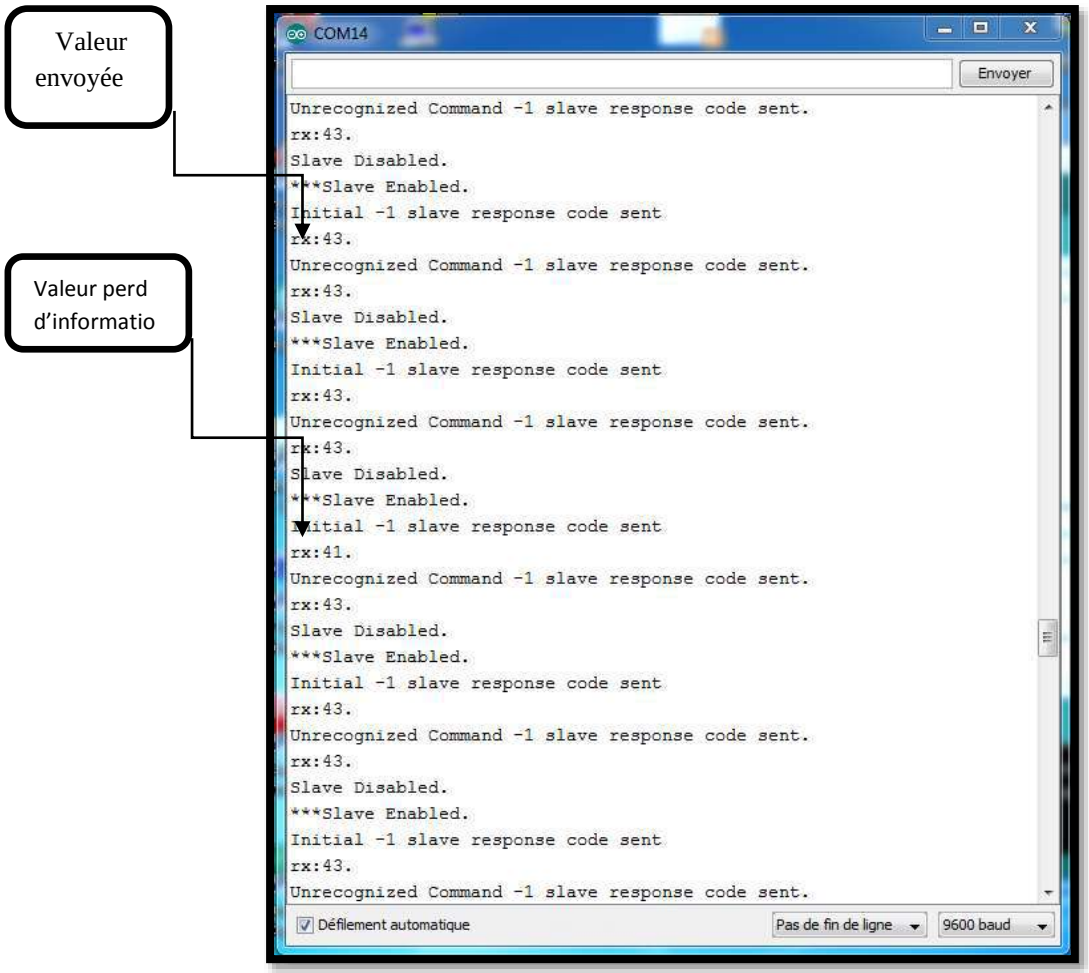


Figure III.19. *L'affichage de mode maitre / esclave*

b. Un Arduino Uno en mode Maitre et l'FPGA en mode esclave

La donnée reçue par l'FPGA est sauvegarde dans un buffer. Dans ce cas un buffer de des cases mémoire est choisir en fois le buffer est chargée l'esclave soit de mode réception et initialise le SPI de nouveau.

L'information envoie

10 fois

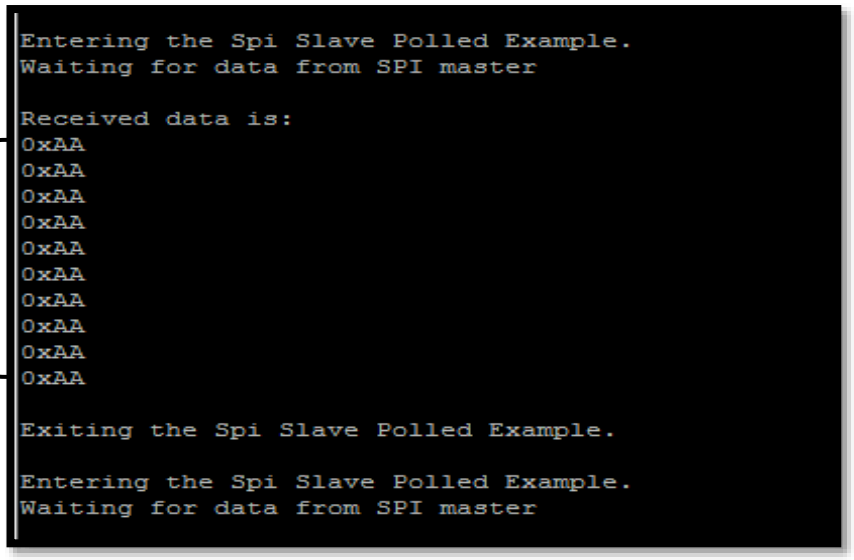


Figure III.20. *L'affichage de mode maitre / esclave*

3. Conclusion :

Dans ce travail nous avons développé et réaliser un système de communication série à base d'un circuit FPGA qui peut communiquer avec réception via une plateforme de communication Arduino uno, les deux cartes réaliser permet l'acquisition d'un ensemble d'entrées analogique et assure la numérisation et l'envoie des données a travers une liaison fil, a la réception ,la visualisation de l' affichage et assuré par hyper terminal .

A travers cette étude nous pouvons déduire que l'existence des IP core gratuite sur l'outil de développement SDK nous a facilités considérablement la tache de réalisation.

Conclusion générale :

Dans ce projet nous avons développé un système de communication série à base d'un circuit FPGA, un test est effectué avec une plateforme Arduino afin de vérifier le bon fonctionnement de notre système. L'existence des IP core gratuite sur l'outil de développement SDK nous a facilités considérablement la tache de réalisation.

Lors des séances de l'étude et réalisation, et en menant notre projet, nous avons pu mettre en œuvre des informations et des connaissances reçues lors des cours d'électroniques et d'informatique ainsi on a pu traduire des études théoriques en études pratiques, Nous avons rencontré un certain nombre de difficultés parmi eux : le manque de connaissance de l'utilisation de plateforme de développement EDK, et les ressources limité de FPGA surtout la taille de la mémoire. On a choisi ce sujet avec des composants étudiés auparavant. Cela nous a permis de familiariser rapidement avec ce type de composants. Ce fût une expérience enrichissante, car nous avons pu réaliser un projet personnel de manière autonome.

Bibliographies

[1] MESSAOUDI K ‘IMPLANTATION DE H.264 SUR MPSOC’ 19 Décembre 2012.

[2] Alibi E,Jawadi S‘Conception et réalisation d’un enregistreur de donnée’ 2010/2011

[3] ‘CHAPITRE1 Introduction à la conception des systèmes embarqués INF3610 Systèmes embarqués’

[4] Rod J Fernando M ‘MicroBlaze Tutorial Creating a Simple Embedded System and Adding Custom PeripheralsUsingXilinx EDK Software Tools’

[6] PILLOUJ’ Transmission de données - Les modes de transmission’

[7] ‘Transmission de données, Interface sérielle et émetteur-récepteur asynchrone universel (UART) ,URL:http://www.technologuepro.com/transmission/chapitre2.htm’

[8] Lycées E. Pérochon et J.Desfontaines, ‘Cours origine Camille Diou’

[6]RAMLI M,AGHAB ‘ETUDE ET REALISATION D’UN NOEUD DE CAPTEUR SANS FIL A BASE D’ARDUINO UNO’2013-2014

[7] MESSAOUDI K ‘IMPLANTATION DE H.264 SUR MPSOC’ 19 Décembre 2012

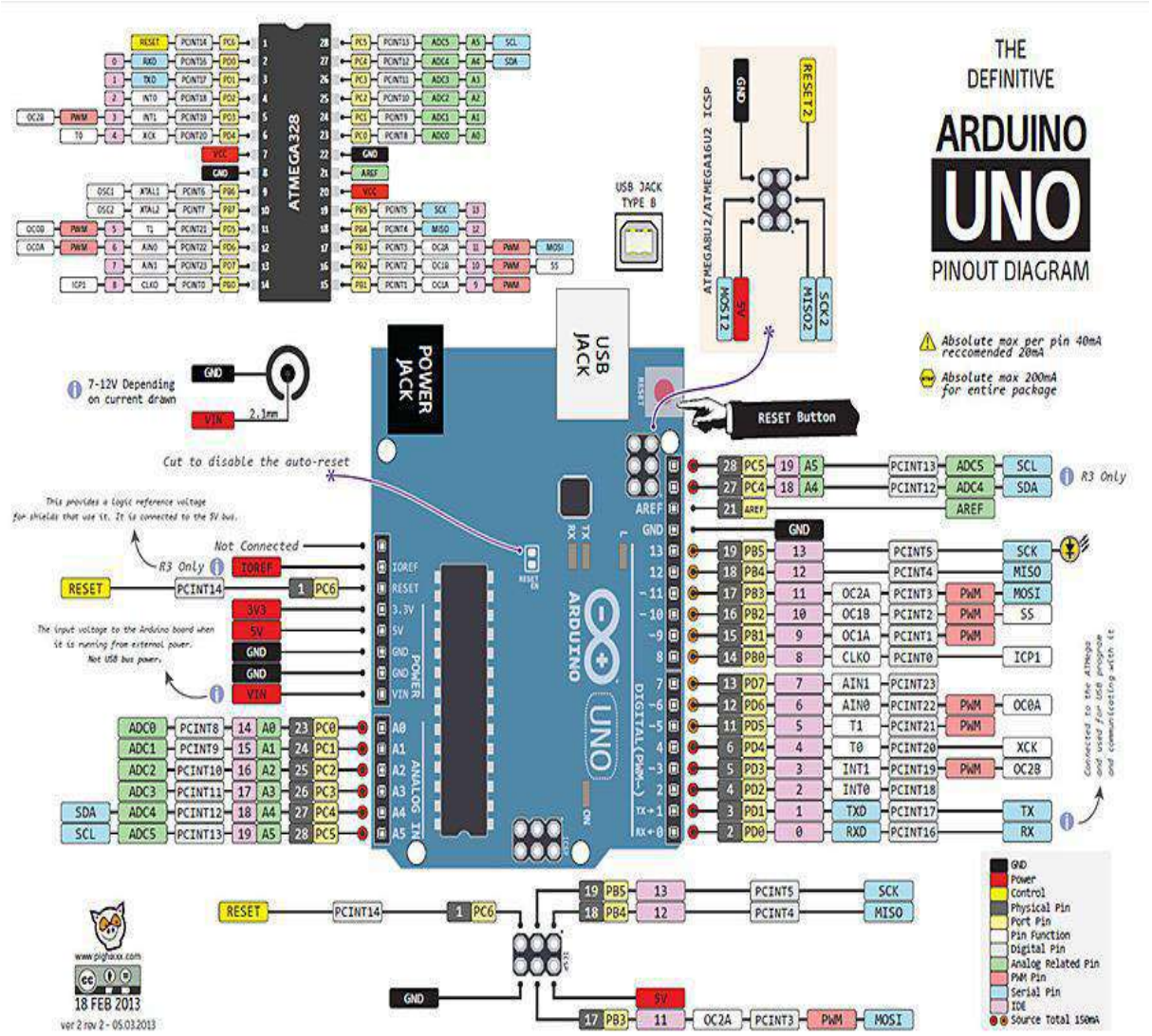
[8] ‘EDK Concepts, Tools, and Techniques UG683 EDK 11.4’

[9] AWAL D ’Implémentation de l’interface SPI sur une carte FPGA’ Juin 2014

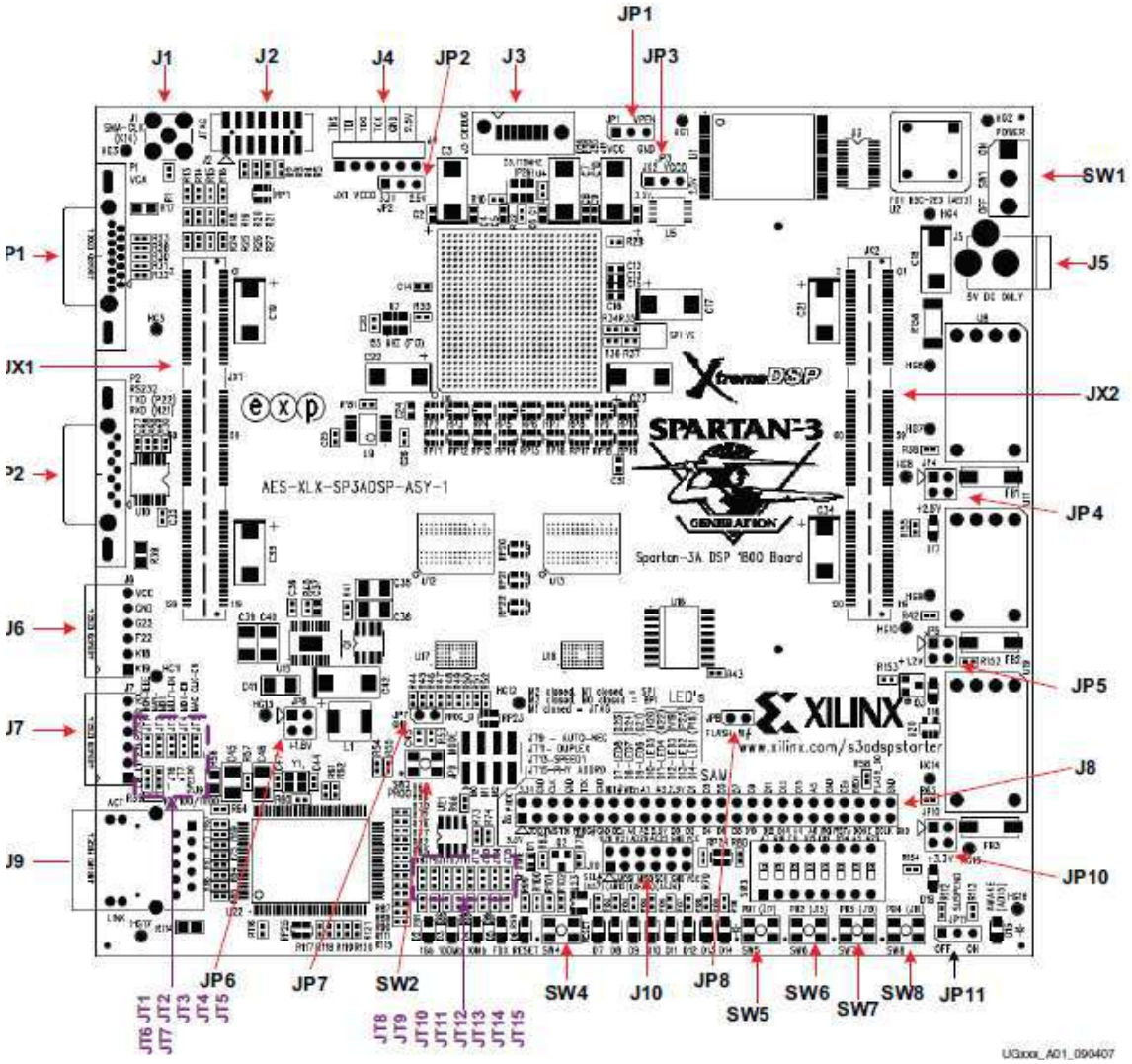
[10]KHIDER A, YAALA M’Implémentation de l’Algorithme décryptage AES sur un circuit FPGA‘ 2012-2013

ANNEXES

ANNEXE A : Le schéma de l'Arduino UNO



ANNEXE B : Le schéma de Spartan-3A DSP



ANNEXE C : Le Brochage de FPGA avec Arduino UNO

FPGA		**Arduino UNO**	
Signal	Pin	**Signal**	Pin
MISO	L18	MISO	12
MOSI	L17	MOSI	11
SCK	E24	SCK	13
SS	F23	SS	10

ANNEXE D : Programme de l'Arduino en mode réception

```
#include <SPI.h>

#include "pins_arduino.h"

void setup (void)

{

Serial.begin(9600);

}

void loop (void)

{

 digitalWrite(SS, HIGH); // ensure SS stays high

 SPI.begin ();

byte cc;

 cc =0xAA;

 SPI.transfer (cc);

}
```

ANNEXE E : Programme de l'Arduino en mode émission

```
#include <SPI.h>

boolean SSlast = LOW

const byte led = 9;

boolean ledState = HIGH;

const byte cmdBtn = 1;

const byte cmdLEDState = 2;

void SlaveInit(void) {

  pinMode(SCK, INPUT);

  pinMode(MOSI, INPUT);

  pinMode(MISO, INPUT);

  pinMode(SS, INPUT);

  SPCR = (1 << SPE);

}

byte SPItransfer(byte value) {

  SPDR = value;

  while(!(SPSR & (1<<SPIF)));

  delay(10);

  return SPDR;

}

void setup() {.

  Serial.begin(9600);

  pinMode(led, OUTPUT);
```

```
  digitalWrite(led, ledState);
  SlaveInit();
  Serial.println("Slave Initialized");
}
void loop() {
  if (!digitalRead(SS)) {
    if (SSlast != LOW) {
      pinMode(MISO, OUTPUT);
      Serial.println("***Slave Enabled.");
      byte rx = SPItransfer(255);
      Serial.println("Initial -1 slave response code sent");
      Serial.println("rx:" + String(rx) + ".");
      if (rx == cmdBtn) {
        byte rx = SPItransfer(cmdBtn);
        Serial.println("cmdBtn Acknowledged.");
        Serial.println("rx:" + String(rx) + ".");
        ledState = !ledState;
        digitalWrite(led, ledState);
      }
      else if (rx == cmdLEDState) {
        byte rx = SPItransfer(cmdLEDState);
        Serial.println("cmdLEDState Acknowledged.");
        Serial.println("rx:" + String(rx) + ".");
```

```
        rx = SPItransfer(ledState);
        Serial.println("ledState:" + String(ledState) + " Sent.");
        Serial.println("rx:" + String(rx) + ".");
      }
      else {
        byte rx = SPItransfer(255);
        Serial.println("Unrecognized Command -1 slave response code sent.");
        Serial.println("rx:" + String(rx) + ".");
      }
      SSlast = LOW;
    }
  }
  else {
    if (SSlast != HIGH) {
      pinMode(MISO, INPUT);
      Serial.println("Slave Disabled.");
      SSlast = HIGH;
    }
  }
}
```