République Algérienne Démocratique et Populaire
Ministère de l'Enseignement Supérieur et de La Recherche Scientifique

Ecole Nationale Polytechnique

Département d'Electronique

Laboratoire des Dispositifs de Communication et Conversion Photovoltaïque

Mémoire de Magister en Electronique

Option : Systèmes de Traitement et d'Analyse du Signal

Présenté par :

MAKHLOUFIA Seddik

Ingénieur d'Etat en Electronique

Thème :

Implémentation d'un Modem 16-QAM sur un DSP TMS320VC5402

Soutenu le 22. 12. 2005 devant le jury d'examen composé de :

Pr M. HADDADI	Professeur à l'ENP	Président
Dr Z. TERRA	Chargé de Cours à l'ENP	Rapporteur
Dr A. BELOUCHRANI	Maître de Conférence à l'ENP	Rapporteur
Dr L. HAMMAMI	Maître de Conférence à l'ENP	Examinatrice
Mr S. AIT-CHEIKH	Chargé de Cours à l'ENP	Examinateur
Mr B. MAGAZ	Chargé de Recherche au CRD-DAT	Examinateur

Ecole Nationale Polytechnique,
10 Avenue Hassen Badi, BP 182, El-Harrach, Alger, Algérie.

Dédicaces

A la mémoire de ma chère mère que dieu le tout puissant

l'accueille en son vaste paradis

A mon cher père

A ma grand-mère

A toute ma famille

A tous mes proches

A mes amis

A tous ceux qui me sont chers

Je leur dédie ce modeste travail.

M.Seddik

Remerciements

Je tiens à exprimer mes sincères remerciements et ma profonde gratitude à mes promoteurs Dr Z.TERRA chargé de cours à l'Ecole Nationale Polytechnique et Dr A.BELOUCHRANI maître de conférence à l'Ecole Nationale Polytechnique pour leurs encouragements et pour les précieux conseils qu'ils n'ont cessé de me prodiguer tout au long de ce projet. Ils trouveront ici ma reconnaissance profonde.

Je remercie très vivement Monsieur M.HADDADI, Professeur à l'Ecole Nationale Polytechnique pour l'honneur qu'il me fait en acceptant de présider le jury de soutenance de ce mémoire.

Mes remerciements vont également au Dr L. HAMMAMI, Maître de conférence à l'Ecole Nationale Polytechnique, Mr S. AIT-CHEIKH, chargé de cours à l'Ecole Nationale Polytechnique et Mr B. MAGAZ Chargé de Recherche au CRD-DAT pour la confiance et l'honneur qu'ils m'accordent en acceptant d'examiner ce travail et faire partie du jury.

Je remercie enfin toute personne ayant contribué de près ou de loin à l'accomplissement de ce travail, en particulier toute ma famille pour son soutien moral durant toute la durée de préparation de ce mémoire.

ملخص :

لوحظ خلال العشرية الأخيرة تزايد مستمر في عدد واستعمال أجهزة الإعلام الآلي. وتماشيا مع هذا التزايد سجل كذلك طلب متزايد على تبادل المعلومات بين مختلف أجهزة الحواسب واللواحق. تعتبر شبكة الهاتف التبادلي العمومي(PSTN) واحدة من أهم الوسائل المناسبة والكثيرة الإستعمال في تبادل المعلومات بين مختلف الحواسب المتباعدة جغرافيا. العنصر الأساسي في هذه الطريقة هو المودام (modem). مؤخرا أصبحت المعالجات الرقمية للإشارة التي تمتلك قدرة فائقة في الحسابات وتنفيد التقنيات الرقمية بواسطة سرعتها الفائقة وتعليماتها المتوافقة مع المعالجة الرقمية للإشارة تستخدم في انجاز المودامات السريعة والمعقدة. الهدف من هذا العمل هو تصميم وإنجاز مودام 16-QAM باستعمال المعالج الرقمي للإشارة DSP TMS320VC5402. سوف نستغل البطاقة التجريبية DSK TMS320VC5402 المتوفرة في المخبر في تجريبه و إنجازه في الوقت الطبيعي.

الكلمات المفتاحية : مودام 16-QAM، ترميز تفاضلي، Quadrature Amplitude Modulation، DSK TMS320VC5402, DSP TMS320VC5402.

Résumé :

Au cours de la dernière décennie, il y a eu une prolifération dans le nombre et l'utilisation des systèmes informatiques. Accompagnant cette croissance, il y a eu une demande croissante de transmissions de données entre les divers systèmes informatiques. Une des méthodes les plus commodes et les plus fréquemment utilisées pour la transmission des données entre les ordinateurs géographiquement distant est via le Réseau Téléphonique Public Commuté (RTPC). L'élément essentiel pour cette méthode de transmission est le modem. Récemment , les processeurs de traitement numérique du signal (DSPs) qui présentent une puissance importante de calcul et d'implémentation des techniques numériques de traitement du signal grâce à ses vitesses et leur jeu d'instructions optimisé pour le traitement numérique du signal sont utilisés pour réaliser des modems rapides et sophistiqués. L'objectif de ce travail est de concevoir et implémenter un modem 16-QAM en utilisant le DSP TMS320VC5402. Nous exploitons le Kit de Démarrage DSK TMS320VC5402 disponible au laboratoire pour le test et l'implémentation du modem en temps réel.

Mots clés : Modem 16-QAM, Codage Différentiel, Modulation d'Amplitude en Quadrature, DSP TMS320VC5402, DSK TMS320VC5402.

Abstract :

Over the past decade there has been a proliferation in the number and the use of computer systems. Accompanying this growth, there has been an increasing demand for data communications between the various computer systems and terminals. One of the most convenient and frequently used methods of data communications between geographically separated computer equipment is via the Public Switched Telephone Network (PSTN). The essential element for this method of data communications is the modem. Recently, digital signal processors (DSPs) which present an important power of calculation and implementation of DSP techniques due to their speeds and their instructions optimized for digital signal processing are used to realize fast and sophisticated modems. The goal of this work is to design and implement a 16-QAM modem using the DSP TMS320VC5402. We exploit the Starter Kit DSK TMS320VC5402 available in the laboratory for test and real time implementation.

Keywords : 16-QAM Modem, Differential Encoding, Quadrature Amplitude Modulation, DSP TMS320VC5402, DSK TMS320VC5402.

Abréviations

ADC	**A**nalog to **D**igital **C**onverter
AIC	**A**nalog **I**nterface **C**ircuit
ALU	**A**rithmetic and **L**ogic **U**nit
ANSI	**A**merican **N**ational **S**tandards **I**nstitute
DARAM	**D**ual **A**ccess **R**andom **A**ccess **M**emory
AT	**AT**tention Code inventé par le constructeur des modems Hayes
BTLZ	**B**ritish **T**elecom **L**empel-**Z**iv
CCITT	**C**omité **C**onsultatif **I**nternational de **T**éléphonie et de **T**élégraphie
CCS IDE	**C**ode **C**omposer **S**tudio **I**ntegrated **D**evelopment **E**nvironnement
ARQ	**A**utomatic **R**epeat **R**equest
Codec	**Co**der/**Dec**oder
CPU	**C**entral **P**rocessing **U**nit
DAA	**D**ata **A**ccess **A**rranged
DAC	**D**igital to **A**nalog **C**onverter
DCE	**D**ata **C**ommunication **E**quipment
DSK	**D**SP **S**tarter **K**it
DSP	**D**igital **S**ignal **P**rocessor
DTE	**D**ata **T**erminal **E**quipment
f_c	**C**arrier **F**requency
$\mathrm{f_s}$	**S**ampling **F**requency
FSK	**F**requency **S**hift **K**eying
I	Composante en phase
I/O	**I**nput/**O**utput
ITU	**I**nternational **T**elecommunication **U**nion
LED	**L**ight **E**mitting **D**iode
MAQ	**M**odulation d'**A**mplitude en **Q**uadrature
McBSP	**M**ulti-**c**hannel **B**uffered **S**erial **P**ort
MNP	**M**icrocom **N**etworking **P**rotocol
MODEM	**MOD**ulateur/**DEM**odulateur
MP/$\overline{\mathrm{MC}}$	**M**icro**P**rocessor/**M**icro**C**omputer mode
PAM	**P**ulse **A**mplitude **M**odulation
PC	**P**ersonal **C**omputer
PSK	**P**hase **S**hift **K**eying
PSTN	**P**ublic **S**witched **T**elephone **N**etwork
Q	Composante En quadrature de phase
QAM	**Q**uadrature **A**mplitude **M**odulation
UART	**U**niversal **A**synchronous **R**eceiver **T**ransmitter
RAM	**R**andom **A**ccess **M**emory
ROM	**R**ead **O**nly **M**emory
TCM	**T**rellis **C**oded **M**odulation
TI	**T**exas **I**nstrument

Liste des figures

Liste des Tableaux

Table des matières

Introduction générale

Avec l'utilisation prédominante des systèmes informatiques et particulièrement les PCs, la transmission et l'échange de données à distance acquéraient une grande importance. La communication entre les divers systèmes informatiques et terminaux est fréquemment accomplie au moyen du réseau téléphonique public commuté (RTPC). L'élément essentiel pour cette méthode de transmission de données est le modem qui joue le rôle d'une interface entre le système informatique ou le terminal et le réseau téléphonique. Il convertit les données digitales qu'il reçoit du système informatique ou terminal en un signal analogique modulé qui sera transmis via le réseau téléphonique. À la destination, le modem de réception démodule le signal reçu et transfère les données digitales au terminal ou système informatique de réception.

Dans le passé, les modems ont été traditionnellement mis en oeuvre dans le domaine analogique en employant des composants discrets. Récemment, les fabricants des modems ont les rendu plus flexibilibles avec une haute performance en utilisant des approches numériques. Les processeurs de traitement numérique du signal (DSPs) présentent une puissance importante de calcul et d'implémentation des techniques numériques de traitement du signal, cette puissance est assurée par ses grandes vitesses d'exécution et leur jeu d'instructions optimisé pour le traitement numérique du signal. Avec la réduction des coûts des processeurs de traitement numérique du signal, la puissance de ces processeurs est exploitée pour la mise en oeuvre des modems rapides et sophistiqués.

Ce mémoire présente la conception et l'implémentation d'un modem 16-QAM basé sur la recommandation V.22bis (standard de base des modems contenant tous les composants de base d'un modem moderne) de CCITT en utilisant le DSP TMS320VC5402 de Texas Instruments (DSP à 16 bit, virgule fixe et 100 MHz de vitesse).

Nous utiliserons l'environnement de développement intégré Code Composer pour tester en simulation les différents algorithmes du modem et le Kit de Développement DSK TMS320VC5402 pour l'implémentation en temps réel. L'implémentation sera faite en langage C compatible ANSI de TI pour les DSPs de la plate forme C5000 (les familles C54x et C55x).

Ce mémoire se compose de quatre chapitres. Le premier chapitre présente des généralités sur les modems, le rôle d'un modem, son fonctionnement et ses différents types ainsi que les standards internationaux de communications des modems. Le deuxième chapitre consiste à la description et la conception du modem 16-QAM avec les détails des différents blocs fonctionnels d'émission et de réception ainsi que leurs représentations mathématiques correspondantes. Le troisième chapitre est consacré à la simulation du modem 16-QAM sur Code Composer et l'implémentation sur la carte DSK TMS320VC5402. Enfin, les différents résultats obtenus par simulation du modem 16-QAM sur Code Composer et implémentation sur la carte DSK TMS320VC5402 avec ses interprétations font l'objet du dernier chapitre.

Chapitre 1

Généralités sur les Modems

1.1 Introduction

Le modem est le périphérique utilisé pour transférer des informations entre plusieurs ordinateurs via un support de transmission (lignes téléphoniques par exemple). Il module les informations numériques en ondes analogiques. En sens inverse, il démodule les données analogiques pour les convertir en numérique. Le mot « modem » est ainsi un acronyme pour « **MO**dulateur/**DEM**odulateur ».

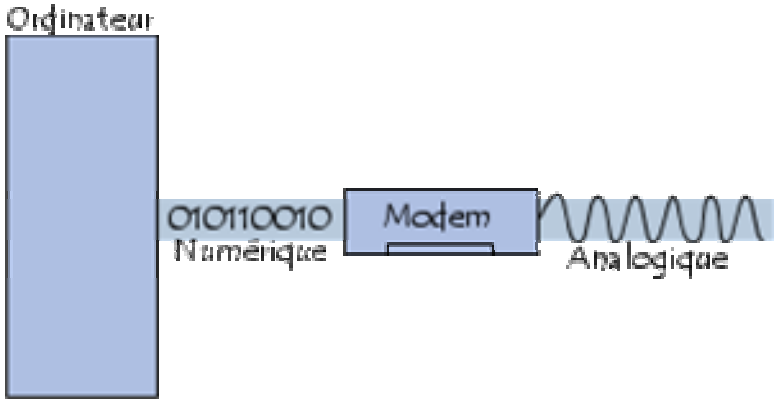


Figure 1.1 : Rôle d'un modem

Pour pouvoir utiliser une ligne téléphonique ordinaire, il faut installer un modem, entre le micro-ordinateur et la prise téléphonique. Il se présente soit sous la forme d'un boîtier externe se branchant sur une prise « série » du micro-ordinateur, soit sous la forme d'une carte d'extension s'insérant dans le micro-ordinateur ; par ailleurs, il existe un modèle sur carte « PCMCIA » destiné aux micro-ordinateurs portables. Son rôle consiste essentiellement à transformer les signaux informatiques en signaux téléphoniques et vice-versa.

1.2 Bref historique sur les Modems

C'était dans les années 1950 que les premiers modems ont été développés. Il y avait un besoin de transmettre des données pour la Défense Aérienne Nord-Américaine, les

efforts ont été faits pour accomplir le but de transfert de données à travers les fils téléphoniques. La Défense Aérienne employait des modems à la fin des années 1950, mais le premier modem commercial n'était pas disponible jusqu'à 1962, c'est le Bell 103 de Bell Labs. Ce premier modem a permis la transmission full-duplex à des débits jusqu'à 300 bps. Après le Bell 103, il apparaît le Bell 212, qui a atteint la vitesse 1200 bps. Il a employé une modulation de phase (PSK).

Pendant les quinze ans suivants, les efforts étaient pour construire des modems qui transmettent des données à des débits plus haut. Pour atteindre ces débits, le système téléphonique a exigé quelques améliorations. A cause des interférences mutuelles, des signaux sont atténués aux divers débits par le système de transmission. Pour compenser cette atténuation, on a besoin d'appliquer des égaliseurs aux lignes téléphoniques. L'égaliseur adaptatif a été inventé en 1965 aux Laboratoires Bell par Robert Lucky. Tandis que les égaliseurs existants exigeaient l'intervention humaine pour les ajuster convenablement. Avec l'apparition de l'égaliseur adaptatif, les données pourraient être transmises aux hauts débits. En 1980, il existait des modems pourrant transmettre jusqu'à 14.4 kilobits par seconde sur des lignes à quatre fils.

En 1984, les modems étaient au point de transmettre 9.6 kilobits par seconde sur un simple-paire circuit sur le système téléphonique. Pour atteindre cet objectif, des avances ont été faites dans l'annulation d'écho (empêchant le modem de capter son signal transmis sur son propre récepteur). Cependant, une nouvelle modulation codée avec correction d'erreur des codes a été développée. Cette correction d'erreur a rendu le signal moins susceptible au bruit.

Les vitesses du modem ont été augmentées à 14.4 kilobits par seconde en 1991. Alors, en 1994, elles ont été doublées à 28.8 kilobits par seconde, bientôt après, elles sont atteints 33.6 Kbps qui a été considérée comme étant une limite supérieure pour la transmission via la ligne téléphonique. Quelques années après, des modems à 56Kbps sont apparus [19].

1.3 Types des modems

Selon leur emplacement et ses caractéristiques de fonctionnement, il existe 5 types des modems :

1.3.1 Les modems internes

Les modems internes (cartes d'extension s'insérant dans le micro-ordinateur) sont en général les moins onéreux. Ils sont au format PCI en deux catégories. Il existe la catégorie des WinModem qui utilisent le processeur central pour décoder les signaux téléphoniques. L'autre catégorie n'utilise plus le processeur central, la navigation sur Internet est plus fluide et moins dépendante du processeur. Les modems internes peuvent aussi servir de fax.

1.3.2 Les modems externes

Les modems externes (boîtiers externes se branchant sur une prise « série » du micro-ordinateur) proposent plus de fonctionnalités que les modems internes. Ils peuvent fonctionner de manière autonome (quand l'ordinateur est éteint, le modem reçoit des fax, ou des messages sur son répondeur virtuel). Ils offrent par contre la même vitesse de transmission et d'émission que les modems internes. Leur interface est pour la plupart du temps en USB, mais un port série est présent pour assurer le maximum de compatibilité.

1.3.3 Les modems ADSL

Les modems ADSL sont encore peu nombreux à être commercialisés. C'est en général le fournisseur d'accès qui livre ses propres modems agréés. Les vitesses de download supportées sont largement au-dessus de la vitesse du réseau actuel, ce qui laisse présager d'une évolution possible.

1.3.4 Les modems pour portables

Comme pour les cartes réseau, un modem pour ordinateur portable est, soit directement intégré, soit au format PCMCIA. Il existe même des cartes PCMCIA qui font modem et en même temps réseau. La vitesse de transmission théorique de ces cartes est identique à celle des cartes modem traditionnelles. Cependant, ces cartes sont moins performantes que les modems intégrés au portable, car elles sont plus dépendantes du processeur central. C'est le seul moyen d'ajouter un modem à un portable si celui-ci n'en dispose pas (ancien modèle).

1.3.5 Les routeurs

Les routeurs sont des hubs qui permettent de partager une connexion ADSL. On n'a plus besoin d'un PC qui doit rester allumer pour partager la connexion internet. Le routeur se

configure à distance et attribue lui-même les adresses Internet locales. C'est la solution idéale pour monter un petit réseau local si on dispose d'une ligne ADSL.

1.4 Fonctionnement d'un modem

La figure suivante présente un simple schéma d'une connexion entre un PC (ou terminal) et un ordinateur hôte en utilisant le réseau téléphonique public.

Figure 1.2 : Connexion d'un PC et un ordinateur hôte via le réseau téléphonique public

Le Modem appelé *DCE* (*Data Communication Equipment*) connecté au premier PC appelé DTE (Data *Terminal Equipment*) est configuré en mode appelant et celui raccordé au deuxième PC en mode appelé.

Mode appelant :

- Génération des tons ou des impulsions de composition du numéro de téléphone.
- Adaptation de la vitesse de transmission en fonction des conditions.
- Gestion de la réception de la porteuse *(Carrier Detect)*.

Mode appelé :

- Détection de sonnerie *(Ring indicator)* pour signaler la réception d'un appel.
- Etablissement de la connexion avec l'appelant, échange des modes de fonctionnement.
- En cas de dégradation de la qualité de la ligne : *Fallback* (message indiquant la dégradation de la qualité de transmission).
- Gestion de la réception de la porteuse *(Carrier Detect)*.

Fréquences vocales et Bande passante :

La bande passante garantie par les opérateurs du téléphone se situe entre 800 et 3800 Hz, mais bien souvent, la limite supérieure n'est pas atteinte. C'est pourquoi il a été défini deux canaux (Figure 1.3) séparés par des filtres passe bande.

On pourra ainsi utiliser simultanément les deux canaux pour pouvoir travailler en mode *Full Duplex*.

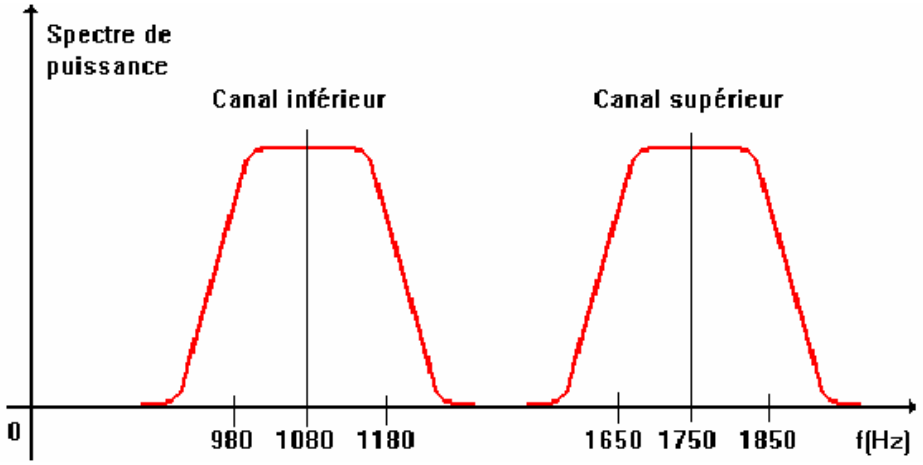


Figure 1.3 : Les deux canaux utilisés en Full Duplex

Modems dits intelligents :

Un modem moderne contient essentiellement les fonctions suivantes :

- Un processeur rapide capable d'effectuer les fonctions de modulation/démodulation.
- De la mémoire vive pour établir des tampons en émission et réception.
- De la mémoire non volatile pour conserver les configurations.
- Un amplificateur et convertisseur digital/analogique pour l'émission.
- Un amplificateur à contrôle automatique de gain et un convertisseur analogique/digital pour la réception.
- Des circuits pour la fonction téléphonique (décrocher, raccrocher la ligne).
- Des circuits pour la gestion des signaux RS232.

Programmation des modems :

Par exemple on prend les séquences de programmation *AT* pour *Attention Code* inventées par le constructeur des modems Hayes. Hayes a défini un certain nombre de commandes qui permettent de dialoguer avec le modem, qui commencent toutes par les deux lettres AT.

Exemple : *AT* DT 7773456 signifie Dial Tone No de téléphone 7773456.

A chaque requête acceptée le modem répond par OK ou ERROR en cas de fonction erronée. Hayes a défini un certain nombre de codes résultants de base aux commandes AT :

(0) OK	Commande acceptée
(1) CONNECT	Connecté
(2) RING	Détection d'appel
(3) NO CARRIER	Pas de porteuse
(4) ERROR	Erreur de commande
(5) CONNECT 1200	Connecté à 1200
(6) NO DIALTONE	Pas de tonalité
(7) BUSY	Signal "Occupé"
(8) NO ANSWER	Pas de réponse
(9) CONNECT 2400	Connecté à 2400

Tableau 1.1 : Codes résultants de base aux commandes AT

Il va sans dire que les codes résultants de base du tableau ci-dessus ont évolué avec les modems. Il faut se référer au manuel du modem pour obtenir les séquences AT et les codes résultants de cet appareil. Les constructeurs proposent généralement une séquence d'initialisation standard appelée *Factory Default* qui est stockée dans une ROM. On peut appeler cette configuration par la commande AT&F.

Fallback : En cas de dégradation de la qualité de la ligne durant la transmission ou à l'initialisation, un des modem partenaire peut demander le *Fallback*, ce qui veut dire que la vitesse de transmission sera réduite pour tenter d'améliorer la qualité des messages.

Mode de transmission :

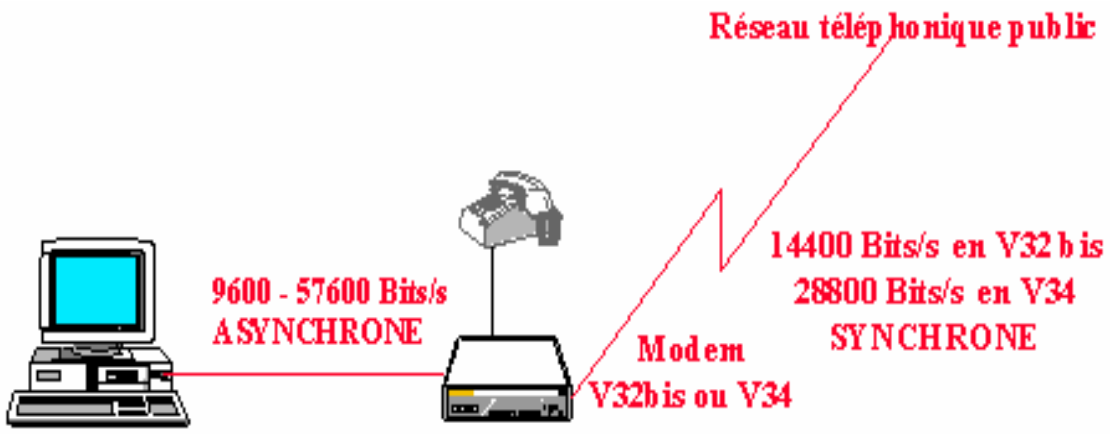
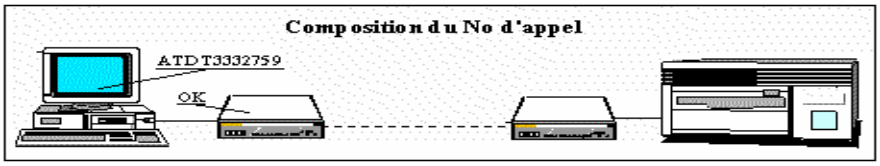


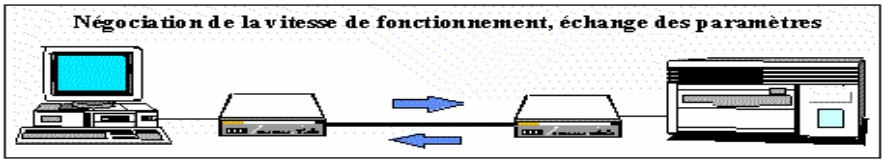
Figure 1.4 : Mode de transmission avec modems

Etape 1

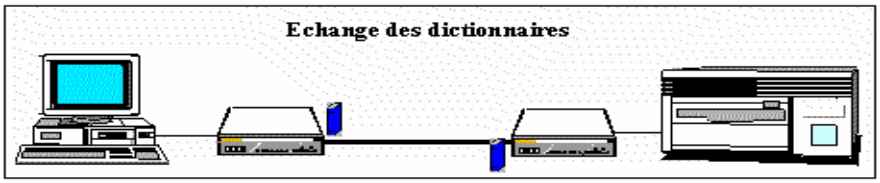


Etape 2

Etape 3



Etape 4

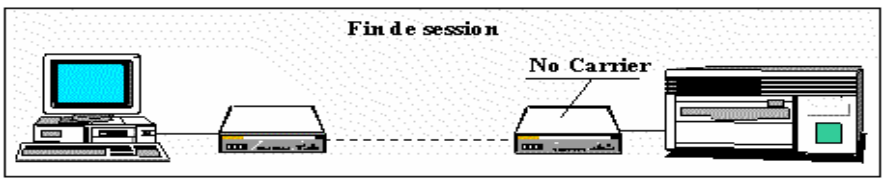


Etape 5

Etape 6

Etape 7

Etape 8



Etape 9

Figure 1.5 : Démonstration d'une session modem

Le mode de transmission est asynchrone (RS232) entre le Modem et le PC (figure 1.4). La vitesse de transmission et le contrôle de flux et ainsi que la taille des caractères et la parité sont définis par la première séquence AT.
Le mode de transmission est synchrone entre les deux Modems, puisque celui qui émet crée une trame issue d'une série de caractères asynchrone auxquels on a enlevé les *start et stop bits.*

Correction d'erreurs :

Les modems sont capables de demander la retransmission du paquet entre-eux en cas de problème, ceci de manière transparente au niveau de l'utilisateur.
Il existe plusieurs standards pour la correction des erreurs :

- **MNP 3&4** de *Microcom Networking Protocol*
- **V42** du CCITT combinaison de MNP3&4 et LAP-M
- **ARQ** *Automatic Repeat Request*

Compression :

On peut optimiser un transfert en faisant appel aux techniques de compression sur la ligne téléphonique :

- **MNP5** *Run-Lenght Encoding* qui est le codage des répétitions (Ex: AAAABBCCC= 4A2B3C) - Compression 2 à 1
- **MNP7** idem - Compression 3 à 1
- **V42bis** du CCITT LAPM *Link Access Procedure* (V42 + BTLZ *British Telecom Lempel-Ziv*) - Compression 4 à 1

De plus, la méthode dite d'échange des dictionnaires, qui a lieu au début de la connexion (phase de négociation), permet d'optimiser la transmission, puisque chaque modem va échanger une table contenant à son début les caractères qu'il a le plus souvent transmis[24].

1.5 Techniques de modulation numérique

La modulation numérique est l'étape qui fait correspondre aux informations numériques, les formes d'onde analogiques qui seront servis à transmettre ces informations physiquement via le canal. En effet, même si l'information est numérique, ce qui est transmis physiquement sur le canal doit être analogique. La correspondance est établie entre des blocs de $k = \log_2 M$ bits (chaque bloc contient k bits) et M formes d'onde notées $s_m(t)$, avec $m \in [1,.....,M]$.

Lorsque la forme d'onde transmise pendant un intervalle de temps dépend d'un ou plusieurs symboles précédents, on dit de la modulation qu'elle a de la mémoire. Inversement, on dit qu'elle est sans mémoire. Le fait d'introduire de la mémoire dans une modulation peut être motivée par la mise en forme de la densité spectrale du signal émis. De plus, cette mémoire introduite dans le signal peut être utilisée au récepteur pour effectuer une estimation plus performante des symboles émis. Les formes d'ondes qui sont mises en correspondance avec les symboles numériques de départ, diffèrent par leur amplitude, leur phase, leur fréquence, ou encore une combinaison de ces paramètres [6].

1.5.1 Modulation PAM

La modulation PAM (Pulse Amplitude Modulation) établit une correspondance entre les symboles A_m et les formes d'ondes données par [6] :

$$\begin{aligned} s_m(t) &= \Re[A_m g(t)\exp(2\pi j f_c t)] \\ &= A_m g(t)\cos(2\pi f_c t) \end{aligned} \tag{1.1}$$

pour $0 \le t \le T$, où f_c est la fréquence de la porteuse, les symboles notés A_m pour $m \in [1,.....,M]$. Les symboles peuvent prendre les valeurs données par :

$$A_m = (2m - 1 - M)d \tag{1.2}$$

où $2d$ est la distance entre symboles. Le signal $g(t)$ est une impulsion (forme d'onde) qui influence le spectre occupé par le signal modulé. Si le débit binaire de départ est R bits/sec, R/k est le débit symbole qui s'exprime en symboles/sec ou bauds. Si la durée des bits d'entrée est notée T_b, la durée des symboles est donnée par $T = kT_b$ et le débit symbole est donc $1/T = 1/kT_b = R/k$. Les énergies ε_m des différentes formes d'onde sont données par :

$$\begin{aligned} \varepsilon_m &= \int_0^T s_m^2(t)dt \\ &= \frac{A_m^2}{2}\int_0^T g^2(t)dt \\ &= \frac{A_m^2}{2}\varepsilon_g \end{aligned} \tag{1.3}$$

où ε_g est l'énergie de la forme d'onde de mise en forme. Cette modulation porte également le nom ASK (Amplitude Shift Keying).

La mise en correspondance des mots de *k* bits avec les symboles A_m peut se faire de différentes manières. Il y'a cependant un arrangement préféré en pratique, appelé encodage de Gray. Il consiste à associer à des symboles A_m voisins, des mots binaires qui ne diffèrent que par un seul bit. Cette façon de faire est justifiée, car en pratique les erreurs les plus probables consistent à confondre des symboles voisins lors de la détection. Si l'encodage de Gray est utilisé, ce type d'erreur se soldera par un seul bit erroné. L'encodage de Gray est donné à la figure 1.6 pour des symboles à 2, 4 et 8 niveaux.

Le signal modulé défini par l'équation 1.1 est à double bande latérale (DSB-double sideband), ce qui n'est pas efficace du point de vue de l'occupation du spectre. On peut rendre ce signal à bande latérale unique en lui ajoutant sa transformée de Hilbert [6] :

$$s_m(t) = \Re\{A_m[g(t) + j\hat{g}(t)]\exp(2\pi j f_c t)\} \tag{1.4}$$

$$s_m(t) = A_m g(t)\cos(2\pi j f_c t) - A_m \hat{g}(t)\sin(2\pi j f_c t) \tag{1.5}$$

pour $0 \le t \le T$, où $\hat{g}(t)$ est la transformée de Hilbert de $g(t)$. Dans ce cas, la bande du signal SSB (single sideband) n'est autre que la moitié de celle occupée par le signal DSB.

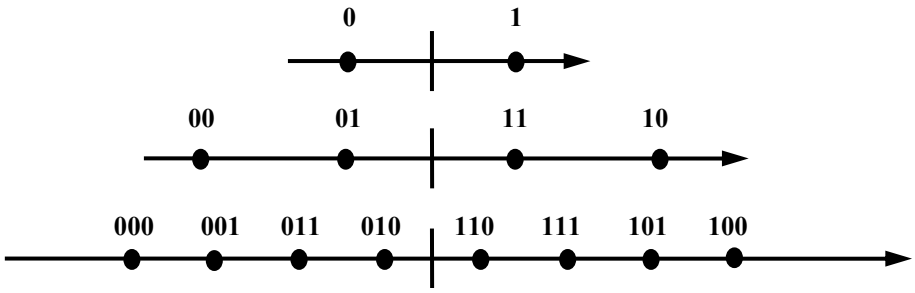


Figure 1.6 : Encodage de Gray pour des symboles à 2, 4 et 8 niveaux

La modulation PAM peut aussi être employée sans porteuse [6]. On parle alors de PAM en bande de base et les formes d'onde sont simplement données par :

$$s_m(t) = A_m g(t) \tag{1.6}$$

D'autre part, dans le cas particulier des symboles $+d$ et $-d$, on a :

$$s_2(t) = -s_1(t) \tag{1.7}$$

Ces signaux sont dits opposés et ont la même énergie ε donnée par :

$$\varepsilon = \int_0^T s_1^2(t)dt = \int_0^T s_2^2(t)dt \tag{1.8}$$

et un coefficient de corrélation ρ donné par :

$$\rho = \frac{1}{\varepsilon}\int_0^T s_1(t)s_2(t)dt = -1 \tag{1.9}$$

La figure 1.7 représente une porteuse sinusoïdale modulée en PAM par un signal binaire, la durée de chaque bit est égale à quatre fois la période de la porteuse.

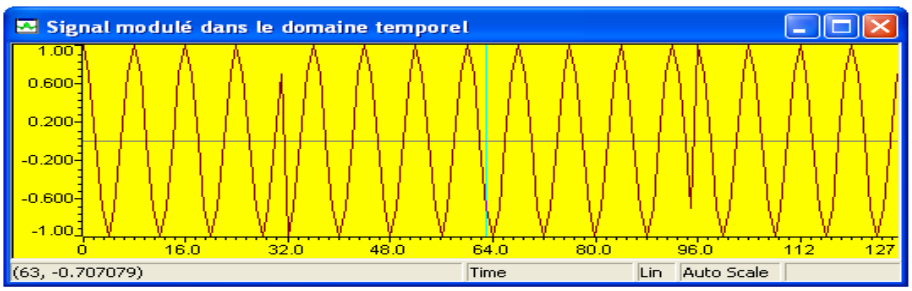


Figure 1.7 : Exemple d'un signal modulé en PAM par un signal binaire

1.5.2 Modulation PSK

La modulation PSK (Phase Shift Keying) ou modulation de phase (MDP) est une modulation pour laquelle les formes d'ondes sont données par [6] :

$$s_m(t) = \Re\left[\exp(2\pi j(m-1)/M)g(t)\exp(2\pi j f_c t)\right]$$
$$= g(t)\cos\left[2\pi f_c t + \frac{2\pi(m-1)}{M}\right] \qquad (1.10)$$

où $\theta_m = 2\pi(m-1)/M$ sont les différentes phases possibles de la porteuse qui véhicule l'information. Ces signaux ont tous même énergie donnée par :

$$\varepsilon = \int_0^T s_m^2(t)dt$$
$$= \frac{1}{2}\int_0^T g^2(t)dt$$
$$= \frac{\varepsilon_g}{2} \qquad (1.11)$$

L'ensemble des symboles possibles, que l'on appelle constellation est représenté à la figure 1.8 pour des nombres d'état $M = 2$, 4 et 8 ainsi que la mise en correspondance avec les mots binaires par encodage de Gray.

La figure 1.9 représente une porteuse sinusoïdale modulée en 4-PSK par un signal à quatre niveaux, la durée de chaque symbole (niveau) est égale à quatre fois la période de la porteuse.

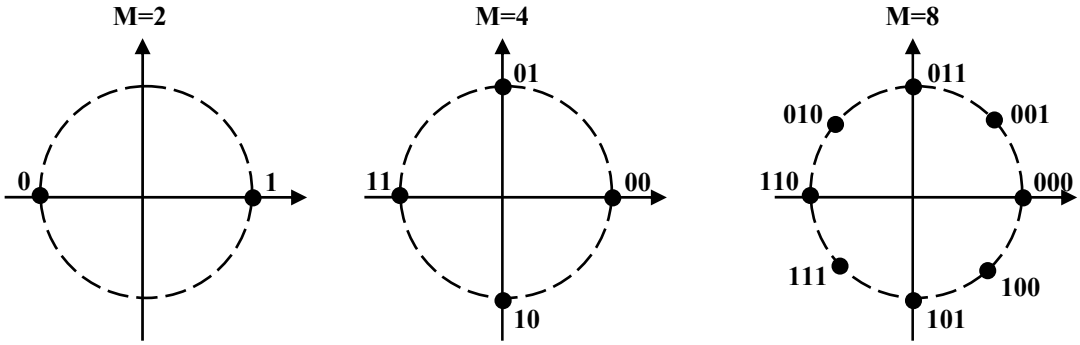


Figure 1.8 : Encodage de Gray pour des symboles à 2, 4 et 8 niveaux et modulation PSK

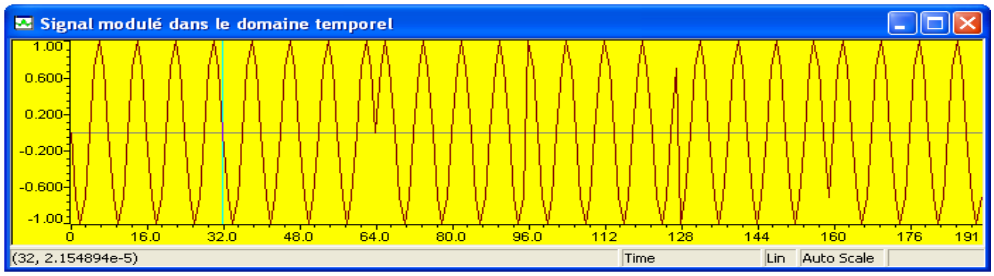


Figure 1.9 : Exemple d'un signal modulé en 4-PSK par un signal à 4 niveaux

1.5.3 Modulation FSK

Une autre quantité sur laquelle l'information peut être imprimée est la fréquence. Une version très simple de la modulation de fréquence numérique est celle où les formes d'ondes sont données par [6] :

$$s_m(t) = \Re\left[s_{lm}(t)\exp(2\pi j f_c t)\right]$$

$$= \sqrt{\frac{2\varepsilon}{T}}\cos\left[2\pi f_c t + 2\pi m\Delta_f t\right] \qquad (1.12)$$

Les formes d'onde en bande de base $s_{lm}(t)$ sont données par :

$$s_{lm}(t) = \sqrt{\frac{2\varepsilon}{T}}\exp\left(2\pi j m\Delta_f t\right) \qquad (1.13)$$

Ce type de modulation porte le nom de FSK (Frequency Shift Keying). L'écart de fréquence entre les différentes formes d'onde est donné par Δ_f. Toutes les formes d'onde ont la même énergie donnée par :

$$\varepsilon = \int_0^T s_m^2(t)dt \qquad (1.14)$$

Elles ont des coefficients de corrélation ρ_{kr} donnés par :

$$\rho_{kr} = \frac{1}{\sqrt{\varepsilon\varepsilon}}\int_0^T s_k(t)s_r(t)dt$$

$$= \frac{2}{T}\int_0^T \cos\left[2\pi f_c t + 2\pi k\Delta_f t\right]\cos\left[2\pi f_c t + 2\pi r\Delta_f t\right]dt$$

$$= \frac{1}{T}\int_0^T \cos\left[2\pi(k-r)\Delta_f t\right]dt$$

$$= \frac{1}{2\pi(k-r)\Delta_f T}\sin[2\pi(k-r)\Delta_f T] \qquad (1.15)$$

L'écartement minimal consiste à prendre $(k-r)=1$. Dans ce cas, la corrélation s'annulera à condition que :

$$2\pi\Delta_f T = n\pi, n \neq 0 \qquad (1.16)$$

L'espacement minimal est donné par le choix : $\Delta_f T = 1/2$ (1.17)

Les différents signaux $s_m(t)$ sont alors dits orthogonaux. Comme mentionné ci avant, cette définition de la FSK est simplifiée. Elle consisterait à commuter au rythme des symboles entre différents oscillateurs. Le signal produit contiendrait des discontinuités de phase, qui auront pour effet d'élargir le spectre.

La figure 1.10 montre une modulation de fréquence d'un signal binaire, on voit bien sur la figure 1.10 dans le domaine fréquentiel même dans le domaine temporel deux composantes fréquentielles, une composante correspond au bit 0 et l'autre au bit 1.

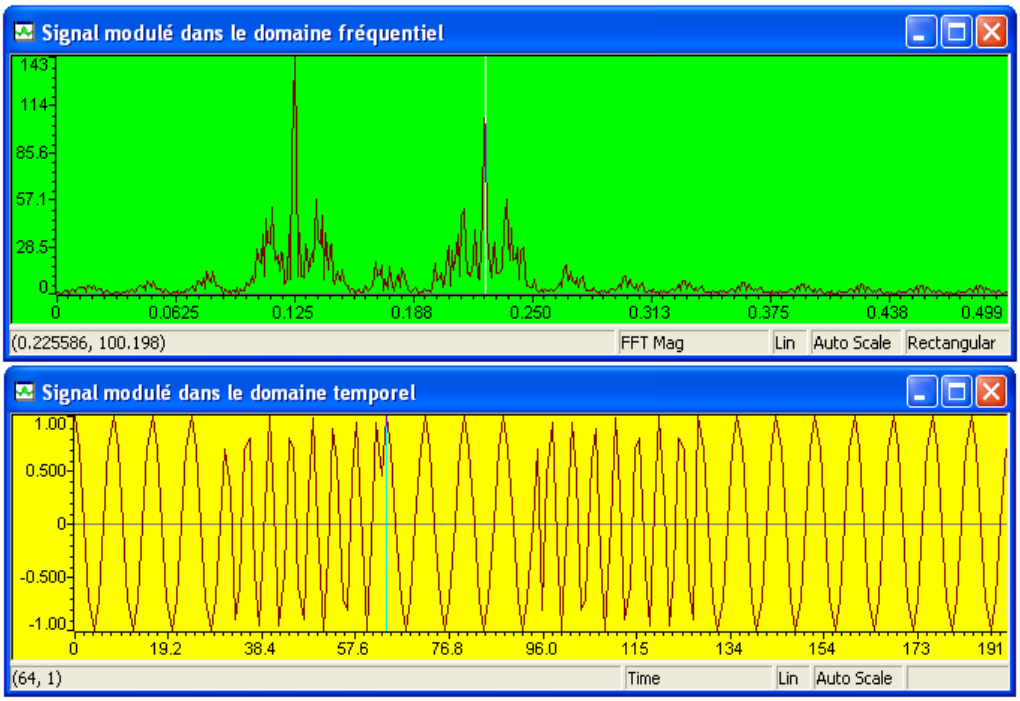


Figure 1.10 : Exemple d'un signal modulé en FSK par un signal binaire dans les domaines fréquentiel et temporel

1.5.4 Modulation QAM

La modulation QAM (Quadrature Amplitude Modulation) ou modulation d'amplitude en quadrature (MAQ) est une modulation où l'on effectue de la PAM sur les deux porteuses en quadrature associées à une même fréquence c'est-à-dire à la fois sur le cosinus et le sinus de la même porteuse. Les formes d'onde sont données par [6] :

$$\begin{aligned} s_m(t) &= \Re\left[\left(A_{mc} + jA_{ms}\right)g(t)\exp(2\pi j f_c t)\right] \\ &= A_{mc}g(t)\cos(2\pi f_c t) - A_{ms}g(t)\sin(2\pi f_c t) \\ &= V_m g(t)\cos(2\pi f_c t + \theta_m) \end{aligned} \tag{1.18}$$

avec :

$$V_m = \sqrt{A_{mc}^2 + A_{ms}^2}$$

$$\theta_m = \tan^{-1}\left(A_{ms} / A_{mc}\right) \tag{1.19}$$

Ces différentes équations servent à mettre en évidence que cette modulation peut aussi être vue comme une modulation combinée de phase et d'amplitude.

En combinant M_1 niveaux d'amplitude avec M_2 valeurs de phase, on a $M_1 M_2$ symboles dans la constellation. Des exemples de constellations pour M = 4, 8 et 16 sont représentés à la figure 1.11.

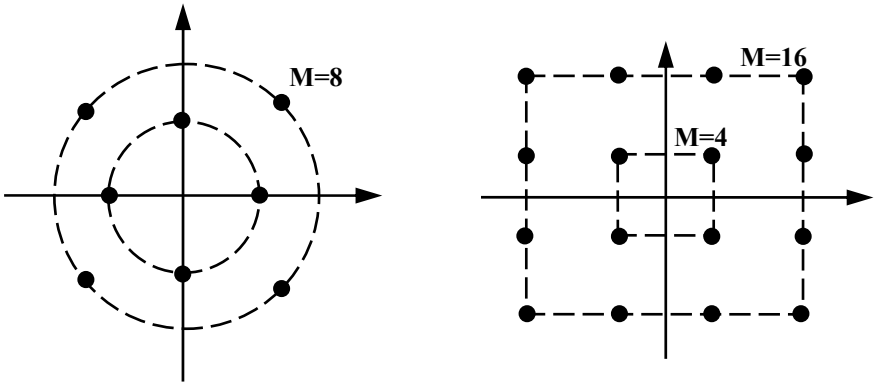


Figure 1.11 : Constellations pour la QAM

La figure 1.12 représente une porteuse sinusoïdale modulée en 16-QAM par un signal à 16 niveaux, la durée de chaque symbole (niveau) est égale à quatre fois la période de la porteuse.

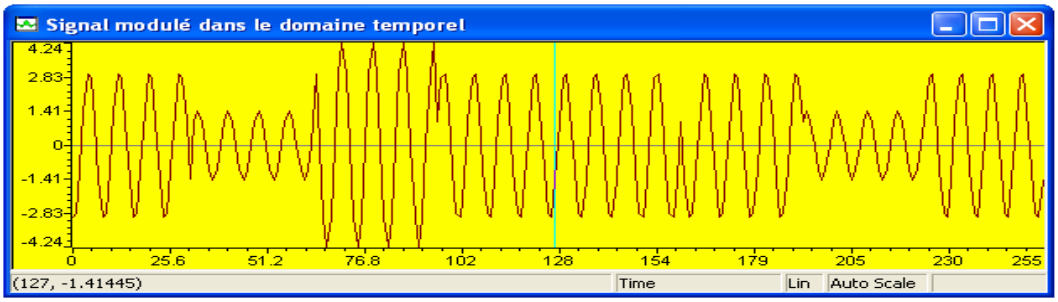


Figure 1.12 : Exemple d'un signal modulé en 16-QAM par un signal à 16 niveaux

1.5.5 Modulation TCM

La modulation TCM (**T**rellis **C**oded **M**odulation) est une technique moderne de modulation qui utilise les techniques présentées précédemment comme QAM ou PSK en combinaison avec le codage pour améliorer les performances de la transmission de données.
Elle a été incluse dans plusieurs standards des modems de la série V de l'ITU depuis sa découverte. Elle consiste à l'utilisation d'un codage convolutif à l'émetteur pour choisir des points de la constellation et l'utilisation de décodage de Veterbi décision-douce (soft-decision Veterbi decoding) au récepteur [7].

1.6 Standards de communication des modems

La multiplication des modems a nécessité une standardisation des protocoles de communication par modem afin qu'ils parlent tous un langage commun. C'est pourquoi deux organismes ont mis au point des standards de communication :

- Les laboratoires **BELL**, précurseurs en matière de télécommunication.
- Le Comité Consultatif International de Téléphonie et de Télégraphie (**CCITT**), renommé depuis 1990 en Union Internationale des Télécommunication (International Telecommunication Union **ITU**). L'ITU a pour but de définir les standards de communications internationaux.

Le tableau 1.2 présente une liste des principaux standards des modems [20] [21] [22] :

Standard	Date	Vitesse (bps)	Modulation	Mode	Description
Bell 103	1962	300	FSK	Full duplex	Standard américain et canadien utilisant un codage à changement de fréquence. Il permet ainsi d'envoyer un bit par baud.
CCITT V.21	1964	300	FSK	Full duplex	Standard international proche du standard *Bell 103*.
Bell 212A		1200	PSK	Full duplex	Standard américain et canadien fonctionnant selon le codage à changement de phase différentiel. Il permet de cette façon de transmettre 2 bits par baud
ITU V.22	1980	1200	PSK	Half duplex	Standard international proche du standard *Bell 212A*.
ITU V.22bis	1984	2400	QAM	Full duplex	Standard international constituant une version améliorée du standard V.22 (d'où l'appellation *V.22bis*).

ITU V.23	1964	1200	FSK	Half duplex	Standard international fonctionnant en half-duplex, c'est-à-dire permettant de transférer les données sur une seule voie à la fois. Possibilité d'une voie de retour à 75 bauds facultative.
ITU V.23	1964	1200	FSK	Full duplex	Standard international fournissant un duplex intégral asymétrique, c'est-à-dire qu'il permet de transférer des données à 1200 bps dans un sens et 75 bps dans l'autre sens.
ITU V.29	1976	9600	QAM	Half duplex	Standard international fonctionnant en half-duplex, c'est-à-dire permettant de transférer les données sur une seule voie à la fois. Ce standard a été mis au point particulièrement pour les faxs.
ITU V.32	1984	9600	QAM	Full duplex	Standard international fonctionnant en full-duplex et incorporant des standards de correction d'erreur. La transmission de données se fait selon une technique de correction d'erreurs appelée *modulation d'amplitude en quadrature codée en treillage*. Cette technique consiste à envoyer un bit supplémentaire pour chaque groupe de 4 bits envoyés sur la ligne de transmission.
ITU V.32bis	1991	14400	TCM	Full duplex	Standard international améliorant le standard V.32, en permettant d'envoyer 6 bits par baud, pour atteindre une vitesse de transmission de 14400 bps.
ITU V.32fast		28800	TCM	Full duplex	Standard international, nommé parfois V.FC (*Fast Class*), permettant d'atteindre une vitesse de transmission des données de 28800 bps.
ITU V.34	1994	28800	TCM	Full duplex	Standard international permettant d'obtenir un débit de 28800 bps. Grâce à un processeur DSP, les modems utilisant ce standard peuvent atteindre un débit de 33600 bps.

ITU V.42		14400			Même taux de transfert que V.32, V.32bis et d'autres standards mais avec meilleure correction d'erreur et donc plus fiable.
ITU V.90	1999	56000	TCM	Full duplex	Standard international permettant d'obtenir des vitesses de transmission de 56000 bps introduit en février 1999 (V.34 + 56 kbit/s).
ITU V.92	2000	56000	TCM	Full duplex	Standard international permettant d'obtenir des vitesses de transmission de 56000 bps introduit en janvier 2000 (V.34 + 56 kbit/s). Plus de compression et connexion plus rapide.

Tableau 1.2 : Liste des principaux standards des modems

1.7 Conclusion

Dans ce travail, nous allons implémenter un modem conforme au standard V.22bis qui utilise la modulation 16-QAM et le codage différentiel et transmit des bits à un débit de 2400 bps en mode full-duplex. Nous avons choisi ce standard puisqu'il présente le modèle de base. Les standards qui le suivent augmentent la vitesse de transmission, adaptent la modulation correspondante et ajoutent des blocs supplémentaires comme l'annulateur d'écho, le bloc de correction d'erreurs,…etc. Le modem 16-QAM à implémenter sera décris en détails dans le chapitre suivant.

Chapitre 2

Description et Conception d'un Modem

16-QAM

2.1 Introduction

Le modem 16-QAM à implémenter utilise la modulation d'amplitude en quadrature de phase (QAM). Ce choix est justifié par une large utilisation de cette modulation pour la transmission des données numériques sur des canaux passe-bande. Elle exploite efficacement la bande passante. Les distorsions linéaires dues au canal peuvent être corrigées par une égalisation adaptative au récepteur. Elle peut être considérée comme une généralisation de la modulation PAM dans des canaux passe-bande. Tous les modems téléphoniques actuels basés sur les recommandations des séries V de CCITT (actuellement ITU) pour la transmission aux débits de 2400 bps ou plus utilisent la modulation QAM. La modulation QAM est utilisée également dans les transmissions par : câbles ultrarapides, micro-ondes et systèmes satellitaires[7].

La conception du modem 16-QAM comprend la définition des architectures de son émetteur et son récepteur ainsi que les détails et les différents paramètres nécessaires pour l'implémentation du modem sur DSP TMS320VC5402.

2.2 Architecture du système modem

La figure 2.1 présente le schéma bloc du système modem à implémenter. Le système modem contient les sous-blocs suivants :

- L'interface hôte (Host Interface).
- Le modem.
- L'interface analogique (Analog Interface Circuit).

Le concepteur doit mettre une interface entre le PC et le modem pour convertir les données du PC en séquences séries de bits asynchrones exploitables par le modem.

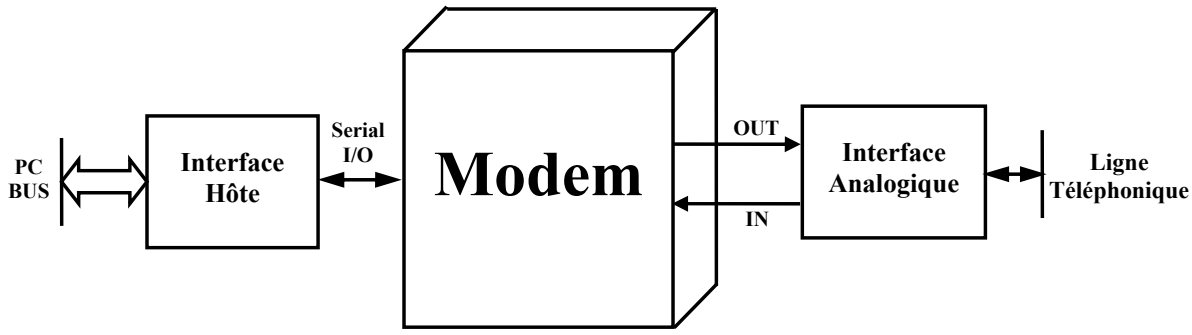


Figure 2.1 : Schéma Bloc des composants d'un système modem

Dans notre carte DSK TMS320VC5402, il existe une UART (Universal Asynchronous Receiver Transmitter) TL16C550 qui se connecte à travers un convertisseur série standard RS-232F avec un connecteur DB-9.

L'interface analogique est composée d'un Codec TLC320AD50 [12] (AIC : Analog Interface Circuit) et une interface téléphonique DAA (Data Access Arranged). Le Codec convertit un mot de 16 bits en un signal analogique et vice versa selon une technologie segma-delta avec une vitesse de conversion maximale de 22.05 KHz. L'interface téléphonique DAA (Data Access Arranged) est utilisée pour connecter la carte avec le réseau téléphonique PSTN ou un simulateur de ligne via une liaison RJ-11. Elle est basée sur un circuit intégré isolé optiquement CPC5604A [11].

Le bloc principal (le modem) réalise toutes les opérations d'un modem 16-QAM conforme au standard V.22 bis de l'ITU (modulation, codage, démodulation,…etc.) qui seront présentées dans les parties suivantes de ce chapitre. Notre modem contient deux parties : l'émetteur et le récepteur du modem.

2.3 Architecture de l'émetteur

L'architecture de l'émetteur du Modem est présentée par le schéma de la figure 2.2. On commence par une description générale de l'émetteur puis on détaille les blocs principaux de ce schéma.

2.3.1 Description générale de l'émetteur

La partie émettrice du modem est la partie qui convertit les données numériques de l'ordinateur en signaux analogiques capables d'être transmis par des canaux analogiques comme les lignes téléphoniques. Le modulateur est le module principal de la partie émettrice du modem [7].

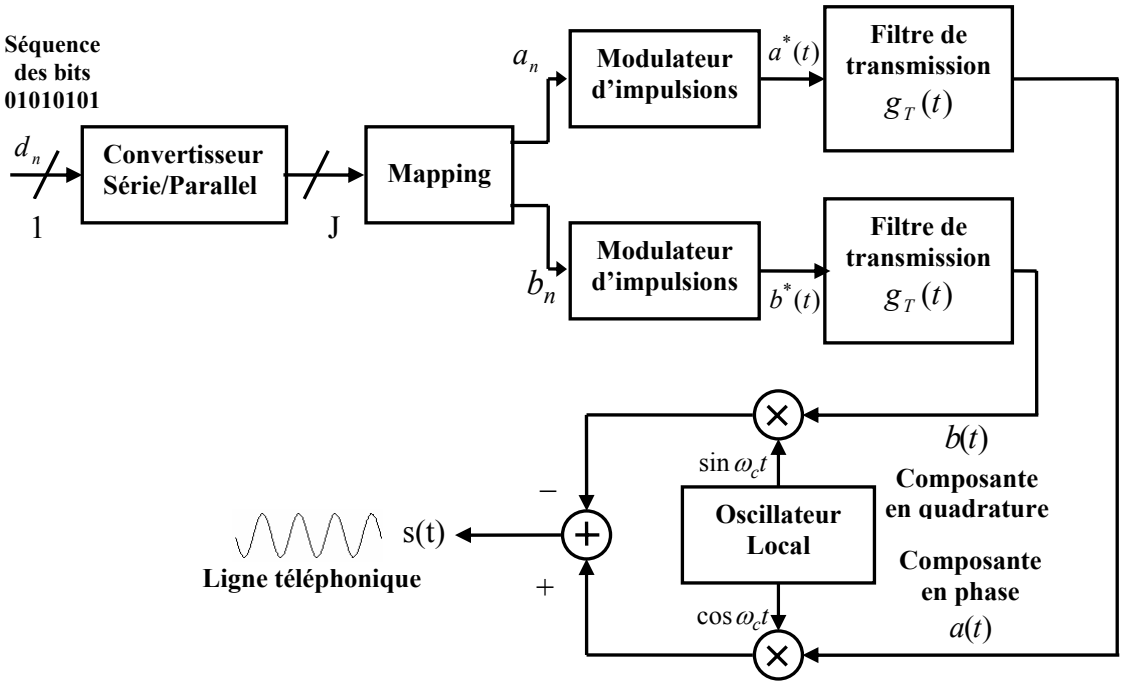


Figure 2.2 : Schéma de l'émetteur du modem 16-QAM

L'entrée de l'émetteur est une séquence binaire de données d_n parvenant à un débit égale à R_d bps avec une durée de bit $T_b = 1/R_d$. Le convertisseur Série-Parallèle groupe les bits d'entrée dans des mots binaires de taille J-bits. Chaque mot binaire de J-bits est représenté par un symbole parmi un alphabet de 2^J symboles. Le débit des symboles est $R_s = R_d / J$ baud (symboles/seconde) où $T_s = 1/R_s = J.T_b$ est la période d'un symbole.

L'alphabet contient des paires de nombres réels représentées par des points dans un espace des signaux de dimension 2 appelé constellation du signal. Il est commode de représenter l'espace de dimension 2 par un plan complexe et les symboles par des nombres complexes $c_n = a_n + jb_n$. On appelle la partie réelle a_n, la composante en phase ou la composante I et la partie imaginaire b_n, la composante en quadrature de phase ou la composante Q.

Les composantes I et Q passent séparément à travers des modulateurs PAM qui les convertissent en formes d'ondes (signaux) :

$$a^*(t) = \sum_{k=-\infty}^{\infty} a_k \delta_k (t - kT) \tag{2.1}$$

$$b^*(t) = \sum_{k=-\infty}^{\infty} b_k \delta_k (t - kT) \tag{2.2}$$

Ces signaux sont filtrés par des filtres identiques de mise en forme en bande de base. La réponse impulsionnelle de chaque filtre est $g_T(t)$. Les sorties $a(t)$ et $b(t)$ des filtres de mise en forme sont appelées les composantes en phase et en quadrature de phase du signal transmis $s(t)$. Elles sont données par les équations :

$$a(t) = \sum_{k=-\infty}^{\infty} a_k g_T(t - kT) \tag{2.3}$$

$$b(t) = \sum_{k=-\infty}^{\infty} b_k g_T(t - kT) \tag{2.4}$$

Le filtre de mise en forme en bande de base est typiquement un filtre passe bas approximativement le filtre à cosinus surélevée (Raised Cosine) ou à racine carrée de la cosinus surélevée (Square Root of Raised Cosine).

Par conséquent, $a(t)$ et $b(t)$ sont des signaux passe bas. Pour transposer ses spectres a la bande passante d'un canal passe bande, on les module en DSBSC-AM par les porteuses en quadrature $\cos \omega_c t$ et $\sin \omega_c t$ et on les soustrait pour former le signal transmis QAM :

$$s(t) = a(t)\cos \omega_c t - b(t)\sin \omega_c t \tag{2.5}$$

La pré-enveloppe du signal QAM est [7]:

$$\begin{aligned} s_+(t) &= s(t) + j\hat{s}(t) \\ &= [\,a\,(t)\,+\;\; jb\,(t)]\,e^{\,j\omega_c t} \end{aligned} \tag{2.6}$$

Donc, le signal transmis QAM est exprimé par :

$$\begin{aligned} s(t) &= \Re e\{s_+(t)\} \\ &= \Re e\{[a(t) + jb(t)]e^{j\omega_c t}\} \end{aligned} \tag{2.7}$$

L'enveloppe complexe de $s(t)$ est :

$$\tilde{s}(t) = s_+(t)e^{-j\omega_c t} = a(t) + jb(t) \tag{2.8}$$

En combinant (2.6), (2.3) et (2.4), on obtient :

$$s_+(t) = \sum_{k=-\infty}^{\infty} (a_k + jb_k) \times g_T(t - kT)e^{j\omega_c t} \tag{2.9}$$

Donc, le modulateur QAM est représenté d'une façon compacte en termes des signaux complexes comme représenté sur la figure 2.3. Le signal complexe $\tilde{s}(t)$ est simplement un signal PAM complexe généré par les symboles d'entrée complexes $c_n = a_n + jb_n$.

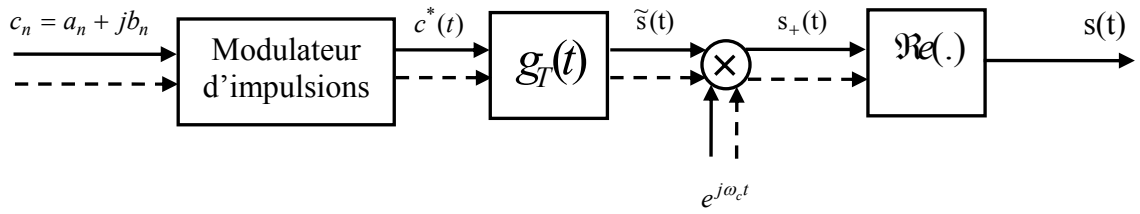


Figure 2.3: Représentation du modulateur QAM en termes des signaux complexes

2.3.2 Embrouillage et Codage Différentiel

Pour réaliser un système performant et faire une bonne assignation des symboles aux points de la constellation (Mapping), on ajoute au système de la figure 2.2 deux modules : l'embrouilleur (the scrambler) et le codeur différentiel [7].

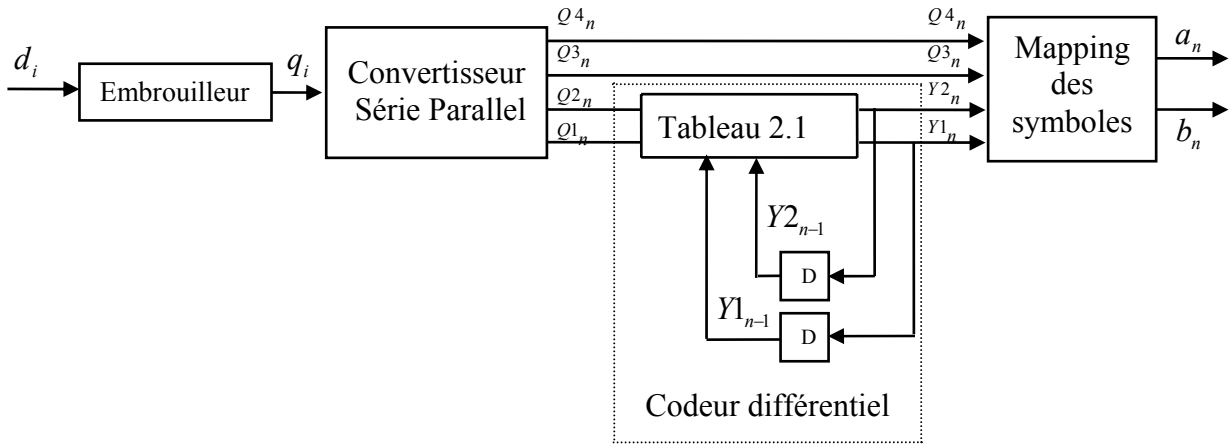


Figure 2.4 : Emplacement de l'embrouilleur et du codeur différentiel dans l'émetteur

2.3.2.1 Embrouillage

La séquence d_i des bits d'entrée est rendue aléatoire q_i par l'embrouilleur (scrambler). La raison d'utiliser un embrouilleur est d'éviter les grandes séquences de 1 ou de 0 et de rendre le choix des points de la constellation pseudo-aléatoire. Les blocs du récepteur exigent que cette variation de symboles fonctionne correctement pour une meilleure détection des données transmises.

L'embrouilleur de la norme V.22bis utilise l'équation aux différences [7] :

$$q_i = d_i \oplus q_{i-14} \oplus q_{i-17} \tag{2.10}$$

Où $\oplus$ représente l'adition modulo-2 ou la fonction logique "OU exclusif". Le polynôme de connexion correspondant est : $h(D) = 1 + D^{14} + D^{17}$ (2.11)

L'émetteur du modem V.32 utilise le polynôme de connexion suivant [7]:

$$h_c(D) = 1 + D^{18} + D^{23} \tag{2.12}$$

et le récepteur du modem V.32 utilise le polynôme [7]:

$$h_a(D) = 1 + D^{5} + D^{23} \tag{2.13}$$

2.3.2.2 Codage Différentiel

La sortie de l'embrouilleur q_i est groupée en mots de 4 bits par un convertisseur série/parallèle.

$$(Q1_n, Q2_n, Q3_n, Q4_n) = (q_{4n}, q_{4n+1}, q_{4n+2}, q_{4n+3}) \tag{2.14}$$

Ces mots sont générés à un débit de $R_s = R_d / 4$ baud.

Les deux premiers bits $(Q1_n, Q2_n)$ sont utilisés pour spécifier le changement du quadrant de la constellation par rapport au quadrant du symbole précédemment transmis. Le tableau 2.1 [7] montre le rapport entre les bits d'entrée actuels $(Q1_n, Q2_n)$, les bits du quadrant précédent $(Y1_{n-1}, Y2_{n-1})$ et les bits du nouveau quadrant $(Y1_n, Y2_n)$. Cette opération est appelée le codage différentiel. Le récepteur peut décider que la paire des bits $(Q1_n, Q2_n)$ est transmise par détermination du changement du quadrant entre le symbole courant et le symbole précédemment reçu. Les deux bits restant $(Q3_n, Q4_n)$ sont utilisés pour spécifier un point dans le quadrant $(Y1_n, Y2_n)$. Si on examine la figure 2.5, on verra que $(Q3_n, Q4_n)$ sont assignés pour qu'ils ne changent pas avec des rotations de constellation de 90°.

Entrées		Sorties précédentes		Changement de la phase (quadrant)	Sorties actuelles	
$Q1_n$	$Q2_n$	$Y1_{n-1}$	$Y2_{n-1}$		$Y1_n$	$Y2_n$
0	0	0	0	+90º	0	1
0	0	0	1	+90º	1	1
0	0	1	0	+90º	0	0
0	0	1	1	+90º	1	0
0	1	0	0	0º	0	0
0	1	0	1	0º	0	1
0	1	1	0	0º	1	0
0	1	1	1	0º	1	1
1	0	0	0	+180º	1	1
1	0	0	1	+180º	1	0
1	0	1	0	+180º	0	1
1	0	1	1	+180º	0	0
1	1	0	0	+270º	1	0
1	1	0	1	+270º	0	0
1	1	1	0	+270º	1	1
1	1	1	1	+270º	0	1

Tableau 2.1 : Codage différentiel utilisé au modem V.22bis

2.3.3 Modulateur

Notre modulateur utilise la constellation 16-QAM rectangulaire représentée sur la figure 2.5. Elle est utilisée dans les modems V.22bis de CITT pour la transmission à un débit de 2400 bps ou 600 baud et dans les modems V.32 -option non codée- pour la transmission à un débit de 9600 bps ou 2400 baud. Cette constellation utilise $J = 4$ bits par symbole [7].

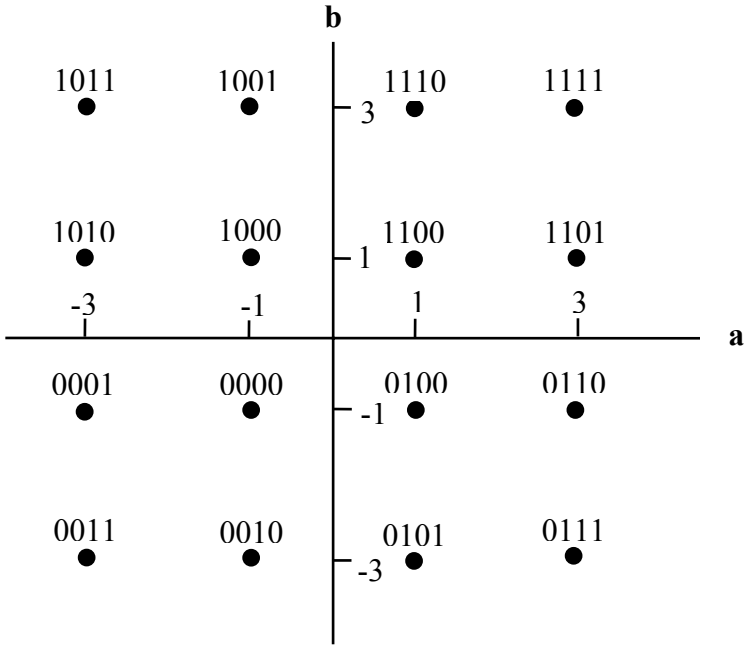


Figure 2.5 : Constellation 16-QAM

Le système peut être fait transparent par rapport des rotations de la phase de 90° par une combinaison du codage différentiel de deux bits d'entrée pour spécifier le quadrant de la constellation et l'utilisation des deux bits restants pour coder les points dans le quadrant.

2.3.4 Génération des porteuses en quadrature de phase par la méthode de la table de consultation (Lookup-table)

Nous utilisons la méthode de la table de consultation pour générer les porteuses en phase et en quadrature de phase utilisées dans la modulation 16-QAM. Cette méthode est considérée plus flexible et conceptuellement plus simple pour générer des signaux sinusoïdaux [5]. La technique est simplement la lecture d'une série des valeurs de données stockées dans une mémoire. Ces valeurs représentent les échantillons discrets du signal sinusoïdal à générer. Elles peuvent être obtenues en échantillonnant l'onde analogique appropriée ou généralement, en calculant les valeurs désirées en utilisant les programmes Matlab ou C. Assez d'échantillons sont générées et stockées dans une mémoire pour

représenter une période complète de la forme d'onde. Le signal périodique est alors généré par la répétition cyclique des données dans les emplacements mémoire en utilisant un pointeur circulaire. Cette technique est aussi utilisée pour générer la musique par ordinateur.

Une table de sinus contient des valeurs des échantillons également espacées, ces échantillons représentent une période complète de la forme d'onde. Une table de sinus de N points peut être calculée en utilisant la fonction suivante :

$$x(n) = \sin\left(\frac{2\pi n}{N}\right), \quad n = 0,1,.....,N-1 \tag{2.15}$$

Ces valeurs doivent être représentées en format binaire. L'exactitude de la fonction de sinus est déterminée par la longueur des mots utilisés pour représenter les échantillons et la longueur de la table de représentation. Le signal sinusoïdal désiré est généré en lisant les valeurs stockées de la table à une période d'échantillonnage constante de pas Δ, retournant autour à la fin de la table lorsque l'indicateur excède $N-1$. La fréquence du signal sinusoïdal généré dépend de la période d'échantillonnage T, la longueur de la table N et le pas d'incrémentation Δ :

$$f = \frac{\Delta}{NT} \text{ Hz} \tag{2.16}$$

Pour la table sinus conçue de longueur N, un signal sinusoïdal de fréquence f avec une fréquence d'échantillonnage f_s peut être généré en utilisant le pas d'incrémentation du pointeur d'adresse :

$$\Delta = \frac{Nf}{f_s}, \quad \Delta \le \frac{N}{2} \tag{2.17}$$

Pour générer L échantillons du signal sinusoïdal $x(l)$, $l = 0,1,.....,L-1$, on utilise un pointeur circulaire k tel que :

$$k = (m + l\Delta)_{\text{mod}\, N}, \tag{2.18}$$

Où m représente la phase initiale du signal sinusoïdal. Il est important de noter que le pas Δ donné en (2.17) peut être non entier, donc $(m + l\Delta)$ dans (2.18) est un nombre réel. C'est-à-dire un nombre contient une partie entière et une partie fractionnaire. Quand la partie fractionnaire de Δ est utilisée, les échantillons entre les éléments de la table doivent être estimés en utilisant les valeurs de la table. La solution facile est d'arrondir cette valeur non entière au plus proche entier.

Cependant, il existe une solution mieux mais plus complexe qui est l'interpolation des deux échantillons adjacents.

La méthode de la table de consultation est soumise aux contraintes imposées par l'aliasing, exigeant au moins deux échantillons par période dans la forme d'onde générée. Deux sources d'erreur dans l'algorithme de table de consultation causent la distorsion harmonique :

1. Une erreur de quantification d'amplitude est générée quand on représente les valeurs de la table de sinus par des nombres en précision finis.
2. Les erreurs de quantification de temps sont générées quand les points entre les éléments de la table sont échantillonnés, qui augmentent avec le pas d'incrémentation Δ. Quand la longueur de la table augmente, cette erreur devient moins significative.

Pour réduire l'occupation mémoire quand on génère un signal sinusoïdal avec une grande précision, nous pouvons profiter de la symétrie de la forme d'onde, qui aboutit en effet à une duplication de valeurs stockées. En effet, les valeurs sont répétées (sans regarder le changement de signe) quatre fois par période. Donc seulement un quart de la mémoire est nécessaire pour représenter une période de la forme d'onde. Cependant, le coût est une grande complexité de l'algorithme pour trouver dans quel quart de cercle on trouve la valeur à générer et avec le bon signe. Le meilleur compromis sera déterminé par la mémoire disponible et la puissance de calcul exigée pour une application donnée sur un système DSP.

Pour diminuer les distorsions pour une taille de table donnée N, un arrangement d'interpolation peut être utilisée pour plus de précision quand on calcule des valeurs entre les éléments de la table. L'interpolation linéaire est la méthode la plus simple à implémenter. Pour l'interpolation linéaire, la valeur de sinus pour un point entre deux éléments successifs de la table est considérée sur la ligne droite entre les deux valeurs. Supposons que la partie entière du pointeur est i $(0 \le i < N)$ et la partie fractionnaire est f $(0 < f < 1)$, la valeur de sinus est calculée comme suit :

$$x(n) = s(i) + f.\left[s(i+1) - s(i)\right], \tag{2.19}$$

Où $\left[s(i+1) - s(i)\right]$ est le segment de la ligne entre deux éléments successifs de la table i et $i+1$ [5].

2.4 Architecture du récepteur

L'architecture globale du récepteur correspondant au système de transmission 16-QAM (Emetteur 16-QAM) est présentée sur le schéma de la figure 2.6.

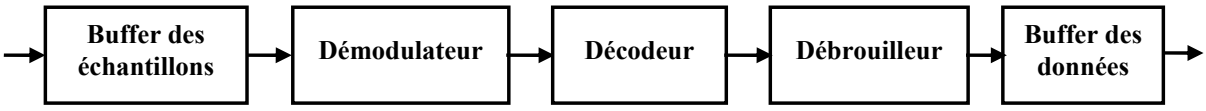


Figure 2.6 : Architecture globale du récepteur du modem

2.4.1 Description générale du récepteur

Selon le schéma de la figure 2.6 le buffer des échantillons stocke les échantillons du signal modulé reçu après la conversion analogique/numérique. Lorsqu'il reçoit un nombre suffisant des échantillons qui correspond à un nombre des symboles, il lance l'exécution du programme du démodulateur. Le démodulateur démodule le signal reçu et génère un symbole à la fois et donne un mot de 4 bits qui sera décodé par un décodeur différentiel. Le débrouilleur est appliqué aux bits décodés pour reconstruire la séquence des bits transmise.

2.4.2 Démodulateur

En supposant que le récepteur connaît exactement la fréquence et la phase de la porteuse. Le schéma du démodulateur cohérent est représenté sur la figure 2.7. Le démodulateur démodule les échantillons du signal reçu puis génère des symboles avant les décoder et les débrouiller. Les échantillons reçus sont filtrés par un filtre passe-bande pour éliminer le bruit hors bande et l'écho de la transmission. La démodulation consiste à la multiplication du signal filtré par -2*sinus et 2*cosinus de la porteuse. Cela crée deux composantes (I et Q).

$$I = s(t)2\cos\omega_c t = a(t) + a(t)\cos 2\omega_c t - b(t)\sin 2\omega_c t \tag{2.20}$$

$$Q = -s(t)2\sin\omega_c t = b(t) - b(t)\cos 2\omega_c t - a(t)\sin 2\omega_c t \tag{2.21}$$

$a(t)$ et $b(t)$ sont des signaux passe-bas. Les deuxième et troisième termes ont des spectres de fréquences autour de $2\omega_c$. Puisque la fréquence de la porteuse doit être choisie plus grande que la fréquence du signal passe-bas, les termes hautes fréquences sont éliminées par les filtres de détection passe-bas qui ont des fréquences de coupure choisies pour laisser passer $a(t)$ et $b(t)$ et éliminer les termes hautes fréquences indésirables. Enfin la reconstitution des symboles émis, le décodage différentiel, la conversion parallèle-série et le débrouillage pour déterminer la séquence des bits transmise.

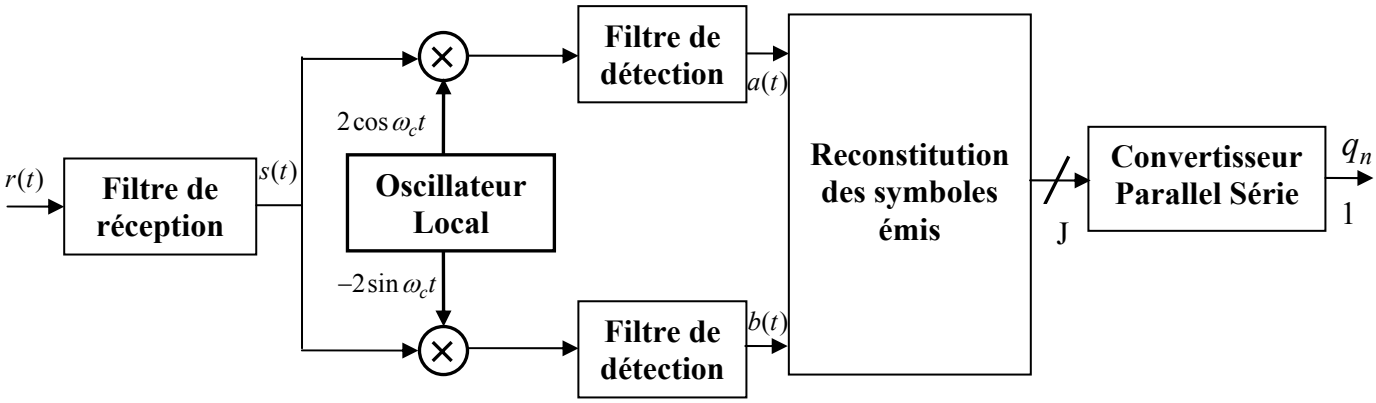


Figure 2.7 : Schéma du démodulateur

2.4.3 Décodeur

Le tableau 2.1 est utilisé pour décoder les bits reçus. Le récepteur décide que la paire des bits $(Q1_n, Q2_n)$ est transmise par détermination du changement du quadrant entre le symbole courant et le symbole précédemment reçu selon le tableau 2.1. Les deux bits restant $(\mathrm{Q3_n}, \mathrm{Q4_n})$ sont les troisième et quatrième bits du symbole reçus.

2.4.4 Débrouilleur

Le débrouilleur utilisé au récepteur du modem est identique à l'embrouilleur de V.22bis utilisé à l'émetteur avec l'équation aux différences [7] :

$$q_i = d_i \oplus q_{i-14} \oplus q_{i-17} \tag{2.22}$$

Où $\oplus$ représente l'adition modulo-2 ou la fonction logique "OU exclusif". Le polynôme de connexion correspondant est :

$$h(D) = 1 + D^{14} + D^{17} \tag{2.23}$$

2.5 Conclusion

L'objectif de ce chapitre était de présenter les blocs fonctionnels du modem 16-QAM à implémenter et les modélisations mathématiques correspondantes pour faciliter l'implémentation par la suite. Dans le prochain chapitre, nous allons passer à l'implémentation logicielle et matérielle du modem.

Chapitre 3

Implémentation du Modem 16-QAM sur un DSP TMS320VC5402

3.1 Introduction

L'implémentation du modem est faite par un DSP TMS320VC5402. Ce chapitre présente l'élaboration et la mise en oeuvre des codes sources de l'implémentation de notre modem. Les résultats de l'implémentation font l'objectif du dernier chapitre.

Avant de parler sur l'implémentation logicielle et matérielle de notre modem, nous donnerons une brève description sur les outils matériels et logiciels de développement nécessaires pour l'implémentation de notre modem. Pour une description détaillée sur ces outils, voir les références [9], [10] et [25].

3.2 Outils matériels de développement

3.2.1 Carte DSK TMS320VC5402

Nous utilisons dans notre implémentation le kit de démarrage DSK TMS320VC5402 disponible au laboratoire. Ce kit est destiné au domaine de la parole, ce qui le rendre valable pour l'implémentation des modems téléphoniques. C'est un système de développement matériel très faible coût, contenant une carte DSP, une alimentation, un déboggeur et des outils de génération de code assembleur spécifiques au système DSK. Il peut être considéré comme un système idéal pour un apprentissage des processeurs de traitement du signal, permettant de développer des applications temps réel sur des signaux physiques [1]. Cette carte se connecte à un PC via un port parallèle. Elle contient un DSP TMS320VC5402 fonctionnant à une vitesse maximale de 100 MHz. Elle possède également une mémoire externe SRAM de 64K mots et une mémoire FLASH de 256K mots.

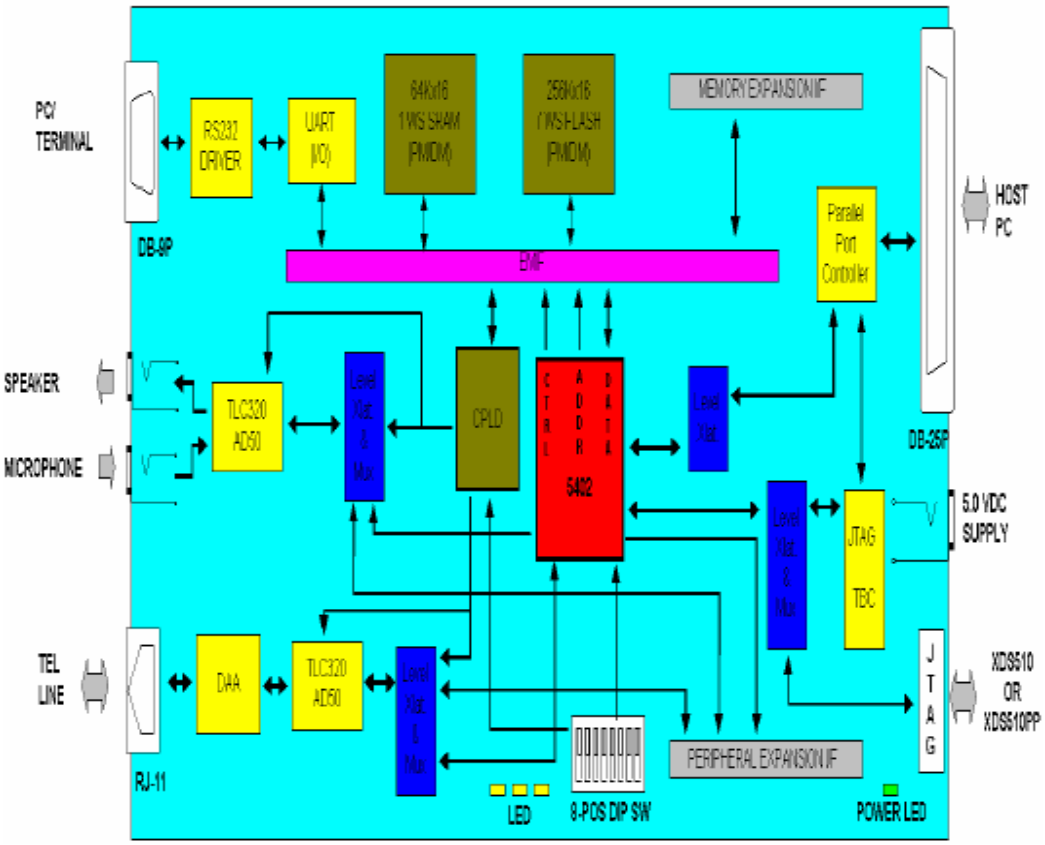


Figure 3.1 : Schéma bloc du DSK TMS320VC5402

La carte comprend de plus une interface analogique formée du circuit TLC320AD50 permettant la conversion analogique/numérique et numérique/analogique, une interface série UART(Universal Asynchronous Receiver Transmitter), un convertisseur de niveau asynchrone RS-232 ainsi qu'une interface téléphonique DAA avec un connecteur RJ11 [25].

Le déboggeur utilisé avec cette carte est le déboggeur C*ode Composer* avec son interface graphique conviviale et ses possibilités de tracés, d'animation et de profiling. Les outils de génération de code assembleur sont spécifiques au DSK. Ils permettent de faire en une seule passe l'assemblage et l'édition de liens d'un fichier source [1].

3.2.2 DSP TMS320VC5402

Le TMS320VC5402 est un DSP à virgule fixe, il peut être cadencé jusqu'à 100 MHz avec une alimentation principal de 1.8 volts et une alimentation d'entrée-sortie de 3.3V. Il a une architecture de base de type Harvard modifiée, comprenant un bus pour la mémoire programme, trois bus pour la mémoire donnée et quatre bus d'adresse, les mots de programme et de données sont sur 16 bits. Il possède une mémoire interne formée de 4 Kmots de ROM et de 16 Kmots de RAM (DARAM) et deux ports séries qu'on va utiliser le port 0 (McBSP0)

pour le transfert des échantillons entre le DSP et le TLC320AD50 AIC (l'interface analogique) et deux timers. Pour une description détaillée sur le DSP TMS320VC5402, voir la référence [14] de la bibliographie.

3.2.3 Interface analogique TLC320AD50 AIC

Le TLC320AD50 est un circuit d'interface analogique (AIC : Analog Interface Circuit) réalise des conversions analogique digitale (A/D) et digitale analogique (D/A) avec une haute résolution en utilisant la technologie d'échantillonnage sigma-delta [12]. Ce circuit consiste à deux conversions séries de 16 bits synchrones (une pour chaque direction) et contient un filtre d'interpolation avant le DAC et un filtre de quantification après l'ADC, la vitesse de conversion est variable jusqu'à 22.05 Khz. Il est doté d'un port série qui permet la liaison avec le DSP TMS320VC5402 (le port0 McBSP0) [12].

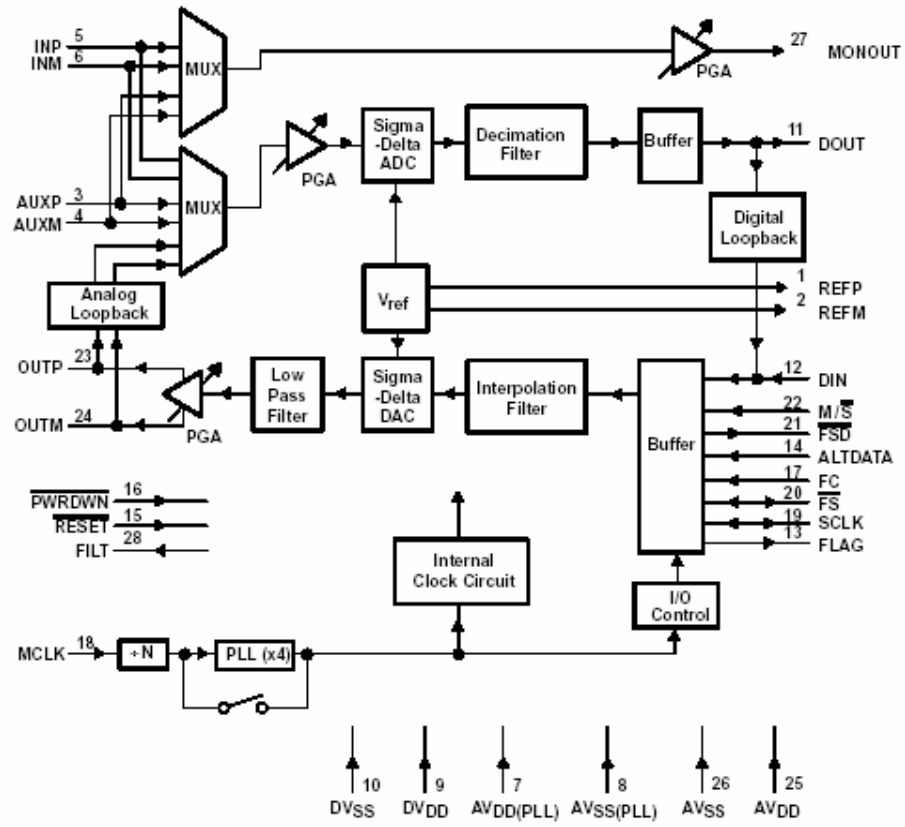


Figure 3.2 : Bloc diagramme du TLC320AD50

3.2.4 Interface UART

L'interface UART (Universal Asynchronous Receiver Transmitter) est une interface série utilisée pour le transfert asynchrone des données entre le DSP et le PC à travers un convertisseur de niveau RS 232F (circuit TI/Maxim MAX3238) et un connecteur DB-9. Elle

est basée sur un circuit standard TL16C550. La vitesse de transmission (en bauds) de l'UART est contrôlée par un registre programmable de génération de Vitesse de transmission. Le signal d'horloge de ce registre est généré par un oscillateur interne fonctionnant à 3.6864 MHZ [25].

3.2.5 Interface Téléphonique DAA (Data Access Arranged)

L'interface téléphonique DAA (Data Access Arranged) est utilisée pour connecter la carte DSK avec le réseau téléphonique ou un simulateur de ligne via une liaison RJ-11. Elle est conçue selon les standards de communication des ETATS-UNIS, du Japon et de la norme européenne CTR21 [25]. L'interface DAA est basée sur un circuit intégré isolé optiquement CPC5604A [11].

Le DAA constitue seulement une connexion. Le traitement de la voix et des données doit être réalisé par le DSP.

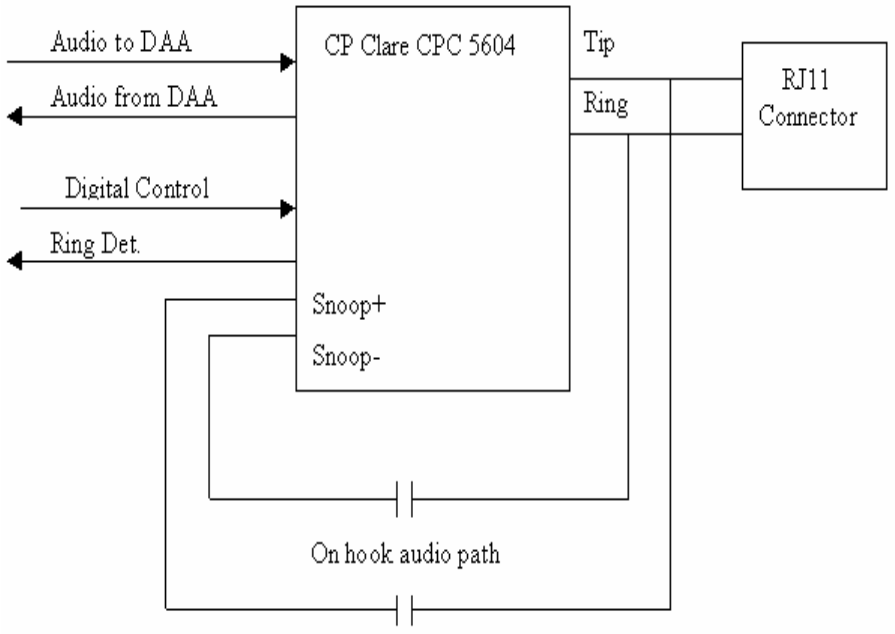


Figure 3.3 : Schéma bloc du DAA

3.3 Outils logiciels de développement

Pour programmer les DSPs, il faut utiliser tout un environnement logiciel de développement. Cet environnement permet de supporter les tâches d'édition de code source en langage "C" ou en assembleur, la compilation, l'édition des liens et la simulation système.

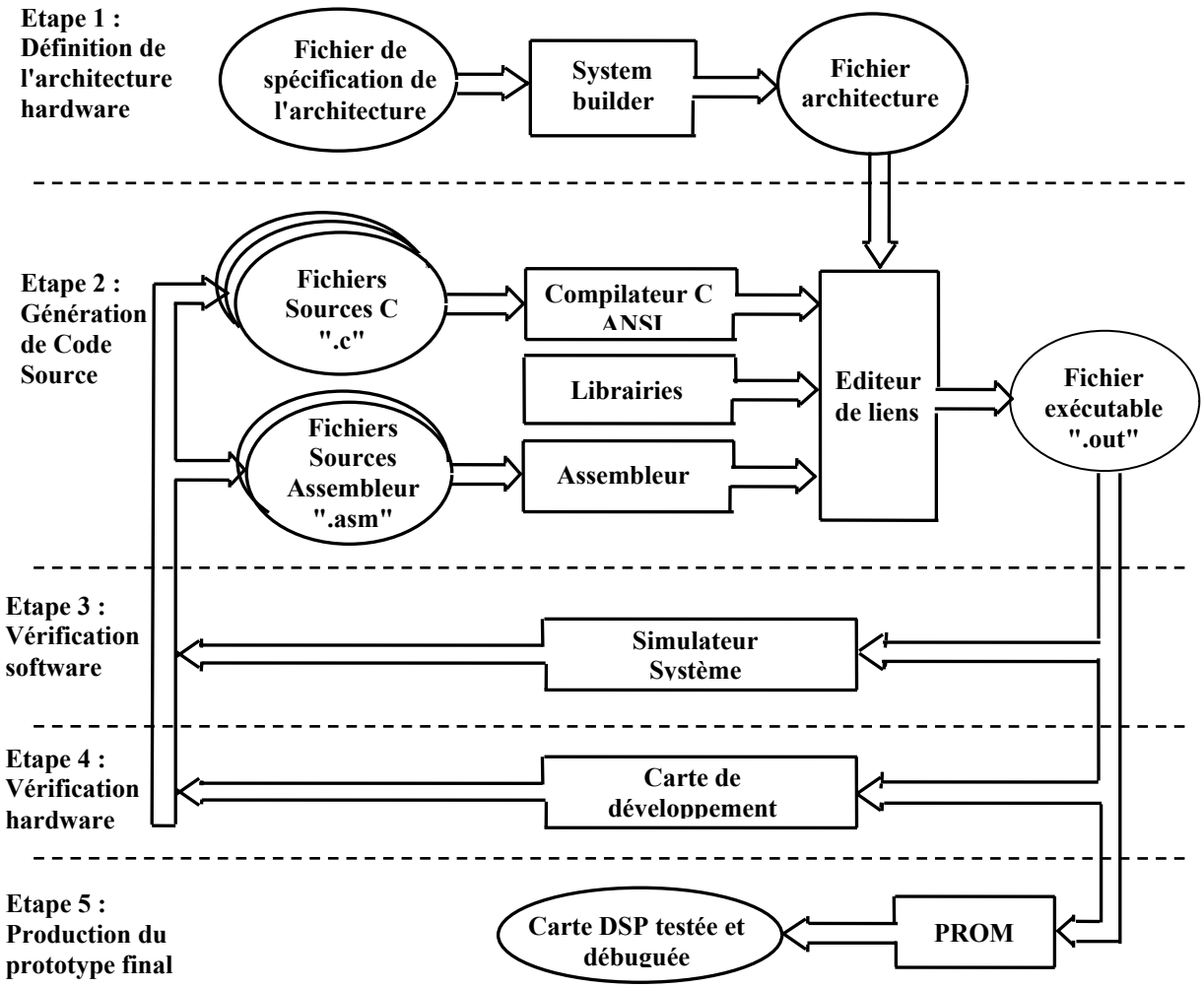


Figure 3.4 : Environnement de développement pour le DSP TMS320VC5402

Les phases de développement d'un programme exécutable sur un DSP sont les suivantes (figure 3.4) :

- Définition de l'architecture hardware.
- Génération du code source : le programmeur fournit un fichier source en langage C ".c" ou en assembleur ".asm". Ce fichier est compilé par l'environnement de développement pour donner un fichier exécutable ".out".
- Vérification software : dans cette étape on utilise un simulateur système "Code Composer Studio" dans lequel on charge le fichier exécutable et on le teste en observant les registres ainsi que la mémoire.
- Vérification hardware : on charge le fichier exécutable sur la carte de développement DSK TMS320VC5402. Selon les résultats de test on peut retourner aux étapes précédentes.

3.3.1 Environnement de Développement Code Composer Studio IDE

Code composer est un environnement de développement intégré commun aux différents DSPs de Texas Instruments, qui peut être utilisé seul ou avec une cible matériel fonctionnant en temps réel telle qu'une carte d'évaluation. Cet environnement permet de construire un projet, d'éditer les fichiers, de les compiler ou les assembler, de faire de l'édition de liens, de tester et de débogger une application mono ou multiprocesseur en simulation ou en temps réel sur une cible matérielle. Les cibles matérielles supportées sont les cartes d'évaluation EVM, les kits de développement DSK, ainsi que les émulateurs de Texas Instruments …etc.

Code Composer est un déboggeur symbolique permettant de référencer les variables et les constantes par leurs noms, plutôt par leurs adresses physiques. Le test des applications est facilité par des outils d'analyse des performances des différentes parties d'une application : *profiling.*

Code Composer offre des outils graphiques permettant de tracer et d'analyser des signaux contenus en mémoire. Ces tracés peuvent se faire en temps ou en fréquence (après application par *Code Composer* d'une transformée de Fourier au signal). D'autres types de tracés sont disponibles, comme les diagrammes de l'oeil ou les constellations [1].

Code Composer dispose d'ailleurs de toutes les possibilités suivantes :

- Simulation des instructions DSP sur un calculateur hôte (PC par exemple) au niveau C ou assembleur.
- Interface graphique Windows configurable et multifenêtres.
- Visualisation du programme C et /ou assembleur.
- Visualisation, modification, et initiation des programmes.
- Visualisation et modification des registres du DSP.
- Visualisation des variables C.
- Exécution du programme pas à pas.
- Mesure du nombre de cycles de temps nécessaire à l'exécution des instructions pour évaluer les performances en vitesse.
- Possibilité de positionner ou d'effacer des points d'arrêt programmables sur l'acquisition.
- La possibilité de suivre les valeurs de l'accumulateur, du compteur programme ou des registres auxiliaires.

- Un modèle d'analyse et de *profiling* dynamique, permettant d'analyser les performances du programme et d'identifier les zones critiques à optimiser.
- Possibilité de connecter un fichier à un port d'E/S, permettant de simuler les E/S comme la lecture ou l'écriture sur un port série asynchrone.

3.3.2 Compilateur C

Les étapes de génération d'un fichier exécutable à partir d'un code sources en C ou C/C++ sont représentées sur la figure 3.5.

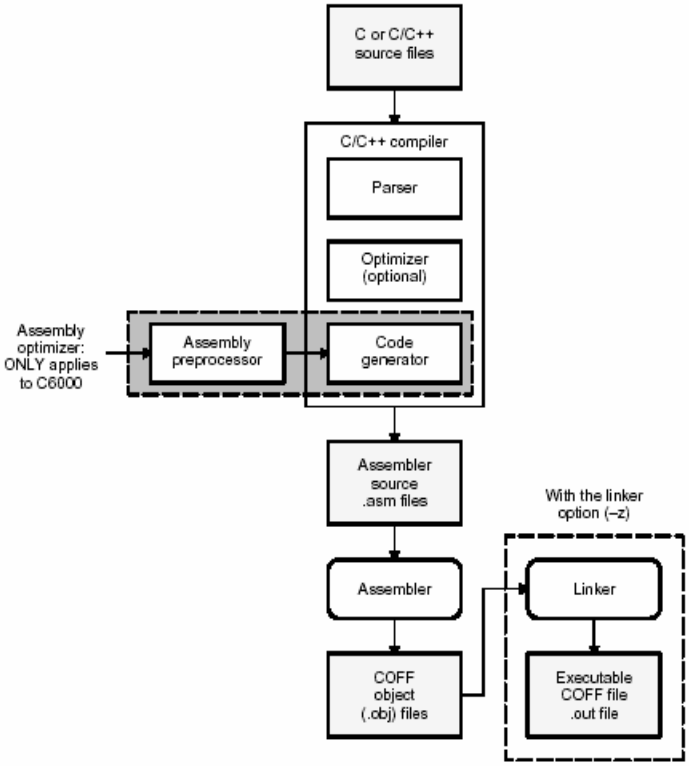


Figure 3.5 : Les étapes de génération d'un fichier exécutable à partir d'un code source en C

Ecrire un programme en langage C est plus simple et plus rapide qu'écrire le même algorithme en assembleur. De plus, l'écriture en C garantit la portabilité du programme sur différents processeurs. Le langage C utilisé dans notre projet est le compilateur C compatible ANSI de TI pour les DSP de la plate forme C5000 (les familles C54x et C55x).

3.4 Mise en oeuvre des codes sources du modem

L'organigramme illustré sur la figure 3.6 décrit le fonctionnement séquentiel des tâches programmées pour le modem16-QAM.

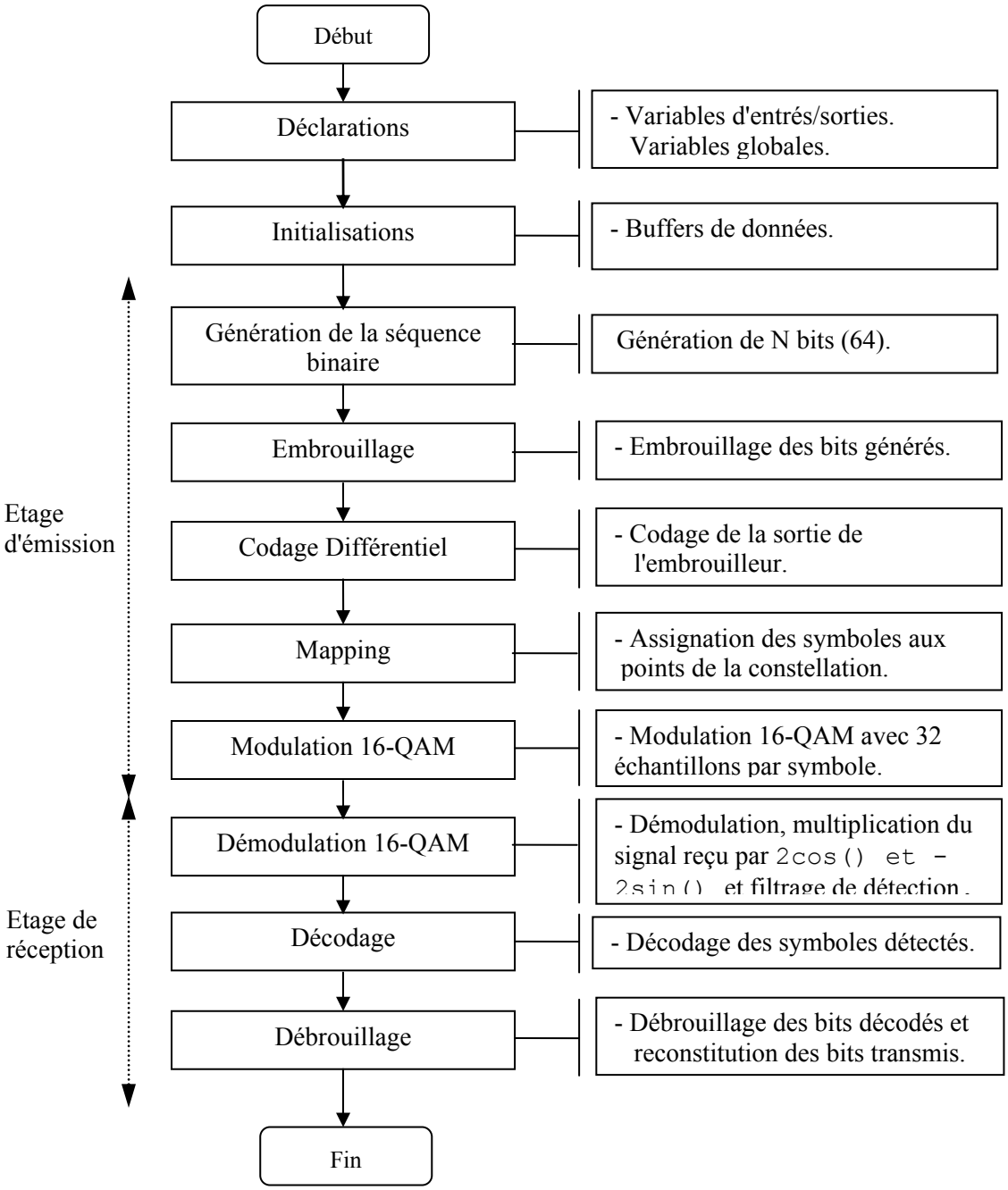


Figure 3.6 : Organigramme principal du modem 16-QAM

Après avoir défini l'organigramme général du modem 16-QAM ainsi que ses différents blocs, nous allons nous focaliser sur la programmation de ces blocs en donnant et expliquant les différentes fonctions qui les réalisent et les implémentent.

3.4.1 Définition des paramètres du modem

Au début, on définit les paramètres du modem (fichier d'entête "data.h", Figure 3.7). On commence par la table Sinus qui contient 128 éléments (valeurs réelles de la fonction sinus)

représentés en hexadécimal en format Q_{15} (l'élément dans ce format est représenté comme un nombre entier sur 16 bits, les quinze bits à droite représente les chiffres après la virgule et le seizième bit à gauche indique le bit de signe). En format Q_{15} sur 16 bits, les nombres réels représentés appartiennent à l'intervalle [-1 et 1[. L'objectif de la table Sinus est de générer des échantillons des fonctions Sinus et Cosinus utilisées comme des porteuses en quadrature de phase pour la modulation 16-QAM. Cette table est stockée dans la mémoire et utilisée pour générer les échantillons des porteuses en quadrature en évitant l'appel des fonctions `sin()` et `cos()` de la bibliothèque standard de C qui provoque un retard en temps d'exécution. Une structure a été définie pour représenter la constellation 16-QAM. Un vecteur contenant les coefficients d'un filtre moyenneur est également déclaré. Enfin les buffers des bits, des symboles et des échantillons ainsi que les différents paramètres et variables utilisés pour l'implémentation du modem sont définis.

```
#define Taille_Table_Sinus 128
#define Nombre_Points_Constellation 16
#define Echantillons_Par_Symbole 32
#define Nombre_Symboles 16
#define K 8         // La taille du filtre

/* Le contenu de ce fichier est donné dans l'annexe 1 (Page 70)*/
```

Figure 3.7 : Fichier d'entête "data.h"

3.4.2 Etage d'émission

L'organigramme de l'étage d'émission est illustré sur la figure 3.8.
L'implémentation de l'émetteur consiste à programmer chaque bloc de celui-ci par une fonction en langage C. Les fonctions de l'émetteur sont organisées et appelées par la fonction principale d'émission représentée sur la figure 3.19. Les fonctions sont les suivantes :

3.4.2.1 Fonction d'initialisation des variables globales du modem

La fonction d'initialisation des variables (Figure 3.9) initialise tous les paramètres du modem pour éviter des valeurs indésirables qui peuvent influer sur les résultats. Tous ces paramètres sont définis dans le fichier d'entête "data.h" (Figure 3.7) avec ses types, tailles et dimensions.

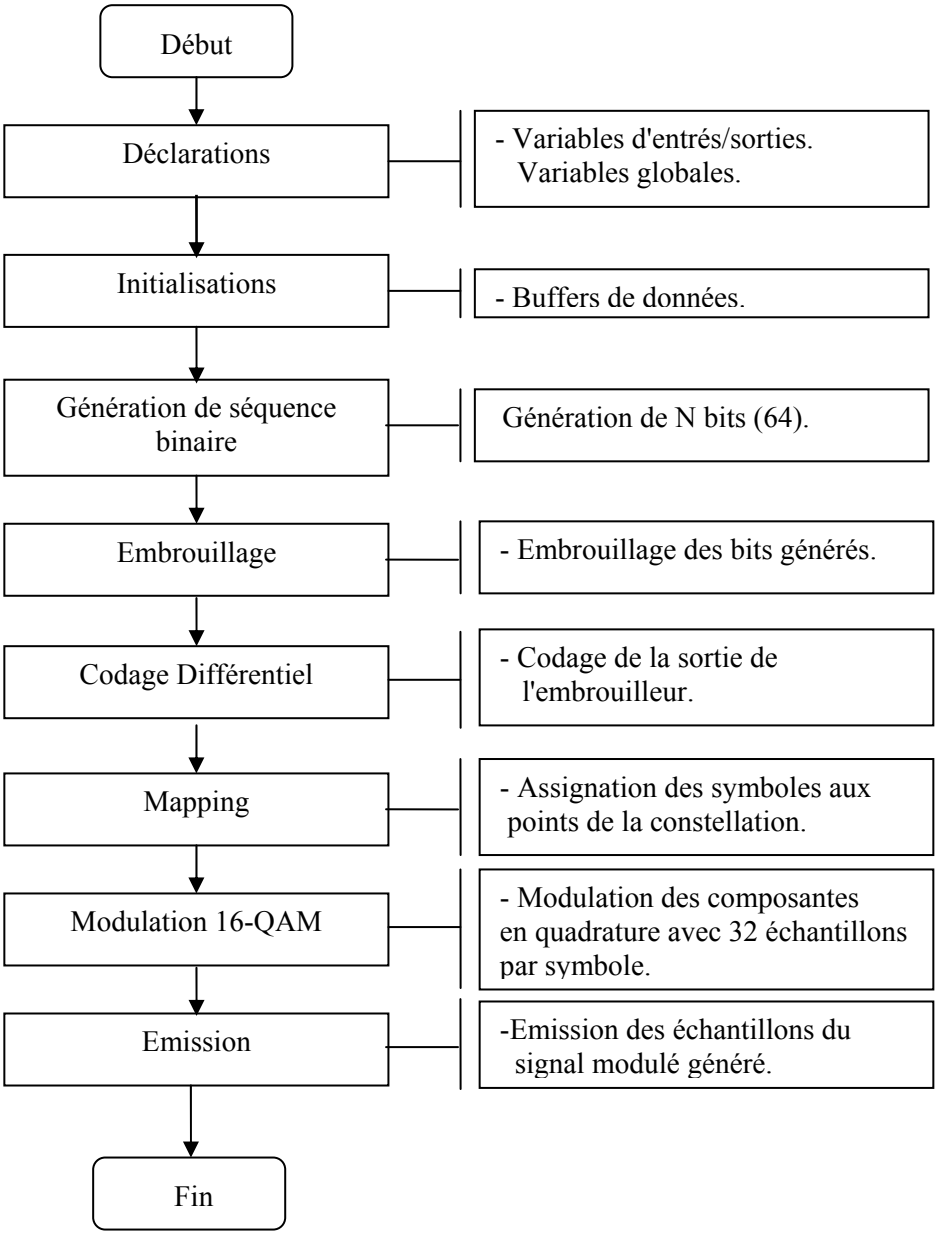


Figure 3.8 : Organigramme de l'émetteur du modem

3.4.2.2 Fonction de lecture des données

La fonction de lecture des données (Figure 3.10) lit les données (bits) à transmettre et les stocke dans un buffer de données (`modem.bufferDonnees[]`) pour les rendre prêtes à traiter ou à transmettre.

```
void Initialisation(void)
{
     int i;
     modem.frequencePorteuse = 64;
     modem.phase = 0;      // Initialiser la phase de la porteuse
     modem.echantillonsParSymbole = Echantillons_Par_Symbole;
     // Initialiser les Buffers des bits
     for( i = 0;i < 4*Nombre_Symboles; i++ )
    {
           modem.bufferDonnees[i] = 0;
           modem.sScr[i] = 0;
           modem.sCod[i] = 0;
           modem.eDec[i] = 0;
           modem.sDec[i] = 0;
           modem.sDescr[i] = 0;
     }
     // Initialiser les Buffers des symboles et de la constellation
     for( i = 0;i < Nombre_Symboles; i++ )
     {
           modem.symbolesE[i] = 0;
           modem.symbolesR[i] = 0;
           modem.pointsConstellation[i].I = 0;
           modem.pointsConstellation[i].Q = 0;
     }
     // Initialiser les Buffers des échantillons
     for( i = 0;i < Echantillons_Par_Symbole*Nombre_Symboles; i++ )
     {
           modem.bufferSortie[i] = 0;
           modem.bufferR1[i] = 0;
           modem.bufferR2[i] = 0; modem.a[i]=0;
           modem.signalFiltreI[i] = 0; modem.b[i]=0;
           modem.signalFiltreQ[i] = 0; modem.s[i]=0;
     }
}
```

Figure 3.9 : Fonction d'initialisation des variables globales (`Initialisation()`)

```
void LireDonnees(void)
{
int n;
for (n=0;n<4*Nombre_Symboles;n+=8)
     {
           modem.bufferDonnees[n]=1;
           modem.sScr[n] = modem.bufferDonnees[n];
           modem.bufferDonnees[n+1]=1;
           modem.sScr[n+1] = modem.bufferDonnees[n+1];
           modem.bufferDonnees[n+2]=1;
           modem.sScr[n+2] = modem.bufferDonnees[n+2];
           modem.bufferDonnees[n+3]=1;
           modem.sScr[n+3] = modem.bufferDonnees[n+3];
     }
}
```

Figure 3.10 : Fonction de lecture des données (`LireDonnees()`)

3.4.2.3 Fonction d'embrouillage

La fonction d'embrouillage (Figure 3.11) rend les bits d'entrée (les données) du buffer de données (`modem.bufferDonnees[]`) aléatoires en utilisant l'embrouilleur de la norme V.22bis décris en (2.3.2.1). Les bits de sortie seront stockés dans le buffer de sortie de l'embrouilleur (`modem.sScr[]`).

```
void Scrambler(void)
{
   int n;
     for (n=17;n<4*Nombre_Symboles;n++)
     {
modem.sScr[n] =(modem.sScr[n]+modem.sScr[n-14]+modem.sScr[n-17])%2;
     }
}
```

Figure 3.11 : Fonction d'embrouillage (`Scrambler()`)

3.4.2.4 Fonction de codage différentiel

La fonction de codage (Figure 3.12) passe les bits sortant de l'embrouilleur à travers un codeur différentiel (voir codage différentiel (2.3.2.2) et Tableau 2.1) pour faire du codage différentiel afin de minimiser les erreurs de détection à la réception. Les bits codés générés seront stockés dans le buffer de sortie du codeur (`modem.sCod[]`).

```
void Codeur(void)
{
/* Le corps de cette fonction est donné dans l'annexe 1 (Page 71)*/
}
```

Figure 3.12 : Fonction de codage différentiel (`Codeur()`)

3.4.2.5 Fonction de constitution des symboles émis

La fonction de constitution des symboles émis (Figure 3.13) spécifie à partir des bits codés sortant du codeur les symboles à émettre (voir la méthode de désignation des points de la constellation 16-QAM sur la figure 2.5). Les symboles constitués seront sauvegardés dans le buffer des symboles émis (`modem.symbolesE[]`).

```
void GenererSymboles(void)
{
/* Le corps de cette fonction est donné dans l'annexe 1 (Page 73)*/
}
```

Figure 3.13 : Fonction de constitution des symboles émis (`GenererSymboles()`)

3.4.2.6 Fonction de mapping

La fonction de mapping (Figure 3.14) représente chaque symbole par un point dans la constellation, c'est-à-dire représenter chaque symbole par deux coordonnées de la constellation (les composantes en phase et en quadrature de phase). La constellation utilisée est celle définie en (2.3.3), c'est une constellation 16-QAM rectangulaire ou carrée utilisée dans les modems V.22bis de ITU.

```
void LireConstellation(void)
{
     int i;
     for( i = 0;i < Nombre_Symboles; i++ )
     {
     /* Convertir la donnée en point de la constellation */
 modem.pointsConstellation[i]=
Constellation[(modem.symbolesE[i])&15];
     }
}
```

Figure 3.14 : Fonction de mapping (`LireConstellation()`)

3.4.2.7 Fonction de génération des échantillons d'une sinusoïde

La fonction de génération des échantillons d'une sinusoïde (Figure 3.15) génère une valeur de la fonction Sinus qui a un entier positif (représentant le temps) à son entrée. Cette fonction avec la fonction de génération des échantillons d'une cosinus sont appelées par la fonction `Modulation()` pour former les porteuses en phase et en quadrature de phase du signal émis. L'algorithme de la fonction de génération des échantillons d'une sinusoïde est expliqué avec un peu de détails en (2.3.4).

```
int Sinus(int t)
{
     int valeurSinus;

     /* Ajustement pour les quadrants 2 et 4 */
     int a = t & (Taille_Table_Sinus-1);
     if (t & Taille_Table_Sinus)
          a = Taille_Table_Sinus-1-a;

     /* Ajustement pour les quadrants 3 et 4 */
     valeurSinus = Table_Sinus[a];
     if (t & (Taille_Table_Sinus*2))
          valeurSinus = -valeurSinus;

     return(valeurSinus);
}
```

Figure 3.15 : Fonction de génération des échantillons d'une sinusoïde (`Sinus()`)

3.4.2.8 Fonction de génération des échantillons d'une Cosinus

La fonction de génération des échantillons d'une Cosinus (Figure 3.16) appelle la fonction `Sinus()` en décalant la phase par 90° pour générer les valeurs d'une fonction Cosinus utilisée comme la porteuse en quadrature de phase. Elle a une valeur entière positive à son entrée comme la fonction `Sinus()`.

```
int Cosinus(int t)
{
     return( Sinus(t + Taille_Table_Sinus) );
}
```

Figure 3.16 : Fonction de génération des échantillons d'une Cosinus (`Cosinus()`)

3.4.2.9 Fonction de Modulation 16-QAM

La fonction de modulation 16-QAM (Figure 3.17) génère le signal modulé 16-QAM en utilisant les composantes en phase et en quadrature de phase du symbole émis et en appelant les fonctions `Cosinus() et Sinus()` selon l'équation (2.5).

```
int Modulation(int Iechantillon, int Qechantillon, int phase)
{
     long resultat = (short)Iechantillon * (short)Cosinus(phase);
     resultat -= (short)Qechantillon * (short)Sinus(phase);
     resultat >>= 14;
     return((int)resultat);
}
```

Figure 3.17 : Fonction de Modulation 16-QAM (`Modulation()`)

3.4.2.10 Fonction d'émission

La fonction d'émission (Figure 3.18) génère les échantillons du signal de sortie du modulateur en appelant la fonction `Modulation()`. Les résultats sont stockés dans le buffer de sortie `modem.bufferSortie[]`.

```
void Transmitter(int indexSymbole, int *buffersortie)
{
  int Iamplitude;          // Amplitude de la composante en phase
  int Qamplitude;     // Amplitude de la composante en quadrature
  int n;              // de phase.
  Iamplitude = modem.pointsConstellation[indexSymbole].I;
  Qamplitude = modem.pointsConstellation[indexSymbole].Q;
  for (n = 0; n < Echantillons_Par_Symbole; n++ )
  { buffersortie[n] = Modulation(Iamplitude,
    Qamplitude,modem.phase);modem.phase+=modem.frequencePorteuse; }
}
```

Figure 3.18 : Fonction d'émission (`Transmitter()`)

3.4.2.11 Fonction principale d'émission

La fonction principale de l'émetteur (Figure 3.19) réalise tous les blocs fonctionnels de l'émetteur pour générer les échantillons du signal émis en appelant les fonctions nécessaires décrites précédemment dans l'ordre suivant : `Initialisation()`, `LireDonnees()`, `Scrambler()`, `Codeur()`, `GenererSymboles()`, `LireConstellation()` et enfin `Transmitter()`. La fonction `Modulation()` est appelée par `Transmitter()` et les fonctions `Cosinus()` et `Sinus()` sont appelées par la fonction `Modulation()`.

```
void main(void)
 {
      int i;
      Initialisation();//Initialiser les paramètres globaux du modem
      LireDonnees();    // Lire les données à transmettre
      Scrambler();      // Rendre les bits d'entrée aléatoire
      Codeur();         // Codage des bits générés par le Scrambler
      GenererSymboles();    // Génération des symboles à transmettre
     LireConstellation();  // Obtenir les composantes en I et en Q
  for( i = 0; i < Nombre_Symboles; i++ )
  {
    Transmitter(i,(modem.bufferSortie[i*Echantillons_Par_Symbole]));
   }
 }
```

Figure 3.19 : Fonction principale d'émission (`main()`)

3.4.3 Etage de réception

L'organigramme de l'étage de réception est illustré sur la figure 3.20.

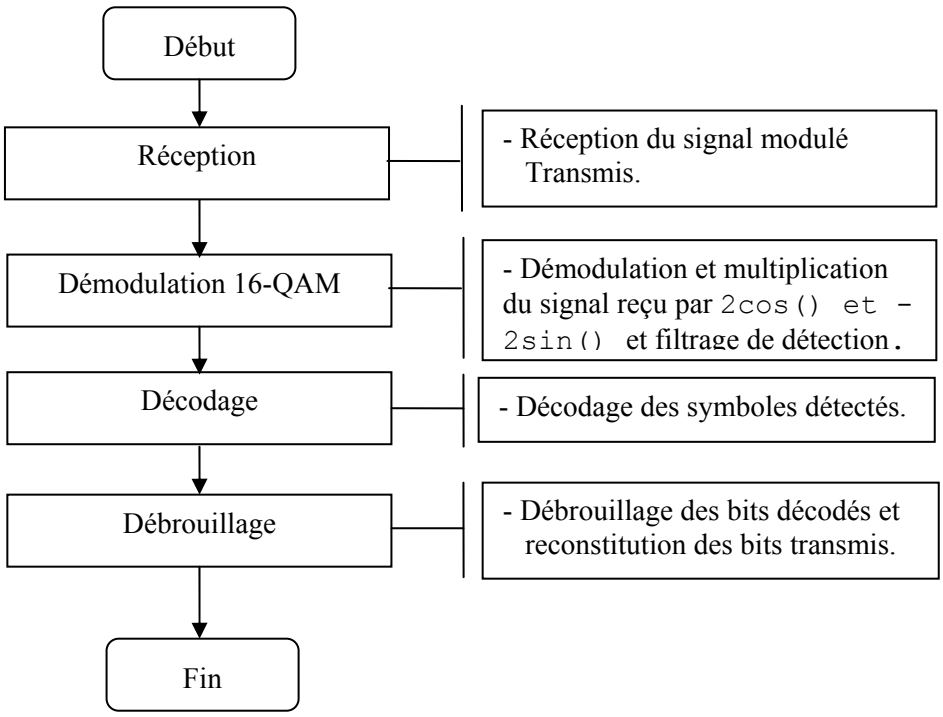


Figure 3.20 : Organigramme du récepteur du modem

L'implémentation du récepteur est faite de la même manière que l'émetteur. La fonction principale du récepteur est représentée sur la figure 3.26. Les fonctions implémentées au récepteur sont :

3.4.3.1 Fonction de réception

La fonction de réception (Figure 3.21) reçoit les échantillons du signal modulé et multiplie le signal reçu par `2*cosinus et -2*sinus` selon les équations (2.20) et (2.21) pour montrer les signaux passe bas (les composantes en phase et en quadrature de phase du signal modulé) à reconstituer et d'autres composantes hautes fréquences qui peuvent être éliminées en utilisant des filtres passe bas comme les filtres moyenneurs utilisés par la suite. Les échantillons résultants seront stockés dans les buffers `modem.bufferR1[]` et `modem.bufferR2[]`.

```
void Receiver(int indexSymbole)
 {
     int n;
     for (n = 0; n < Echantillons_Par_Symbole; n++ )
     {
     int echantillon =
    modem.bufferSortie[n+indexSymbole*Echantillons_Par_Symbole];

     long r1 = (short)echantillon * (short)(2*Cosinus(modem.phase));
     long r2 = (short)echantillon * (short)(-2*Sinus(modem.phase));
     r1 >>= 14;r2 >>= 14;

   modem.bufferR1[n+indexSymbole*Echantillons_Par_Symbole]          =
(short)r1;
   modem.bufferR2[n+indexSymbole*Echantillons_Par_Symbole]          =
(short)r2;

   // Incrementer la phase pour l'échantillon suivant
      modem.phase += modem.frequencePorteuse;
      }
 }
```

Figure 3.21 : Fonction de réception du signal modulé (`Receiver()`)

3.4.3.2 Fonction de filtrage de détection

La fonction de filtrage de détection (Figure 3.22) filtre les signaux produits par la fonction précédente (`Receiver()`) en utilisant un filtre moyenneur de taille 8 et génère deux signaux passe-bas. Ces derniers sont les composantes en phase et en quadrature de phase du signal modulé. Ils seront sauvegardés dans les buffers `modem.signalFiltreI[]` et `modem.signalFiltreQ[]`.

```
void FiltreDetection()
{
     int k, n;
     long ar[Nombre_Symboles*Echantillons_Par_Symbole];
     long br[Nombre_Symboles*Echantillons_Par_Symbole];
     for (k=0;k<Nombre_Symboles*Echantillons_Par_Symbole;k++)
     {
         ar[k]=0;br[k]=0;
         for (n=k;n<(k+K);n++)
         {
            // Composante en phase filtrée
            ar[k] += (short)f[n-k]*(short)modem.bufferR1[n];
            // Composante en quadrature filtrée
            br[k] += (short)f[n-k]*(short)modem.bufferR2[n];
          }
          modem.signalFiltreI[k]=(short)(ar[k] >>= 14);
          modem.signalFiltreQ[k]=(short)(br[k] >>= 14);
     }
}
```

Figure 3.22 : Fonction de filtrage de détection (`FiltreDetection()`)

3.4.3.3 Fonction de détection des symboles émis

La fonction de détection des symboles émis (Figure 3.23) reconstitue à partir des signaux filtrés passe-bas les coordonnées des symboles dans la constellation. Ces coordonnées seront rangées dans les buffers `modem.a[]` et `modem.b[]`. Les valeurs des symboles reçus sont rangés dans le buffer `modem.symbolesR[]`.

```
void ReconstituerSymboles(void)
{
/* Le contenu de cette fonction est donné dans l'annexe 1 (Page 74)*/
}
```

Figure 3.23 : Fonction de détection des symboles émis (`ReconstituerSymboles()`)

3.4.3.4 Fonction de décodage

La fonction de décodage (Figure 3.24) décode les bits reçus selon l'opération inverse du codage différentiel (voir codage différentiel (2.3.2.2) et Tableau 2.1). Les bits d'entrée du décodeur qui correspondent aux symboles détectés sont stockés dans le buffer `modem.eDec[]` et les bits décodés seront stockés dans le buffer `modem.sDec[]`.

```
void Decodeur(void)
{
/* Le corps de cette fonction est donné dans l'annexe 1(Page 75) */
}
```

Figure 3.24 : Fonction de décodage (`Decodeur()`)

3.4.3.5 Fonction de débrouillage

La fonction de débrouillage (Figure 3.25) fait l'opération de débrouillage pour reconstituer les bits émis (les données transmises) (voir le débrouilleur de la norme V.22bis en 2.4.4). Les bits reconstitués seront stockés dans le buffer `modem.sDescr[]`.

```
void Descrambler(void)
{
   int n;

     for (n=0;n<=17;n++)
     {    modem.sDescr[n] = modem.sDec[n];    }

     for (n=4*Nombre_Symboles-1;n>=17;n--)
     {
modem.sDescr[n]=(modem.sDec[n]+modem.sDec[n-14]+modem.sDec[n-17])%2;
     }
}
```

Figure 3.25 : Fonction de débrouillage (`Descrambler()`)

3.4.3.6 Fonction principale de réception

La fonction principale du récepteur (Figure 3.26) réalise tous les blocs fonctionnels du récepteur en commençant par l'acquisition des échantillons du signal reçu jusqu'à la reconstitution de la séquence des bits transmise (les données) en appelant les fonctions nécessaires décrites pour le récepteur dans l'ordre suivant : `Receiver(i)`, `FiltreDetection()`,`ReconstituerSymboles()`, `Decodeur()` et enfin `Descrambler()`.

```
void main(void)
{
   int i;

   for( i = 0; i < Nombre_Symboles; i++ )
     {
           Receiver(i);
     }

     FiltreDetection();       // Application des filtres de détection
     ReconstituerSymboles();// Reconstitution du symboles émis après
                              le codage différentiel
     Decodeur();      // Application du décodeur
     Descrambler();   // Application du déscrambler
}
```

Figure 3.26 : Fonction principale de réception (`main()`)

3.5 Implémentation du modem sur une carte DSK TMS320VC5402

Pour l'implémentation du modem sur une carte DSK TMS320VC5402, nous devons ajouter aux fonctions principales d'émission (Figure 3.19) et de réception (Figure 3.26) deux fonctions (`Init()` et `delay()`) permettant la configuration des paramètres du DSP et des convertisseurs analogique numérique et numérique analogique. Les fichiers entêtes qui contiennent les fonctions nécessaires à cette configuration sont inclus.

Les paramètres de configuration sont :

- La fréquence du CPU : 40 MHz (fonctionnement nominal suffisant du DSP).
- La fréquence d'échantillonnage des convertisseurs ADC et DAC : 9142 Hz (imposée par la carte DSK proche de 9600 Hz, fréquence d'échantillonnage de notre modem).
- Le gain d'entrée de ADC : 6dB (réglage par défaut).
- Le gain de sortie de DAC : - 6dB (réglage par défaut).

On doit ajouter une fonction de retard (Figure 3.27) pour occuper le processeur entre deux échantillons successifs afin d'ajuster la fréquence désirée du signal de sortie. La fonction d'initialisation (Figure 3.29) contient une boucle de commandes permettant d'allumer et d'éteindre les trois LEDs de la carte pour signaler le commencement d'exécution des programmes de l'implémentation.

```
void delay(s16 period)
{
    int i, j;

    for(i=0; i<period; i++)
    {
       for(j=0; j<period>>1; j++);
    }
}
```

Figure 3.27 : Fonction de retard (`delay()`)

Figure 3.28 : Carte DSK utilisée pour l'implémentation du Modem

L'image de la figure 3.28 montre la carte DSK TMS320VC5402 utilisée au laboratoire pour l'implémentation du modem.

```
void Init()
{
  int cnt=2;
  brd_set_cpu_freq(40);

    /* Allumer les LEDs puis les éteindre
       pour indiquer le départ d'exécution */
    while ( cnt-- )
    {
         brd_led_toggle(BRD_LED0);
         delay(600);

         brd_led_toggle(BRD_LED1);
         delay(600);

         brd_led_toggle(BRD_LED2);
         delay(600);
    }

   // Ouvrir le Codec
   hHandset = codec_open(HANDSET_CODEC);        // Handle du Codec

   // Régler les parameters du Codec
   codec_dac_mode(hHandset, CODEC_DAC_16BIT); // DAC en mode 16-bit
   codec_adc_mode(hHandset, CODEC_ADC_16BIT); // ADC en mode 16-bit
   codec_ain_gain(hHandset, CODEC_AIN_6dB);//gain d'entrée ADC 6dB
   codec_aout_gain(hHandset, CODEC_AOUT_MINUS_6dB);// gain de
                                                    sortie DAC -6dB
   codec_sample_rate(hHandset,SR_9142);
}
```

Figure 3.29 : Fonction d'initialisation du CPU et du Codec

3.5.1 Implémentation de l'émetteur

L'implémentation de l'émetteur du modem sur la carte DSK consiste à :

- Modifier la fonction principale d'émission pour la rendre comme représentée sur la figure 3.30.
- Transférer les échantillons du signal modulé transmis à partir du buffer `modem.bufferSortie[]` au registre de sortie (d'émission) du Codec.
- Retarder (fonction (`delay()`) les échantillons à transmettre dans le registre de sortie (d'émission) du Codec pour pouvoir visualiser le signal transmis sortis du Codec sur un oscilloscope en fréquence désirée.

```
void main(void)
{
     int i;

     Init();// Régler la fréquence du CPU et les paramètres du Codec
     Initialisation();//Initialiser les paramètres globales du modem
     LireDonnees();        // Lire les données à transmettre
     Scrambler();          // Rendre les bits d'entrée aléatoire
     Codeur();            // Codage des bits générés par le Scrambler
     GenererSymboles();    // Génération des symboles à transmettre
    LireConstellation(); // Obtenir les composantes en phase et en
                            quadrature de phase.

 for( i = 0; i < Nombre_Symboles; i++ )
 {
  Transmitter(i,&(modem.bufferSortie[i*Echantillons_Par_Symbole]));
 }

 while(1)
 {
    for( i = 0; i < Nombre_Symboles*Echantillons_Par_Symbole; i++ )
        {
          data = modem.bufferSortie[i];
          *(volatile u16*)DXR1_ADDR(HANDSET_CODEC) = data;
          delay(10);
        }
    }
}
```

Figure 3.30 : Fonction principale d'émission implantée sur la carte DSK (`main()`)

3.5.2 Implémentation du récepteur

L'implémentation du récepteur du modem sur la carte DSK consiste à :

- Modifier la fonction principale de la réception pour la rendre comme représentée sur la figure 3.31.
- Stocker les échantillons reçus par le registre d'entrée (de réception) du Codec dans le buffer `modem.bufferEntree[]`.
- Lancer les fonctions de démodulation et de reconstitution des données transmises.

```
void main(void)
{
     int i;
     Initialisation();//Initialiser les paramètres globales du modem
     Init();// Régler la fréquence du CPU et les paramètres du Codec

    while(1)
    {
       for (i=0;i<Echantillons_Par_Symbole*Nombre_Symboles;i++)
      {
         // Wait for sample from handset

         while (!MCBSP_RRDY(HANDSET_CODEC)) {};
         // Read sample
         data = *(volatile u16*)DRR1_ADDR(HANDSET_CODEC);
         modem.bufferEntree[i]= data;
         delay(5);
        }

        for( i = 0; i < Nombre_Symboles; i++ )
         {
              Receiver(i);
         }

         FiltreDetection(); // Application des filtres de détection
         ReconstituerSymboles();// Reconstitution des symboles émis
                                // aprés le codage différentiel
         Decodeur();      // Application du décodeur
         Descrambler();   // Application du déscrambler
       }
}
```

Figure 3.31 : Fonction principale de réception implantée sur la carte DSK (`main()`)

3.6 Faisabilité et limites d'implémentation en temps réel du modem 16-QAM sur DSP TMS320VC5402

- **Définition du temps réel**

On dit que le traitement est fait en temps réel si le temps de traitement est inférieur à la période d'échantillonnage (Figure 3.32). Si le traitement est fait par bloc (par trame), il faut que le temps de traitement d'une trame soit inférieur au temps d'acquisition d'une nouvelle trame (Figure 3.33).

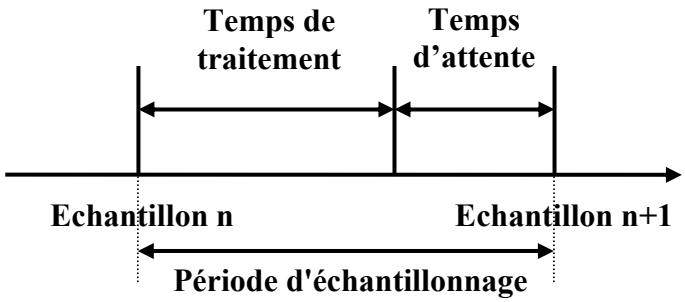


Figure 3.32 : Notion du temps réel (traitement par échantillon)

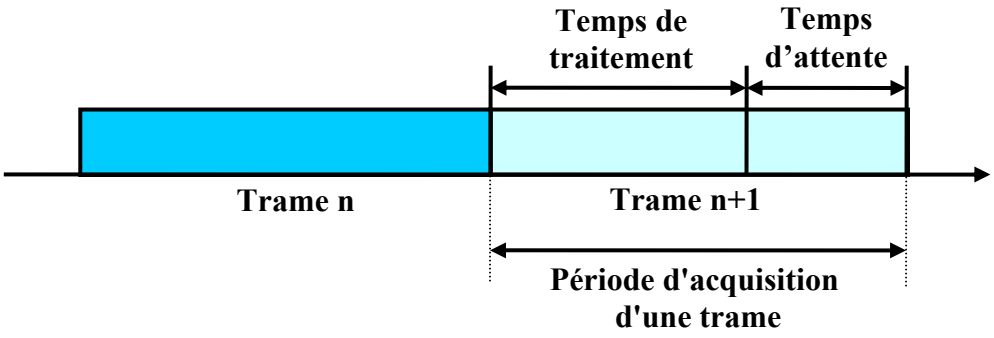


Figure 3.33 : Notion du temps réel (traitement par trame)

- **L'implémentation en temps réel et l'évaluation du temps d'exécution**

Pour implémenter des applications DSP en temps réel, il faut choisir un *DSP* capable d'exécuter les programmes de traitement en temps réel. Dans notre application (l'implémentation d'un Modem 16-QAM sur TMS320VC5402), la fréquence d'échantillonnage est de 9600 Hz, le traitement est fait par bloc (trame de 512 échantillons qui correspond à 8 octets de données, 8 échantillons par bit) ce qui correspond à un intervalle de 53,33333 ms entre deux trames consécutives. Cet intervalle est le temps tolérable maximal pour une opération d'émission ou de réception en temps réel d'un bloc de 8 octets. Cet intervalle correspond à 5333333 cycles d'horloge sur un DSP TMS320VC5402(cycle =10 ns). Dans le but d'exécuter les tâches de traitement dans l'intervalle indiqué, il est nécessaire de réduire et optimiser la durée d'exécution des programmes d'émission et de réception. Pour ce faire, il est intéressant de connaître quelles sont les parties du programme qui consomment le plus de temps CPU. On optimise alors en priorité ces parties du programme.

Les outils de *Profiling* de *Code Composer* aident à analyser les programmes et permettent d'obtenir des statistiques d'exécution en déterminant combien l'exécution des programmes passe en moyenne de cycles CPU (10 ns pour le temps d'un cycle du DSP TMS320VC5402).

Le tableau 3.1 présente les temps d'exécution des programmes d'émetteur et du récepteur du modem 16-QAM pour des différents niveaux d'optimisation du Compilateur.

	Temps d'exécution du programme d'émetteur (en cycles d'horloge du TMS320VC5402)	Temps d'exécution du programme de récepteur (en cycles d'horloge du TMS320VC5402)
Aucun (None)	11346	4431
Niveau Register (-o0)	54074	13855
Niveau Local (-o1)	35859	41925
Niveau Function (-o2)	12053	52385
Niveau File (-o3)	61205	40609

Tableau 3.1 : Statistiques d'exécution

Le temps d'exécution des programmes d'émetteur et du récepteur est largement inférieur au temps maximal tolérable qui est de 5333333 cycles environ. Donc, le traitement se fait en temps réel. Le meilleur temps d'exécution est enregistré pour le niveau d'optimisation Aucun(None), donc on choisit l'optimisation (None) pour l'implémentation des programmes d'émetteur et de récepteur sur le DSP TMS320VC5402.

3.7 Conclusion

Nous avons présenté dans ce chapitre les implémentations logicielle et matérielle du modem 16-QAM. Les résultats obtenus par simulation de notre modem sur Code Composer et par son implantation sur la carte DSK TMS320VC5402 seront discutés dans le chapitre suivant.

Chapitre 4

Résultats et Interprétations

4.1 Introduction

Après avoir implémenté notre modem sur le simulateur Code Composer et sur la carte DSK TMS320VC5402. Nous avons procédé à des tests sur ce simulateur et cette carte.

4.2 Génération des porteuses en quadrature de phase

Nous avons commencé les tests par la génération des porteuses en quadrature de phase utilisées pour la modulation 16-QAM. La fréquence des porteuses selon la norme V.22bis est égale à : $f_c = 1200$ Hz.

Donc, on va prendre une fréquence d'échantillonnage qui vérifie largement le théorème de Shannon : $f_s = 8 f_c = 9600$ Hz.

La taille de la table de consultation utilisée est de $N = 512$. Elle représente une période de la sinusoïde à générer.

L'Offset Δ est calculée par : $\Delta = \frac{N f_c}{f_s} = \frac{512 \times 1200}{9600} = 64$.

La symétrie de la sinusoïde permet de prendre seulement le quart de cette table, c'est-à-dire le 1/4 de la période de la sinusoïde à générer. Les valeurs de la sinusoïde sont représentées en format Q_{15} (la partie entière sur 1 bit (bit de signe) et la partie fractionnaire sur 15 bits).

Le graphe d'un quart de la table est représenté sur la figure4.1. A partir de cette table, 16 périodes des deux porteuses en quadrature sont générées comme représentées sur la figure 4.2.

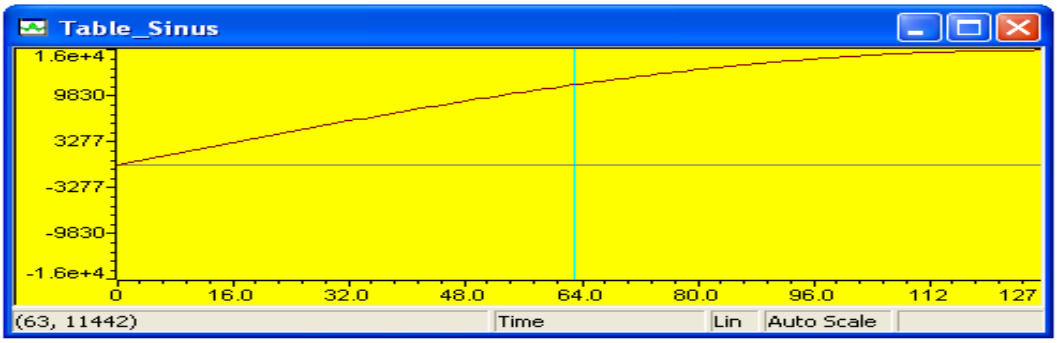


Figure 4.1 : Représentation graphique de la table `Table_Sinus`

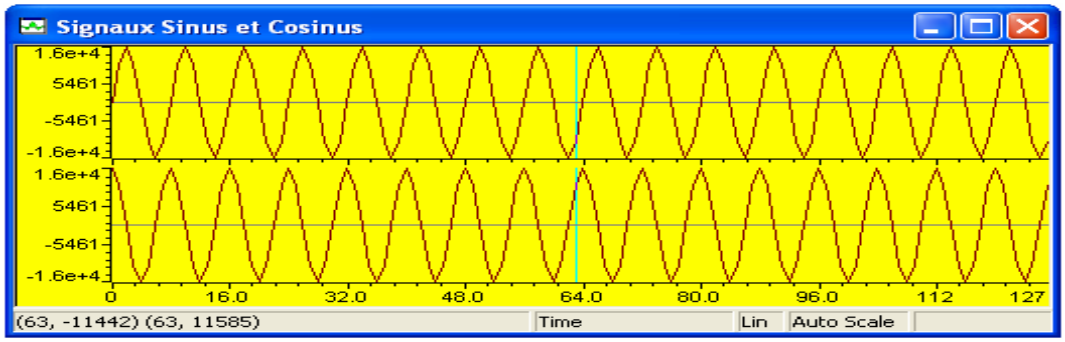


Figure 4.2 : Représentation graphique des 16 périodes des porteuses en quadrature de phase

Ces deux porteuses en quadrature sont utilisées pour la modulation et pour la démodulation aux niveaux d'émetteur et de récepteur du modem.

4.3 Test de l'implémentation de l'émetteur du modem avec Code Composer

L'environnement de développement intégré Code composer de Texas Instruments permet de tester et de débogger une application mono ou multiprocesseur en simulation ou en temps réel sur une cible matérielle. La cible matérielle utilisée dans notre implémentation est le kit de développement DSK TMS320VC5402.

Les étapes suivies pour le test de l'émetteur en simulation avec Code Composer sont :

1. Génération d'un signal binaire de 64 bits : Pour la simulation et le test, un signal binaire de 64 bits est généré contenant en alternance 4 bits des 1 et 4 bits des 0 comme représentés sur la figure 4.3. Ce choix est justifié par le test de l'efficacité de l'embrouilleur.

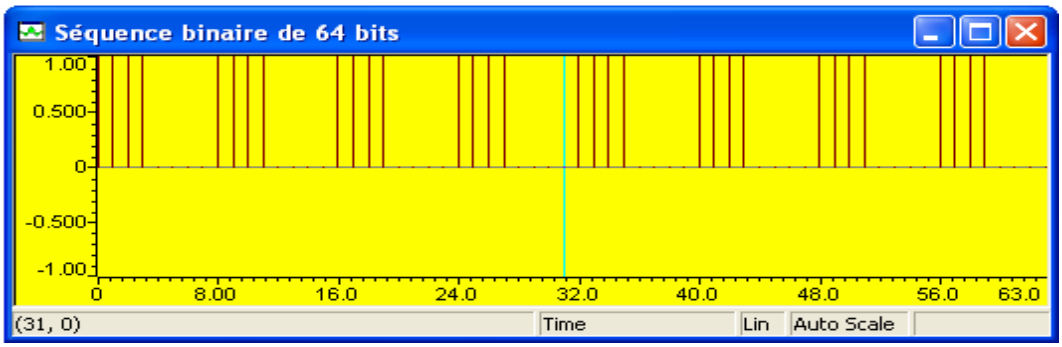


Figure 4.3 : Séquence binaire d'entrée de 64 bits (4 bits des 1 et 4 bits des 0 en alternance)

2. Rendre les bits d'entrée aléatoires en utilisant l'embrouilleur défini en (2.3.2.1). On remarque d'après la figure 4.4 que les 17 premiers bits d'entrée sont inchangés à la sortie de l'embrouilleur puisque l'embrouillage commence au $18^{ème}$ bit (voir l'équation

de l'embrouillage 2.10). Les autres bits sont devenus aléatoires après cette opération d'embrouillage.

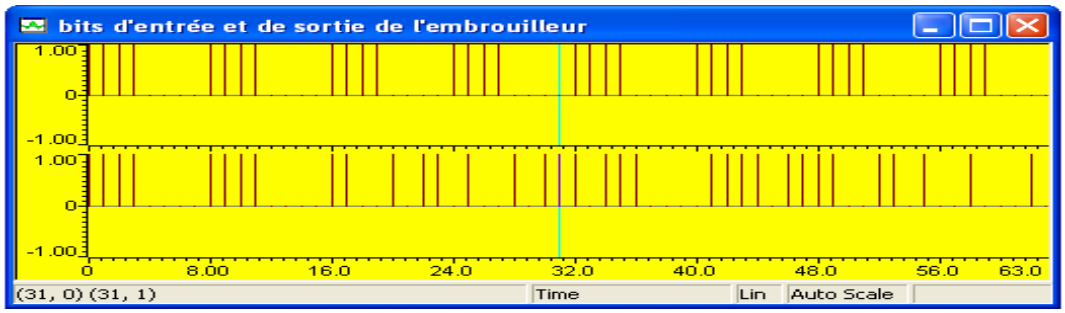


Figure 4.4 : Séquence d'entrée et de sortie de l'embrouilleur

3. Codage des bits générés par l'embrouilleur par l'application du codeur différentiel. La figure 4.5 montre la séquence d'entrée du codeur différentiel (en haut) non codée et la séquence de sortie codée par le codeur différentiel.

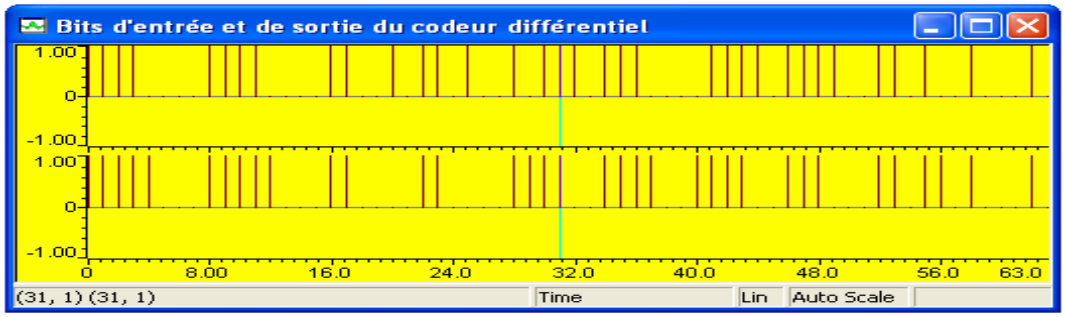


Figure 4.5 : Séquence d'entrée et de sortie du codeur différentiel

4. Mapping : l'opération de mapping consiste à générer les deux composantes en quadrature de phase `a` et `b` à partir des bits codés de la sortie du codeur différentiel. Pour faciliter la programmation, chaque 4 bits de la sortie du codeur sont représentés par un nombre décimal entre 0 et 15 qui correspond à la même valeur des 4 bits en binaire. La figure 4.6 représente les 16 nombres décimaux correspondants aux 64 bits de la sortie du codeur différentiel.

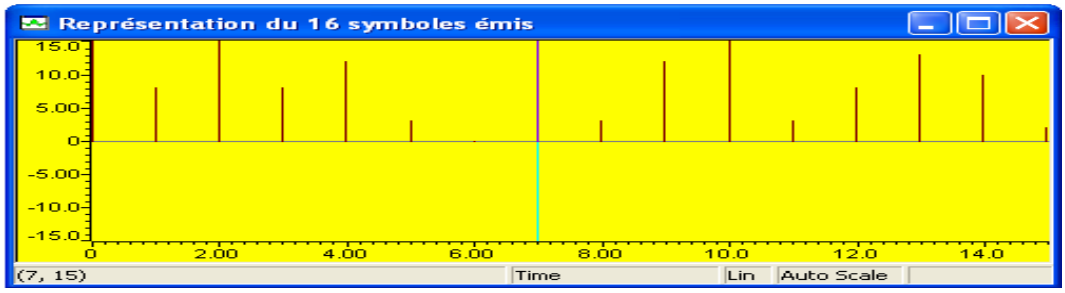


Figure 4.6 : Les 16 symboles émis.

Après la représentation en nombres décimaux des bits codés, les coordonnées `a` et `b` correspondantes sont prises à partir de la constellation 16-QAM (Figure 2.5).

5. Génération des porteuses en quadrature avec une fréquence $f_c = 1200$ Hz. La figure 4.7 montre la représentation fréquentielle des porteuses (à gauche) et la représentation temporelle (à droite). Dans la représentation fréquentielle les fréquences sont normalisées par rapport à la fréquence d'échantillonnage d'où la fréquence normalisée des porteuses est $f_N = 0.125$.

$f_N = \frac{f_c}{f_s} = 0.125$ où $f_c = 1200$ Hz et $f_s = 9600$ Hz.

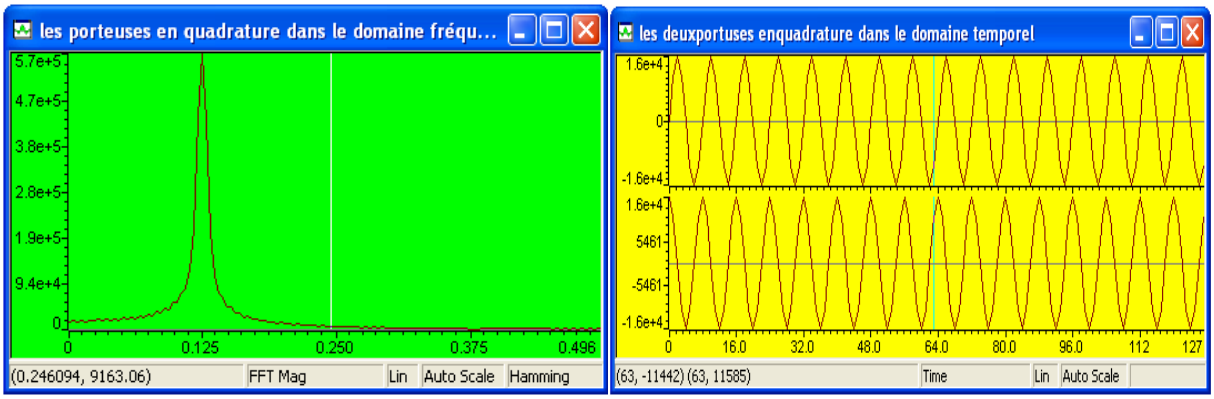


Figure 4.7 : Les porteuses en quadrature avec une fréquence normalisée $f_N = 0.125$ dans les domaines fréquentiel et temporel.

6. Modulation 16-QAM : cette modulation utilise un ensemble de 16 signaux ($a(t)$ et $b(t)$). Chaque signal module quatre périodes des porteuses en quadrature de phase. Ce choix est justifié par un débit des bits $R_d = 1200$ bps supporté par la norme V.22bis et une fréquence des porteuses $f_c = 1200$ Hz. Ces conditions impliquent un bit par période des porteuses ou un symbole par 4 périodes des porteuses. La figure 4.8 montre la représentation fréquentielle du signal modulé (en haut) et la représentation temporelle (en bas).

Nous remarquons sur la figure 4.8 que le spectre du signal modulé est situé autour de la fréquence porteuse normalisée ($f_N = 0.125$).

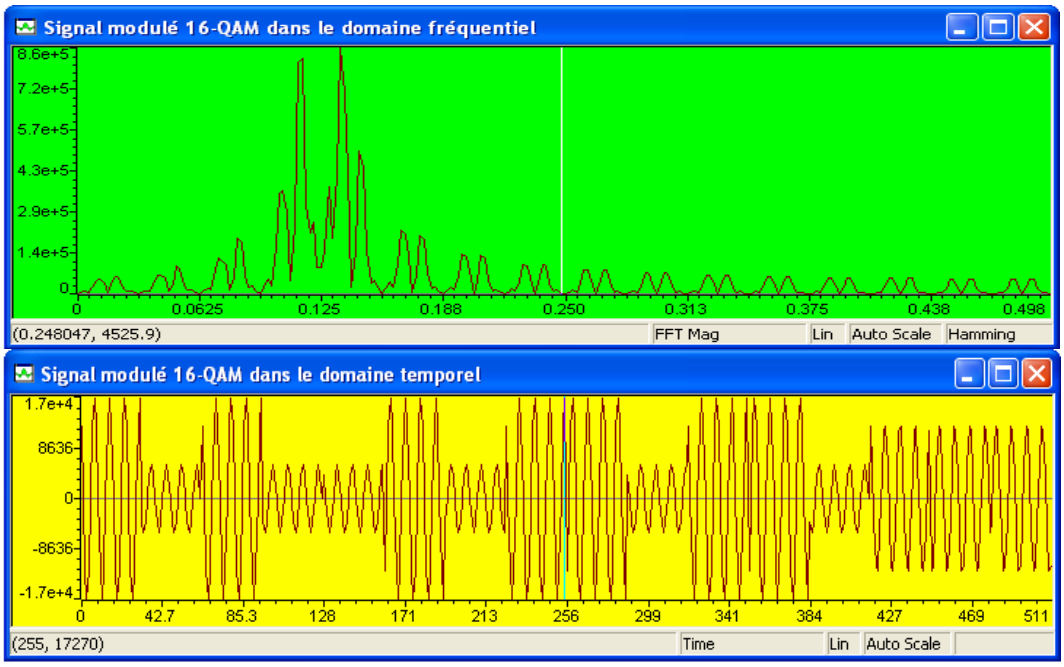


Figure 4.8 : Signal modulé 16-QAM dans les domaines fréquentiel et temporel

4.4 Test de l'implémentation du récepteur du modem avec Code Composer

Pour tester le récepteur du modem, des échantillons d'un signal modulé comme représenté sur la figure 4.8 (en bas) sont nécessaires. Donc les 512 échantillons du signal modulé émis par l'émetteur sont utilisés à cet effet. La détection des données transmises (les bits émis) est faite selon les étapes suivantes :

1. Multiplication du signal modulé 16-QAM reçu par les porteuses `2*cosinus()` et `-2*sinus()` pour obtenir les composantes basse et haute fréquences.

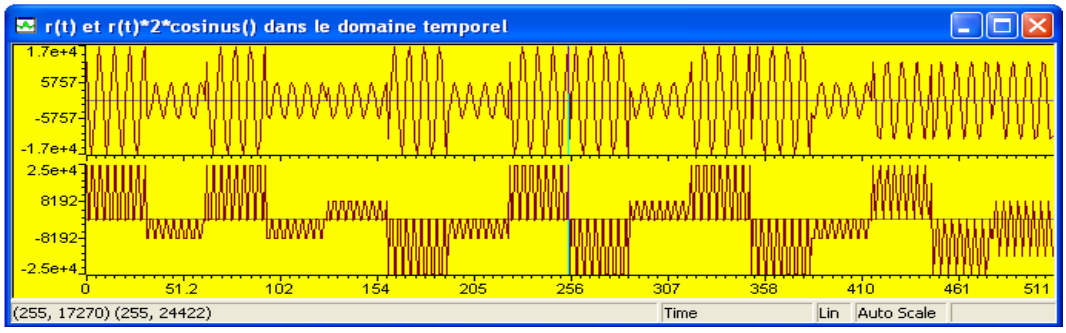


Figure 4.9 : Signal reçu (en haut) et signal reçu multiplié par `2*cosinus()` (en bas) dans le domaine temporel

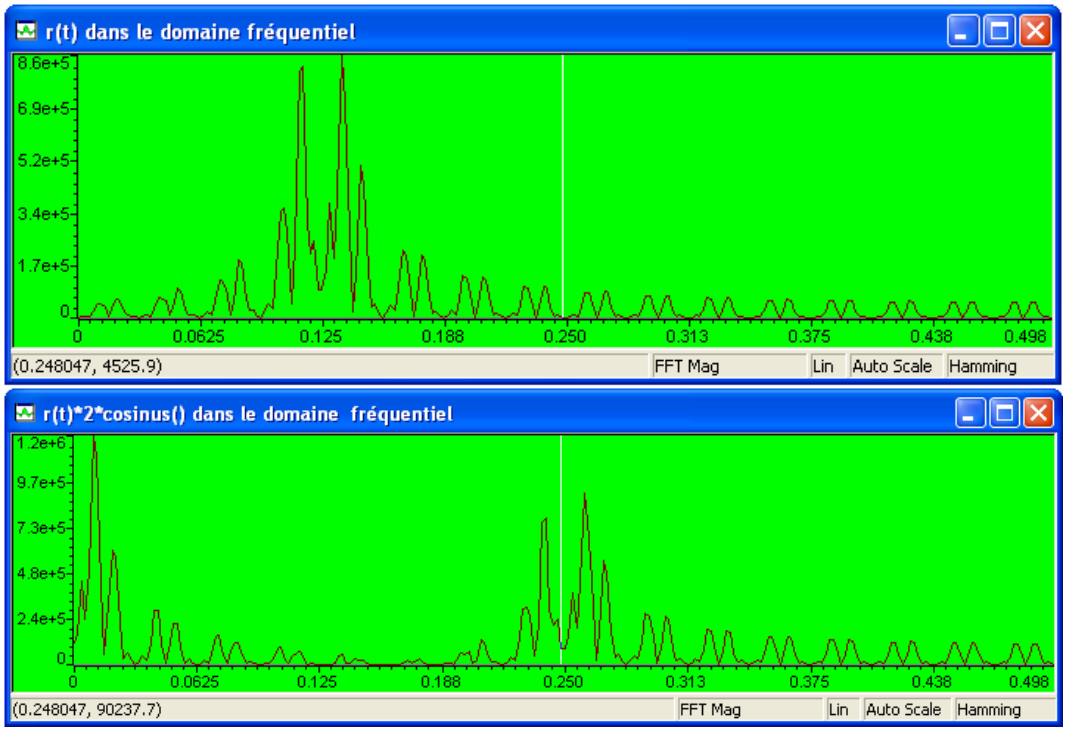


Figure 4.10 : Signal reçu (en haut) et signal reçu multiplié par `2*cosinus()` (en bas) dans le domaine fréquentiel

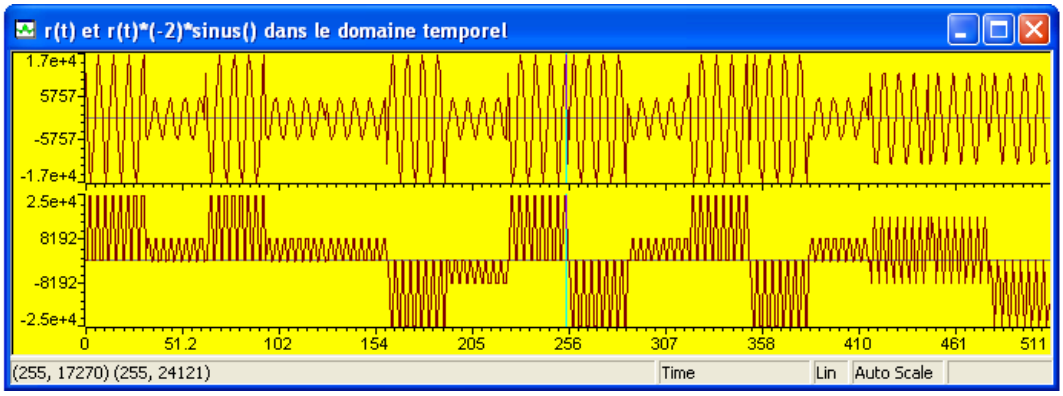


Figure 4.11 : Signal reçu (en haut) et signal reçu multiplié par `(-2)*sinus()` (en bas) dans le domaine temporel

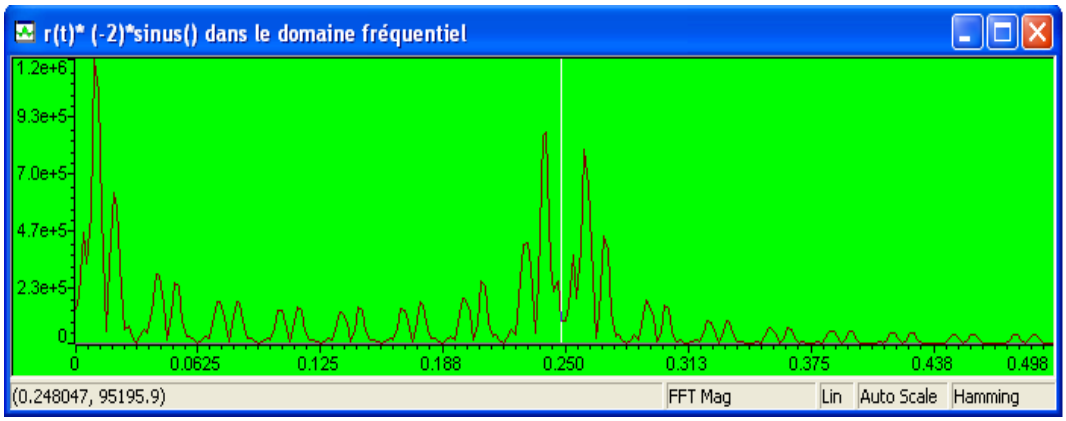


Figure 4.12 : Signal reçu multiplié par `(-2)*sinus()` dans le domaine fréquentiel

Les figures 4.9, 4.10, 4.11 et 4.12 montrent les signaux résultant de la multiplication du signal modulé reçu par les porteuses `2*cosinus()` et `(-2)*sinus()`. Ces signaux se composent de deux composantes fréquentielles, une composante basse fréquence utile représentant le signal message et l'autre haute fréquence (deux fois la fréquence porteuse) non désirée qui doit être éliminée lors de la récupération du signal transmis.

2. Filtrage des composantes basse-fréquences : Après filtrage passe-bas, les signaux messages sont obtenus comme les montrent les figures 4.13 et 4.14. Ces signaux constituaient les enveloppes des signaux obtenus par la multiplication du signal reçu par les porteuses `2*cosinus()` et `(-2)*sinus()` (Figures 4.9 et 4.11).

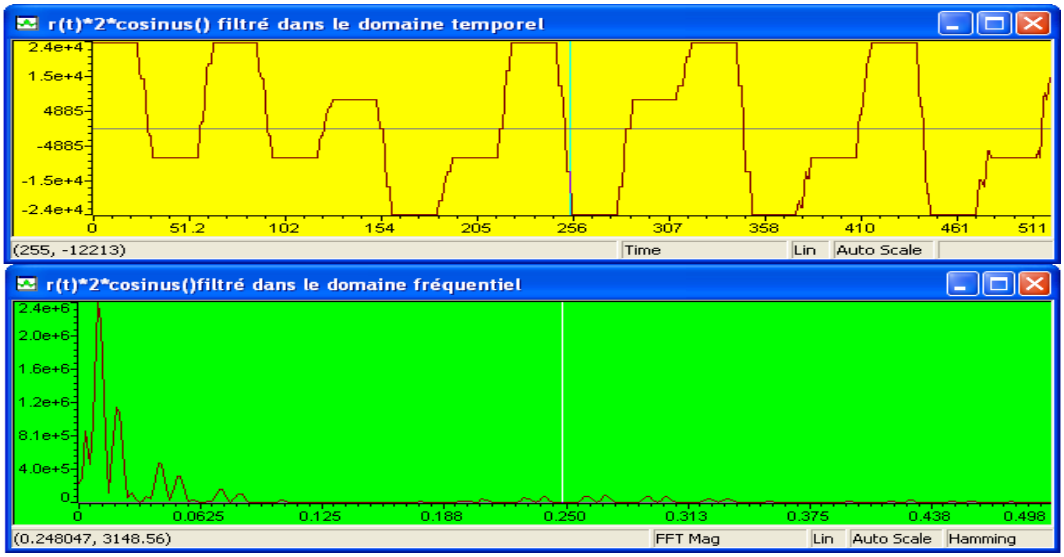


Figure 4.13 : Signal reçu multiplié par `2*cosinus()` et filtré par un filtre passe-bas dans les domaines temporel et fréquentiel

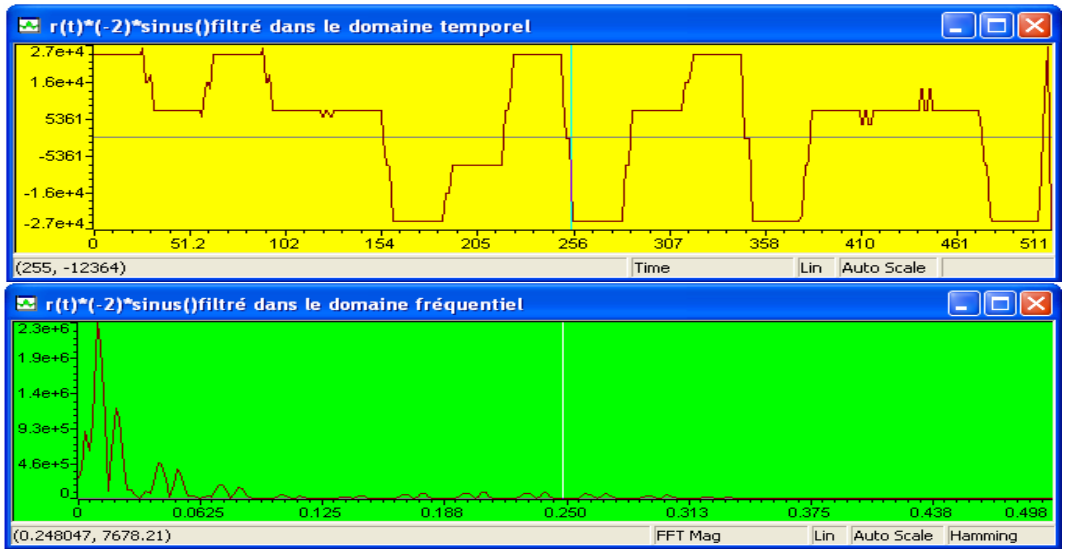


Figure 4.14 : Signal reçu multiplié par `(-2)*sinus()` et filtré par un filtre passe-bas dans les domaines temporel et fréquentiel

3. Détection des symboles et bits émis : A partir des signaux passe-bas filtrés, on reconstitue les symboles et bits transmis. Cette reconstitution est faite par deux décisions. La première décision consiste à diviser l'espace de la constellation en régions de décisions. Si l'échantillon filtré appartient à une région, la décision est faite en faveur du point de la constellation contenant dans cette région. Cette décision est appliquée aux deux composantes filtrées (la composante en phase et en quadrature de phase) séparément. La deuxième décision est une décision majoritaire, c'est-à-dire chaque 32 échantillons des signaux passe-bas filtrés sont testés, la décision est faite en faveur du symbole ayant le grand nombre d'apparition. Les 16 symboles détectés et les 64 bits correspondants sont représentés sur la figure 4.15.

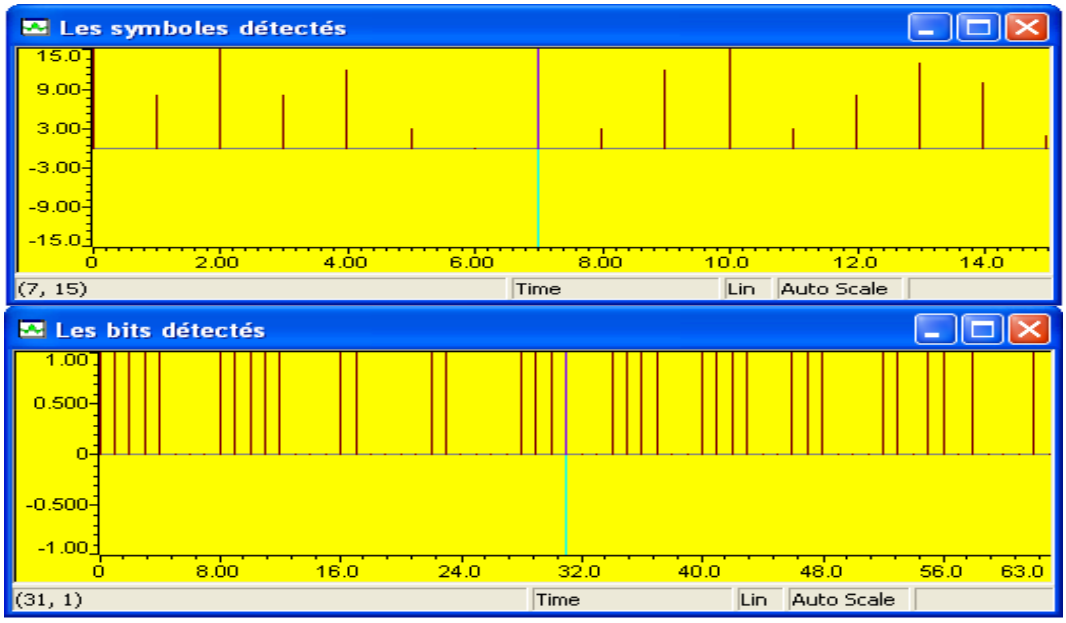


Figure 4.15 : Les symboles détectés et les bits correspondants

4. Décodage des bits détectés par l'application du décodeur différentiel. La figure 4.16 montre la séquence d'entrée (en haut) codée et la séquence de sortie décodée (en bas) par le décodeur différentiel.

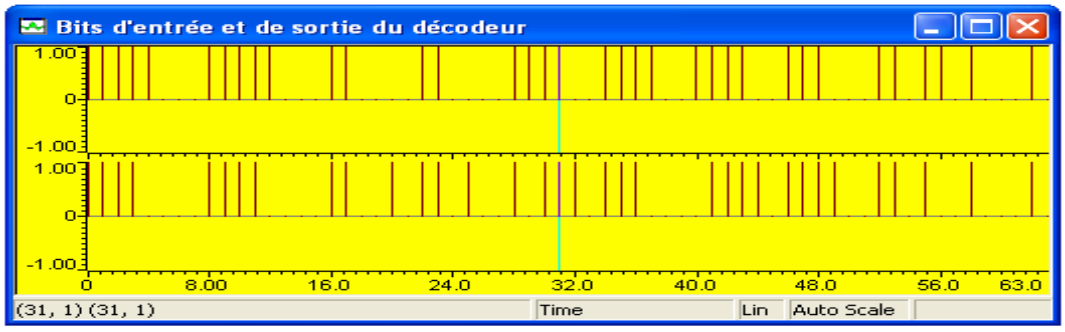


Figure 4.16 : Séquence d'entrée et de sortie du décodeur

5. Reconstitution des bits originaux (les bits transmis) en utilisant le débrouilleur. La figure 4.17 montre la séquence d'entrée (en haut) et la séquence de sortie du débrouilleur (en bas).

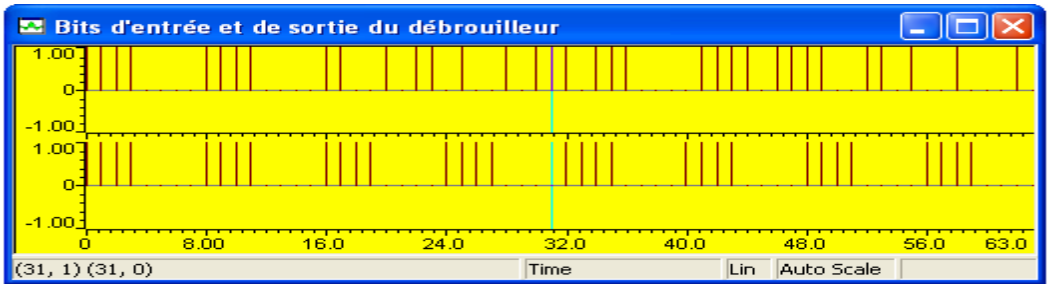


Figure 4.17 : Séquence d'entrée et de sortie du débrouilleur

6. Comparaison entre la séquence binaire originale et la séquence binaire reconstituée. La figure 4.18 montre la séquence des bits reconstituée par le récepteur (en haut) et la séquence des bits transmise (bits originaux) par l'émetteur (en bas).

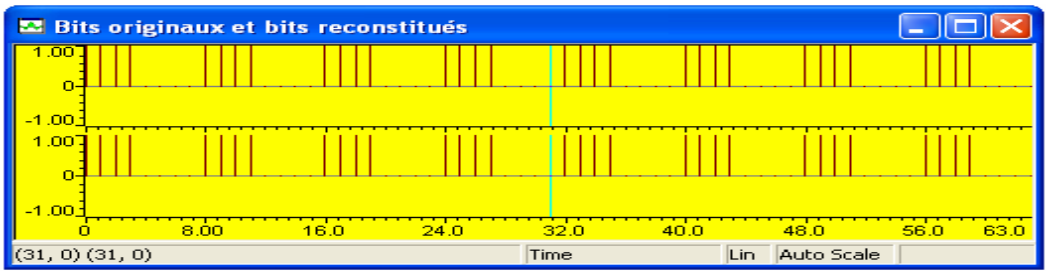


Figure 4.18 : Signal binaire original et signal binaire reconstitué

Ces deux séquences sont identiques. Cela montre des bons résultats du test de l'implémentation du modem avec Code Composer. Maintenant, on passe au test de l'implémentation du modem sur la carte DSK.

4.5 Test de l'implémentation du modem sur la carte DSK TMS320VC5402

Dans notre montage (figure 4.19), le test de l'implémentation du Modem 16-QAM a été réalisé par une carte DSK TMS320VC5402, un PC sur lequel le Simulateur Code Composer a été installé et un oscilloscope pour visualiser les signaux aux différents étages du modem. Dans un premier temps, nous avons testé l'implémentation de l'émetteur sur la carte DSK et nous avons visualisé le signal modulé transmis sur l'oscilloscope. Pour le test de l'implémentation du récepteur nous avons stocké les échantillons du signal modulé transmis par le montage d'émission dans un fichier puis nous l'avons rechargé sur la carte DSK pour reconstituer les données numériques transmises par l'émetteur. Nous avons également visualisé les signaux aux différents étages du récepteur sur l'oscilloscope.

4.5.1 Test de l'implémentation de l'émetteur sur la carte DSK TMS320VC5402

Pour le test de l'implémentation de l'émetteur du modem sur la carte DSK, nous avons implanté les programmes de l'étage d'émission sur la carte. D'après les résultats obtenus, nous les constatons en concordance avec les résultats obtenus par simulation. La figure 4.19 montre le montage utilisé au laboratoire pour le test de l'implémentation du modem.

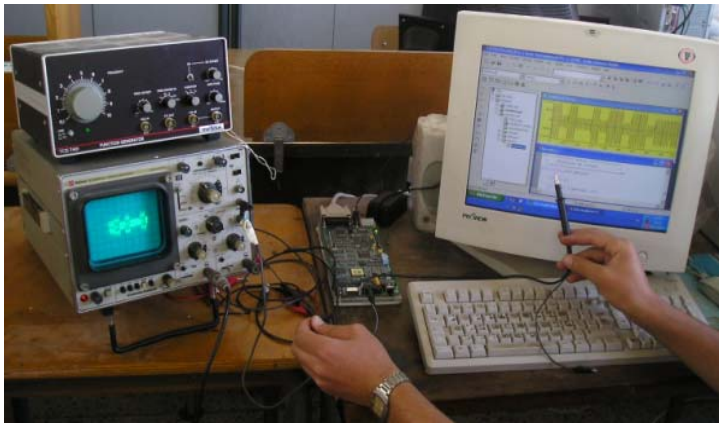


Figure 4.19 : Montage de test utilisé au laboratoire

Le signal modulé délivré par la sortie du convertisseur numérique analogique de la carte DSK est visualisé sur un oscilloscope. Les photos prises de ce signal par un appareil photo numérique à partir de l'écran de l'oscilloscope sont représentées sur la figure 4.20.

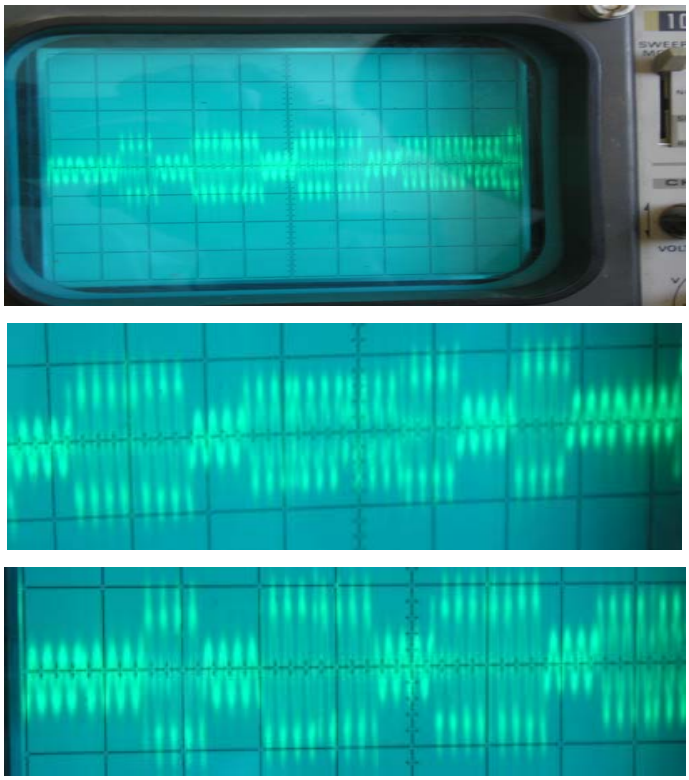


Figure 4.20 : Signal modulé 16-QAM généré par la carte DSK et visualisé sur un oscilloscope 0.5v/div et 5ms/div.

4.5.2 Test de l'implémentation du récepteur sur la carte DSK TMS320VC5402

Les résultats obtenus par implémentation du récepteur du modem sur la carte DSK après visualisation sur un oscilloscope sont présentés sur les figures suivantes :

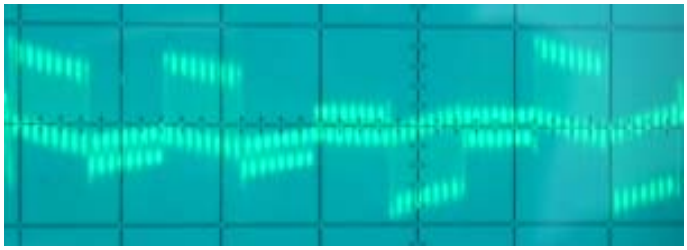


Figure 4.21 : Signal reçu multiplié par `2*cosinus()`
0.5v/div et 5ms/div.

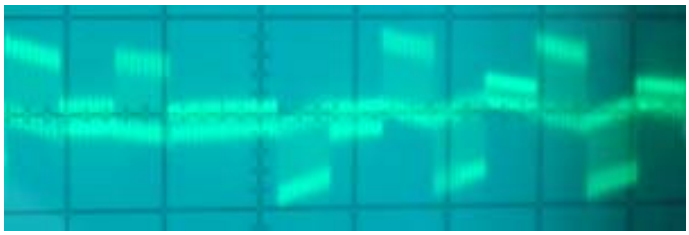


Figure 4.22 : Signal reçu multiplié par `(-2)*sinus()`
0.5v/div et 5ms/div.

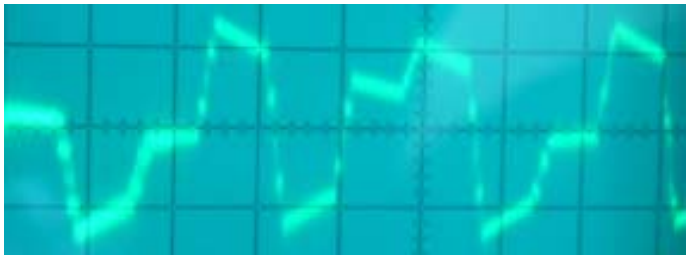


Figure 4.23 : Signal reçu multiplié par `2*cosinus()` et filtré par un filtre passe-bas
0.5v/div et 5ms/div.

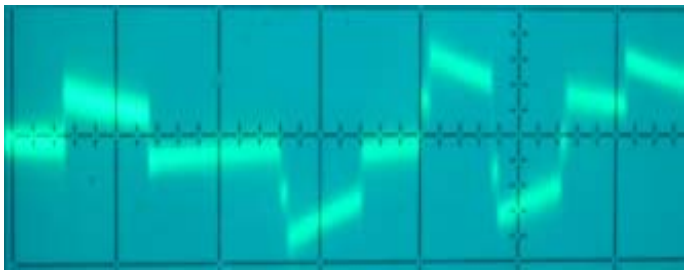


Figure 4.24 : Signal reçu multiplié par `(-2)*sinus()` et filtré par un filtre passe-bas
0.5v/div et 5ms/div.

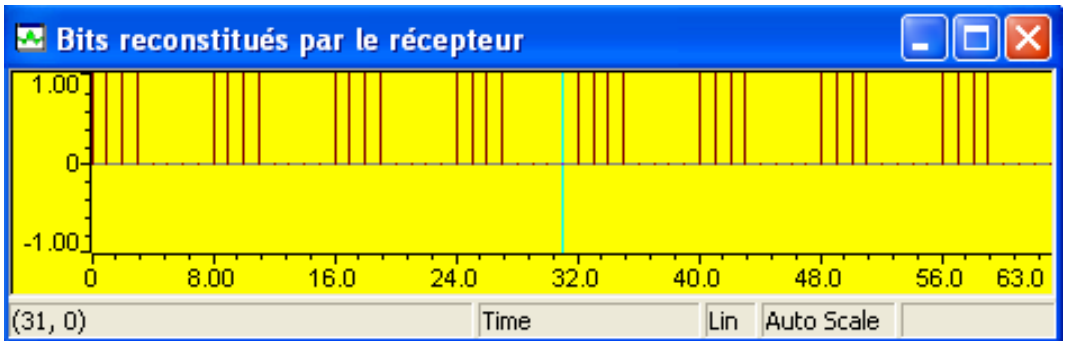


Figure 4.25 : Bits reconstitués par le récepteur du modem et stockés dans la mémoire du DSP

Les inclinaisons apparaîtrant aux nivaux des signaux sont dues par des défauts du matériel (oscilloscope). Nous remarquons que les résultats obtenus par implémentation du récepteur sur la carte DSK sont comparables à ceux obtenus par simulation.

4.6 Conclusion

L'objectif de ce chapitre a été la présentation des résultats d'implémentation du modem 16-QAM sur la Carte DSK TMS320VC5402. Pour atteindre cet objectif, nous avons commencé par tester l'implémentation du modem avec le Simulateur Code Composer pour valider les programmes. La deuxième étape était l'implantation des programmes sur la carte DSK qui a été validée à la fin du chapitre par des tests expérimentaux sur la carte. Ces tests ont donné des bons résultats. Nous constatons que les signaux obtenus par implémentation du modem sur la carte DSK sont en conformité avec les signaux obtenus par simulation sur Code Composer. Les bits transmis ont été reconstitués parfaitement (sans erreurs) au récepteur.

Conclusion générale

Tout au long de ce mémoire, nous avons exposé les différentes étapes de la conception et l'implémentation du modem 16-QAM pour la transmission de données sur une ligne téléphonique bifilaire. L'environnement de développement intégré Code Composer et le kit de développement DSK TMS320VC5402 ont été utilisés pour la simulation et l'implémentation en temps réel du modem. L'implémentation est faite en langage C, le compilateur C compatible ANSI de TI pour les DSP de la plate forme C5000 (les familles C54x et C55x).

Le modem est un dispositif sophistiqué contenant beaucoup de blocs fonctionnels qui doivent être correctement implémentés. L'interface des blocs fonctionnels doit être convenablement ajustée pour que la structure complète fonctionne correctement. Le calcul mathématique effectué par le modem est très complexe. Par utilisation du processeur TMS320VC5402, avec un temps de cycle de 10 ns, un multiplicateur on chip et un jeu d'instruction spécialisé pour le traitement numérique du signal, on a pu implémenter les blocs fonctionnels du modem avec succès.

L'implémentation matérielle du modem 16-QAM a nécessité dans un premier temps la familiarisation et la maîtrise de l'environnement de développement intégré Code Composer Studio. Compte tenu des caractéristiques du processeur TMS320VC5402, nous avons mis au point un ensemble de codes sources en C relatifs aux différents modules du Modem. Nous avons accordé un intérêt bien particulier à l'optimisation de nos codes sources pour mieux les adapter aux contraintes du système réel et réussir l'implémentation sur le DSP TMS320VC5402 (16 bit virgule fixe et 100 MHz de vitesse) que nous avons utilisé. Nous constatons que les résultats relevés par implémentation du Modem sur la carte DSK à la sortie des différents étages sont en conformité avec les résultats de simulation sur Code Composer.

Notre travail constitue une base d'implémentation des techniques numériques (utilisées dans les modems) sur le processeur de traitement du signal TMS320VC5402. Grâce à la flexibilité des programmes de l'implémentation, le travail peut être exploité pour ajouter d'autres blocs fonctionnels complémentaires qui peuvent être insérés en ajoutant leurs codes. D'autres modules supplémentaires pourrants être insérés tels que la correction d'erreurs, la compression de données, l'égalisation adaptative,..etc. Ces derniers permettront l'amélioration des performances de la transmission. La puissance offerte par le DSP TMS320VC5402 permet l'implémentation de modem à vitesse variable sur un seul DSP.

Bibliographie

[1] G. Baudoin et F. Virolleau, "Les DSP Famille TMS320C54x, Développement d'applications", Dunod, Paris, 2000.

[2] C. J. Buehler, J. Keating, H. Metz, C. J. Nuzman, et H. Sampath "V.34 Transmitter and Receiver Implementation on the TMS320C50 DSP", Application Report : SPRA159 , Texas Instruments, June 1997.

[3] P. Evans et A. Lovrich, "Implementation of an FSK Modem Using the TMS320C17", Application Report : SPRA080, Texas Instruments, 1997.

[4] H. Garreta, "Le langage C", Université de la Méditerranée, 2003.

[5] S. M. Kuo et B. H. Lee, "Real Time Digital Signal Processing", John Wiley & Sons, Ltd, Chichester, 2001.

[6] P. Sobieski et L. Vandendorpe, "Signaux de communications (ELEC 2880)", Université catholique de Louvain, Laboratoire de Télécommunications et de Télédétection. http://www.tele.ucl.ac.be/ELEC2880/tout_2880.pdf.

[7] S. A. Tretter, "Communication System Design Using DSP Algorithms With Laboratory Expériment for the TMS320C30", Plenum Press, New York, 1995.

[8] G. Troullinos, P. Ehlig, R. Chirayll, J. Bradley et D. Garcia, "Theory and Implementation of a Splitband Modem Using the TMS32010", Application Report : SPRA013, Texas Instruments, 1997.

[9] "Code Composer Studio Getting Started Guide", Literature Number : SPRU509C, Texas Instruments, November 2001.

[10] "Code Composer Studio IDE", Literature Number : SPRU405A, Texas Instruments, February 2001.

[11] "CPC5604, Optical Data Access Arrangement I.C", Specification ANDS-CPC5604-XX, Data Sheet, Clare, Inc, 2001.

[12] "TLC320AD50C/I, TLC320AD52C, Sigma-Delta Analog Interface Circuits With Master-Slave Function" Data Manual, Texas Instruments, 2000.

[13] "TMS320C54x Assembly Language Tools User's Guide", Literature Number : SPRU102E, Texas Instruments, June 2001.

[14] "TMS320C54x DSP Reference Set Volume 1 : CPU and Peripherals", Literature Number : SPRU131G, Texas Instruments, March 2001.

[15] "TMS320C54x DSP Reference Set Volume 2 : Mnemonic Instruction Set", Literature Number : SPRU172C, Texas Instruments, March 2001.

[16] "TMS320C54x Optimizing C/C++ Compiler User's Guide", Literature Number : SPRU103F, Texas Instruments, June 2001.

[17] "V.22bis Modem on Fixed-Point TMS320C2xx DSPs", Application Report : SPRA484, Texas Instruments, February 1999.

[18] "Guide d'achat des modems", http://alain.saramito.fr/index.htm.

[19] "Introduction and History of Modems", http://www.dementia.org/~julied/tele2100/intro.html

[20] "Le modem", http://www.commentcamarche.net/pc/modem.php3

[21] "Modem Standards", http://www.dementia.org/~julied/tele2100/standards.html

[22] "Modems Standards", http://www.fujitsu-siemens.co.uk/rl/servicesupport/techsupport/ Boards/Modems/ modemdiags/General-Info/Modem-Standards.htm

[23] "MODEMS", http://www2.rad.com/networks/1994/modems/modem.htm.

[24] J. F. L'HAIRE, "Modems", Université de Genève – SEINF, Septembre 1999, http://padina.info.umoncton.ca/elem-res/elem/modem.htm#Normes.

[25] TMS320VC5402 DSK Help, Code Composer Studio, Texas Instrument.

Annexe 1

```
/*=================================================================
  =                Fichier d'entête "data.h"                      =
  =================================================================*/
#define Taille_Table_Sinus 128
#define Nombre_Points_Constellation 16
#define Echantillons_Par_Symbole 32
#define Nombre_Symboles 16
#define K 8         //La taille du filtre

int Table_Sinus[Taille_Table_Sinus] =
{
0x0,0xc9,0x192,0x25b,0x324,0x3ed,0x4b5,0x57e,0x646,0x70e,0x7d6,0x89d
,0x964,0xa2b,0xaf1,0xbb7,0xc7c,0xd41,0xe06,0xeca,0xf8d,0x1050,0x1112
,0x11d3,0x1294,0x1354,0x1413,0x14d2,0x158f,0x164c,0x1708,0x17c3,0x18
7e,0x1937,0x19ef,0x1aa7,0x1b5d,0x1c12,0x1cc6,0x1d79,0x1e2b,0x1edc,0x
1f8b,0x203a,0x20e7,0x2193,0x223d,0x22e6,0x238e,0x2435,0x24da,0x257d,
0x2620,0x26c0,0x2760,0x27fd,0x289a,0x2934,0x29cd,0x2a65,0x2afb,0x2b8
f,0x2c21,0x2cb2,0x2d41,0x2dce,0x2e5a,0x2ee4,0x2f6b,0x2ff2,0x3076,0x3
0f8,0x3179,0x31f7,0x3274,0x32ef,0x3367,0x33de,0x3453,0x34c6,0x3537,0
x35a5,0x3612,0x367c,0x36e5,0x374b,0x37af,0x3811,0x3871,0x38cf,0x392a
,0x3984,0x39db,0x3a30,0x3a82,0x3ad3,0x3b21,0x3b6c,0x3bb6,0x3bfd,0x3c
42,0x3c85,0x3cc5,0x3d03,0x3d3e,0x3d78,0x3dae,0x3de3,0x3e15,0x3e44,0x
3e72,0x3e9d,0x3ec5,0x3eeb,0x3f0f,0x3f30,0x3f4f,0x3f6b,0x3f85,0x3f9c,
0x3fb1,0x3fc4,0x3fd4,0x3fe1,0x3fec,0x3ff5,0x3ffb,0x3fff};
// Structures globales
struct Point
{
     int I;     /* Amplitude de la composante en phase */
     int Q;  /* Amplitude de la composante en quadrature de phase */
};
// Definition de la structure constellation de type Point
struct Point Constellation[Nombre_Points_Constellation] =
{
     // Troisième quadrant de la constellation
     {-0x1000, -0x1000}, {-0x3000, -0x1000},
     {-0x1000, -0x3000}, {-0x3000, -0x3000},

     // Quatrième quadrant de la constellation
     {0x1000, -0x1000}, {0x1000, -0x3000},
     {0x3000, -0x1000}, {0x3000, -0x3000},

     // Deuxième quadrant de la constellation
     {-0x1000, 0x1000}, {-0x1000, 0x3000},
     {-0x3000, 0x1000}, {-0x3000, 0x3000},

     // Premier quadrant de la constellation
     {0x1000, 0x1000}, {0x3000, 0x1000},
     {0x1000, 0x3000}, {0x3000, 0x3000},
};
 int f[K]={0x1000,0x1000,0x1000,0x1000,// Les coefficients du filtre
           0x1000,0x1000,0x1000,0x1000}; // moyenneur de taille 8.
struct Parametres_Modem
{
```

```
      int bufferDonnees[4*Nombre_Symboles];// Buffer des bits generes
                                           //  et entree du Scrambler
      int sScr[4*Nombre_Symboles];          // Sortie du Scrambler
      int sCod[4*Nombre_Symboles];  // Sortie du Codeur Differentiel
      int symbolesE[Nombre_Symboles]; // Les symboles émis
      int echantillonsParSymbole;  // Nombre d'échantilons par
                                      symbole
      int phase;  // La phase de la porteuse
      int frequencePorteuse;   // La fréquence de la porteuse
      struct Point pointsConstellation[Nombre_Symboles];// Définition
          d'un vecteur de taille Nombre_Symboles, ses coéfficients de
          type  structure Point.
      int bufferSortie[Echantillons_Par_Symbole*Nombre_Symboles];
          // Buffer contient les échantillons du signal de sortie de
             l'emetteur du modem.
      int bufferR1[Echantillons_Par_Symbole*Nombre_Symboles];
          // signal reçu aprés multiplication par 2*cosinus
      int bufferR2[Echantillons_Par_Symbole*Nombre_Symboles];
          // signal reçu aprés multiplication par -2*sinus
      int signalFiltreI[Echantillons_Par_Symbole*Nombre_Symboles];
      int signalFiltreQ[Echantillons_Par_Symbole*Nombre_Symboles];
      int a[Nombre_Symboles*Echantillons_Par_Symbole];
          // Echantillons des composantes en phase reconstitués
      int b[Nombre_Symboles*Echantillons_Par_Symbole];
          // Echantillons des composantes en quadrature de phase
             reconstitués
      int s[Nombre_Symboles*Echantillons_Par_Symbole];
          // Echantillons des symboles reconstitués
     int symbolesR[Nombre_Symboles];  // Symboles détectés
     int eDec[4*Nombre_Symboles];      // Entrée du décodeur
     int sDec[4*Nombre_Symboles];      // Sortie du décodeur
     int sDescr[4*Nombre_Symboles];    // Sortie du déscrambler
};
struct Parametres_Modem modem;  // Définition d'une structure de
                                // type Parametres_Modem.
/*================================================================*/


/*================================================================
  =        Fonction de codage différentiel (Codeur())            =
  ================================================================*/
void Codeur(void)
{
     int n;
     for (n=0;n<=3;n++)
          {
           modem.sCod[n]=modem.sScr[n];
          }
      for (n=4;n<4*Nombre_Symboles;n=n+4)
      {
if (modem.sScr[n]==0 & modem.sScr[n+1]==0 & modem.sCod[n-4]==0 &
         modem.sCod[n-3]==0)
        {modem.sCod[n]=0;modem.sCod[n+1]=1;}
else if (modem.sScr[n]==0 & modem.sScr[n+1]==0 & modem.sCod[n-4]==0 &
         modem.sCod[n-3]==1)
```

```
          {modem.sCod[n]=1;modem.sCod[n+1]=1;}
else if (modem.sScr[n]==0 & modem.sScr[n+1]==0 & modem.sCod[n-4]==1 &
          modem.sCod[n-3]==0)
          {modem.sCod[n]=0;modem.sCod[n+1]=0;}
else if (modem.sScr[n]==0 & modem.sScr[n+1]==0 & modem.sCod[n-4]==1 &
          modem.sCod[n-3]==1)
          {modem.sCod[n]=1;modem.sCod[n+1]=0;}
else if (modem.sScr[n]==0 & modem.sScr[n+1]==1 & modem.sCod[n-4]==0 &
          modem.sCod[n-3]==0)
          {modem.sCod[n]=0;modem.sCod[n+1]=0;}
else if (modem.sScr[n]==0 & modem.sScr[n+1]==1 & modem.sCod[n-4]==0 &
          modem.sCod[n-3]==1)
          {modem.sCod[n]=0;modem.sCod[n+1]=1;}
else if (modem.sScr[n]==0 & modem.sScr[n+1]==1 & modem.sCod[n-4]==1 &
          modem.sCod[n-3]==0)
          {modem.sCod[n]=1;modem.sCod[n+1]=0;}
else if (modem.sScr[n]==0 & modem.sScr[n+1]==1 & modem.sCod[n-4]==1 &
          modem.sCod[n-3]==1)
          {modem.sCod[n]=1;modem.sCod[n+1]=1;}
else if (modem.sScr[n]==1 & modem.sScr[n+1]==0 & modem.sCod[n-4]==0 &
          modem.sCod[n-3]==0)
          {modem.sCod[n]=1;modem.sCod[n+1]=1;}
else if (modem.sScr[n]==1 & modem.sScr[n+1]==0 & modem.sCod[n-4]==0 &
          modem.sCod[n-3]==1)
          {modem.sCod[n]=1;modem.sCod[n+1]=0;}
else if (modem.sScr[n]==1 & modem.sScr[n+1]==0 & modem.sCod[n-4]==1 &
          modem.sCod[n-3]==0)
          {modem.sCod[n]=0;modem.sCod[n+1]=1;}
else if (modem.sScr[n]==1 & modem.sScr[n+1]==0 & modem.sCod[n-4]==1 &
          modem.sCod[n-3]==1)
          {modem.sCod[n]=0;modem.sCod[n+1]=0;}
else if (modem.sScr[n]==1 & modem.sScr[n+1]==1 & modem.sCod[n-4]==0 &
          modem.sCod[n-3]==0)
          {modem.sCod[n]=1;modem.sCod[n+1]=0;}
else if (modem.sScr[n]==1 & modem.sScr[n+1]==1 & modem.sCod[n-4]==0 &
          modem.sCod[n-3]==1)
          {modem.sCod[n]=0;modem.sCod[n+1]=0;}
else if (modem.sScr[n]==1 & modem.sScr[n+1]==1 & modem.sCod[n-4]==1 &
          modem.sCod[n-3]==0)
          {modem.sCod[n]=1;modem.sCod[n+1]=1;}
else if (modem.sScr[n]==1 & modem.sScr[n+1]==1 & modem.sCod[n-4]==1 &
          modem.sCod[n-3]==1)
          {modem.sCod[n]=0;modem.sCod[n+1]=1;}

modem.sCod[n+2]=modem.sScr[n+2];modem.sCod[n+3]=modem.sScr[n+3];
      }
}

/*=====================================================================*/
```

```
/*=================================================================
  =       Fonction de constitution des symboles émis              =
  =                    (GenererSymboles())                        =
  =================================================================*/
void GenererSymboles(void)
{
int n, k;

for (n = 0, k = 0; n < 4*Nombre_Symboles; n = n+4, k++)
 {
 if (modem.sCod[n]==0 & modem.sCod[n+1]==0 & modem.sCod[n+2]==0
                      & modem.sCod[n+3]==0)
          modem.symbolesE[k]=0;
 else if (modem.sCod[n]==0 & modem.sCod[n+1]==0 & modem.sCod[n+2]==0
                           & modem.sCod[n+3]==1)
          modem.symbolesE[k]=1;
 else if (modem.sCod[n]==0 & modem.sCod[n+1]==0 & modem.sCod[n+2]==1
                           & modem.sCod[n+3]==0)
          modem.symbolesE[k]=2;
 else if (modem.sCod[n]==0 & modem.sCod[n+1]==0 & modem.sCod[n+2]==1
                           & modem.sCod[n+3]==1)
          modem.symbolesE[k]=3;
 else if (modem.sCod[n]==0 & modem.sCod[n+1]==1 & modem.sCod[n+2]==0
                           & modem.sCod[n+3]==0)
          modem.symbolesE[k]=4;
 else if (modem.sCod[n]==0 & modem.sCod[n+1]==1 & modem.sCod[n+2]==0
                           & modem.sCod[n+3]==1)
          modem.symbolesE[k]=5;
 else if (modem.sCod[n]==0 & modem.sCod[n+1]==1 & modem.sCod[n+2]==1
                           & modem.sCod[n+3]==0)
          modem.symbolesE[k]=6;
 else if (modem.sCod[n]==0 & modem.sCod[n+1]==1 & modem.sCod[n+2]==1
                           & modem.sCod[n+3]==1)
          modem.symbolesE[k]=7;
 else if (modem.sCod[n]==1 & modem.sCod[n+1]==0 & modem.sCod[n+2]==0
                           & modem.sCod[n+3]==0)
          modem.symbolesE[k]=8;
 else if (modem.sCod[n]==1 & modem.sCod[n+1]==0 & modem.sCod[n+2]==0
                           & modem.sCod[n+3]==1)
          modem.symbolesE[k]=9;
 else if (modem.sCod[n]==1 & modem.sCod[n+1]==0 & modem.sCod[n+2]==1
                           & modem.sCod[n+3]==0)
          modem.symbolesE[k]=10;
 else if (modem.sCod[n]==1 & modem.sCod[n+1]==0 & modem.sCod[n+2]==1
                           & modem.sCod[n+3]==1)
          modem.symbolesE[k]=11;
 else if (modem.sCod[n]==1 & modem.sCod[n+1]==1 & modem.sCod[n+2]==0
                           & modem.sCod[n+3]==0)
          modem.symbolesE[k]=12;
 else if (modem.sCod[n]==1 & modem.sCod[n+1]==1 & modem.sCod[n+2]==0
                           & modem.sCod[n+3]==1)
          modem.symbolesE[k]=13;
 else if (modem.sCod[n]==1 & modem.sCod[n+1]==1 & modem.sCod[n+2]==1
                           & modem.sCod[n+3]==0)
          modem.symbolesE[k]=14;
```

```
  else if (modem.sCod[n]==1 & modem.sCod[n+1]==1 & modem.sCod[n+2]==1
                              & modem.sCod[n+3]==1)
          modem.symbolesE[k]=15;
   }
 }
 /*=================================================================*/


/*=================================================================
  =         Fonction de détection des symboles émis               =
  =                  (ReconstituerSymboles())                     =
  =================================================================*/
void ReconstituerSymboles()
{
int k;
for (k=0;k<Nombre_Symboles*Echantillons_Par_Symbole;k++)
     {
if (modem.signalFiltreI[k]< -0x2000)
   {modem.a[k]=-0x3000;goto second;}
if((modem.signalFiltreI[k]>-0x2000)&&(modem.signalFiltreI[k]< 0x0000))
    {modem.a[k]=-0x1000;goto second;}
if ((modem.signalFiltreI[k]>0x0000)&&(modem.signalFiltreI[k]< 0x2000))
    {modem.a[k]=0x1000;goto second;}
if (modem.signalFiltreI[k]> 0x2000) {modem.a[k]=0x3000;goto second;}
second:;
if (modem.signalFiltreQ[k]< -0x2000)
    {modem.b[k]=-0x3000;goto third;}
if((modem.signalFiltreQ[k]>-0x2000)&&(modem.signalFiltreQ[k]< 0x0000))
    {modem.b[k]=-0x1000;goto third;}
if ((modem.signalFiltreQ[k]>0x0000)&&(modem.signalFiltreQ[k]< 0x2000))
    {modem.b[k]=0x1000;goto third;}
if (modem.signalFiltreQ[k]> 0x2000)
    {modem.b[k]=0x3000;goto third;}
third:;
if ((modem.a[k]==-0x1000) && (modem.b[k]==-0x1000))
     modem.s[k]=0;
else if ((modem.a[k]==-0x3000) && (modem.b[k]==-0x1000))
          modem.s[k]=1;
else if ((modem.a[k]==-0x1000) && (modem.b[k]==-0x3000))
          modem.s[k]=2;
else if ((modem.a[k]==-0x3000) && (modem.b[k]==-0x3000))
          modem.s[k]=3;
else if ((modem.a[k]==0x1000) && (modem.b[k]==-0x1000))
          modem.s[k]=4;
else if ((modem.a[k]==0x1000) && (modem.b[k]==-0x3000))
          modem.s[k]=5;
else if ((modem.a[k]==0x3000) && (modem.b[k]==-0x1000))
          modem.s[k]=6;
else if ((modem.a[k]==0x3000) && (modem.b[k]==-0x3000))
          modem.s[k]=7;
else if ((modem.a[k]==-0x1000) && (modem.b[k]==0x1000))
          modem.s[k]=8;
else if ((modem.a[k]==-0x1000) && (modem.b[k]==0x3000))
          modem.s[k]=9;
```

```
else if ((modem.a[k]==-0x3000) && (modem.b[k]==0x1000))
          modem.s[k]=10;
else if ((modem.a[k]==-0x3000) && (modem.b[k]==0x3000))
          modem.s[k]=11;
else if ((modem.a[k]==0x1000) && (modem.b[k]==0x1000))
          modem.s[k]=12;
else if ((modem.a[k]==0x3000) && (modem.b[k]==0x1000))
          modem.s[k]=13;
else if ((modem.a[k]==0x1000) && (modem.b[k]==0x3000))
          modem.s[k]=14;
else if ((modem.a[k]==0x3000) && (modem.b[k]==0x3000))
          modem.s[k]=15;

}
for (k=0;k<Nombre_Symboles;k++)
{
 int n, m, s[16];
 for (n=0;n<16;n++)
     s[n]=0;
  for (n=(Echantillons_Par_Symbole)*k;n<((Echantillons_Par_Symbole)*k+
          Echantillons_Par_Symbole);n++)
      {
       for (m=0;m<16;m++)
           {
             if (modem.s[n]== m)
                         s[m] +=1;
           }
       }
           for (m=0;m<16;m++)
           {
               if (s[m] > 16)
                  {modem.symbolesR[k]=m;goto fin;}
           }
           fin:;
  }
}
/*=====================================================================*/
```

```
 /*=====================================================================
   =                  Fonction de décodage (Decodeur())                  =
   =====================================================================*/
 void Decodeur(void)
 {
      int k, n;
      for (k=0;k<Nombre_Symboles;k++)
      {
          modem.eDec[4*k]   = ((modem.symbolesR[k]&8)>>3);
          modem.eDec[4*k+1] = ((modem.symbolesR[k]&4)>>2);
          modem.eDec[4*k+2] = ((modem.symbolesR[k]&2)>>1);
          modem.eDec[4*k+3] = (modem.symbolesR[k]&1);
      }
      modem.sDec[0]=modem.eDec[0];modem.sDec[1]=modem.eDec[1];
      modem.sDec[2]=modem.eDec[2];modem.sDec[3]=modem.eDec[3];
 for (n=4;n<4*Nombre_Symboles;n=n+4)
```

```
     {
if (modem.eDec[n]==0 & modem.eDec[n+1]==1 & modem.eDec[n-4]==0 &
     modem.eDec[n-3]==0)
    {modem.sDec[n]=0;modem.sDec[n+1]=0 ;}
else if (modem.eDec[n]==1 & modem.eDec[n+1]==1 & modem.eDec[n-4]==0 &
     modem.eDec[n-3]==1)
    {modem.sDec[n]=0;modem.sDec[n+1]=0;}
else if (modem.eDec[n]==0 & modem.eDec[n+1]==0 & modem.eDec[n-4]==1 &
     modem.eDec[n-3]==0)
    {modem.sDec[n]=0;modem.sDec[n+1]=0;}
else if (modem.eDec[n]==1 & modem.eDec[n+1]==0 & modem.eDec[n-4]==1 &
     modem.eDec[n-3]==1)
    {modem.sDec[n]=0;modem.sDec[n+1]=0;}
else if (modem.eDec[n]==0 & modem.eDec[n+1]==0 & modem.eDec[n-4]==0 &
     modem.eDec[n-3]==0)
    {modem.sDec[n]=0;modem.sDec[n+1]=1;}
else if (modem.eDec[n]==0 & modem.eDec[n+1]==1 & modem.eDec[n-4]==0 &
     modem.eDec[n-3]==1)
    {modem.sDec[n]=0;modem.sDec[n+1]=1;}
else if (modem.eDec[n]==1 & modem.eDec[n+1]==0 & modem.eDec[n-4]==1 &
     modem.eDec[n-3]==0)
    {modem.sDec[n]=0;modem.sDec[n+1]=1;}
else if (modem.eDec[n]==1 & modem.eDec[n+1]==1 & modem.eDec[n-4]==1 &
     modem.eDec[n-3]==1)
    {modem.sDec[n]=0;modem.sDec[n+1]=1;}
else if (modem.eDec[n]==1 & modem.eDec[n+1]==1 & modem.eDec[n-4]==0 &
     modem.eDec[n-3]==0)
    {modem.sDec[n]=1;modem.sDec[n+1]=0;}
else if (modem.eDec[n]==1 & modem.eDec[n+1]==0 & modem.eDec[n-4]==0 &
     modem.eDec[n-3]==1)
    {modem.sDec[n]=1;modem.sDec[n+1]=0;}
else if (modem.eDec[n]==0 & modem.eDec[n+1]==1 & modem.eDec[n-4]==1 &
     modem.eDec[n-3]==0)
    {modem.sDec[n]=1;modem.sDec[n+1]=0;}
else if (modem.eDec[n]==0 & modem.eDec[n+1]==0 & modem.eDec[n-4]==1 &
     modem.eDec[n-3]==1)
    {modem.sDec[n]=1;modem.sDec[n+1]=0;}
else if (modem.eDec[n]==1 & modem.eDec[n+1]==0 & modem.eDec[n-4]==0 &
     modem.eDec[n-3]==0)
    {modem.sDec[n]=1;modem.sDec[n+1]=1;}
else if (modem.eDec[n]==0 & modem.eDec[n+1]==0 & modem.eDec[n-4]==0 &
     modem.eDec[n-3]==1)
    {modem.sDec[n]=1;modem.sDec[n+1]=1;}
else if (modem.eDec[n]==1 & modem.eDec[n+1]==1 & modem.eDec[n-4]==1 &
     modem.eDec[n-3]==0)
    {modem.sDec[n]=1;modem.sDec[n+1]=1;}
else if (modem.eDec[n]==0 & modem.eDec[n+1]==1 & modem.eDec[n-4]==1 &
     modem.eDec[n-3]==1)
    {modem.sDec[n]=1;modem.sDec[n+1]=1;}

modem.sDec[n+2]=modem.eDec[n+2];modem.sDec[n+3]=modem.eDec[n+3];
      }
}
/*=================================================================*/
```

Annexe 2

Fonctionnement de l'Environnement de Développement Intégré Code Composer Studio (CCS)

L'environnement de développement intégré Code Composer Studio est décris en bref dans le chapitre 3. Les différentes capacités et possibilités de son utilisation sont citées encore dans le même chapitre. Cette annexe présente son fonctionnement.

1. Création d'une configuration système

Le CCS Setup nous permet de configurer le logiciel CCS IDE afin de travailler sur différents matériels ou cibles de simulateur. En effet, le CCS Setup dispose de plusieurs configurations des deux familles C55x et C54x. Une configuration du système est exigée avant de créer une quelconque application afin de déterminer quels outils seront utilisés par le CCS.

Dés l'installation du CCS, une configuration du simulateur C55x est fournie par défaut. Pour créer une nouvelle configuration du système on suit les étapes suivantes :

- **Etape 1 :** On ouvre le CCS Setup en double cliquant sur l'icône du CCS Setup localisée sur le bureau.
- **Etape 2 :** On enlève l'ancienne configuration (ou celle installée par défaut) en cliquant sur le bouton "Clear" (effacer) de la boite de dialogue "Import configuration".
- **Etape 3 :** On clique ensuite sur le bouton "Yes" pour confirmer la commande "Clear".
- **Etape 4 :** On sélectionne la configuration standard qu'on veut installer sur la liste "Available Configuration".

En sélectionnant une configuration standard, une information sera affichée afin de nous aider à déterminer la configuration voulue pour notre système. Si aucune configuration ne nous convient, on peut créer une configuration personnalisée (voir l'aide ou Tutorial fournis par CCS).

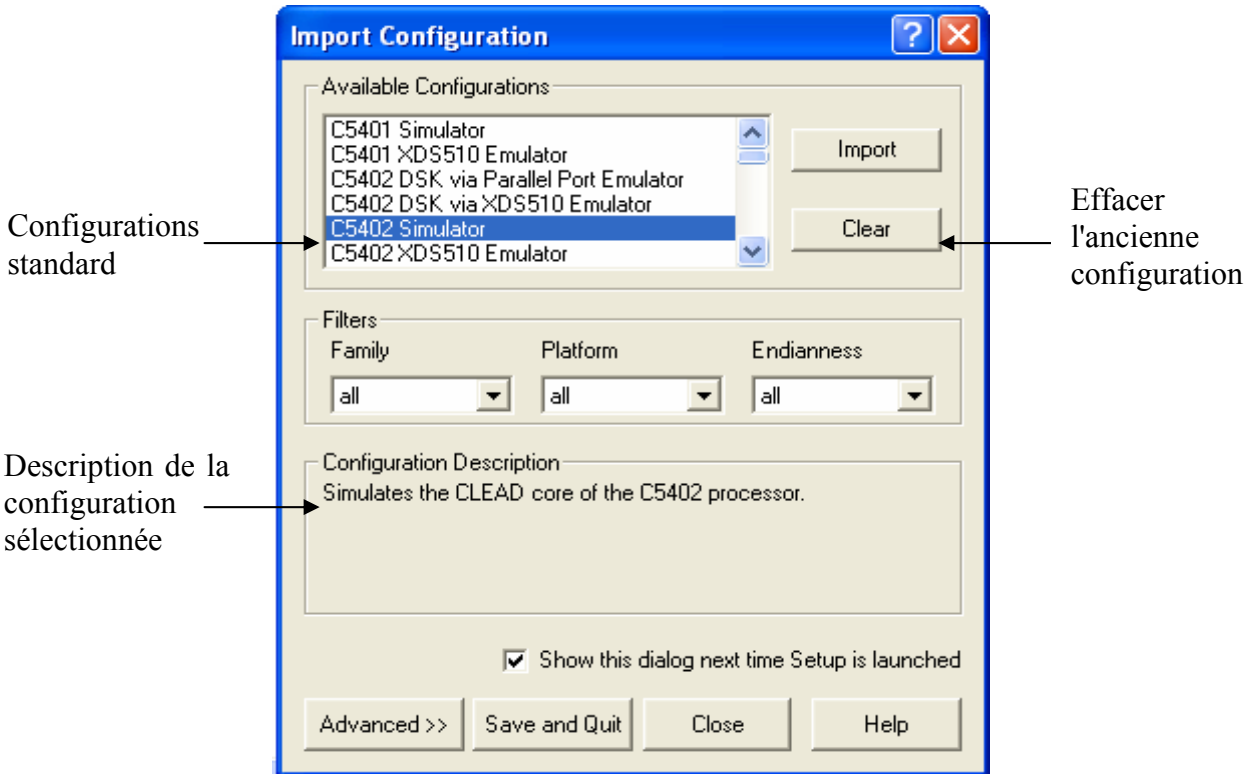


Figure 1 : Boite de dialogue du bouton "Import Configuration"

- **Etape 5 :** On clique sur le bouton "Import" pour importer notre configuration sélectionnée. Si cette dernière a plus d'une cible, on répète les étapes 4 et 5 jusqu'à ce que nous sélectionnons une configuration pour chaque cible.
- **Etape 6 :** on clique sur le bouton "Save and Quit" pour sauvegarder la nouvelle configuration.
- **Etape 7 :** On clique sur le bouton "Yes" quand le message suivant est apparaît : "Start Code Composer Studio on Exit". Le CCS Setup sera fermé et le CCS IDE s'ouvre automatiquement utilisant la nouvelle configuration créée.

Maintenant, on peut commencer à créer un nouveau projet dans le CCS IDE.

2. Accès au Tutorial du CCS

Afin de se familiariser d'avantage avec l'environnement CCS IDE, il est recommandé d'exécuter les exemples proposés par ce dernier en Tutorial. Ceci nous permet de réduire notre temps d'apprentissage, et de prendre connaissances des principales procédures fondamentales. Pour accéder aux exemples de CCS, on suit ces deux étapes :

- **Etape 1 :** On ouvre le CCS IDE en double cliquant sur l'icône "CCS 2" localisée sur le bureau.

♦ **Etape 2 :** Sur le menu "help" du CCS on sélectionne "Tutorial".

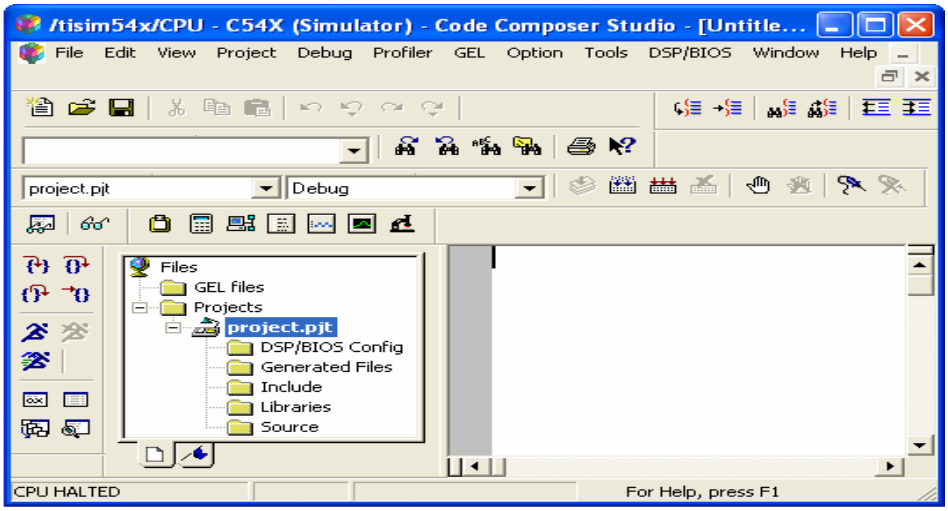


Figure 2 : Interface du CCS IDE

3. Gestion de projets

3.1 Création d'un fichier projet

Les procédures suivantes nous permettent de créer de nouveaux projets, un ou plusieurs à la fois, seulement le nom du dossier de chaque projet doit être unique. L'information pour un projet est enregistrée dans un dossier sous la forme (*.pjt). On peut ouvrir plusieurs projets en même temps.

Les étapes de création d'un projet se déroulent comme suit :

Etape 1 : Sur le menu du CCS, on sélectionne Project→New, la fenêtre si- dessous apparaît :

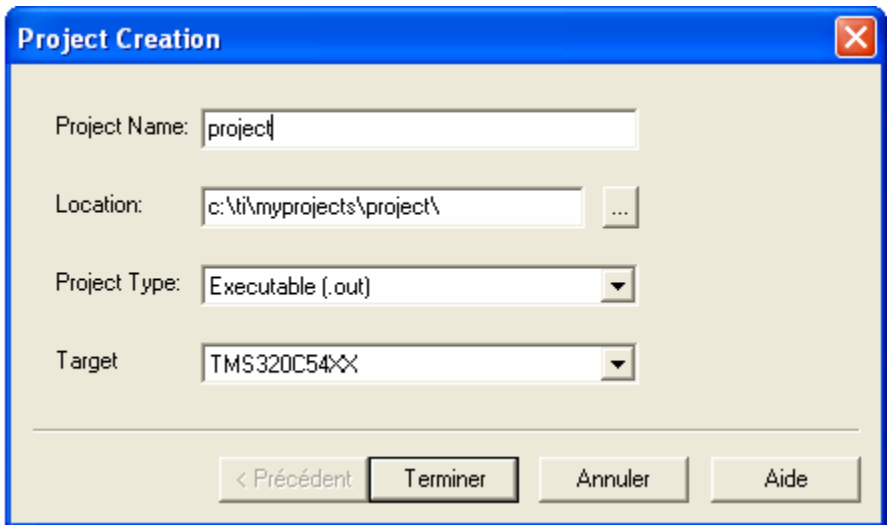


Figure 3 : Boite de dialogue de "Project Creation"

Etape 2 : Sur le champ "Project Name" on écrit le nom qu'on veut attribuer à notre projet.

Etape 3 : Sur le champ "Location" on choisit le chemin où on veut enregistrer notre projet.

Etape 4 : Sur le champ "Project Type" on choisit le type de projet dans la liste proposée (telle que Executable (.out)) pour que le projet produit un fichier exécutable (.out) ou bibliothèque (.lib) pour construire une bibliothèque de l'objet.

Etape 5 : Sur le champ "Target" on choisit la famille qui identifie notre CPU. Cette information s'avère très nécessaire lorsque les outils sont installés pour plusieurs cibles.

Etape 6 : On clique sur "Finish", le CCS crée le dossier de notre projet "Projectname.pjt" qui contiendra tous les dossiers utiles pour ce dernier. Le nouveau projet devient automatiquement le projet actif et aura les options de débogage et de liens (linker) fournies par TI.

Après avoir créé notre projet on ajoute à sa liste : le fichier code source, la bibliothèque et le fichier de commande de l'éditeur de liens.

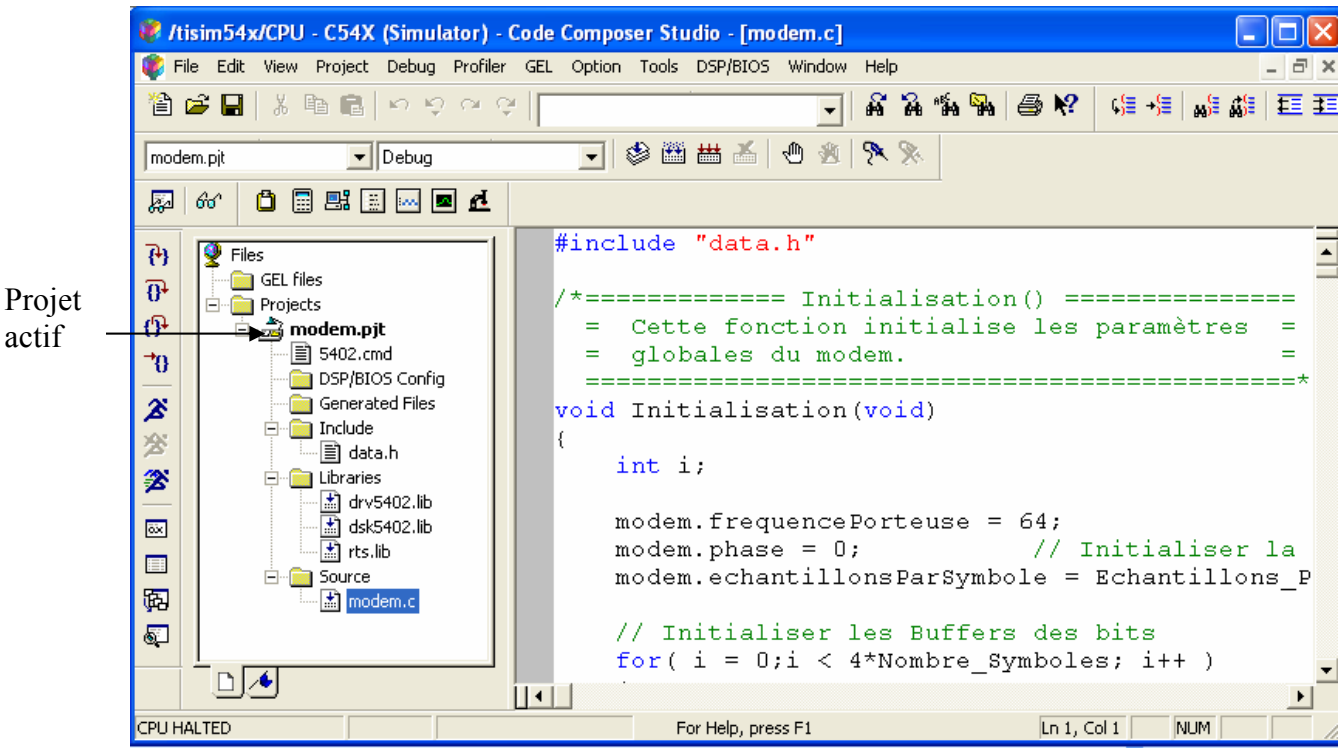


Figure 4 : Fenêtre de base du CCS

3.2 Ajout de fichiers à un projet

On peut ajouter plusieurs et différents fichiers à notre projet. Les types sont montrés dans la boite de dialogue (figure 5). La procédure à suivre pour ajouter des fichiers à notre projet se déroule comme suit :

Etape 1 : On sélectionne Project→Add Files to Project ou on clique par le bouton droit sur le nom du projet affiché sur la fenêtre puis on sélectionne "Add Files". La boite de dialogue suivante apparaît :

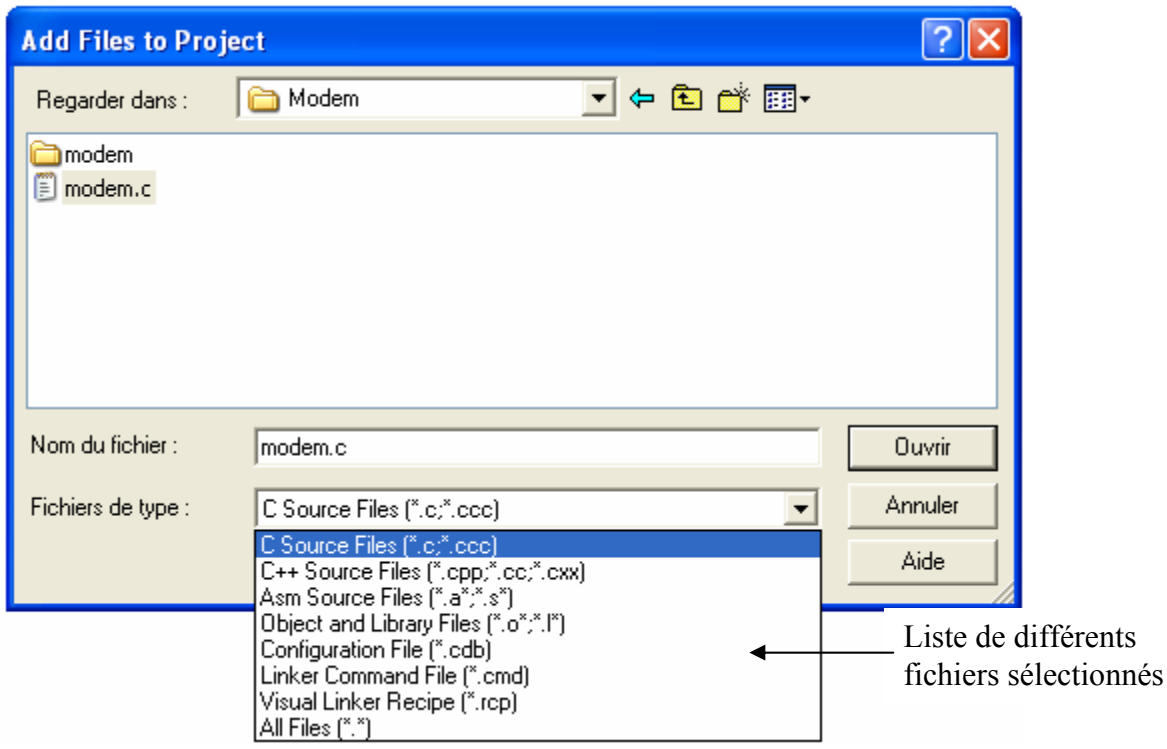


Figure 5 : Boite de dialogue de "Add Files to Project"

Etape 2 : Dans cette boite de dialogue on spécifie le fichier qu'on désire ajouter. Si le fichier n'existe pas dans le répertoire courant, on parcourt jusqu'à trouver son chemin. Ne jamais essayer d'ajouter manuellement les fichiers en-tête/Include (*.h) au projet. Ces fichiers seront ajoutés automatiquement lorsque on clique sur "Scan All dependencies" ou bien lorsque on construit le projet (on va voir ça par la suite).

Etape 3 : On clique sur "Open" pour ajouter le fichier spécifié au projet. L'affichage du projet est automatiquement mis à jour lorsqu'un fichier est ajouté. Le manager du projet organise les fichiers dans les dossiers : source, Include, Librairies et DSP/BIOS Configuration.

Si on veut enlever un fichier du projet on clique sur ce fichier par le bouton droit et on sélectionne sur le pop-up menu "Remove from project".

3.3 Génération et exécution du fichier exécutable (Building and running)

Pour générer et exécuter le programme exécutable, on suit les étapes suivantes :

Etape 1 : On sélectionne Project→Rebuild All ou on clique sur le bouton  dans la barre d'outils.

Le CCS recompile, réassemble et relie tous les fichiers dans notre projet. Des messages sur la progression du processus d'exécution sont affichés dans un cadre au fond de la fenêtre.

Etape 2 : Par défaut, le fichier exécutable (.out) est construit dans le dossier "Debug" du projet courant, pour changer cette location on sélectionne une location différente dans la barre d'outils du CCS.

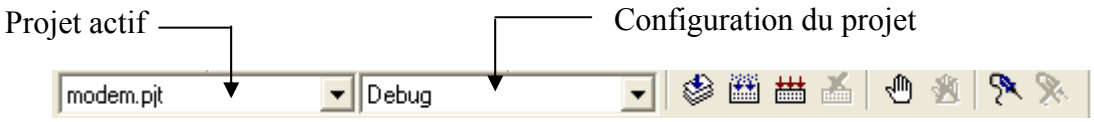


Figure 6 : Location du fichier exécutable (.out)

Etape 3 : On sélectionne Files→Load Program, ensuite on sélectionne le programme que nous venons de créer (.out), et on clique sur "Open". Le CCS charge le programme sur la cible DSP et ouvre une fenêtre "Di-Assembly"qui montre les instructions traduites en assembleur composant le programme.

Etape 4 : On choisit le View/Mixed Source/ASM, cela permet de voir notre code source écrit en C et sa traduction en assembleur simultanément.

Etape 5 : On sélectionne Debug→Go to Main pour commencer l'exécution à partir de Main (ie programme principal). L'exécution s'arrête au Main et est identifiée par la flèche .

Etape 6 : On sélectionne Debug→Run ou on clique sur le bouton (bouton sur la barre d'outils).

Etape 7 : On sélectionne Debug→Halt pour arrêter l'exécution du programme.

4. Points d'arrêt, break points

Les points d'arrêt servent à arrêter l'exécution d'un programme à la lecture d'instructions particulières ou lorsqu'une condition est remplie.

Pour placer un point d'arrêt sur une instruction, on positionne le curseur devant l'instruction désirée dans la fenêtre d'édition contenant le fichier source C ou assembleur. Pour positionner un point d'arrêt sur cette instruction, il suffit de cliquer sur l'icône *toggle break-point* .

Lors d'une exécution du programme, le programme s'arrêtera juste avant exécuter l'instruction sur laquelle on met le point d'arrêt. Pour effacer un point d'arrêt, il suffit de faire la même opération, cliquer une deuxième fois sur l'icône *toggle break-point* .

5. Chargement et sauvegarde des données

Le CCS nous permet de charger et de stocker des données sous format (.dat) par trois différents moyens :

5.1 Utilisation de Load/Save à partir du menu File

On sélectionne File→Data→Load pour charger les données manuellement à partir d'un fichier (.dat) qui doit contenir une ligne d'entête du DSP formée de 5 champs :

- Le nombre magique fixé à 1651.
- Le format qui est un entier entre 1 et 4, spécifiant le format des données dans le fichier (1 : hexadécimal, 2 : entier, 3 : entier long, 4 : nombre flottant).
- L'adresse de départ du bloc qui a été sauvegardée.
- Le numéro de page correspond à la page source du bloc.
- La longueur du bloc.

Exemple d'une entête : 1651 1 0 1 100.

On doit préciser la variable de destination (ou son adresse mémoire) de notre chargement et le nombre d'échantillons à charger.

Pour la sauvegarde des données traitées, on sélectionne File→Data→Save, on précise la variable qu'on veut sauvegarder (ou son adresse mémoire) et le nombre d'échantillons à sauvegarder.

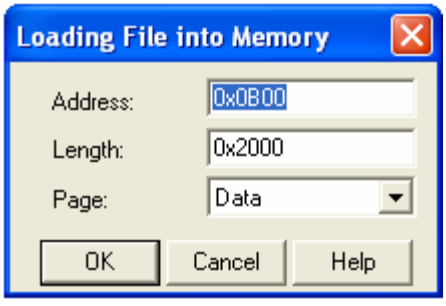


Figure 7 : Chargement de données

Figure 8 : sauvegarde de données

5.2 Utilisation des points sondes (Probe Points)

Ces points sondes servent aussi à charger et sauvegarder manuellement les fichiers de données Input/Output (*.dat), elles seront placés sur le curseur positionné sur le code source de notre projet, il suffit de cliquer sur le bouton . Ensuite on clique sur File→File I/O on

obtient la fenêtre (figure 9) sur laquelle on choisit les fichiers à connecter aux points sondes pour le chargement (Input) ou pour le sauvegarde (Output). Ces fichiers doivent contenir l'en-tête expliqué précédemment.

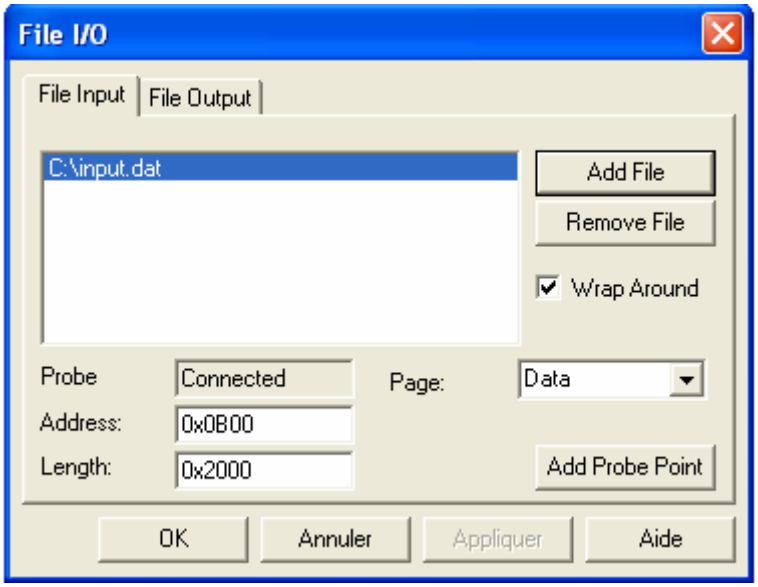
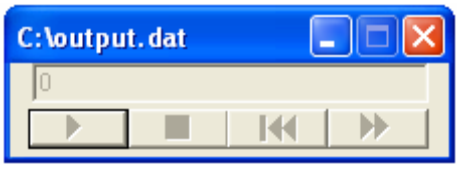


Figure 9 : Boite de dialogue "File I/o"

Pour chaque point sonde connecté à un fichier, on obtient les icônes suivantes :



Il suffit de cliquer sur  pour charger ou sauvegarder les données.

Remarque : Si notre fichier source contient des données qui se répètent dans le temps, on sélectionne l'option "Warp around" sur la boite de dialogue, par exemple cas d'un signal sinusoïdal.

5.3 Utilisation des instructions de la directive "stdio.h" dans le code source

Les instructions : fopen, fscanf, fprintf, fclose,…etc, servent à charger et sauvegarder automatiquement les fichiers de données Input/Output (*.dat). Ces fichiers lus ne doivent pas contenir l'en-tête du DSP.

6. Tracés et graphiques

Pour ajouter une fenêtre graphique et tracer un signal, on utilise la commande *View-Graph-Time/Frequency*. Dans la boite de dialogue, on remplie les différents champs proposés,

comme par exemple l'adresse de départ de la zone mémoire à tracer. On clique ensuite sur Ok, une fenêtre de tracé s'ouvre.

Les options du tracé sont :

Single Time : tracer les signaux dans le domaine temporel.

FFT magnitude : tracer la transformée de Fourier du signal.

Coplex FFT : tracer la transformée de Fourier complexe du signal.

FFT magnitude and Phase : tracer les transformées de Fourier en amplitude et en phase du signal.

FFT Waterfall : un autre type de tracé de la transformée de Fourier du signal.

Eye Diagram : Tracer le diagramme de l'oeil.

Constellation : tracer la constellation du signal.

Image : tracer une image.

7. Analyse de performances :

Pour mettre aux point une application et pouvoir travailler en temps réel, il faut optimiser la durée d'exécution du programme. Les outils de *Profiling* de Code Composer aident à analyser le programme et permettent d'obtenir des statistiques d'exécution. On utilise pour cela le menu Profiler de la barre des menus.