Etude d'un analyseur de protocole pour réseaux Bluetooth

Travail de Diplôme

J.Stettler

Professeur responsable : Prof. S. Ventura
Expert responsable : Prof. Dr J.-D. Decotignie

Table des matières

Liste des Figures

Liste des Tables

Département : E+I
Filière : Télécommunications
Candidat : **Jérôme Stettler**

TRAVAIL DE DIPLOME 2000

Télécommunications et téléinformatique

Analyseur d'accès et protocoles Bluetooth

Données de base :

Ce projet supervisé par le groupe Real-time and Networking du CSEM (Centre Suisse d'Electronique et de Microtechnique) s'inscrit dans le cadre des communications à courte distance et plus particulièrement de la norme de communication Bluetooth. Bluetooth est le nom adopté par une spécification technologique qui permet, pour un encombrement réduit et un faible coût, d'établir des communications sans fil entre appareils électroniques proches les uns des autres (quelques dizaines de mètres). Créé au printemps de l'an dernier par Ericsson, IBM, Intel, Nokia et Toshiba, Bluetooth est aujourd'hui adopté par des centaines d'entreprises.

Le candidat devra réaliser un dispositif d'analyse permettant de surveiller le trafic entre plusieurs stations communiquant selon le standard Bluetooth.

Cahier des charges :

Ce projet comprend les activités suivantes :

- Développement d'un firmware permettant l'analyse des basses couches du protocole :
  1. accès aux piconets, signaux de synchronisation,
  2. découverte de nouveaux correspondants,
  3. décodage, dans la mesure des possibilités offertes par l'interface physique d'accès à Bluetooth, des trames (header et header du payload) de transports spécifiques à Bluetooth dans le cadre des communications ACL,
  4. décodage sommaire des trames MAC et éventuellement LLC.

Remarques : Le firmware sera développé sur un kit de développement, mis à disposition par le CSEM, comportant un circuit HF et qui permet la conversion du signal reçu en bande de base.

Pour des raisons de performance, le développement se fera sur une plaque munie d'un microprocesseur. Le firmware de décodage sera implémenté sur cette plaque.

- Les résultats seront envoyés sur un PC par ligne sérielle pour affichage et stockage.
- Le même PC devra permettre de configurer les paramètres d'acquisition du système de développement.

Dans ce projet l'accent sera mis sur la capture du plus d'événements possibles relatifs au niveau physique (détection de porteuse, synchronisation, etc.) et de la stabilité du firmware. Pour cela il s'agira de proposer et réaliser un dispositif de test permettant de valider le développement effectué.

Résultats :

Les résultats du développement feront l'objet d'un rapport circonstancié. Un aperçu du logiciel et de ses caractéristiques sera publié sur Internet, au travers du World Wide Web. Cette publication sera accessible sur internet et présentée lors de la défense du travail de diplôme.

Le rapport contiendra toutes les indications nécessaires pour un éventuel développement futur du logiciel par de tierces personnes, ainsi qu'un mode d'emploi suffisamment précis pour pouvoir servir de base à une éventuelle publication. L'utilisation d'un éditeur de texte agréé par l'école est requise.

Les produits réalisés sont à livrer en même temps que le rapport, sous une forme utilisable par le laboratoire :

- les logiciels sous une forme exécutable, accompagnés des sources et des fichiers nécessaires à la recompilation et à la régénération du produit, seront déposés sur le serveur du laboratoire de téléinformatique ;
- les logiciels, accompagnés des sources et des fichiers nécessaires à la recompilation et à la régénération du produit, seront également joints au rapport sur des disquettes avec une procédure d'installation ;
- le rapport sera mis à disposition sous forme de CD lors de la défense du travail de diplôme ;
- la présentation et le contenu du rapport doivent correspondre aux consignes reçue en annexe.

Introduction

Ce travail de diplôme a pour but l'utilisation des connaissances acquises durant le travail de semestre pour créer un analyseur de protocole pour réseaux sans-fils.

Lors du travail de semestre, les réseaux sans-fils Bluetooth et 802.11 ont été étudiés et comparés. Il s'est avéré qu'il serait intéressant de créer un analyseur de protocole pour Bluetooth. Ce travail assez long est prévu sur plusieurs diplômes.

La première partie est le sujet de ce diplôme. Elle consiste à développer le *firmware* qui met en forme le signal reçu d'un transceiver radio pour qu'il soit utilisable par un microprocesseur. Puis une analyse précise du fonctionnement de la baseband de l'analyseur est faite et en partie implémantée.

Remerciements

Ce diplôme n'aurait pas été possible sans l'aide de plusieurs personnes. Je tiens à remercier :

M. Amre El-Hoiydi pour son aide, ses explications sur Bluetooth et ses remarques et idées sur la façon de mener ce diplôme;

M. Philippe Dallemagne pour la relecture de mon rapport et ses explications;

M. Guevara Noubir pour l'aide sur l'algorithme de recouvrement de l'horloge ainsi que sur les algorithmes de codage et de détections d'erreurs;

M. Alexandre Pollini pour l'aide lors du montage et des tests du circuit de mise en forme ainsi que pour la relecture d'une partie de mon rapport;

M. Laurent Mealares pour l'aide en C. et les explications sur le DSP;

M. Jean Hernandez pour la surveillance de la bonne marche de mon diplôme;

M. Yan Brand pour l'explication sur le fonctionnement du transceiver radio de TEMIC;

Mme Anne Monin pour la correction et la relecture de mon rapport;

Ainsi que toutes les personnes travaillant dans le secteur 241 et 231 du CSEM.

1. Points atteints

Les buts de ce travail de diplôme étaient très élevés et il n'était pas possible en trois mois de parvenir à les réaliser entièrement. Mais un grand travail a été accompli puisque les signaux reçus par le frontend radio sont maintenant mis en forme et compatibles avec une entrée série d'un DSP. Le DSP mémorise les données reçues et détecte déjà le début des trames. De plus la Baseband de l'analyseur a été décrite précisément.

1.1. Plan de travail

Ce travail est divisé en plusieurs parties qui font chacune l'objet d'un chapitre de ce rapport.

- La première partie a été le traitement du signal de données généré par le Transceiver radio pour le rendre compatible avec un port série et le recouvrement du signal d'horloge.

- la deuxième partie du travail consiste en la création d'un programme permettant la recherche dans le signal reçu du début des paquets.

- La troisième partie présente l'analyse et la description du fonctionnement de la Baseband de l'analyseur.

1.2. Résumé du travail de semestre

1.2.1. Rapport

Un rapport a été écrit décrivant la norme Bluetooth ainsi que 802.11 et une comparaison des deux normes.

1.2.2. Analyseur de protocole

Le but de ce travail de semestre est aussi d'étudier la possibilité existante de réaliser un analyseur de protocole. Les questions principales sont :

- Quel protocole choisir ?

Bluetooth ou 802.11

- Que devra faire cet analyseur ?

1.2.3. Choix du protocole

Les diverses possibilités offertes par les deux protocoles ont été étudiées. Les questions étaient:

- Les drivers possibles ?
- Quelles documentations ?
- A quel niveau de la pile de protocole peut-on accéder ?

Voici les résultats :

Bluetooth

Bluetooth est un protocole nouveau et très à la mode. On trouve beaucoup d'informations générales sur ce protocole et tous les grands constructeurs ont des projets de cartes BT. Malheureusement on ne trouve pas beaucoup de documentation précise sur le fonctionnement interne des implémentations de BT.

Voici le résumé des possibilités les plus intéressantes :

802.11

802.11 contrairement à Bluetooth, est un protocole un peu plus vieux et on peut donc espérer plus de documentation et de driver qui permettrait de commander des cartes 802.11. Il existe une liste des cartes et de leurs drivers linux :

http://www.hpl.hp.com/personal/Jean_Tourrilhes/Linux/Linux.Wireless.drivers.html

1.2.4. Que doit faire un analyseur 802.11 ?

Il serait intéressant de pouvoir afficher toutes les trames MAC de 802.11 qui passent à la portée de la carte. La plupart des analyseurs ne donnent aucune information sur les trames de services et de contrôles, il serait donc intéressant de pouvoir les afficher. Il peut être aussi intéressant de dater très précisément chaque paquet reçu (en utilisant par exemple real-time Linux), ceci pour permettre de développer des applications temps réel.

2. Bluetooth

2.1. Introduction

Ce chapitre explique le fonctionnement de quelques points précis particuliers à Bluetooth[1] et qui sont utiles à la compréhension du fonctionnement de l'analyseur. Il s'agit des piconets, du format des paquets, de l'interface HCI et des procédures de connexion (**Page**) et de recherche(**Inquiry**). Les chapitres sur les piconets et sur le format des paquets est extrait du travail de semestre [2] dont les parties sur le format des paquets et sur la procédure **Inquiry** ont été approfondies.

2.2. Les piconets

2.2.1. Introduction

Un piconet regroupe un ensemble d'appareils utilisant Bluetooth dans un réseau ad hoc. Un piconet existe dès que deux appareils se connectent ensemble et peut contenir jusqu'à 8 appareils. Lors de la création du réseau, l'appareil qui démarre la connexion deviendra le maître du réseau et tous les autres appareils seront ses esclaves pendant toute la durée de vie du piconet.

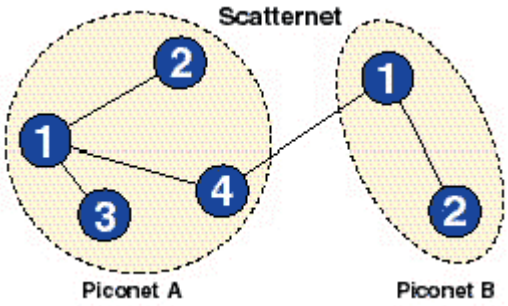


Figure 2-1 Exemple de piconet et de scatternet.

Deux piconets peuvent être connectés ensemble, ils forment alors un scatternet. Il existe plusieurs possibilités pour créer un scatternet. L'esclave d'un piconet peut devenir le maître d'un autre piconet ou esclave dans plusieurs piconets.

2.2.2. Canaux physiques

Les appareils Bluetooth d'un même piconet utilisent un canal commun pour échanger les paquets. Ce canal est divisé en intervalle de temps, *slots*, de 625 µs dont l'utilisation est gérée par le maître du piconet. Il interroge les esclaves (pooling) pour leur permettre de

[1] 'BLUETOOTH est une marque déposée de Telefonaktiebolaget L M Ericsson, Sweden

transmettre leurs paquets. Le maître émet durant les intervalles pairs et les esclaves durant les intervalles impairs. Une transmission ne peut se faire qu'entre un maître et un esclave ou entre un esclave et son maître. Les communications esclave-esclave n'existent pas.

La numérotation des *slots* correspond à la valeur de l'horloge Bluetooth du maître. Cette horloge de 27 bits, numérote les paquets de 0 à 2^{27-1}. Elle fait un tour en 23h et a une résolution de 312.5µs (1/2 *slot*).

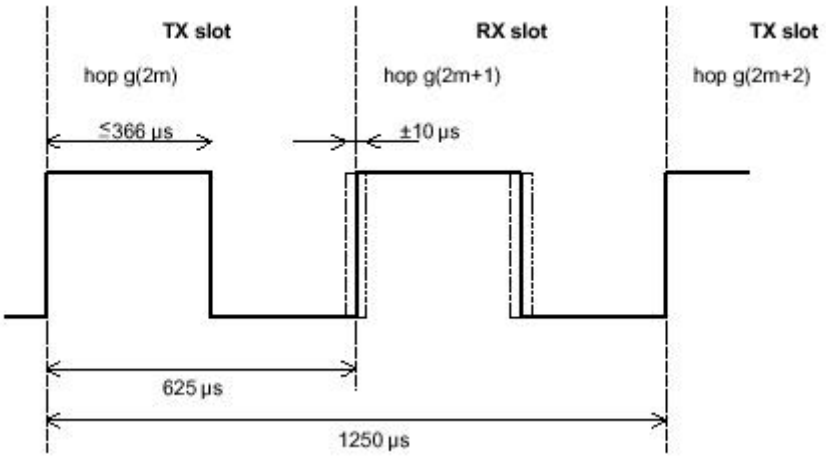


Figure 2-1 Cycle de Rx/Tx d'un maître. Pour un esclave, les cycles Rx et Tx sont inversés. 366µs sont utilisés pour la transmission du paquet, le reste est utilisé pour changer la fréquence du transceiver radio.

Bluetooth utilise le frequency hopping, technique qui consiste à changer la fréquence du canal de transmission après chaque paquet transmis. Un canal physique est défini par sa séquence de sauts de fréquences. La séquence est déterminée par l'adresse du maître et la phase de cette séquence dépend du numéro du *slot* utilisé, donc de l'horloge du maître du piconet. Les esclaves doivent pour pouvoir transmettre des paquets, connaître l'horloge et l'adresse du maître. Pour cela, lorsqu'ils entrent dans le piconet, ils reçoivent un paquet (paquet FHS, voir chap. 2.4.2) du maître qui contient l'horloge et l'adresse du maître, ainsi ils sont synchronisés avec lui et peuvent participer au piconet.

2.3. Format des paquets

Les données sur un canal du piconet sont transmises par paquets. Les paquets ont le format général suivant:

Figure 2-1 Format général d'un paquet.

Chaque paquet contient les trois parties *Access code*, *Header* et *Payload*, sauf le paquet ID qui ne contient que l'*Access Code*.

Une description exacte du contenu de chaque paquet se trouve en annexe 7.11

Les champs *Access Code* et *Header* ont des tailles fixes de 72 et 54 bits, sauf le paquet ID dont l'*Access Code* ne contient que les 68 premiers bits. La taille de la partie utile appelée *Payload* peut varier de 0 à 2 745 bits.

Dans les paquets, les bits sont ordonnés selon le format *Little Endian.* Ceci implique que :

- Le bit LSB correspond à b_0 ;
- Le bit LSB est le premier émis par l'émetteur;
- Dans les figures le LSB est représenté sur le côté gauche ;

2.3.1. Champs Access Code

Chaque paquet débute avec un *Access Code*. Celui-ci sera utilisé pour la synchronisation, la suppression de la composante continue et pour l'identification du piconet. Il identifie les paquets échangés dans le piconet. Tous les paquets émis dans un même piconet sont précédés du même *Access Code*.

L'*Access Code* contient 72 bits s'il est suivi d'un en-tête et 68 bits s'il est émis seul. Il contient trois parties: un fanion d'en-tête; un mot de synchronisation et un fanion de queue.

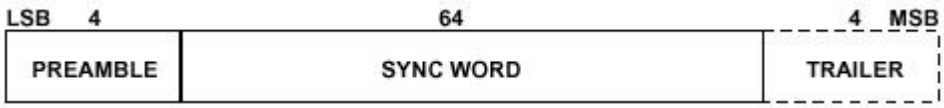


Figure 2-1 Format de l'*Access Code*.

Il y a trois types d'*Access Code* définis :

- *Channel Access Code* (CAC);
- *Device Access Code* (DAC);
- *Inquiry Access Code* (IAC).

Le type utilisé dépend du mode d'opération de l'appareil Bluetooth.

Le *Channel Access Code* identifie un piconet. Il est inséré dans tous les paquets transmis dans un piconet.

Le *Device Access Code* est utilisé pour la création d'une nouvelle connexion (*Page* et *Page Scan*) ou pour la transmission de signalisation spéciale.

Le *Inquiry Access Code* est utilisé lors de la recherche d'autres appareils Bluetooth (*Inquiry*). Il existe en plusieurs versions. Une version générale (GIAC, *General Inquiry Access Code*) qui indique à tous les appareils qui le reçoivent qu'ils doivent répondre, et des versions pour chaque classe d'appareils (DIAC, Device Inquiry *Access Code*) qui permet de ne rechercher qu'une classe d'appareils. Dans ce cas, tous les appareils recoivent le DIAC mais seul les appareils correspondant à la classe appelée répondent au message.

Preamble

Le fanion de tête (*Preamble*) est une séquence de bits définie qui permet de faciliter la compensation de la composante continue. La séquence dépend du LSB du *Sync Word* suivant.

Figure 2-1 Format des fanions d'en-tête selon que le LSB du *Sync Word* est à 1 (fig de gauche) ou à 0 (fig de droite).

Sync word

Le mot de synchronisation (*Sync Word*) est un mot de 64 bits créé à partir des 24 bits de *Lower Address Part* (LAP) de l'adresse de base. Les bits d'adresses LAP sont la partie basse de l'adresse de 48 bits d'un appareil Bluetooth, appelé *Bluetooth device adresse* (BD_ADDR), bits A0 à A23.

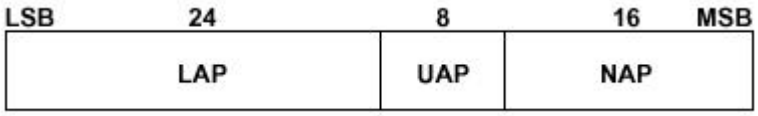


Figure 2-1 Format de l'adresse de base d'un appareil Bluetooth BD _ADDR.

Pour le CAC, on utilise les bits LAP du maître comme base pour créer le *Sync Word*; pour le GIAC et le DIAC, on utilise des adresses dédiées, réservées spécialement à cette effet comme base du *Sync Word* et pour le DAC on utilise la partie LAP de l'adresse de l'unité que l'on veut connecter. Les *Sync Word* sont créés de façon à avoir une grande distance de Hamming entre eux.

Table 2-1 LAP de base pour les différents *Sync Word*

Type de code	LAP de base
CAC	Maître
DAC	Unité que l'on veut connecter
GIAC	0x9E8b3F
DIAC	Adresses dédiées. Dépend de la classe d'appareils.

Les classes d'appareils utilisés pour le DIAC ne sont pas encore définies dans la norme

Génération du mot de synchronisation Sync Word

Pour créer le *Sync Word*, on utilise une adresse LAP qui dépend du type d'opérations que l'on veut effectuer. Les adresses à utiliser sont répertoriées dans laTable 2-1.

1. A ce LAP de 24 bits (A_0 à A_{23}), on concatène une séquence de 6 bits qui est 001101_b si A23 vaut 0 ou 110010_b si A_{23} vaut 1. Ces 6 bits plus le bit A_{23} forment un Barker code de longueur 7 qui facilite la synchronisation de la trame.

2. A ces 30 bits, on additionne modulo 2, un bruit pseudo-aléatoire. Cette séquence vaut 0x83848D96BBCC54FC. On utilise les bits P_{34} à P_{63}. On obtient ainsi les bits $\tilde{x}_0 - \tilde{x}_{29}$.

3. On rajoute devant les bits $\tilde{x}_0 - \tilde{x}_{29}$, 34 bits $\tilde{c}_0 - \tilde{c}_{33}$ qui sont le reste de la division de $D^{34} \cdot \tilde{x}(D)$ par $g(D)$. $g(D)$ vaut 260534236651_{octal}, le chiffre le plus à gauche est le MSB.

4. Au mot de 64 bits obtenu au pt.3, on additionne modulo 2, le bruit pseudo-aléatoire du pt. 2. Cette fois on utilise les bits P_0 à P_{63}.

5. Le résultat de l'addition au pt 4 nous donne le *Sync Word* à envoyer.

On remarque que l'adresse LAP et le Barker code apparaissent en clair dans le *Sync Word*. Le but des pt.2 à 5 n'est que de calculer les bits de redondance c_0 à c_{33}.

Un exemple complet de calcul du *Sync Word* à partir d'une adresse se trouve en annexe 7.14 Exemple complet de calcul du *Sync Word*, page 143.

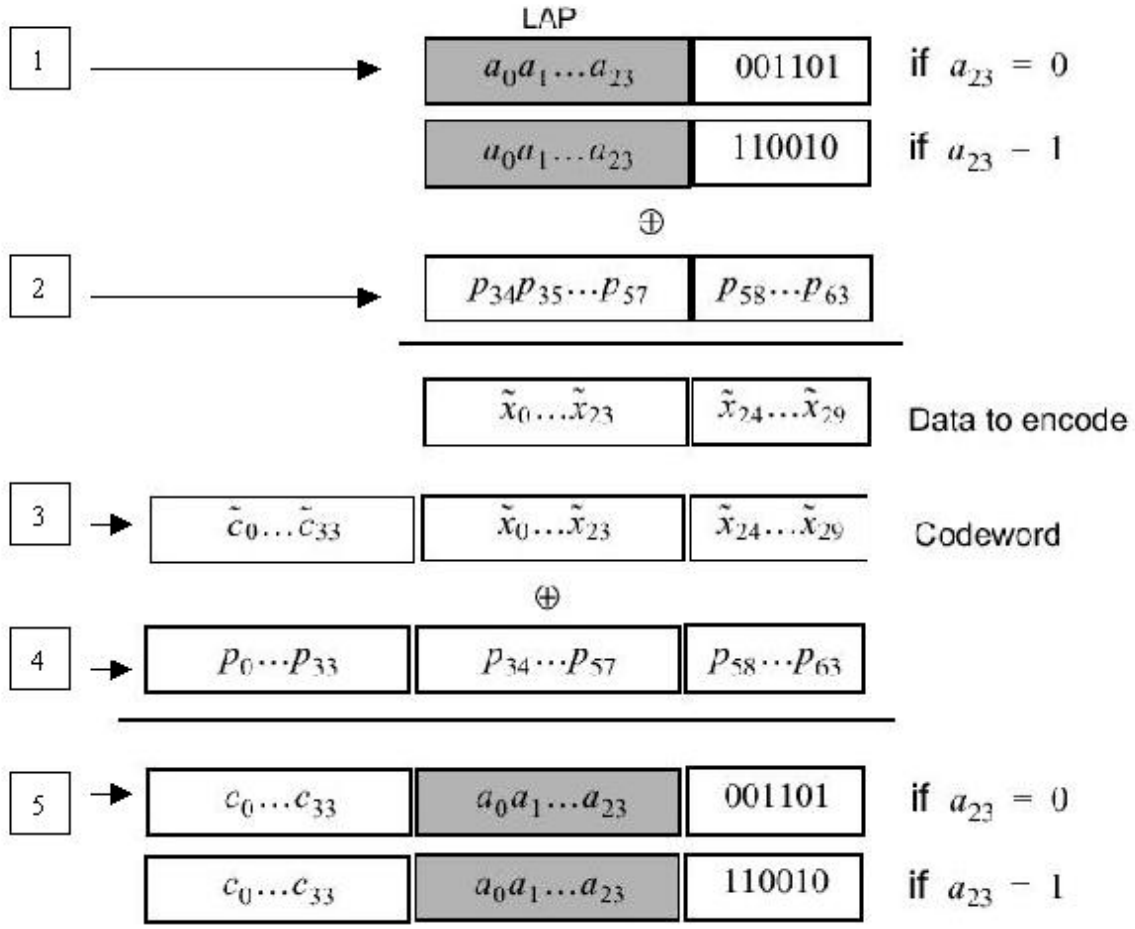


Figure 2-1 Calcul du *Sync Word*

Trailer

Le fanion de fin (*Trailer*) est ajouté au *Sync Word* seulement si un *Header* suit l'*Access Code*. Le fanion est une suite de 4 symboles. Les 4 bits du *Trailer* et les 3 MSB du mot de *Sync Word* créent une séquence de 7 bits qui comme dans le fanion de tête peut-être utilisée pour diminuer la composante continue. La séquence du fanion dépend du MSB du *Sync Word*.

Figure 2-1 Fanion de queue lorsque le MSB du mot de sync est à 0 (a) ou lorsqu'il est 1 (b).

2.3.2. Champs Header

Le *Header* contient 6 champs d'informations. La taille du *Header* est de 18 bits encodés avec le FEC 1/3 ce qui crée un paquet de 54 bits.

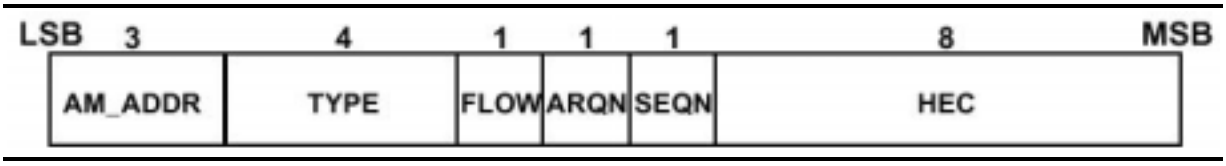


Figure 2-1 Format de l'en-tête.

AM_ADDR: Active Membre Address, ce champ de 3 bits, contient l'adresse temporaire de l'esclave, LSB en premier. On utilise son AM_ADDR dans chaque communication entre le maître et l'esclave ou entre l'esclave et le maître. Cette adresse de 3 bits permet de distinguer les différents appareils Bluetooth actifs dans le piconet. L'AM_ADDR est temporaire. Chaque appareil Bluetooth obtient une AM_ADDR lorsqu'il se trouve dans l'état **Actif** et la libère lorsqu'il se déconnecte ou passe dans l'état **Park**.

L'adresse 000_b est une adresse de Broadcast, sauf si elle se trouve dans l'en-tête d'un paquet FHS. Dans ce cas, il s'agit alors d'un paquet FHS destiné à un appareil auquel on n'a pas encore assigné d'AM_ADDR.

Type: Ce champ de 4 bits définit le type de paquet envoyé. La Table 2-1 indique les codes de chaque paquet.

Table 2-1 Code des paquets utilisés par le champ Type

Code TYPE $b_3b_2b_1b_0$	Type de paquet
0000	Null
0001	Poll
0010	FHS
0011	DM1
0100	DH1
0101	HV1
0110	HV2
0111	HV3

Code TYPE $b_3b_2b_1b_0$	Type de paquet
1000	DV
1001	AUX1
1010	DM3
1011	DH3
1100	Réserve
1101	Réserve
1110	DM5
1111	DH5

Flow: bit de régulation de flux pour une transmission *Asynchronous Connection-Less Link* (ACL). Régulation de flux au niveau de la Baseband.

Flow = 0 indique que le buffeur de réception est plein. Arrête temporairement la transmission.

Le contrôle de flux ne concerne que les paquets ACL, les *paquets Synchronous Connection-Oriented Link* (SCO) et les paquets de contrôle (NULL, POLL, FHS, DM1) ne sont pas concernés et peuvent toujours être transmis.

Il existe aussi un champ *Flow* dans le *Payload* des paquets ACL, mais ce champ contrôle le flux par canal logique au niveau L2CAP. Un canal logique ACL peut arrêter de transmettre pendant que les autres canaux logiques ACL continuent de transmettre normalement. Alors que la régulation de flux dans l'en-tête est commune à tous les canaux logiques.

ARQN: Bit d'acquittement, le contrôle se fait à l'aide du champ CRC.

Il indique au récepteur de ce bit, que le message qu'il a envoyé a bien été reçu.

- 1 : ACK, Indique à la source que le champ *Payload* a été correctement reçu;
- 0 : NAK, Indique que le paquet reçu est erroné.
- Pas de réponse indique que le paquet reçu est erroné, NAK.

Le bit d'ARQN est transmis dans l'en-tête du paquet de retour qui suit le paquet reçu.

SEQN: Bit de numérotation des paquets. On utilise une numérotation modulo 2 qui permet de faire la distinction entre messages pairs et impairs et ainsi de repérer les messages répétés.

HEC (*Header Error Check*) : Ce champ de 8 bits permet de détecter les erreurs de transmissions dans l'en-tête.

2.3.3. Champs de data

On peut distinguer deux champs principaux:

- Le champ réservé à la voix (transmission synchrone);
- Le champ réservé aux données (transmission asynchrone).

Les paquets ACL ne contiennent que le champ de données et les paquets SCO ne transmettent que le champ voix, avec une exception pour le paquet DV qui transporte la voix et les données.

Champ voix

Ce champ est de longueur fixe, 240 bits pour un paquet HV et 80 bits pour un paquet DV.

Champ data

Ce champ est divisé en trois parties, l'en-tête (*Payload Header*), les données (*Payload*) et le CRC.

Le *Payload Header* contient le numéro du canal logique (L_CH) sur 2 bits, un contrôle de flux du canal logique (FLOW) sur 1 bit et l'indication de la taille des données.

Le nombre de bits alloué à l'indication de la taille des données peut varier. Il est de 5 bits pour un paquet envoyé sur un slot ou de 9 bits si le paquet est transmis sur plusieurs slots.

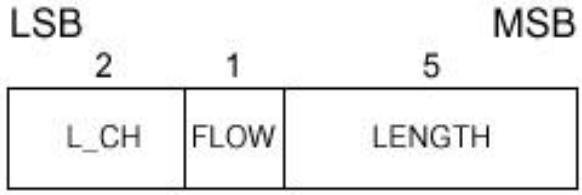


Figure 2-1 En-tête du *Payload* d'un paquet transmis dans un seul intervalle de temps.

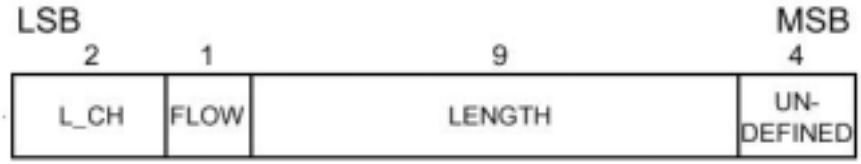


Figure 2-2 En-tête du *Payload* d'un paquet transmis sur plusieurs intervalles de temps.

Le *Payload* contient les informations à transmettre. Sa longueur est indiquée dans *Payload Header*.

Le contrôle de flux transporté dans ce paquet régule le flux au niveau du canal logique, contrairement au contrôle de flux du *Header* qui régule le flux du canal ACL physique.

Le champ CRC contient 16 bits qui sont calculés en fonction du champ de données.

2.4. Les Paquets

Il existe 15 types de paquets. Il y a 4 paquets de contrôle (POLL, NULL, FHS, DM1) qui servent au contrôle du canal et 12 paquets utilisés pour le transport des données. De plus il existe un paquet, le paquet ID qui est utilisé pour les procédures d'*Inquiry* et de *Page*.

Ci-dessous sont décrits en détail les deux paquets les plus importants pour le fonctionnement de notre analyseur. Il s'agit du paquet utilisé pour annoncer un *Inquiry*, le paquet ID et du paquet utilisé pour transmettre la synchronisation et répondre aux *Inquiry*, le paquet FHS.

2.4.1. Paquet ID

Le paquet ID contient l'*Access Code* raccourci de 68 bits (*Preamble* et *Sync Word*). Le *Sync Word* est calculé à partir du DAC (*device Access Code*, c'est l'adresse de l'appareil auquel on veut se connecter) ou de l'IAC (*Inquiry Access Code*, code réservé aux *Inquiry* comme le GIAC ou le DIAC). C'est un paquet très robuste aux erreurs de transmissions car le récepteur utilise un corrélateur pour déterminer l'arrivée d'un paquet ID. Ce paquet est utilisé pour l'émission d'un *Inquiry* et dans l'émission et la réponse à un *Page*.

2.4.2. Paquet FHS

Le paquet FHS est un paquet de contrôle qui contient des informations importantes sur l'émetteur du paquet. Le *Payload* contient 144 bits d'informations plus un code CRC de 16 bits. Le *Payload* est en plus protégé par un code FEC 2/3 qui amène la taille du *Payload* à 240 bits ([144+16]*3/2=240).

Ce paquet est utilisé dans les réponses aux *Page* et aux *Inquiry* ainsi que pour les échanges des rôles maître-esclave. Le champ CLK_{27-2} de ce paquet contient une partie de l'horloge de l'émetteur. Ce champ est mis à jour à chaque transmission du paquet. Le paquet FHS est utilisé pour la synchronisation de la séquence de saut de fréquence avant l'établissement du piconet ou lorsqu'un piconet existant change de maître.

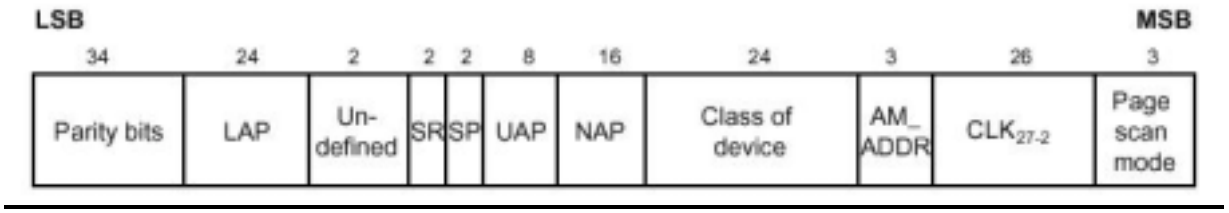


Figure 2-1 Format du *Payload* du paquet FHS

Parity bits

Ce champ de 34 bits contient les bits de redondance qui forment la première partie du *Sync Word* de l'*Access Code* de l'appareil qui a envoyé ce paquet.

LAP (Lower Adresse Part)

Ce champ contient les 24 bits de la partie inférieure de l'adresse Bluetooth BD_ADDR.

Undefined

Ce champ de 2 bits est réservé à des utilisations futures.

SR (Scan Repetition Field)

Ce champ de 2 bits indique l'intervalle entre deux états **Page Scan**.

SP (Scan Periode Field)

Ce champ de 2 bits indique la période pendant laquelle l'appareil se trouve dans l'état **Page Scan** après avoir répondu à un *Inquiry*.

UAP (Upper Address Part)

Ce champ de 8 bits contient la partie supérieure de l'adresse Bluetooth BD_ADDR de l'appareil qui envoie ce paquet.

NAP (Non-significant adresse part)

Ce champ de 16 bits contient la partie non-significative de l'adresse Bluetooth BD_ADDR.

Class of device

Ce champ de 24 bits indique la classe de l'appareil qui envoie ce paquet FHS. Les classes d'appareils ne sont pas encore normalisées.

AM_ADDR (Active Membre Address)

Ce champ de 3 bits contient l'adresse de membre actif (AM ADDR) de l'appareil qui a envoyé ce paquet FHS.

CLK_{27-2}

Contient les bits 2 à 27 de l'horloge CLK de l'émetteur à l'instant du début de la transmission de l'*Access Code* de ce paquet FHS. Ce champ est mis à jour à chaque transmission.

Page Scan Mode

Ce champ de 3 bits indique quel *Scan Mode* est utilisé par défaut par l'émetteur de ce paquet FHS.

Le LAP, UAP et NAP forment l'adresse BD_ADDR de l'appareil qui a envoyé le paquet FHS.

Les champs *Parity* et LAP permettent de créer directement l'*Access Code* de l'émetteur de ce paquet.

2.5. Interface HCI

Le Host Controller Interface (HCI) décrit une interface permettant d'accéder au Hardware Bluetooth de manière uniforme. L'interface HCI reçoit des *HCI Command Packets* et retourne des *HCI Event Packets*. Les commandes HCI permettent de contrôler les connexions avec un autre appareil et de modifier le comportement de la Baseband de l'hôte.

Les commandes HCI peuvent ne prennent pas toutes le même temps pour s'effectuer. C'est pourquoi le résultat d'une commande sera retourné à l'hote sous la forme d'un *HCI event*.

La liste complète des *HCI commands* et des *HCI events* ainsi que la description de chaque paramètre se trouve dans [3].

La description précise des commandes HCI à envoyer pour un *Inquiry* et pour créer une connexion se trouve en annexe 7.1.

2.6. Inquiry

La procédure *Inquiry* permet de découvrir les appareils qui sont dans le rayon d'action d'un appareil Bluetooth. Cette procédure est utilisée lorsque l'adresse du ou des appareils recherchés est inconnue. Après cette procédure l'émetteur de l'*Inquiry* peut s'il le désire, démarrer une connexion avec un des appareils découverts, grâce à la procédure *Page.*

Dans l'état **Inquiry**, l'initiateur émet une suite de paquet ID, contenant le GIAC s'il désire contacter tous les appareils ou un DIAC s'il désire contacter seulement un certain type d'appareils. Ces paquets sont envoyés sur une séquence de 32 fréquences.

Les appareils qui reçoivent le paquet ID contenant le GIAC ou un DIAC correspondent à leur classe d'appareils, répondent en envoyant un paquet FHS. Ce paquet contient, comme décrit au paragraphe 2.4.2, l'horloge de l'émetteur, son adresse BD_ADDR ainsi que les bits *Parity*.

Le paquet FHS n'est pas acquitté par l'initiateur de l'*Inquiry*. De même, le récepteur du paquet ID peut répondre plusieurs fois dans une même procédure *Inquiry* puisque l'initiateur émet plusieurs paquets ID.

2.6.1. Inquiry scan

Dans l'état **inquiry scan**, l'appareil se trouve sur une fréquence fixe à l'écoute d'un paquet ID contenant le GIAC ou le DIAC de sa classe d'appareils. Il y a 32 fréquences possibles d'écoute. Il détermine sa fréquence d'écoute selon son horloge interne. Pour détecter un paquet ID, il utilise un *Sliding Correlator*. Lorsqu'il reçoit un paquet ID qui lui correspond, l'appareil a l'obligation de répondre, pour cela, il passe en mode **Inquiry Response.**

2.6.2. Inquiry Response

La réponse au paquet ID reçu, est envoyée sous la forme d'un paquet FHS.

Un problème de collision peut se produire si plusieurs appareils répondent à l'*inquiry* en même temps. Les appareils ont tous une horloge CLK différente, il est donc peu probable qu'ils répondent en même temps puisqu'ils écoutent sur des fréquences différentes.

Mais pour éviter ce problème, on utilise le mécanisme suivant: lorsque l'appareil reçoit un paquet ID, il génère un nombre RAND aléatoire compris entre 0 et 1023; de plus, il mémorise la valeur de son horloge CLK. Ensuite, l'esclave retourne en mode **Connection** ou **Standby** pour RAND *slots*. Après au minimum RAND *slots*, l'appareil retourne en mode **Inquiry Response.** Si à ce moment il ne reçoit pas de paquet ID, il retourne en mode **Connection** ou **Standby.** Par contre, s'il reçoit un paquet ID, il répond immédiatement par un paquet FHS, et passe à la prochaine fréquence d'écoute dans la suite des 32 fréquences d'*Inquiry*. A ce moment il recommence à attendre pour un paquet ID.

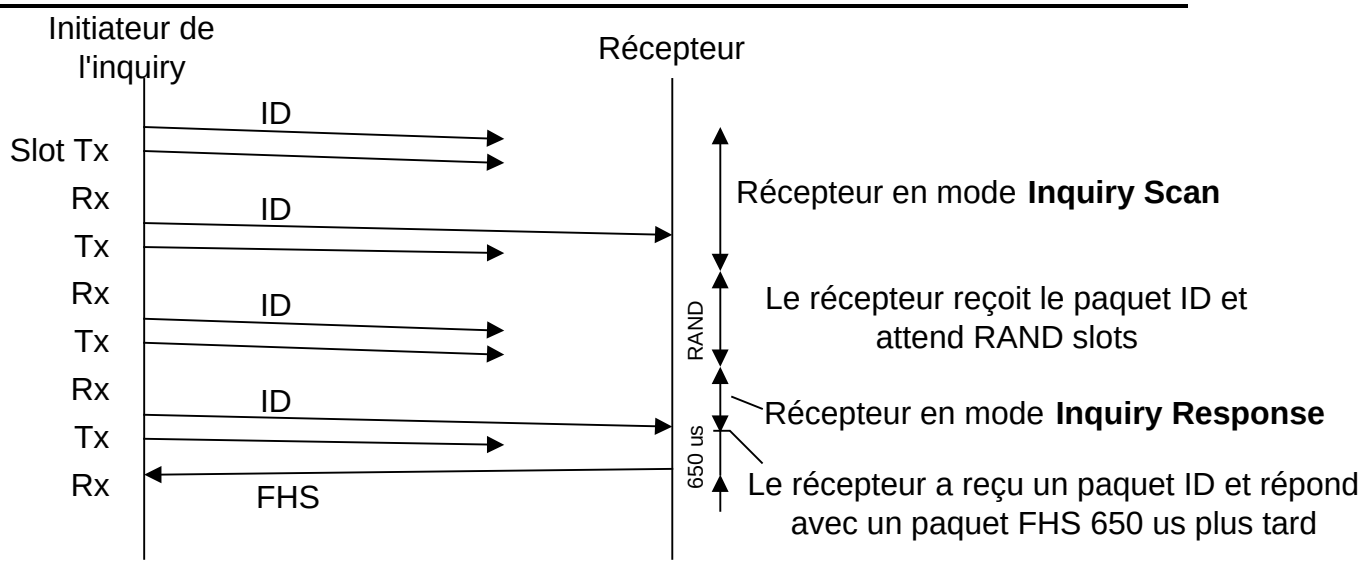


Figure 2-1 Diagramme en flèche d'une procédure Inquiry

2.6.3. Inquiry

L'unité Bluetooth qui démarre un *Inquiry*, émet des paquets ID rapidement et sur un grand nombre de fréquence pour augmenter ses chances de contacter les autres unités. Les paquets ID sont très courts, 68bits, il est donc possible d'augmenter la vitesse des sauts de fréquence de 1600 sauts/s à 3200 sauts/s.

Timing Rx/Tx

La partie de la norme qui explique les *timings* de transmission (Tx) et de réception (Rx) n'est pas claire. Pour s'assurer des explications données ci-dessous, des mesures ont été faites. Ces mesures sont décrites au chap 5.3.7 Mesure des timings Rx/Tx d'un Inquiry, page 81.

Dans un *slot Tx* de 625µs, l'unité envoie un paquet ID sur les fréquences $f_{(k)}$ et $f_{(k+1)}$. Dans un *slot Rx*, l'appareil écoute sur les fréquences $f'_{(k)}$ et $f'_{(k+1)}$ dans l'attente de paquets FHS.

L'esclave qui répond à un paquet ID renvoie un paquet FHS exactement 625µs après la réception du paquet ID auquel il répond.

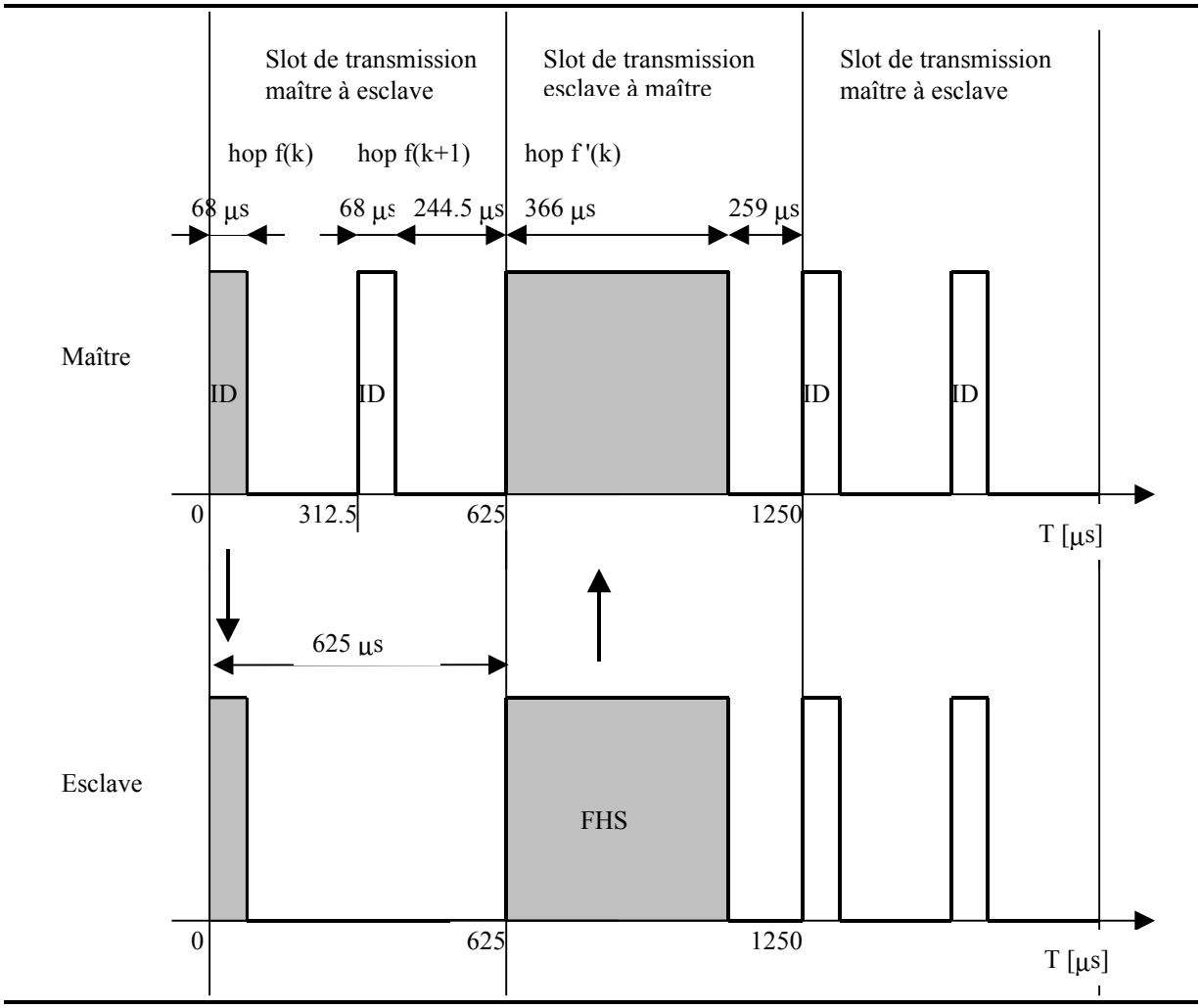


Figure 2-1 Timing d'un Inquiry avec un paquet ID reçu dans la première moitié du slot.

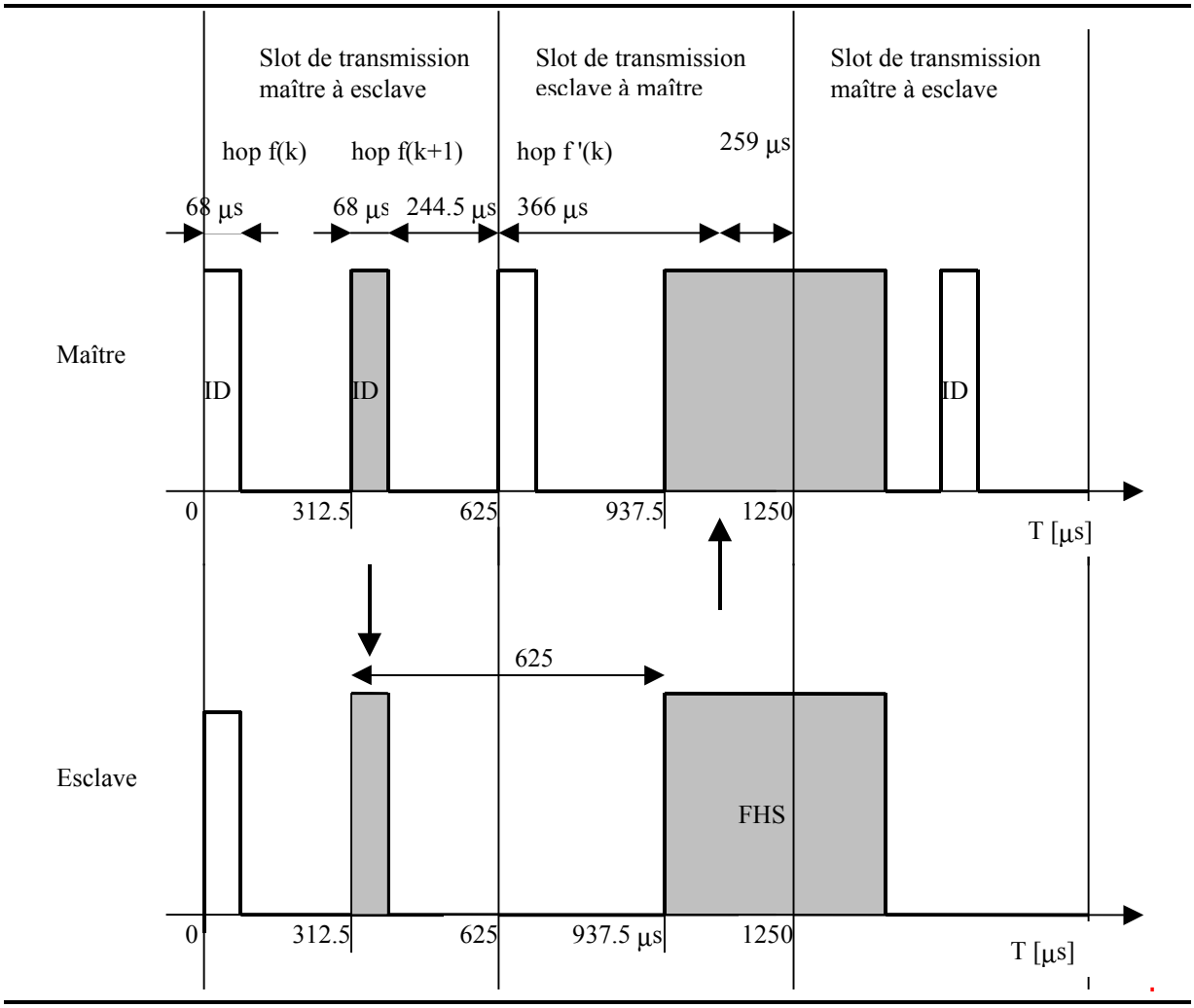


Figure 2-2 Timing d'un Inquiry avec un paquet ID reçu dans la seconde moitié du slot

La Figure 2-2 montre ce qui se passe lorsque le paquet ID est reçu dans la deuxième moitié du slot. Le paquet FHS retourné, empiète sur le *slot* Tx du maître et l'empêche d'émettre le paquet ID de la première moitié de ce slot.

Fréquences utilisées

La séquence d'émission des paquets ID contient 32 fréquences. Ces 32 fréquences sont divisées en deux groupes, A et B.

- Le groupe A contient les fréquences $f_{(k-8)}$, $f_{(k-7)}$,...$f_{(k)}$,...$f_{(k+7)}$
- Le groupe B contient les fréquences $f_{(k-15)}$, $f_{(k-15)}$,..., $f_{(k-9)}$,$f_{(k+8)}$

K est déterminé par l'horloge CLK_{16-12}de l'initiateur de l'*Inquiry*.

L'initiateur commence par émettre sur les fréquences de la série A. Il utilise 256 fois au minimum cette série. Après cela, il passe à la série B, qu'il parcourt aussi au minimum 256 fois.

Pour permettre une réception sans erreur, un *Inquiry* contient au minimum 4 séries A et 4 séries B parcourues alternativement.

Une procédure *Inquiry* dure 10.24s à moins de recevoir assez d'adresses et de décider d'arrêter l'*Inquiry*.

Durée d'une série, :	$t_{série}$=(312.5µ*2) * 16 = 10ms
Chaque série est répétée 256 fois	$t_{série}$*256 = 2.56s
Un *Inquiry* comporte 4 séries	2.56 *4 = 10.24s

Figure 2-1 Calcul de la durée d'un Inquiry

L'adresse utilisée pour le calcul des fréquences est la partie LAP du GIAC pour les bits (A_{23}-A_0) et les 4 LSB du *Default Check Initalization* (DCI) pour les bits A_{27}-A_{24}.

Le DCI a la valeur : `0x00`

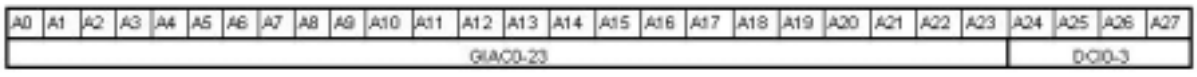


Figure 2-2 Adresse utilisée pour la génération de la séquence de saut de l'Inquiry

Paquets ID

Les paquets ID utilisés pour l'*Inquiry* sont les paquets ID standard. Le *Sync Word* de l'*Access Code* est calculé sur la base du GIAC ou du DIAC.

- Le GIAC vaut : `0x9E8B33`.
- Les adresses réservées pour les DIAC sont comprises entre `0x9E8B00` et `0x9E8B3F`.

Toutes ces adresses sont données avec le LSB à droite

Paquets FHS

Les paquets FHS reçu sont décrits au chap. 2.4.2 Paquet FHS, page29.

L'*Access Code* de ces paquets sont générés en utilisant le GIAC ou le DIAC utilisé dans le paquet ID émis par l'initiateur de l'Inquiry.

3. Analyse

3.1. Objectifs

L'objectif de cette analyse est de décrire précisément la Baseband d'un analyseur de réseau Bluetooth. Cet analyseur devra recevoir les paquets transmis entre le maître du piconet et ses esclaves. Le contenu de ces paquets sera décodé puis transmis à un PC pour être traité.

L'analyseur suivra la séquence de saut du piconet, pour cela il devra être synchronisé avec le maître.

3.2. Particularité de l'analyseur de réseau Bluetooth.

Un analyseur de trafic Bluetooth ne fonctionne pas de la même façon qu'un analyseur de protocole pour réseau câblé. Un réseau Bluetooth utilise l'air comme média de transmission, il utilise donc des mécanismes particuliers à ce genre de réseaux.

3.2.1. Le canal de transmission

Le canal de transmission utilisé est formé d'une suite de saut de fréquence dont la séquence est déterminé par l'horloge et l'adresse du maître. Sans ces deux paramètres, il ne sera pas possible d'analyser un canal physique puisqu'on ne pourra pas recevoir tous les paquets transmis sur ce canal.

Si l'on veut analyser le trafic d'un piconet Bluetooth, il est obligatoire de se synchroniser avec le maître du piconet pour connaître son adresse et son horloge.

3.2.2. Décodage des paquets

L'air qui est le média de transmission choisi, contient beaucoup d'interférences qui peuvent détruire ou modifier les paquets transmis. Pour protéger ces paquets et les rendre plus résistants aux interférences, les paquets envoyés sur le canal de transmission sont *scramblés* (*whitening*) et codés. Le *whitening* diminue la composante continue et le codage permet de détecter et parfois corriger les erreurs de transmission. Malheureusement, l'algorithme de *whitening* est initialisé avec une partie de l'horloge du maître. Il faut donc connaître l'horloge du maître pour effectuer le *de-whitening*.

Le *Header* des paquets contient un HEC qui permet de détecter les erreurs de transmission. L'algorithme qui génère le HEC est initialisé avec la partie UAP de l'adresse du maître du piconet. Il faut connaître cette partie si on veut vérifier le *Header*. En utilisant l'UAP de l'adresse du maître du piconet, le domaine d'adressage des appareils Bluetooth est agrandit. Deux piconet dont les maîtres ont des adresses LAP identiques mais des parties UAP différentes ne mélangeront pas leurs paquets. Les paquets émis dans un piconet seront ignorés par les appareils de l'autre piconet lorsque leur Baseband vérifiera le *Header* de ces paquets.

3.3. Fonctionnement global de l'analyseur

Pour répondre aux particularités de ce réseau, la synchronisation de l'analyseur sur le maître est obligatoire.

3.3.1. Propositions de synchronisation

Pour se synchroniser, l'analyseur devra interroger le maître pour qu'il lui envoie son horloge, c'est le moyen le plus rapide et le plus simple. Mais cela impose que l'analyseur se fasse connaître du maître. L'analyseur devra intervenir dans le piconet et le maître devra accepter de répondre à cette demande de synchronisation.

Il existe d'autres moyens. Si, par exemple, l'adresse du maître est connue, il est possible de calculer la séquence de saut. La deuxième étape consisterait à chercher où se trouve le maître dans cette séquence.

L'horloge Bluetooth est composée de 28 bits qui s'incrément à une fréquence de 3.2kHz. L'horloge fait un tour complet de la séquence de saut en, 2^{28}*1/3.2 k= 23h 18min 6sec.

Si l'analyseur est placé sur une fréquence connue f_k et qu'il calcul la fréquence f_{k+1}. A chaque détection d'un paquet sur la fréquence f_k l'analyseur passe à la fréquence f_{k+1}, puis s'il détecte à nouveau un paquet il continue à la fréquence f_{k+2} et ainsi de suite. Si par contre, ce qui sera le cas le plus fréquent, il ne détecte rien à f_{k+1}, il retourne à f_k et recommence son attente.

L'horloge ayant une période de 24h environ, l'analyseur devrait avoir trouvé la synchronisation au plus tard après 24h. Ce système oblige le piconet a fonctionner 24h au moins pour que l'analyseur puisse s'y synchroniser. Ce qui n'est pas vraiment prévu pour Bluetooth puisqu'il s'agit d'un protocole imaginé pour connecter et déconnecter facilement deux appareils. Les connexions seront donc en générale, relativement courtes.

Pour contourner ce problème, il faudrait étudier le fonctionnement d'appareils Bluetooth du commerce. Il est possible que leur horloge recommence à 0 ou à une valeur fixe à chaque remise en route, ce qui permettrait de trouver plus rapidement leur phase.

3.3.2. Solutions retenues

Pour cet analyseur, la première possibilité a été retenue. Elle impose une interaction avec le maître, mais elle a l'avantage d'être beaucoup plus rapide. Comme le but n'est pas de créer un espion Bluetooth, il n'est pas gênant de devoir communiquer avec le maître.

Le fonctionnement de l'analyseur de trafic Bluetooth est divisé en trois phases distinctes qui sont exécutées séquentiellement.

- Dans la première phase, l'analyseur détermine les adresses des maîtres des différents piconets présents sur le média.
- Dans la deuxième phase, l'analyseur se synchronise sur un des piconets.
- Finalement lorsque l'analyseur est synchronisé avec le piconet, il commence à capter les paquets transmis entre le maître et ses esclaves.

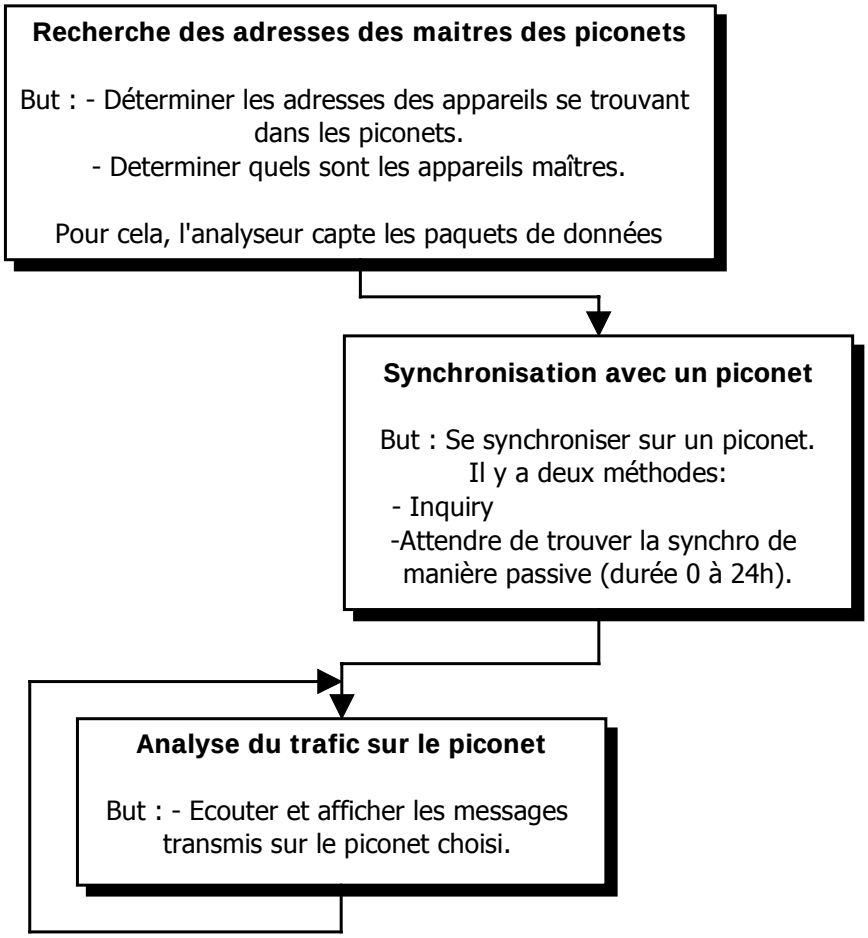


Figure 3-1 Fonctionnement général de l'analyseur de trafic

3.4. Détection des adresses des maîtres des piconets

3.4.1. Introduction

Cette étape permet à l'analyseur de connaître les adresses des maîtres des différents piconets. Ces adresses seront utilisées dans la prochaine phase pour synchroniser l'analyseur sur le piconet choisi.

Chaque paquet transmis dans le piconet comporte en en-tête l'adresse LAP du maître du piconet. Ceci permet de différencier les paquets dans le cas où il y aurait plusieurs piconets qui utiliserait le même canal au même instant. Les adresses LAP se trouvent dans les bits 38 à 61 de l'*Access Code* comme indiqué sur la Figure 3-1.

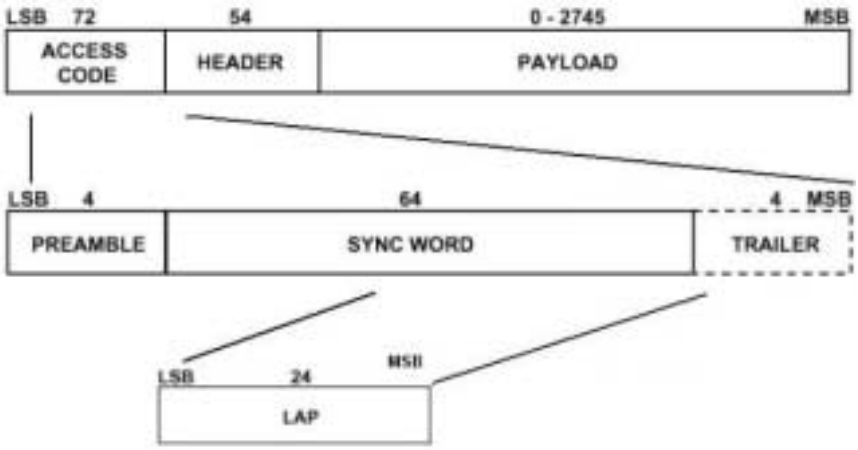


Figure 3-1 Emplacement de l'adresse LAP dans un paquet de données

L'analyseur va capter les paquets de données pour en extraire l'adresse. Pour cela, il va se mettre en écoute sur une fréquence donnée pendant 500 ms (ce temps est déterminé au paragraphe 3.4.6) et capturer tous les paquets transmis sur ce canal. A ce moment l'analyseur ne peut pas suivre une transmission complète, il ne fait que recevoir de manière aléatoire les paquets du piconet qui sont transmis sur la fréquence qu'il écoute.

3.4.2. Fréquences utilisées

Le choix de la fréquence sur laquelle l'analyseur écoute n'est pas très important, les canaux ayant une équiprobabilité d'être utilisés. Mais il se peut que la fréquence choisie soit brouillée par un autre appareil. C'est pourquoi l'analyseur teste le nombre d'adresse trouvé après 500ms. S'il n'a pas trouvé d'adresse, il va choisir une autre fréquence. Cette fréquence devra si possible être assez éloignée de la fréquence précédente. Si une fréquence est brouillée, il est probable que les fréquences adjacentes le seront aussi, il est donc inutile de rester dans cette région. On aura donc plus de chance de trouver une fréquence non brouillée si l'on fait un grand saut fréquence.

Une suite de fréquence possible pourrait être la suite de fréquence utilisée dans l'état **Connection.** Cette suite de fréquence est décrite dans [3] page 939.

Pour éviter que l'analyseur ne cherche indéfiniment une adresse alors qu'il n'y a pas de piconet en fonction, un temps maximum de recherche est défini, *timeOut*, après quoi l'appareil cesse de rechercher un paquet.

3.4.3. Programmation du transceiver en réception

La programmation du transceiver se fait en envoyant deux mots sur le bus série du transceiver. La description exacte de ces deux mots se trouve au chap. 5.2.2 Programmation du transceiver, page 72.

Calcul des valeurs de MC et SC

Connaissant la fréquence sur laquelle l'analyseur veut écouter, il faut calculer les valeurs des compteurs MC et SC qui contrôle la fréquence d'émission du Transceiver.

Soit un registre de 9 bits appelé `Compteur`. Ce registre est chargé avec la valeur de la fréquence désirée moins 2048. Les bits de $compteur_{0-4}$ contiennent la valeur de `SC` et les bits $compteur_{5-8}$ contiennent la valeur de `MC`.

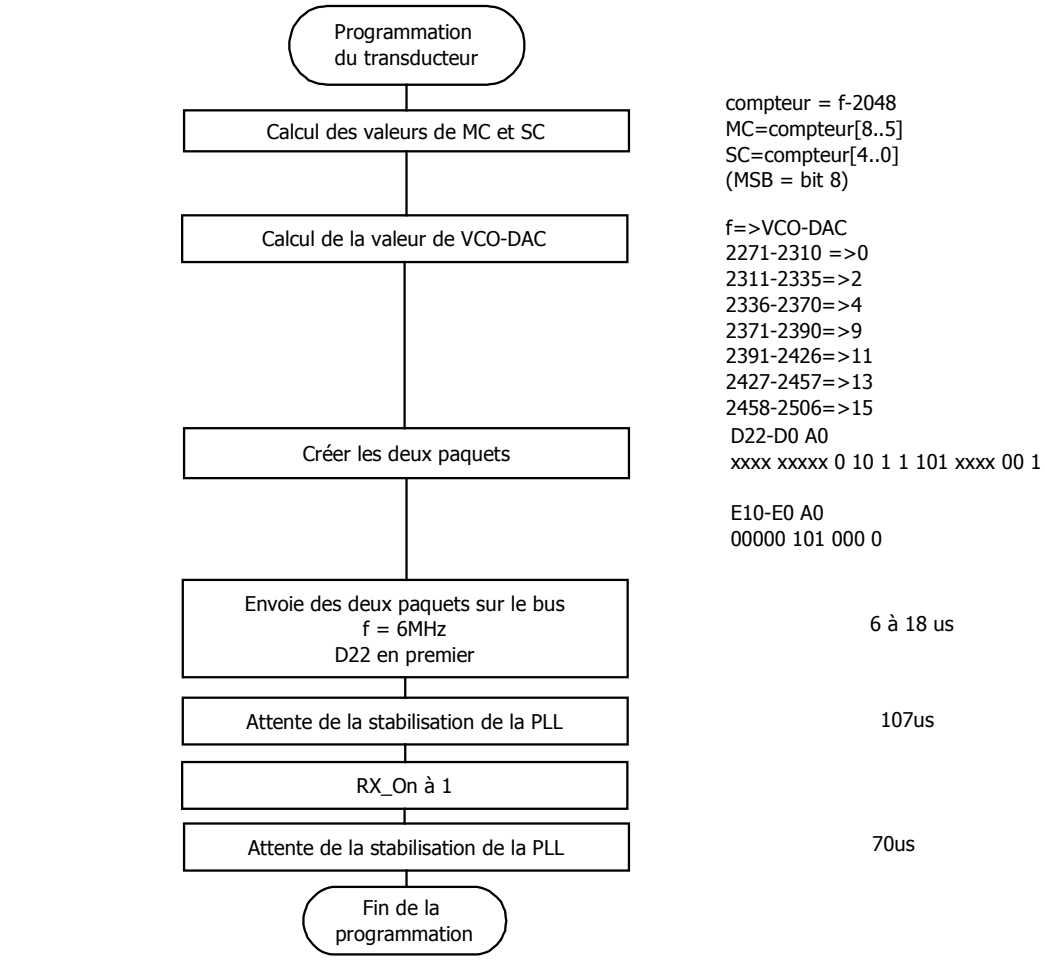


Figure 3-1 Structogramme de la programmation du transceiver en réception

Calcul de la valeur de VCO-DAC

La valeur du VCO-DAC peut être trouvée à l'aide de la Table 3-1.

Table 3-1 Correspondance entre fréquence de réception et VCO-DAC

Fréquence désirée	Valeur VCO-DAC [dec]
2271 – 2310	0000
2311 – 2335	0010
2336 – 2370	0100
2371 – 2390	1001
2391 – 2426	1011
2427 – 2457	1101
2458 – 2506	1111

Créer les deux paquets

Format des deux paquets à envoyer

Table 3-1 Mot 1

D22	D21	D20	D19	D18	D17	D16	D15	D14	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0	A0
MC				**SC**					**PS**	**PA**		**GF**	**MCC**	**GFCS**			**VCO-DAC**				**CPCS**		**1**
									0	1	0	1	1	1	0	1					0	0	1

MC, SC et VCO-DAC contiennent les valeurs calculées précédemment.

Table 3-2 Mot 2

												E10	E9	E8	E7	E6	E5	E4	E3	E2	E1	E0	A0
												DEMODDAC					**MCCS**			**TEST**			**0**
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	0	0	0

Envoi des données sur le bus

Les deux mots sont transmis sur le bus. D22 est émis en premier.

Entre chaque mot, le signal *Enable* repasse à 1 pendant 231ns au minimum.

La Figure 3-1 représente une transmission sur le bus série du transceiver. Cette transmission dure de 6 à 18 μs selon la fréquence de l'horloge du bus.

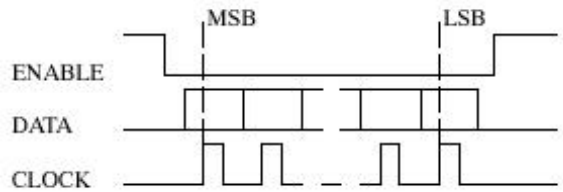


Figure 3-1 Chronogramme du bus de programmation du transceiver.

Attente de la stabilisation de la PLL

A la fin de l'émission des données sur le bus, il faut attendre 107μs avant de mettre le signal Rx_on à 1. Puis après 70μs, le transceiver est prêt à recevoir les données.

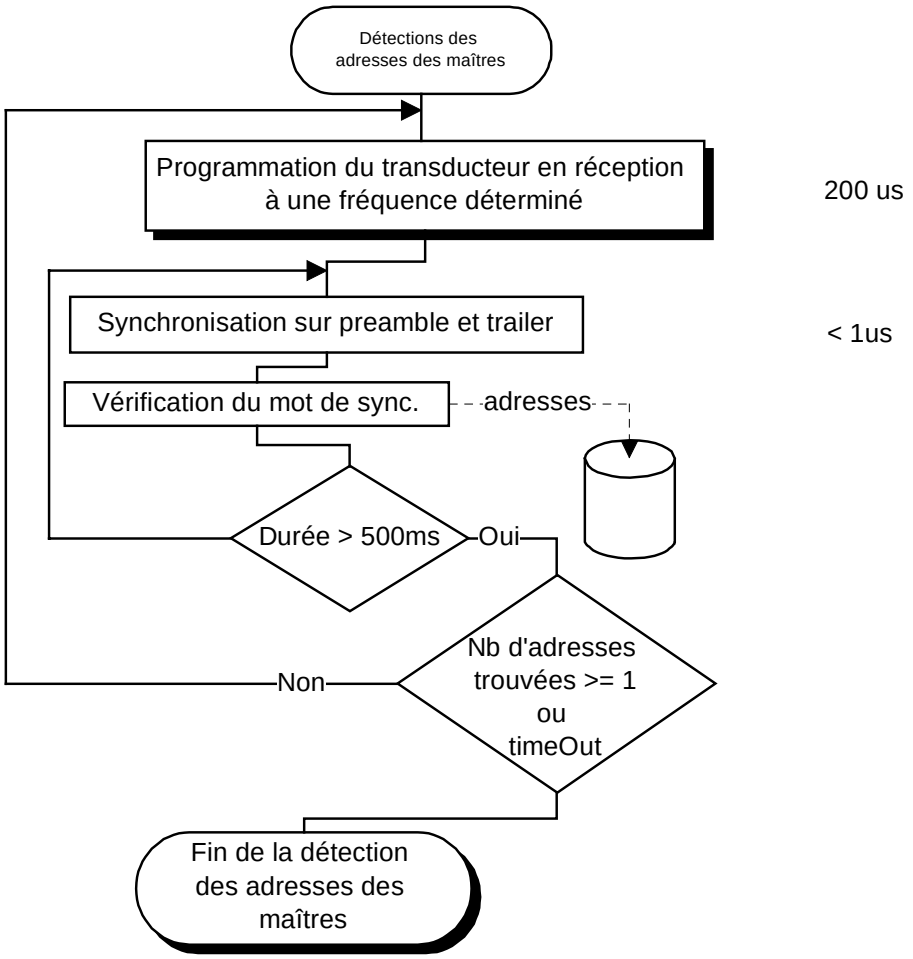


Figure 3-1 Fonctionnement de la recherche d'adresses des maîtres.

3.4.4. Synchronisation des paquets reçus

En fonctionnement normal, la détection de début d'un paquet se fait en utilisant un *sliding correlator* qui recherche l'*Access Code.* Cette détection déclenche le traitement du paquet lorsqu'un certain niveau de similitude entre le mot reçu et le mot recherché est atteint. Un analyseur ne connaît pas l'adresse utilisée pour générer le *Sync Word*. Il recherchera donc seulement les parties de l'*Access Code* qui ne dépendent pas de l'adresse. Il s'agit du *Preamble*, du *Trailer* et des 6 derniers bits de *Sync Word*.

- Il existe deux *Preamble* différents en fonction du LSB du *Sync Word*. Il faut donc rechercher la suite 10101_b ou 01010_b.

- Le *Trailer* dépend du MSB du *Sync Word*. Il faut donc rechercher 01010 ou 10101_b.

- Les 7 derniers MSB du *Sync Word* sont un code de Barker. Cette séquence peut être 0001101_b ou 1110010_b selon le MSB de l'adresse utilisée.

- Puisque le *Trailer* dépend du *Sync Word*, la fin du paquet peut être soit 00011010101_b ou 11100101010_b.

En synchronisant sur ces 16 bits, tous les paquets sont détectés à l'exception du paquet ID qui ne contient pas de *Trailer*. Mais le paquet ID n'est pas intéressant car il ne contient jamais l'adresse du maître du piconet, comme le montre la Table 3-1.

Table 3-1 Adresses utilisées pour le *Sync Word* du paquet ID.

Utilisation du paquet ID	Adresse utilisée pour le *Sync Word*
Inquiry	Adresses réservées pour l'inquiry (GIAC et DIAC)
Page	Adresse de l'appareil auquel l'initiateur du Page veut se connecter

Contrainte de temps

Le débit de transmission de la ligne est de 1Mb/s, un bit arrive donc tous les 1μs. L'algorithme de synchronisation des paquets vérifie à chaque bit reçu s'il s'agit d'un début de paquet. Pour ne pas prendre de retard, cet algorithme devra s'effectuer en moins d'1μs.

Probabilité de faux départ dû au nombre restreint de bits testés

Puisque seul 15 bits sur les 72 bits de l'*Access Code* sont testés, il y a des risques de détection erronée de début de trame. Cette probabilité est de

$$p_{(\text{faux départ})} = \frac{1}{2^{15}} = 30.5 \cdot 10^{-6}$$

3.4.5. Vérification du Sync Word

Le but de la vérification du *Sync Word* est de diminuer le taux de faux départ en s'assurant que le *Sync Word* contient une valeur plausible. Comme expliqué au chapitre 2.3.1, le *Sync Word* est constitué de trois parties : 6 bits de Barker code, 24 bits d'adresse et 34 bits de redondance.

Les bits de redondance dépendent exclusivement des 24 bits d'adresse. Pour vérifier le *Sync Word* on soustrait aux 64 bits du *Sync Word* reçu le bruit pseudo-aléatoire qui a été rajouté lorsqu'il a été généré. Puis on divise ce mot par un polynôme $g^*(x)$. Si le reste de la division vaut 0, le *Sync Word* est correcte, sinon il contient une erreur et on élimine ce paquet.

Le polynôme $g^*(x)$ est le polynôme complémentaire de g(x). Il est créer selon la formule :

$$g^*(X) = X^{n-k} g(X^{-1})$$

Exemple de polynôme complémentaire avec un polynôme (7,4) :

$$g(x) = 1 + X + X^3 \Rightarrow 1011$$
$$g^*(X) = X^{n-k} g(X^{-1})$$
$$g^*(X) = X^{7-4} \cdot (1 + X^{-1} + X^{-3})$$
$$g^*(X) = X^3 + X^2 + 1 \Rightarrow 1101$$

On remarque que pour trouver le polynôme complémentaire de g(x) il suffit de croiser l'ordre des bits. Dans notre exemple les coefficients du polynôme g(X) sont 1011_b qui deviennent 1101_b pour g*(X).

Nous allons procéder de la même manière pour calculer le polynôme complémentaire.

Coefficients de g(X): `1001 0101 1011 1100 1000 1110 1010 0001 101`$_b$

Coefficient de g*(X) : `101 1000 0101 0111 0001 0011 1101 1010 1001`$_b$

Probabilité de faux départ :

Puisque 34 bits de plus sont testés, la probabilité de faux départ diminue. Elle sera de :

$$p_{(\ fauxdépart)} = \frac{1}{2^{15+34}} = 1.77 \cdot 10^{-15}$$

3.4.6. Calcul de la durée d'écoute sur un canal donné.

Le choix des fréquences utilisées par un piconet est un processus stochastique. Ce calcul permet de déterminer la probabilité que la fréquence choisie soit utilisée par le piconet après un temps To.

Il y a 79 fréquences différentes. La probabilité d'utilisation de chaque fréquence est identique.

$$p = \frac{1}{79}$$

Temps entre deux sauts de fréquence: $\Delta t = 625\mu s$

Probabilité que la première fréquence sur lequel le piconet émette soit la fréquence sur laquelle l'analyseur écoute:

$$n_{T=0} = p$$

Probabilité que la deuxième fréquence sur lequel le piconet émette soit la fréquence sur laquelle l'analyseur écoute:

$$n_{T=1} = (1-p) \cdot p$$

Formule généralisée:

$$n_{T=i} = (1-p)^{i-1} \cdot p$$

Probabilité que la fréquence écoutée par l'analyseur soit utilisée au moins une fois par le piconet dans un délai T0.

$$p_{(T \le T0)} = \sum_{i=1}^{T0/\Delta t} (1-p)^{i-1} \cdot p$$

Pour avoir une probabilité de 0.999 que la fréquence écoutée par l'analyseur soit utilisée par le piconet, il faudra attendre :

$$1-10^{-4} = 1-(1-p)^{\frac{T0}{\Delta T}}$$

$$T0 = \Delta T \cdot \frac{\log(10^{-4})}{\log(1-\frac{1}{79})}$$

T0 = 451.9 ms

La durée d'écoute sur une fréquence sera de 500 ms, assurant ainsi une probabilité supérieure à $1\text{-}10^{-4}$ que cette fréquence soit utilisée par un piconet.

3.5. Synchronisation sur un piconet

Lorsque l'adresse du maître du piconet est connue, l'analyseur va se synchroniser sur lui. Cette étape consiste à synchroniser l'horloge Bluetooth de l'analyseur sur celle du maître du piconet de manière à pouvoir déterminer la phase de la séquence de saut de fréquence du piconet. L'horloge du maître est aussi utilisée dans l'initialisation du *de-whitening* des paquets.

Pour synchroniser l'analyseur avec le maître, le maître doit envoyer un paquet FHS car c'est le seul paquet qui contienne l'horloge de l'émetteur du paquet. Ce paquet de contrôle contient aussi l'adresse Bluetooth complète et d'autres informations importantes, voir chap. 2.4.2.

Ce paquet est envoyé lors d'une procédure *Inquiry*, d'une procédure *Page* ou d'une demande d'inversion des rôles maître-esclave. Ces procédures fonctionnent de la manière suivante:

- La procédure *Page* est utilisée pour démarrer une connexion entre deux appareils. L'initiateur de la connexion deviendra le maître du piconet ainsi créé. Dans cette procédure, c'est l'initiateur qui envoie son paquet FHS pour que son correspondant puisse se connecter à lui. Cette procédure ne nous est d'aucune utilité puisque c'est l'analyseur qui devrait envoyer le paquet FHS au maître du piconet.

- La procédure d'inversion ne fonctionne que lorsque l'esclave et le maître sont déjà synchronisés, ce qui n'est pas le cas.

- La procédure *Inquiry* permet de connaître les appareils qui se trouvent dans le rayon d'action de l'émetteur. Un appareil Bluetooth envoie une suite de paquets ID et chaque appareil lui renvoie un ou plusieurs paquets FHS. C'est cette procédure que nous allons utiliser pour recevoir le paquet FHS du maître.

Nous avons choisi d'utiliser la procédure Inquiry pour récupérer les paquets FHS émis par les appareils du piconet à étudier. Dans cette procédure, l'analyseur reçoit des paquets FHS de tous les appareils présents, l'analyseur devra trier les paquets reçus pour trouver celui émis par le maître du piconet. L'adresse de l'émetteur du paquet FHS se trouve dans l'*Access Code*, l'analyseur pourra facilement trouver le paquet FHS envoyé par le maître du piconet et en extraire les indications sur son horloge.

3.5.1. Procédure Inquiry

La procédure *Inquiry* est décrite au chap 2.6 Inquiry, page 31.

Il est important que la procédure *Inquiry* permette de mémoriser l'état de l'horloge Bluetooth de l'analyseur à l'instant exact du début de la réception de chaque paquet FHS. Ceci pour permettre de synchroniser l'horloge de l'analyseur avec celle du maître du piconet.

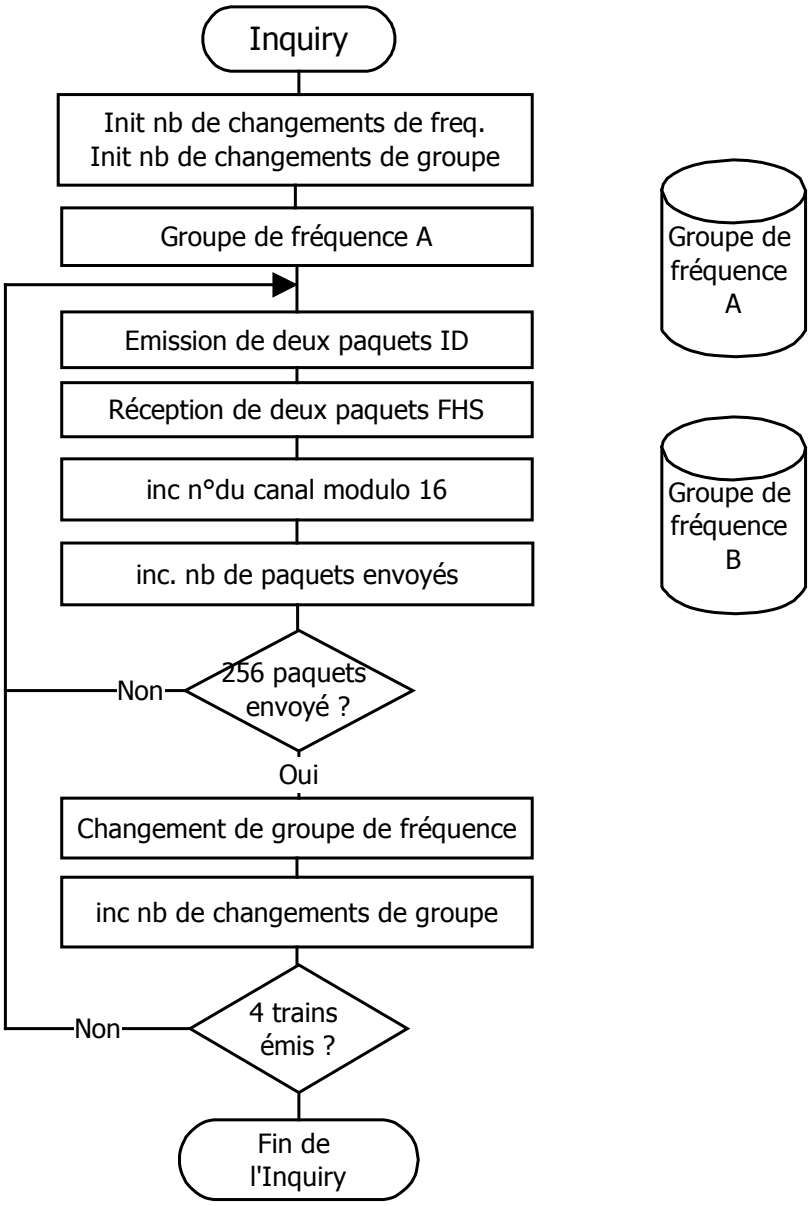


Figure 3-1 Structogramme d'un Inquiry

Les paquets ID sont émis en utilisant le GIAC pour générer l'*Access Code*

Pour recevoir les paquets FHS, le détecteur de trame recherche un *Access Code* généré par un GIAC. Lorsqu'un paquet FHS est reçu, il faut décoder son *Header* puis le reste du paquet. Ces opérations sont expliquées au chap. 3.6.3 et au chap.3.6.7

3.5.2. Tri des paquets FHS

Lorsque la procédure *Inquiry* est terminée et que les paquets FHS sont décodés, il faut trier ces derniers pour trouver celui émis par le maître du piconet qui nous intéresse. Les paquets FHS contiennent dans le champ LAP du *Payload* la partie LAP de l'adresse de l'émetteur du paquet. L'analyseur peut trier les paquets FHS à l'aide de ce champ.

Le format des paquets FHS est expliqué au chap. 2.4.2 Paquet FHS, page 29.

3.5.3. Réglage de l'horloge

Pour synchroniser son horloge, l'analyseur calcul la différence entre son horloge au moment de la réception du paquet FHS et la valeur de l'horloge contenue dans le champ CLK_{27-2} du paquet FHS. Cette différence (offset) sera additionnée à son horloge CLKN pour fournir l'horloge du Piconet, appelé CLK. C'est l'horloge CLK que l'on utilisera pour synchroniser les sauts de fréquences.

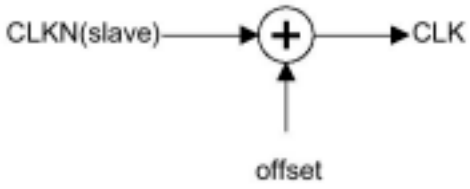


Figure 3-1 Génération de l'horloge synchronisée avec le maître du Piconet

Après cette étape, l'analyseur est prêt à recevoir des paquets de données

3.6. Analyse du trafic sur le piconet

Lorsque l'horloge de l'analyseur est synchronisée sur l'horloge du maître du piconet, il faut encore programmer le détecteur de trame pour que l'analyseur soit prêt à recevoir des données. Puis pour chaque paquet reçu, l'analyseur devra décoder le *Header* et le *Payload* selon le type de paquet reçu.

L'organigramme de cette étape se trouve en annexe 7.1 Exemple d'utilisation des commandes HCI, page 113.

Cette étape est divisée en plusieurs parties. En premier lieu, l'analyseur recherche le début de la trame. Une fois le début de la trame trouvé, il décode le *Header* qui permet de déterminer le type de paquet. Connaissant le type de paquet, il peut décoder le *Payload* et lorsque le paquet est terminé, changer de fréquence et transféré les données reçues.

3.6.1. Initialisation du détecteur de début de trame

Tous les paquets transmis dans un piconet comporte en entête un *Access Code* propre à chaque piconet. Le détecteur de trames est initialisé avec l'*Access Code* du piconet analysé. Les éléments de l'*Access Code* sont tous connus. L'adresse LAP du maître du piconet est connue, les bits de redondance ont été transmis dans le champ *Parity bits* du paquet FHS envoyé par le maître du piconet et le *Preamble*, le *Trailer* et le *Barker Code* sont simples a créer.

Le contenu exact de l'*Access Code* est expliqué au chap 2.3.1 Champs Access Code, page 23.

3.6.2. Détection de début de trame

La détection du début d'une trame se fait en recherchant dans le flot de bits reçu, l'*Access Code* qui se trouve au début de chaque paquet transmis dans le piconet.

La détection du début de trame est divisée en deux parties pour pouvoir détecter les paquets ID qui sont des paquets spéciaux car leur *Access Code* ne contient pas de *Trailer*. Si l'analyseur détecte seulement le *Preamble* et le *Sync Word*, il s'agit d'un paquet ID. S'il détecte aussi le *Trailer*, il s'agit d'un autre type de paquet.

3.6.3. Lecture du Header

Lorsqu'on reçoit un paquet autre qu'un paquet ID, il faut décoder son *Header* pour connaître son type et sa taille.

Le *Header* est protegé par un FEC1/3, un HEC et un *Whitening*.

Le FEC 1/3 est expliqué au chap. 3.6.6 Codes correcteurs d'erreurs, page 52. Ce code permet de corriger les erreurs de transmission simple.

La première partie de la Figure 3-1 utilise ce codage pour corriger les erreurs de transmission simples.

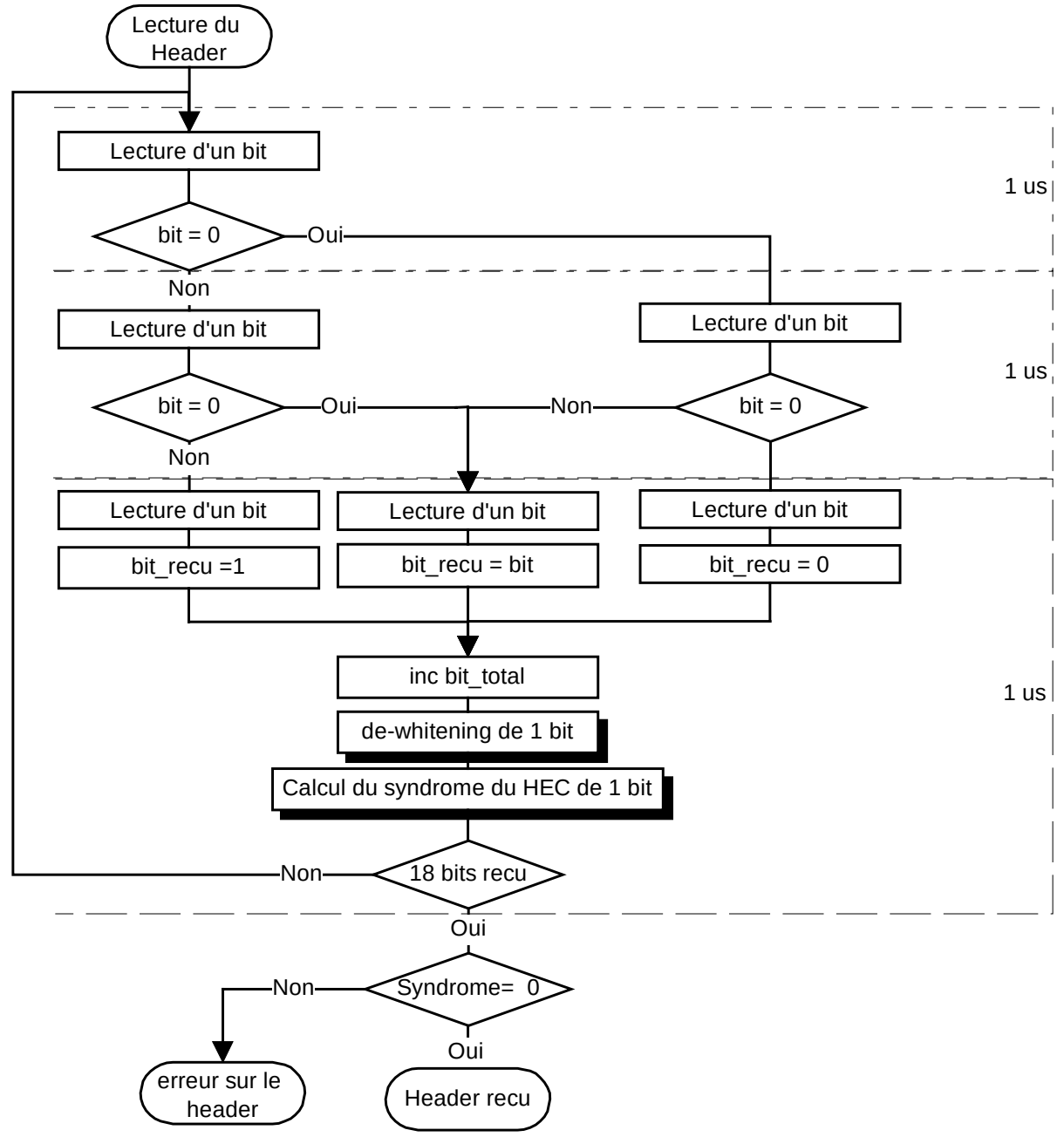


Figure 3-1 Organigramme de la lecture du *Header*

Puis pour chaque bit décodé, l'analyseur utilise l'algorithme de *de-whitening* pour supprimer le bruit pseudo-aléatoire et ensuite calcul le syndrome du HEC.

Lorsque le *Header* a été reçu en entier, l'analyseur test le syndrome du HEC, s'il vaut 0, il n'y a pas eu d'erreur. Si le syndrome est différent de 0, il y a eu une erreur de transmission et le paquet reçu est ignoré.

Le calcul du syndrome de même que le *de-whitening* se fait après chaque bit reçu. De cette façon la charge de travail est répartie durant toute la réception du *Header*.

L'algorithme de calcul du syndrome du FEC 2/3 est expliqué au chap 3.6.6 Codes correcteurs d'erreurs, page 52.

Contrainte de temps

La réception du paquet se fait à un débit de 1Mb/s. Pour ne pas perdre de bits, les opérations effectuées entre deux lectures de bits doivent durer moins de 1µs. Ces contraintes de temps sont représentées sur la Figure 3-1.

3.6.4. Traitement du Payload

Lorsque le *Header* est décodé, on peut déterminer le type de paquet reçu grâce aux 4 bits du champ *Type* du *Header*. Ceci permet d'adapter le décodage des paquets puisque chaque paquet possède un *Payload* et un codage différent. La Table 2-1 indique les correspondances entres la valeur du champ *Type* et le paquet envoyé.

L'organigramme en annexe 7.2 répertorie les opérations à effectuer pour chaques paquets.

Les opérations de décodage sont expliquées en détail ci-dessous avec un exemple de décodage pour le paquet FHS.

3.6.5. De-whitening

Les champs *Header* et *Payload* sont brouillés *(whitening)* pour diminuer la composante continue du signal et pour éviter l'apparition trop fréquente de certaines suites de bits. Ce brouillage s'effectue avant le calcul du FEC.

Les données sont brouillées en effectuant un XOR des données avec un mot de *whitening*. Le mot de *whitening* est généré par le polynôme $g(d)=D^7+D^4+1$.

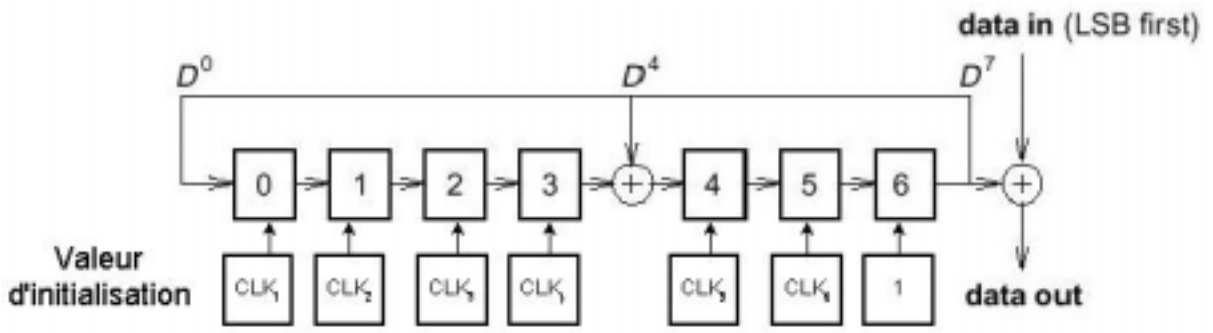


Figure 3-1 Data whitening

Avant chaque début de transmission, le registre à décalage est initialisé avec une portion de l'horloge Bluetooth maître, CLK_{6-1}. La position 0 est initialisé avec CLK_1, la position 1 avec CLK_2 et ainsi de suite. La dernière position, position 6 est initialisé avec un 1.

Une exception à cette initialisation existe pour les paquets FHS transmis lors d'un **Inquiry response** ou d'un **Page response.**

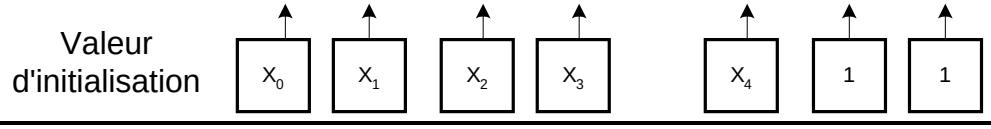


Figure 3-2 Initialisation des registres lors d'un Inquiry Response ou d'un Page Response

Les deux équations ci-dessous permettent de calculer $Xi^{(79)}$ lors d'un Inquiry.

$$Xi^{(79)} = \left[CLKN_{16-12} + k_{offset} + (CLKN_{4-2,0} - CLKN_{16-12}) \bmod 16\right] \bmod 32$$

$$k_{offset} = \begin{cases} 24 \\ 8 \end{cases}$$

La Figure 3-3 représente l'organigramme du *de-whitening* pour traiter un bit. Cette opération doit être effectuée pour chaque bit reçu.

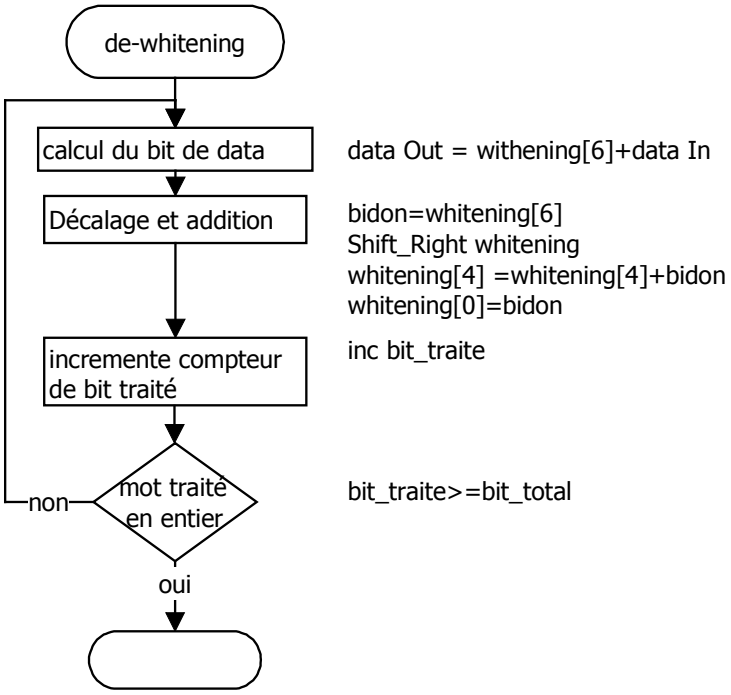


Figure 3-3 Organigramme du de-whitening bit à bit

3.6.6. Codes correcteurs d'erreurs

Introduction

Chaque paquet est protégé contre les erreurs de transmission. Bluetooth utilise trois codes correcteurs qui protègent un paquet selon l'importance des données qu'il transporte. Ce chapitre explique les moyens de décoder ces codages.

HEC

Le codage HEC est un code CRC de 8 bits concaténé à la fin du *Header*. Pour vérifier si le paquet est transmis correctement, le syndrome du *Header* est calculé. S'il vaut 0, le *Header*

est correct. S'il est différent de 0, le *Header* a été mal transmis et ce paquet est ignoré. Le HEC ne permet de pas de corriger des erreurs, seulement de les détecter.

Le calcul du syndrome peut se faire avec un registre à décalage dont on aurait rebouclé la sortie sur certaines cellules. Les bits reçus sont rentrés en série dans le registre. Lorsque le mot est terminé, le syndrome est le mot se trouvant à l'intérieur du registre à décalage.

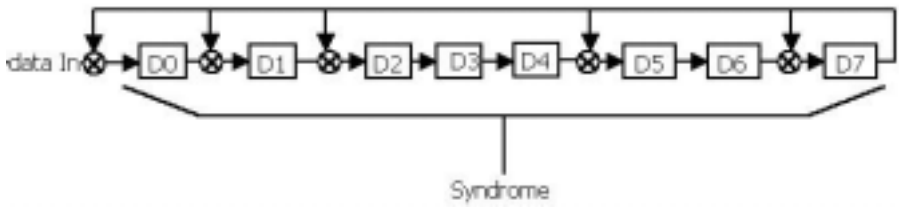


Figure 3-1 Registre à décalage pour le calcul du syndrome du HEC.

Au début de la réception du *Header*, le registre est initialisé avec la partie haute (UAP) de l'adresse du maître du piconet. S'il s'agit d'un paquet FHS transmis lors d'un *Inquiry*, le registre est initialisé avec le DCI qui vaut 0x00.

La Figure 3-2 représente le calcul du syndrome bit à bit.

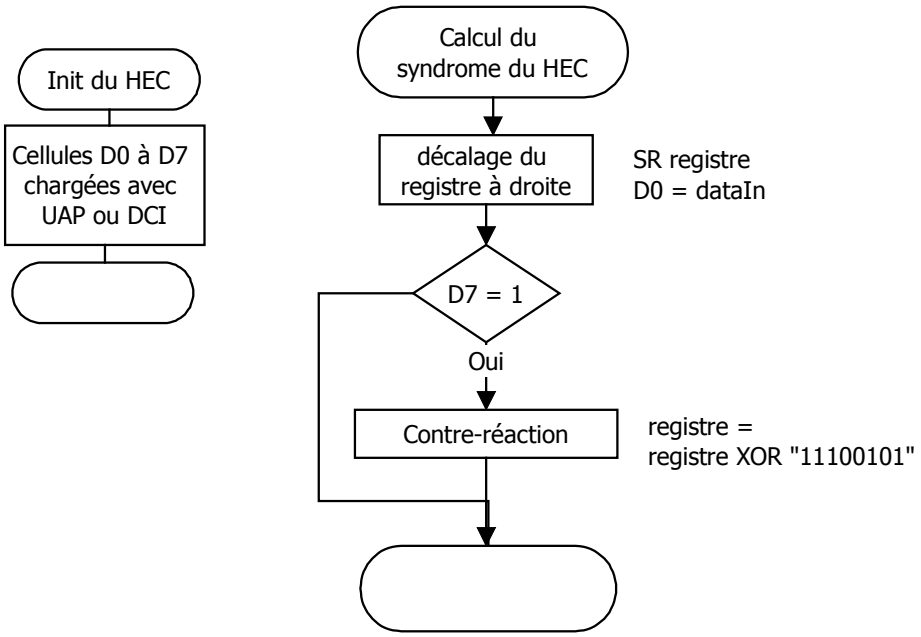


Figure 3-2 Organigramme de l'initialisation du registre (a gauche) et du calcul du syndrome du HEC (a droite)

CRC

Le CRC est un mot de 16 bits concaténé à la fin du *Payload*. Il est généré de la même manière que le HEC.

Comme pour le HEC, la vérification du CRC peut se faire en calculant le syndrome du *Payload*. S'il vaut 0, le paquet a été correctement transmis, sinon le paquet est éliminé car il contient une ou plusieurs erreurs.

Le calcul du syndrome peut se faire avec un registre à décalage dont la sortie est rebouclée sur certaines cellules.

Comme indiqué sur la Figure 3-1, les 8 LSB du registre sont initialisés avec la partie haute (UAP) de l'adresse du maître du piconet. S'il s'agit d'un paquet FHS transmis lors d'un *Inquiry*, les 8 LSB sont initialisés avec le DCI qui vaut `0x00`. Les 8 MSB du registre sont toujours initialisés avec `0x00`.

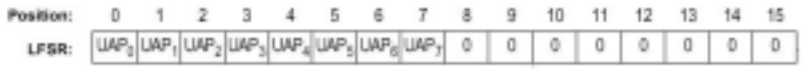


Figure 3-1 Initialisation du registre du CRC

Les bits sont entrés en série dans le registre. Le syndrome est le mot qui se trouve dans le registre après avoir entré le dernier bit du *Payload* qui est le dernier bit du mot de CRC du payload.

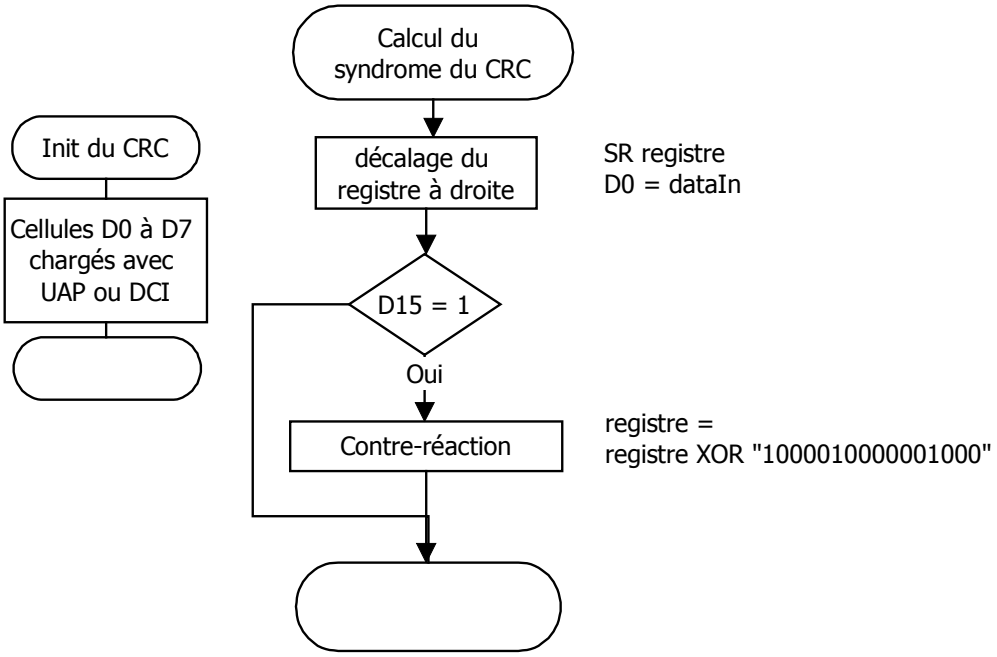


Figure 3-2 Organigramme du calcul du Syndrome du CRC

1/3 FEC

Le codage 1/3 FEC triple chaque bit transmis. Il est utilisé pour le *Header* de chaque paquet et pour le champ *Voice* des paquets **HV.**

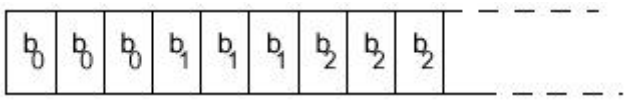


Figure 3-1 Code correcteur 1/3 FEC

Le décodage se fait en calculant la moyenne des valeurs des bits reçus, 3 bits par 3 bits, si elle est supérieure à 1.5, le bit reçu est un 1, sinon c'est un 0.

2/3 FEC

Ce code utilise un code de Hamming(15,10), qui permet de détecter toutes les erreurs doubles et corriger les erreurs simples. Le polynôme générateur est $g_{(D)}=(D+1)(D4^4+D+1)$

La distance de Hamming de ce code vaut :

$$d\min = 4$$

La capacité de détection "λ" et de correction "L" est de :

$$\begin{aligned} d\min &= \lambda + L + 1 \\ 4 &= 2 + 1 + 1 \\ \Rightarrow L &= 1 \\ \Rightarrow \lambda &= 2 \end{aligned}$$

Ce code détecte toutes les erreurs doubles et corrige les erreurs simples.

Il s'agit d'un code systématique, les 10 bits d'informations se retrouvent en tête du message alors que les bits *r(x)* (bits de contrôle) occupent les 5 dernières positions. Si la taille du message n'est pas un multiple de 10, des bits de remplissage d'une valeur 0 sont rajoutés en queue du paquet.

Ce code peut être généré par un registre à décalage de 5 bits comme sur la Figure 3-1.

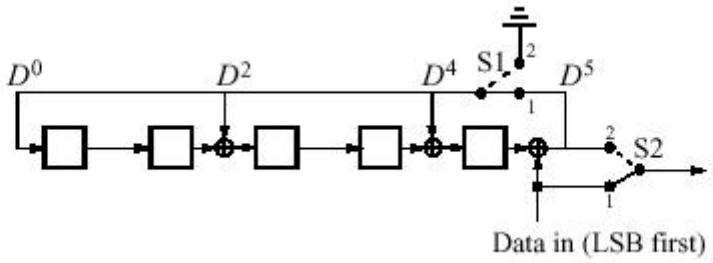


Figure 3-1 Registre à décalage générateur du code de Hamming (15,10)

Pour détecter les erreurs, on divise des blocs de 15 bits par $g_{(D)}$. Si le mot ne contient pas d'erreur (ou si l'erreur $e_{(x)}$ est un des codes cycliques, auquel cas l'erreur est indécelable) le reste de la division sera égal à 0. Si le reste est différent de 0, une erreur est détectée. Le reste est appelé syndrome.

Ce décodage peut aussi être implanté par un registre à décalage, comme sur la Figure 3-2

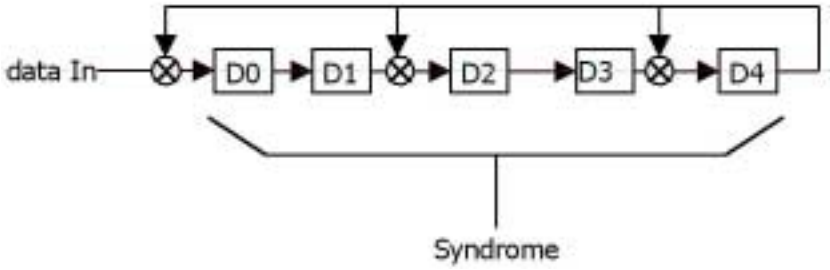


Figure 3-2 Registre à décalage pour calculer le syndrome

Le registre est initialisé à 0. Les données sont entrées, LSB en premier, sur *data In*. A chaque coup d'horloge, le bit de donné suivant est rentré et après 15 coups d'horloge, le mot se trouvant dans le registre est le syndrome.

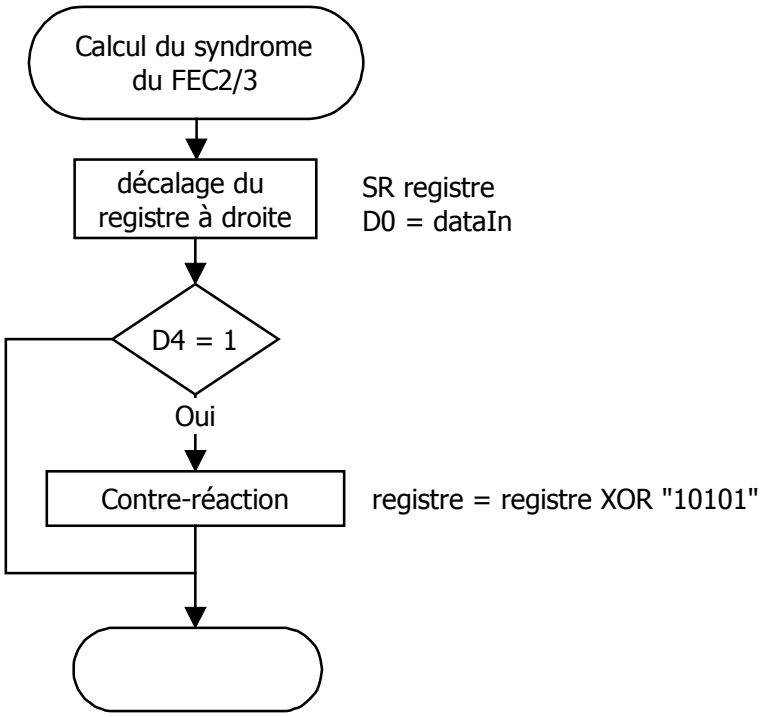


Figure 3-3 Organigramme du calcul du syndrome pour le FEC 2/3

Si le syndrome vaut 0, les 10 premiers bits reçus correspondent aux bits de données. Si le syndrome obtenu est différent de 0 l'analyseur va corriger les données reçues en cherchant dans la Table 3-1 qui répertorie toutes les erreurs simple de transmission *"e"* en fonction du syndrome obtenu.

Pour corriger le code, l'analyseur additionne *"e"* aux donnés reçus.

Les syndromes présents dans le tableau sont crées par des erreurs simples. Si le syndrome obtenu n'est pas dans le tableau cela indique qu'il s'agit d'une erreur double que l'analyseur ne peut pas corriger.

Table 3-1 Tableau des erreurs de transmission en fonction des syndromes.

Erreur e [bin]	Syndrome [hex]
000'0000'0000'000**1**	0B
000'0000'0000'00**1**0	16
000'0000'0000'0**1**00	07
000'0000'0000'**1**000	0E
000'0000'000**1**'0000	1C
000'0000'00**1**0'0000	13
000'0000'0**1**00'0000	0D

Erreur e [bin]	Syndrome [hex]
000'0000'**1**000'0000	1A
000'000**1**'0000'0000	1F
000'00**1**0'0000'0000	15
000'0**1**00'0000'0000	01
000'**1**000'0000'0000	02
00**1**'0000'0000'0000	04
0**1**0'0000'0000'0000	08
100'00000'0000'0000	10

Ce tableau a été calculé grâce au programme FEC2_3.exe. Le code de ce programme se trouve en annexe 7.15 Calcul du syndrome du FEC 2/3.

Fonctionnement du programme FEC2_3.exe

Ce programme, écrit en C, implémente une division polynomiale par un registre à décalage. Ce registre est représenté à la Figure 3-2.

Le registre est implanté par un `unsigned int` représenté sur 16 bits.

Le registre étant sur 5 bits, on utilise les 5 MSB.

```
//Calcule les valeurs dans le registre
        registre=(dataIn | (registre>>1))^((registre&0x0800)?0xA800:0);
```

Code 3-1 Rebouclage du registre a décalage

Le rebouclage est effectué par le Code 3-1. Cette ligne décale d'un bit à droite le registre, insère le prochain bit reçu (`dataIn`) sur la gauche, puis si le LSB vaut 1, rajoute la contre-réaction.

Le décalage est effectué 15 fois, puisqu'on divise un mot de 15 bits.

```
//Initialisation du mot de data;
    dataRecu = pow(2,k);
```

Code 3-2 Initialisation du registre à décalage

Le Code 3-2 crée des syndromes qui n'ont qu'un seul bit à 1 à la fois.

ARQ

Il existe aussi un mécanisme Automatic Repeat Request (ARQ) utilisé pour les paquets DM, DH et DV. Ces paquets sont retransmis jusqu'à ce que leurs réceptions correctes soit confirmées. Ce mécanisme ne code pas les paquets. Un des intérêts d'un analyseur est de voir les trames qui ne sont pas confirmés, ces trames ne demandent donc aucun traitement spécifique

3.6.7. Exemple de Traitement du Payload, le paquet FHS

Le *Payload* du paquet FHS est protégé par un FEC 2/3 et par un mot de CRC de 16 bits.

Le traitement du *Payload* s'effectue en décodant le FEC 2/3 puis en vérifiant le CRC de 16 bits.

Le décodage du FEC 2/3 est expliqué au chap. 3.6.6.

Une partie du syndrome FEC 2/3 est calculé à chaque bit reçu, répartissant la charge de calcul tout au long de la réception du *Payload*. Le FEC 2/3 permet de corriger les erreurs simples. Si une erreur est détecte, l'analyseur vérifie s'il s'agit d'une erreur simple. Si c'est le cas le mot est corrigé, sinon ce paquet est ignoré.

Lorsque le FEC2/3 a été vérifié, on procède au *de-whitening*. Le *de-withening* est expliqué au chap. 3.6.5 De-whitening, page 51.

L'algorithme de *de-whitening* reçoit le message par paquet de 10 bits car c'est la taille des blocs traités par le FEC.

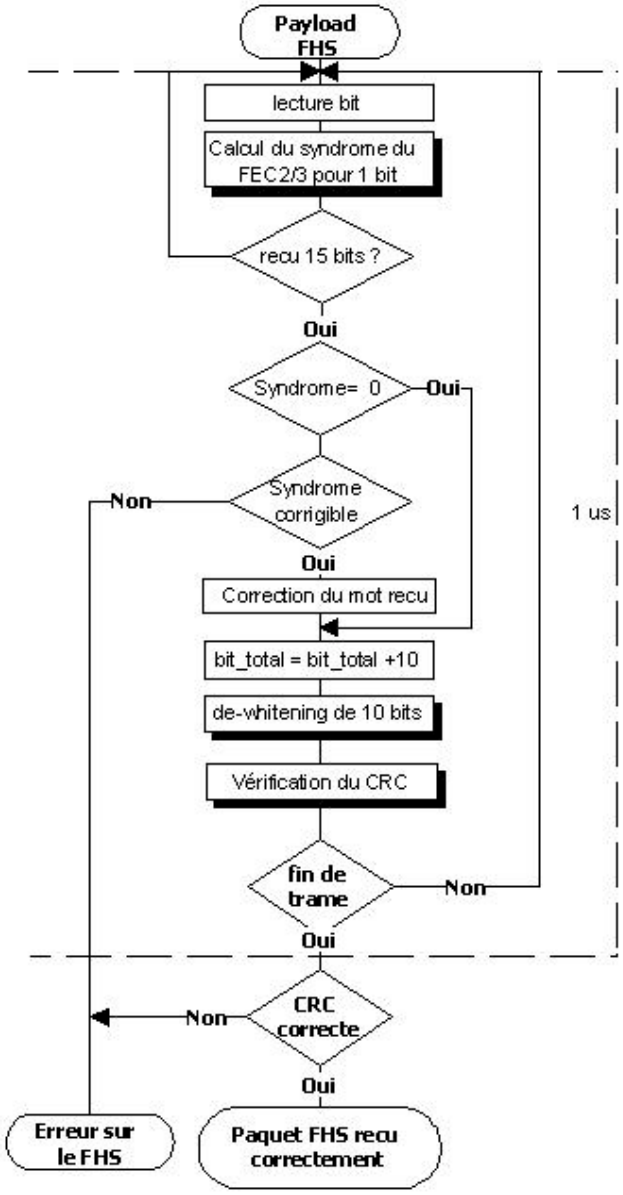


Figure 3-1 Organigramme du traitement d'un paquet FHS

Après le *de-whitening*, une partie du syndrome du CRC est calculé. Puis l'analyseur teste le nombre de bits reçu pour savoir s'il a reçu le *Payload* en entier. Le *Payload* est complet lorsque l'analyseur a lu 240 bits et que 160 bits ont été rentrés dans le registre de calcul du syndrome du CRC.

Si le paquet est complet, l'analyseur teste le syndrome. Ce syndrome ne permet pas de corriger la trame, seulement de détecter les erreurs. Si le syndrome vaut 0 la trame est correcte, sinon le paquet est éliminé.

Contrainte de temps

La réception du paquet se faisant à un débit de 1Mb/s, il est indispensable que, comme lors du traitement du *Header,* les opérations se déroulant entre deux lectures de bits durent moins de 1µs. Ces contraintes de temps sont représentées sur la Figure 3-1.

3.6.8. saut de fréquence

Lorsque le paquet a été reçu en entier, le récepteur radio de l'analyseur doit changer de fréquence. L'algorithme de calcul de la prochaine fréquence, représenté à la Figure 3-1, utilise les partie UAP et LAP de l'adresse du maître ainsi que l'horloge Bluetooth de l'analyseur synchronisé sur celle du maître.

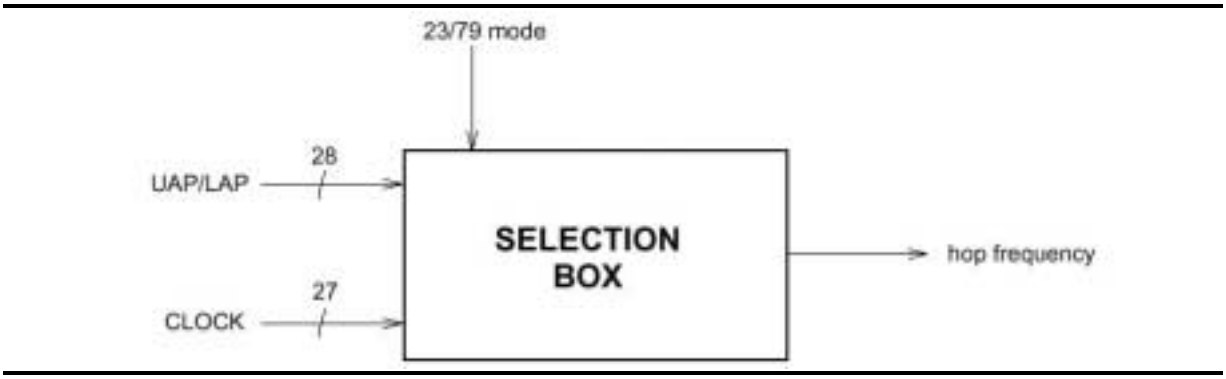


Figure 3-1 Schéma bloc général du sélecteur de saut de fréquence

Le calcul des fréquences est constitué d'une addition, d'une opération XOR, d'une permutation, d'une deuxième addition et d'une sélection d'un registre.

Pour le reste de la description, la notation Ai est utilisée pour le bit i de BD_ADDR qui est l'adresse Bluetooth de 48 bits.

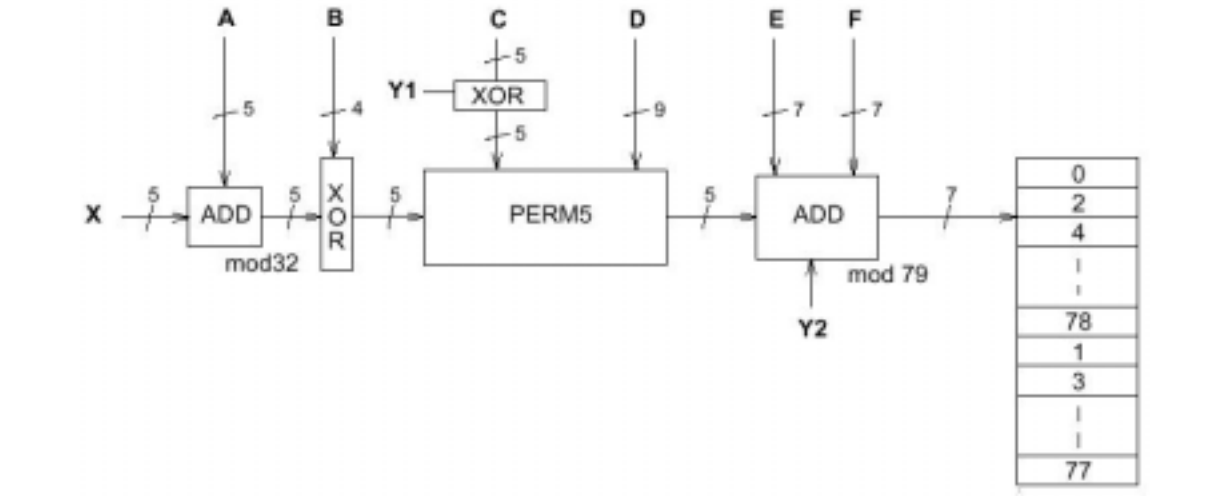


Figure 3-2 Schéma bloc détaillé du sélecteur de saut de fréquence

Première addition

Le premier bloc additionne modulo 32 la phase (l'entrée X) avec une constante (A)

Le résultat de cette addition se nomme Z'

Opération XOR

Les 4 LSB de Z' sont additionnés modulo 2 avec les bits A_{22-19}. Cette opération est représentée par la Figure 3-1.

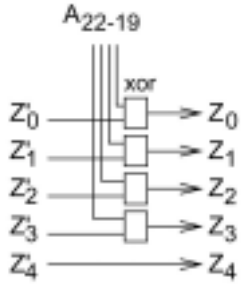


Figure 3-1 Opération XOR

Permutation

Cette opération permute les 5 entrées sur 5 sorties en fonction du mot de contrôle. La permutation est faite en 7 étapes appelées "Butterfly Operation"

La Figure 3-1 montre les permutations à effectuer à chaque étape.

Les bits de contrôle P_0 à P_8 correspondent aux bits D_0 à D_8. Les bits P_{i+9} correspondent à $C_i \oplus Y1$ pour i=0..4.

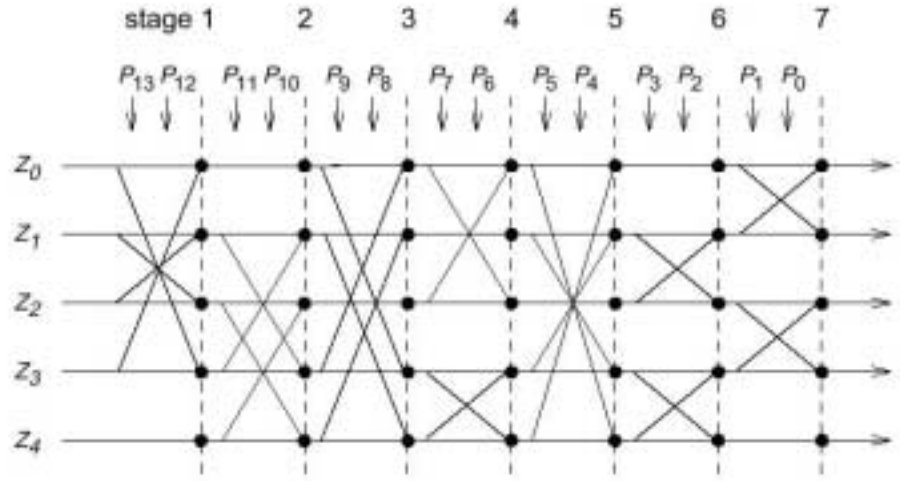


Figure 3-1 Grille de permutation

La Table 3-1 indique les bits de contrôle à utiliser pour chaques permutation.

Table 3-1 Signaux de contrôle des opérations Butterfly

Signal de contrôle	Butterfly
P_0	$\{Z_0,Z_1\}$
P_1	$\{Z_2,Z_3\}$
P_2	$\{Z_1,Z_2\}$
P_3	$\{Z_3,Z_4\}$
P_4	$\{Z_0,Z_4\}$
P_5	$\{Z_1,Z_3\}$
P_6	$\{Z_0,Z_2\}$

Signal de contrôle	Butterfly
P7	$\{Z_3,Z_4\}$
P_8	$\{Z_1,Z_4\}$
P_9	$\{Z_0,Z_3\}$
P_{10}	$\{Z_2,Z_4\}$
P_{11}	$\{Z_1,Z_3\}$
P_{12}	$\{Z_0,Z_3\}$
P_{13}	$\{Z_1,Z_2\}$

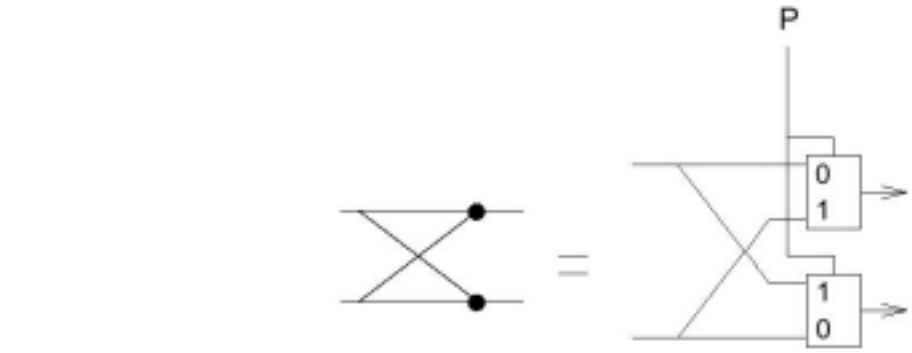


Figure 3-2 Implémentation de l'opération Butterfly à l'aide de deux multiplexeurs.

Deuxième addition

L'addition rajoute une constante au résultat des permutations. C'est une addition modulo 79.

Sélection d'un registre

La sortie de l'additionneur adresse un registre de 79 lignes. Ce registre est chargé avec les codé du Transceiver correspondant aux 79 canaux. La moitié supérieure du registre contient les fréquences paires et la partie inférieur les fréquences impaires.

Mot de contrôle

Le mot de contrôle dépend du mode de fonctionnement de l'appareil. Il existe un mot pour les états **Page Scan** et **Inquiry Scan**, un mot pour les états **Page** ou **Inquiry**, un mot pour les états **Page Response** ou **Inquiry Response** et un mot de contrôle pour l'état **Connection.**

Ce chapitre décrit le mot de contrôle utilisé lors de l'état **Connection**, puisque c'est celui utilisé par l'analyseur pour suivre les sauts du piconet analysé.

CLK représente les bits de l'horloge Bluetooth de l'analyseur qui est synchronisé sur celle du maître du piconet.

A représente les bits d'adresses du maître du Piconet.

$X=CLK_{6-2}$

$Y1=CLK_1$

$Y2=32 \times CLK_1$

$A=A_{27-23} \oplus CLK_{25-21}$

$B= A_{22-19}$

$C= A_{8,6,4,2,0} \oplus CLK_{20-16}$

$D= A_{18-10} \oplus CLK_{15-7}$

$E= A_{13,11,9,7,5,3,1}$

$F=16 \times CLK_{27-7} \bmod 79$

Lorsque la prochaine fréquence a été calculée, elle est envoyée au Transceiver. Cette étape est expliquée au chap. 3.4.3 Programmation du transceiver en réception, page 40.

3.6.9. Transfert des résultats

Lorsque la fréquence a été programmée, il faut un certain temps à la PLL pour se synchroniser. Ce temps peut être mis à profit pour transférer le paquet reçu vers un PC qui s'occuperait de la mise en page des données reçu.

3.7. Architecture de l'analyseur

Avant d'implanter l'analyseur, il est intéressant de voir les architectures qu'il est possible d'utiliser.

On peut numériser directement le signal à la sortie du transceiver. C'est la méthode utilisée par le kit d'INDIEN. Le signal reçu est échantillonné sur 8 bits à 4MHz (oversampling). Tout le filtrage ainsi que le recouvrement de l'horloge se fait numériquement par des portes logiques. Le kit INDIEN contient une grande FPGA (XC4020E pouvant contenir entre 13 000 et 40'000 portes) et un processeur ARM7 qui permet d'implanter toute cette logique.

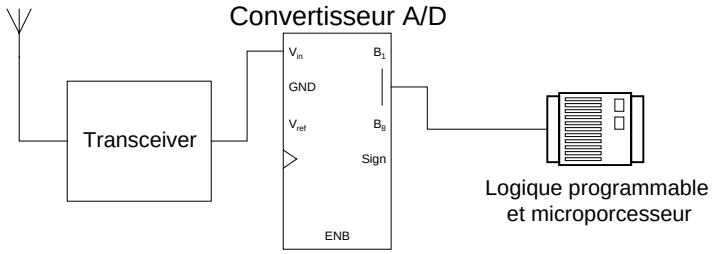


Figure 3-1 Architecture "Tout numérique"

Cette architecture est avantageuse car la Baseband de Bluetooth est prévu pour être implanter avec des portes logiques. Tous les exemples de codage et décodage des algorithmes ainsi que les calculs des sauts de fréquences sont décrits à l'aide de registres à décalage. De plus la logique de commandes peut être facilement décrite par une machine d'états. Mais la numérisation du signal est plus complexe, car le signal reçu comporte une composante basse-fréquence importante, ce qui fait varier continuellement le niveau de décision.

Une autre architecture consiste à filtrer le signal analogique et ensuite à le numériser grâce à un comparateur. Le filtrage analogique supprime la composante basse-fréquence, ce qui rend constant le niveau de décision. Il est donc facile de numériser le signal à l'aide d'un comparateur. Le recouvrement de l'horloge est effectué à l'aide d'une PLL numérique. Le signal peut ensuite être fourni à un microprocesseur pour le traitement des données.

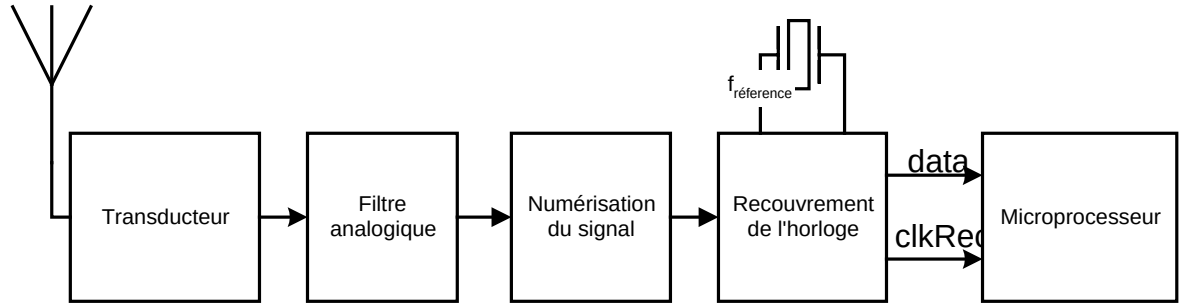


Figure 3-2 Architecture mixte

Cette architecture nécessite une plus faible quantité de portes logiques mais demande plus de puissance de calcul de la part du processeur. C'est lui qui doit détecter le début des trames, gérer les sauts de fréquences et décoder les trames.

Pour ce diplôme, on ne dispose pas d'une grande FPGA mais une carte DSP pour le traitement du signal est à disposition, nous avons donc choisi cette architecture. De plus, cette solution est plus simple pour le filtrage du signal et pour la numérisation. Le signal traverse un filtre passe-bande qui supprime la composante basse-fréquence, puis on utilise un comparateur avec un niveau de décision fixe pour générer le signal numérique. Mais, la détection des débuts de trames et leur décodage sont plus compliqué car prévu pour une implémentation avec des portes logiques.

Il est aussi possible de retrouver l'horloge avec un signal analogique en utilisant une PLL analogique. Cela impose que la fréquence de l'horloge soit transmise dans le signal de donnée. Dans notre cas, les données à la sortie du Transceiver sont transmises avec un codage NRZ (No return to zero), le spectre du signal contient donc une raie à la moitié de la fréquence du signal d'horloge.

4. Plateformes de test

4.1. Introduction

Pour les besoins de ce travail, plusieurs kits de démonstration ont été utilisés. Ce chapitre décrit les possibilités de chaque kit, leurs avantages et leurs inconvénients.

4.2. Digianswer Demo Card

4.2.1. Introduction

Les cartes de démonstration de Digianswer se branchent sur le port PCMCIA d'un PC. Elles sont prêtes à l'emploi. Il n'y a aucune opération de réglage à effectuer sur les cartes qui sont hermétiquement fermées. Chaque carte offre un branchement pour un casque audio avec micro et écouteurs pour tester le canal audio.

4.2.2. Applications

Le CD-ROM livré avec le kit contient plusieurs applications pour piloter ces cartes. Elles permettent d'envoyer des commandes HCI (*HCI Terminal*), de connecter des PC ensemble pour créer un piconet (*Bluetooth Neighborhood*) ou d'utiliser le poste équipé de la carte PCMCIA comme routeur (*HCI router*) relié à un autre poste par l'intermédiaire d'un câble série.

Bluetooth Neighborhood

Permet de connecter et déconnecter des PC équipés de cartes Bluetooth et d'ouvrir des connexions pour la transmission de la voix ou des données.

Les commandes HCI sont cachées à l'utilisateur qui n'a pas besoin de connaître le fonctionnement de Bluetooth.

Bluetooth HCI Terminal

Ce terminal permet d'envoyer et de recevoir des commandes HCI à la Baseband de la carte Bluetooth.

Le terminal est simple d'utilisation, il complète les commandes que l'on écrit et indique quels champs remplir et avec quelles valeurs.

HCI router

Cette application permet d'envoyer des commandes HCI à la Baseband Bluetooth au travers d'une interface RS232. Le poste qui contient l'application à exécuter est branché au travers d'un câble NULL MODEM au poste équipé d'une carte Bluetooth.

Statistic viewer

Statistiques sur le débit des canaux, le nombre de paquets perdus et sur le type de d'erreur survenue.

4.2.3. Problèmes rencontrés

Ces cartes fonctionnent correctement, si ce n'est un fichier DLL qu'il a fallut mettre à jour pour afficher correctement les icônes dans le *Bluetooth Neighborhood* (COMCTL32.DLL)

Ces cartes ont été utilisées principalement pour générer du trafic, reçu ensuite sur la carte TEMIC, toutes les ressources de ce kit n'ont donc pas été testé. La documentation reçue avec ce kit est complète et indique les opérations à effectuer pas à pas pour installer le kit et démarrer un piconet. Le fascicule *HCI Implémentation Notes* explique clairement les commandes HCI et les valeurs des paramètres à utiliser.

4.3. Indien Evaluation Board

4.3.1. Introduction

Le kit d'évaluation d'INDIEN contient deux cartes *Baseband controller* avec leurs cartes transceiver radio RF T2901 et un terminal fonctionnant sous *Windows98* permettant de les piloter. Les deux cartes *Baseband controller* sont connectées par l'intermédiaire du port série au PC et sont pilotable directement depuis le logiciel *IndienTerm* fourni avec le kit. Il est possible de brancher les deux boîtiers sur le même PC, chacun sur un port série différent et de démarrer deux fois l'application *IndienTerm*.

4.3.2. Cartes Baseband controller

Chaque carte se trouve dans un boîtier comportant une antenne, un connecteur d'alimentation et un connecteur DB9 série pour la connexion au PC.

Un boîtier contient un *Baseband controller* avec un processeur *ARM7* et une FPGA ainsi qu'une carte T2901 pour la partie radio.

Le signal de donnée Rx_data reçu de la carte T2901 est directement numérisé par un DAC. Ce DAC échantillonne le signal à une fréquence de 4MHz. Ce signal est ensuite injecté dans la FPGA.

4.3.3. IndienTerm

IndienTerm permet de piloter un appareil Bluetooth. Il permet d'envoyer ou de recevoir des commandes au niveaux HCI ou L2CAP.

La plupart des opérations de base peuvent être exécutées directement depuis des menus contextuels. On peut ainsi, démarrer un *Inquiry* ou un *Page*, se connecter ou accepter une connexion d'un tiers et transférer des fichiers depuis la barre de menu de *IndienTerm*, comme le montre la Figure 4-1.

Figure 4-1 IndienTerm

IndienTerm contient aussi un *HCI terminal* qui permet de voir les commandes HCI qu'il envoie et reçoit. Ceci est très utile pour comprendre le séquencement des opérations ainsi que les valeurs à donner aux paramètres des commandes que l'on veut envoyer.

Un mode d'emploi de l'installation du kit INDIEN se trouve en annexe 7.16 Mise en route du kit INDIEN. Il indique comment installer le kit et les instructions HCI à utiliser pour un *Inquiry* et un *Page*.

4.3.4. Limitations

Ce kit de démonstration (version 1.0) n'implémente pas toutes les fonctionnalités prévues dans la norme. La liste exhaustive des commandes implémentées se trouve dans *AT76C551 Firmware Features and Specification*. Mais il est à relever que seul les paquets ID, POLL, FHS, DV, DH1, DH3 et DH5 sont supportés, aucun paquet HV n'est supporté. De plus, la connexion de plusieurs esclaves sur un seul maître n'est pas possible et il n'est pas non plus possible d'avoir une connexion en cours lorsque l'on démarre une procédure d'accès (*Page*, *Page Scan, Inquiry, Inquiry Scan*).

4.3.5. Problèmes rencontrés

Dans l'ensemble ce kit fonctionne bien et ne comporte pas d'erreur importante. Mais il demande une certaine connaissance du fonctionnement de Bluetooth pour l'utiliser. Pour faire un *Inquiry*, il faut démarrer l'esclave en mode *Inquiry response* et lui indiquer l'adresse (le GIAC) à laquelle il doit répondre. Il en va de même pour le *Page*.

Après une réinitialisation, il arrive parfois qu'un des boîtiers refuse de se connecter au terminal *IndienTerm*. Il faut alors fermer toutes les applications de *Windows*, débrancher l'alimentation du boîtier et réessayer.

Lors de transfert de fichiers supérieurs à 50 Mb, la transmission s'arrête après environ 1 heure. Il faut alors réinitialiser une connexion depuis *IndienTerm* puis recommencer à transmettre le fichier.

4.4. TEMIC RF Development Kit

4.4.1. Introduction

Le kit de développement de TEMIC contient un transceiver HF avec une carte d'interface pour la brancher sur le PC, ainsi qu'un programme s'exécutant sur *Windows98* pour contrôler le transceiver. Ce kit ne contient que la partie radio d'un appareil Bluetooth, il ne comprend pas la partie Baseband.

L'avantage de ce kit est que l'on contrôle tous les paramètres de l'émetteur radio. On peut le configurer facilement grâce au programme fourni et les mesures des signaux reçus ou envoyés peuvent se faire directement sur la carte.

4.4.2. T2901RF Demoboard

La carte T2901 est connectée à une carte d'interface par un connecteur 28 broches. La carte d'interface met à disposition les signaux RX_data et RSSI sur des connecteurs externes et relie le transceiver au PC par le port parallèle. La carte T2901RF ne traite absolument pas les données reçues, elle se contente de les translater en bande de base.

On configure le transceiver, en lui envoyant deux paquets de bits sur un bus série. Ces paquets peuvent facilement être créés depuis le programme fourni avec le kit.

La carte T2901 est aussi utilisée dans le kit INDIEN pour la partie radio fréquence.

4.4.3. T2901 software (U2901 B Manual programming)

Le programme fourni avec le kit permet d'éditer les deux paquets de bits qui servent à programmer la carte radio et affiche des informations sur la synchronisation de la PLL (*PLL lock*) ainsi que sur le mode de fonctionnement de la carte (Tx/Rx/Pause). Les données à transmettre ou à émettre ne sont pas du tout traitées ce programme.

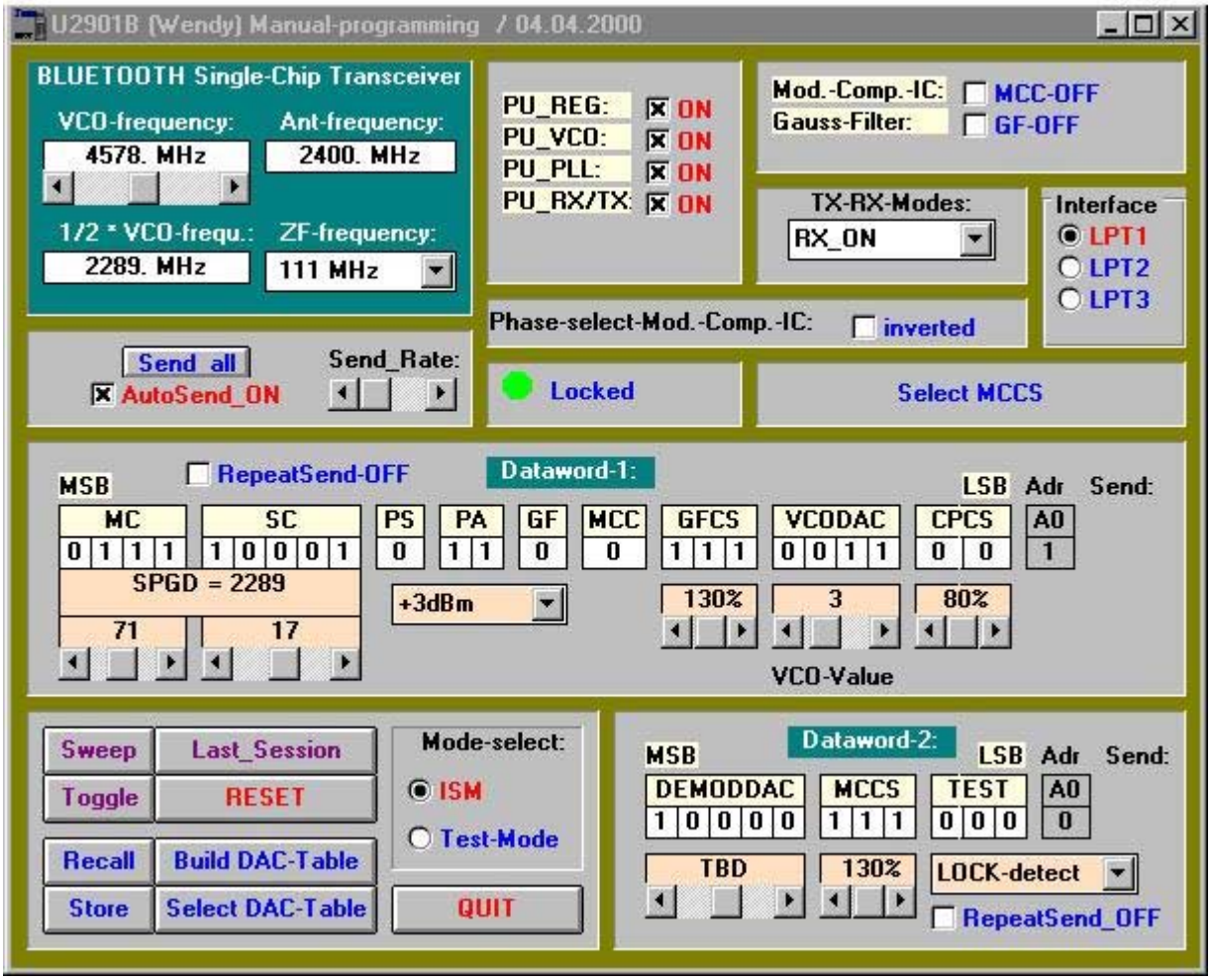


Figure 4-1 Copie d'écran du U2901B Manual programming

La Figure 4-1 présente les deux mots qui seront envoyés au transceiver (Dataword-1 et Dataword-2). Les bits des deux mots sont répartis par champs que l'on peut modifier soit en cliquant directement sur les bits, soit avec les curseurs qui se trouvent sous chaque champ. La fréquence de la PLL, qui influence directement la fréquence d'émission et de réception, peut être réglée par le curseur *VCO-frequency* qui se trouve au haut de la fenêtre. A chaque modification d'un des deux mots, les deux mots sont envoyés au transceiver.

L'état de la PLL est indiqué par le mot *Locked* ou *Unlocked!* et un rond vert ou rouge (gris sur la figure) qui indique si la PLL est synchronisée et prête à émettre ou pas..

Les boutons en bas à gauche de la fenêtre permettent de rechercher (bouton *recall*) des configurations que l'on aurait sauvegardées (bouton *Store*).

4.4.4. Problèmes rencontrés

Aucune erreur d'erreur de fonctionnement n'a été trouvée, mais, il faut bien connaître la norme Bluetooth et les transceiver pour pouvoir utiliser ce kit correctement. Les documents qui accompagnent la carte donnent les noms de chaque champ des mots de programmation et expliquent très clairement comment faire fonctionner la carte et le programme. Mais ils n'expliquent pas du tout les valeurs à utiliser pour programmer le transceiver.

C'est pourquoi il a fallu quelques semaines avant de comprendre parfaitement le fonctionnement du transceiver et de pouvoir la régler correctement.

Les réglages à utiliser pour cette carte sont décrits au chapitre 5.2.2 Programmation du transceiver.

5. Implémentations Haute-Fréquence

5.1. Introduction

La partie haute-fréquence regroupe toutes les parties qui mettent en forme les signaux de façon à ce qu'ils soient utilisables par le DSP. Il s'agit du front-end radio, des filtres de mise en forme et du circuit de recouvrement de l'horloge.

5.2. Transceiver Radio Bluetooth

5.2.1. Introduction

Les bits générés par la Baseband module une porteuse radio à une fréquence déterminée, c'est une partie des fonctions du Transceiver T2901. Inversement, il permet aussi de recevoir un signal radio et de le transformer en un signal analogique en bande de base.

Sa fréquence d'émission ou de réception ainsi que tous les réglages de ce transceiver sont programmables au travers d'un bus série trois fils (*Clock, Data, Enable*).

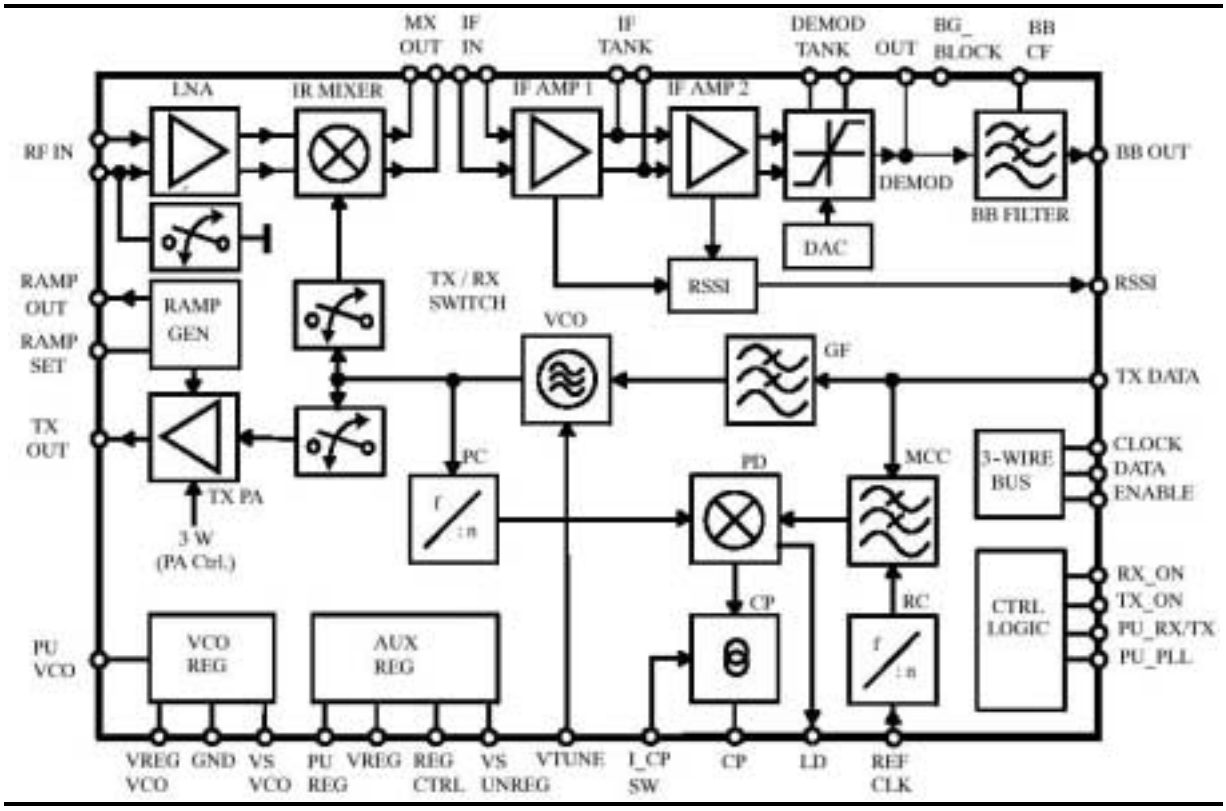


Figure 5-1 Schéma bloc du transceiver T2901

Emission

Les données à transmettre, reçues sur l'entrée TX DATA, sont mises en forme par un filtre Gaussien. Ce filtrage permet de limiter la bande passante occupée par un canal. Après la mise en forme, les données sont transmises au modulateur FM qui va faire varier sa fréquence en fonction des données reçues.

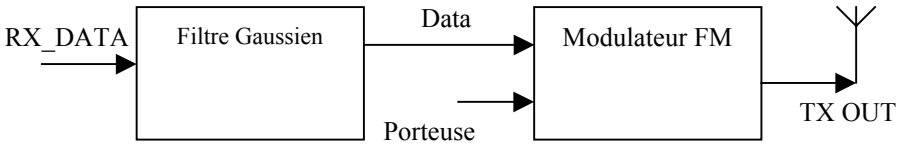


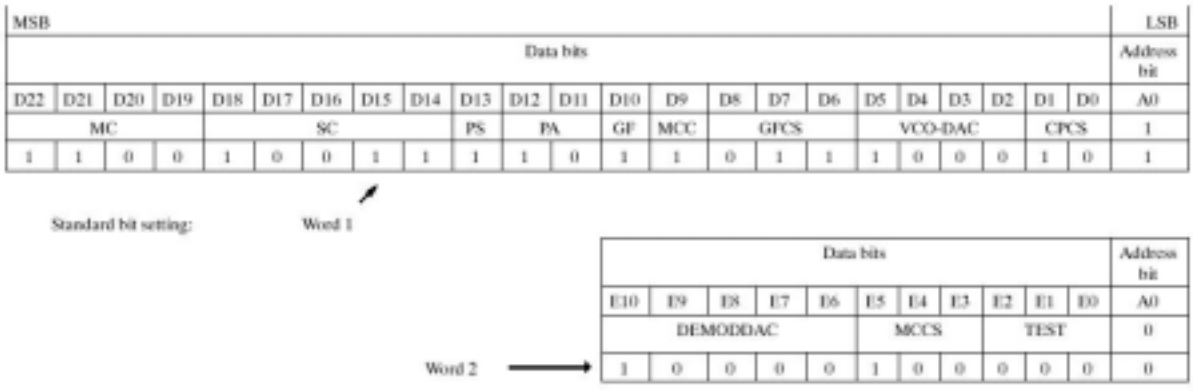
Figure 5-1 Transceiver en émission

Réception

On programme le transceiver avec la fréquence du signal que l'on désire recevoir. Puis, le signal reçu de RF IN est démodulé et converti en un signal analogique BB OUT.

5.2.2. Programmation du transceiver

La programmation du transceiver se fait à l'aide de deux mots envoyés sur le bus série du chip.



MSB																						LSB
Data bits																						Address bit
D22	D21	D20	D19	D18	D17	D16	D15	D14	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0	A0
MC				SC					PS	PA		GF	MCC	GFCS			VCO-DAC				CPCS		1
1	1	0	0	1	0	0	1	1	1	1	0	1	1	0	1	1	1	0	0	0	1	0	1

Standard bit setting: Word 1

	Data bits											Address bit
	E10	E9	E8	E7	E6	E5	E4	E3	E2	E1	E0	A0
	DEMODDAC					MCCS			TEST			0
Word 2 →	1	0	0	0	0	1	0	0	0	0	0	0

Figure 5-1 Format des mots de programmation

Ces deux mots doivent être envoyés sur le bus comme indiqué sur la Figure 5-2.

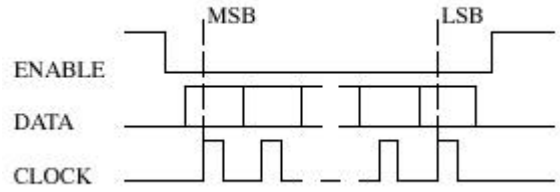


Figure 5-2 Chronogramme du bus de programmation

Ces mots contiennent les informations suivantes :

- Fréquence de la PLL.
- Configuration du DAC
- Configuration du DEMODDAC
- Configuration du filtre Gaussien.
- Configuration du sélecteur de phase.
- Configuration du circuit *Modulation Compensation*.
- Configuration de l'amplificateur de sortie.
- Configuration du *Charge-pump*.
- Configuration du mode de test

Les valeurs proposées et utilisées dans le cadre de ce travail sont les valeurs mesurées sur le kit INDIEN puisqu'il contient exactement le même transceiver. Plusieurs mesures ont été effectuées à l'aide d'un analyseur logique branché sur le bus série du kit INDIEN lors d'émission et de réception.

Fréquence de la PLL

MC (Main Counter)			
D22	D21	D20	D19

SC (Swallow Counter)				
D18	D17	D16	D15	D14

La fréquence de la PLL est inscrite dans les registres *Main counter* (MC) et *Swallow Counter* (SC).

$$F_{VCO} = (MC+64) * 32 + SC$$

En variant f_{VCO} on choisit la fréquence sur laquelle on émet ou reçoit.

$$f_{émission} = ½ * f_{PLL}$$

$$f_{réception} = ½ * f_{PLL} + f_{if}$$

Fréquence intermédiaire du transceiver f_{if} =111MHz.

Configuration du DAC

Pretune DAC Voltage			
D5	D4	D3	D2

Lors de modifications de la fréquence de la PLL, il faut aussi modifier la valeur du VCO pour que la PLL puisse toujours se synchroniser. La valeur du VCO à programmer est indiquée par le graphe de la Figure 5-1 Configuration du DAC, page 74.

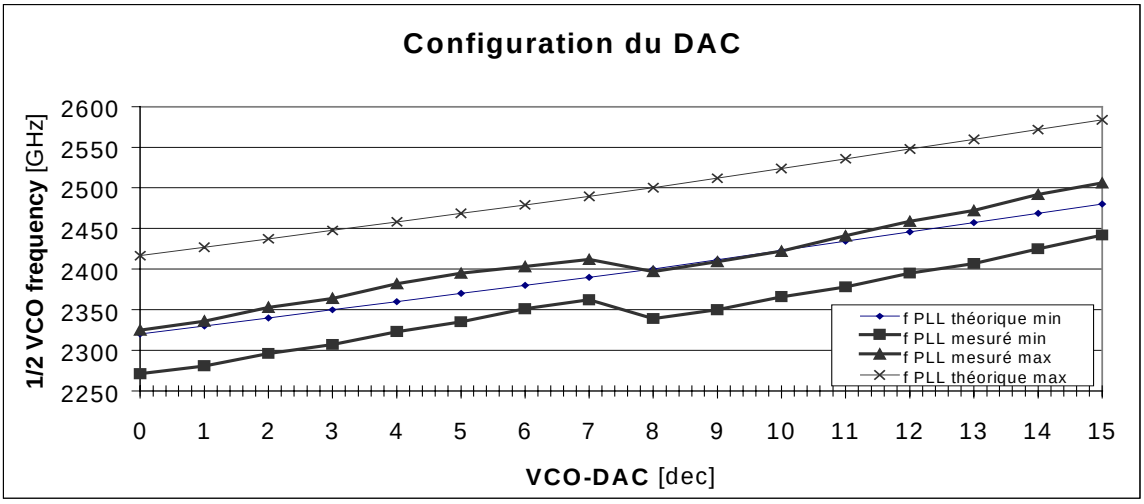


Figure 5-1 Configuration du DAC

Configuration du DEMOD DAC

DEMODDAC (Demod DAC Voltage)					
E10	E9	E8	E7	E6	
0	0	0	0	0	Emission
0	1	0	0	0	Réception

Ce registre n'est pas utilisé en émission.

Configuration du filtre Gaussien.

D10	GF (Gaussian Filter)
0	OFF
1	ON (Valeur Choisie)

Le filtre Gaussien permet de réduire la bande passante utilisée pour transmettre les données en arrondissant les flancs du signal carré.

La valeur du filtre peut être programmée au travers des bits D8 à D6, les valeurs suivantes sont utilisées:

GFCS (Gaussian Filter Current Settings)			
D8	D7	D6	GFCS
1	0	1	Emission
0	0	0	Réception

Configuration du sélecteur de phase.

D13	PS (Phase Select Modulation-Compensation Circuit)
0	Inversé (Valeur choisie)
1	Normale

Configuration du circuit Modulation Compensation.

D9	MCC (Modulation Compensation Circuit)
0	OFF
1	ON (Valeur choisie)

Ce réglage permet d'améliorer le SNR et la durée qu'il faut à la PLL pour changer de fréquence (*frequency Switching*). Les valeurs utilisées sont celles ci-dessous:

MCCS (Modulation Compensation Current Settings)			
E5	E4	E3	MCCS
1	0	1	Emission
0	0	0	Réception

Configuration de l'amplificateur de sortie.

PA (Power Amplifier)		
D12	D11	PA
1	0	-1dBm

L'amplificateur de sortie détermine la puissance radio émise lors de l'émission d'un signal. Cette puissance peut varier lors d'une transmission et peut être réglée par l'application. La valeur ci-dessus a été utilisée.

Configuration du Charge-pump.

CPCS (Charge-Pump Current Settings)		
D1	D0	CPCS
0	0	100%

En augmentant le courant consommé, on peut diminuer le temps de commutation de la PLL mais on augmente le risque d'instabilité de la PLL.

Configuration du mode de test

Test Output Pin			
E2	E1	E0	
0	0	0	Lock detect mode

Lors d'une utilisation normale, le mode de test n'est pas enclenché.

5.2.3. Emission

En mode émission, le transceiver est initialisé comme indiqué dans la Table 5-1.

Puis les deux mots suivants sont transmis au Transceiver:

D22	D21	D20	D19	D18	D17	D16	D15	D14	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0	A0
MC				**SC**					**PS**	**PA**		**GF**	**MCC**	**GFCS**			**VCO-DAC**				**CPCS**		**1**
									0	1	0	1	1	1	0	1					0	0	1

E10	E9	E8	E7	E6	E5	E4	E3	E2	E1	E0	A0
DEMODDAC					**MCCS**			**TEST**			**0**
0	0	0	0	0	1	0	1	0	0	0	0

Les données sont rentrées sur TX_Data.

Les valeurs laissées vides varient à chaque transmission

5.2.4. Réception

En mode réception, le transceiver est initialisé comme indiqué dans la Table 5-1.

Puis les deux mots suivants sont transmis au Transceiver:

D22	D21	D20	D19	D18	D17	D16	D15	D14	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0	A0
MC				**SC**					**PS**	**PA**		**GF**	**MCC**	**GFCS**			**VCO-DAC**				**CPCS**		**1**
									0	1	0	1	1	0	0	0					0	0	1

E10	E9	E8	E7	E6	E5	E4	E3	E2	E1	E0	A0
DEMODDAC					**MCCS**			**TEST**			**0**
0	1	0	0	0	0	0	0	0	0	0	0

Les données sont reçues sur BB OUT et le signal RSSI indique la puissance du signal reçu.

Les valeurs laissées vides varient à chaque transmission.

Table 5-1 Configuration du transceiver en émission et en réception selon les recommandations du fabricant.

Mode	Réception	Emission
PU_PLL, PU_REG	1	1
PU_RX/TX	1	1
RX_ON	1	0
TX_ON	0	1
PU_VCO	1	1
VCO, PLL, prescaler, RX/TX switch	On	On
PA, Ramp Gen, MCC, Gaussian filter	Off	On
LNA ; IR mixer, IF amplifier	On	Off
Demodulator, BB-filter	On	Off

5.3. Mesures du fonctionnement du transceiver

Ces mesures ont été effectuées sur le transceiver du kit TEMIC et sur le transceiver utilisée par le kit INDIEN. Ce sont deux cartes identiques, mais les mesures sur le kit INDIEN ont permis de comprendre les réglages à utiliser sur la carte TEMIC ainsi que le fonctionnement d'un transceiver radio.

De plus ces mesures ont permis de visualiser les paquets reçus et de comprendre les données a traiter. Ceci a permis d'élaborer le filtre de mise en forme.

Finalement, elles montrent le fonctionnement et les timings des paquets reçus.

Une partie des mesures ont été imprimées et se trouvent en annexe.

5.3.1. Protocoles de mesures

Les mesures du kit INDIEN ont été faites sur le connecteur qui relie le transceiver au *Baseband Controller* du boîtier `0xBBBB 0000 BBBB`.
Les mesures du kit TEMIC ont été faites sur la carte d'interface du kit TEMIC.
La transmission ont lieu entre les deux boîtiers INDIEN. Avec le boîtier `0xAAAA 0000 AAAA` maître du piconet qui transmet un fichier texte à son esclave `0xBBBB 0000 BBBB`.
La cartes TEMIC et les deux boîtiers se trouvent dans un rayon de 30 cm les un des autres.

5.3.2. Mesure du signal de données en réception

La vue de l'oscilloscope pour cette mesure se trouve en annexe 7.3, page 119.

Cette mesure montre le signal reçu à la sortie du transceiver. Elle permet de déterminer le filtre à utiliser pour la mise en forme.

Le signal du haut est le signal de données reçues, Rx-data, mesuré à la sortie du transceiver TEMIC. On remarque que le signal à la même amplitude qu'il reçoive du bruit ou des données. Ceci est dû au contrôle de gain automatique (CAG) qui régule l'amplitude du signal

de sortie en fonction de l'inverse du signal RSSI. Ce signal a une amplitude de 0,5V et une composante DC d'environ 1,8V. Mais on remarque que la composante DC varie dans le temps.

La puissance du signal reçu est indiquée par le signal *Received Signal Strength Indicator* (RSSI), au centre sur la mesure. On ne sait pas à quoi est dû le pic contre le bas au centre du signal. On aurait pu supposer que ce signal resterait constant pendant la réception d'un paquet.

Le troisième signal est le signal Rx-data, mesuré sur le kit INDIEN. Ce signal est utilisé pour *trigger* l'oscilloscope, il ne contient pas de bruit car le récepteur n'est enclenché que lorsqu'un paquet est reçu. L'arrivée des paquets est synchronisée avec l'horloge Bluetooth, il est donc possible de connaître exactement leurs instants d'arrivées. On remarque une grande variation de la composante continue pendant les 50µs du début de la trame.

Au sujet des data reçus, on suppose qu'il s'agit d'un paquet POLL. Les paquets NULL et POLL font 126 bits de long et la durée du paquet reçu est d'environ 120 µs. A 1 Mb/s, un paquet de 126 bits dure 126µs, ce qui peut correspondre à notre mesure. Sachant que les mesures sont faites sur l'esclave, on peut déterminer qu'il s'agit d'un paquet POLL et non pas d'un paquet NULL, comme le montre la Figure 5-1.

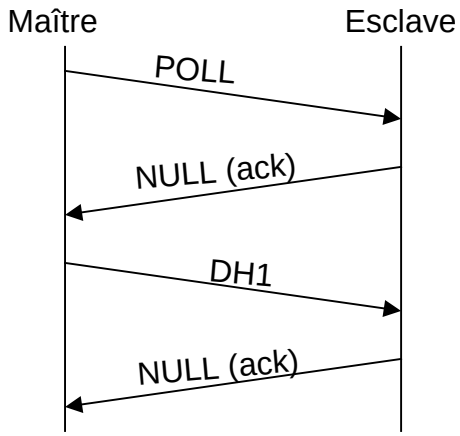


Figure 5-1 Diagramme en flèche d'une partie de la transmission de fichier. La transmission d'un fichier complet se fait en envoyant plusieurs paquet DH1.

Le maître permet à l'esclave d'émettre en lui envoyant un paquet POLL. L'esclave acquitte le paquet reçu avec un paquet NULL. Puis le maître lui envoie une partie du fichier texte dans un paquet DH1, l'esclave acquitte ce paquet en émettant un paquet NULL.

On constate avec cette mesure que le frontend de la carte TEMIC reçoit les données correctement. Le signal Rx-data n'est pas encore aussi clair que celui mesuré sur le kit INDIEN, mais il y a une très bonne ressemblance.

Il faut encore améliorer la programmation du frontend, ceci peut se faire en mesurant les valeurs programmées sur le frontend radio du kit INDIEN. Voir le chap. 5.3.8.

5.3.3. Mesure de l'activité de l'esclave recevant un fichier

La vue de l'oscilloscope pour cette mesure se trouve en annexe 7.4, page 121.

Cette mesure confirme le diagramme en flèche des paquets émis lors d'une transmission de fichier. Cette mesure est faite sur le kit INDIEN, boîtier esclave.

Le signal du haut est le signal Rx_data reçu par le transceiver. Le signal du bas est le signal Tx_data émis par le transceiver.

La structure du dialogue entre le maître et l'esclave apparaît clairement. Le maître émet, l'esclave répond, le maître ré-émet, ainsi de suite.

L'observation de la taille des paquets montre que le maître émet des paquets DH1 qui contiennent une partie du fichier texte transféré, puis régulièrement, il émet un paquet POLL, ces paquets sont plus courts, pour interroger l'esclave et lui permettre de transmettre un paquet.

5.3.4. Mesure de la structure d'un paquet POLL

La vue de l'oscilloscope pour cette mesure se trouve en annexe 7.5, page 123.

Cette mesure à permis de confirmer que les signaux mesurés étais vraiment des trames Bluetooth. Cette mesure est faite sur le kit INDIEN, boîtier esclave.

Le signal mesuré est le signal Rx_data. En observant la forme lors de plusieurs mesures, on constate les choses suivantes :

- La première partie de la trame a une composante continue qui varie pour chaque trame et qui varie aussi à l'intérieur d'une même trame. Cette partie est une suite régulière d'oscillations à 500 kHz qui forment une suite de 1010_b.
- La forme de la deuxième partie, environ 72 µs, est constante pour toutes les trames. Il pourrait s'agir de l'*Access Code* puisqu'il est identique pour tous les paquets à l'intérieur d'un piconet.
- La troisième partie, d'environ 54 µs, varie d'une trame à l'autre, il pourrait s'agir du *Header*, puisqu'il contient des champs qui varient pour chaque trame.

5.3.5. Mesure du signal I_CP_SW

On a mesuré ce signal pour essayer de comprendre son utilité, comment il était généré et s'il dépendait du RSSI. On suppose que ce signal pourrait être activé au début de la réception d'une trame par le transceiver. Il serait alors très utile pour la détection du début des trames reçues.

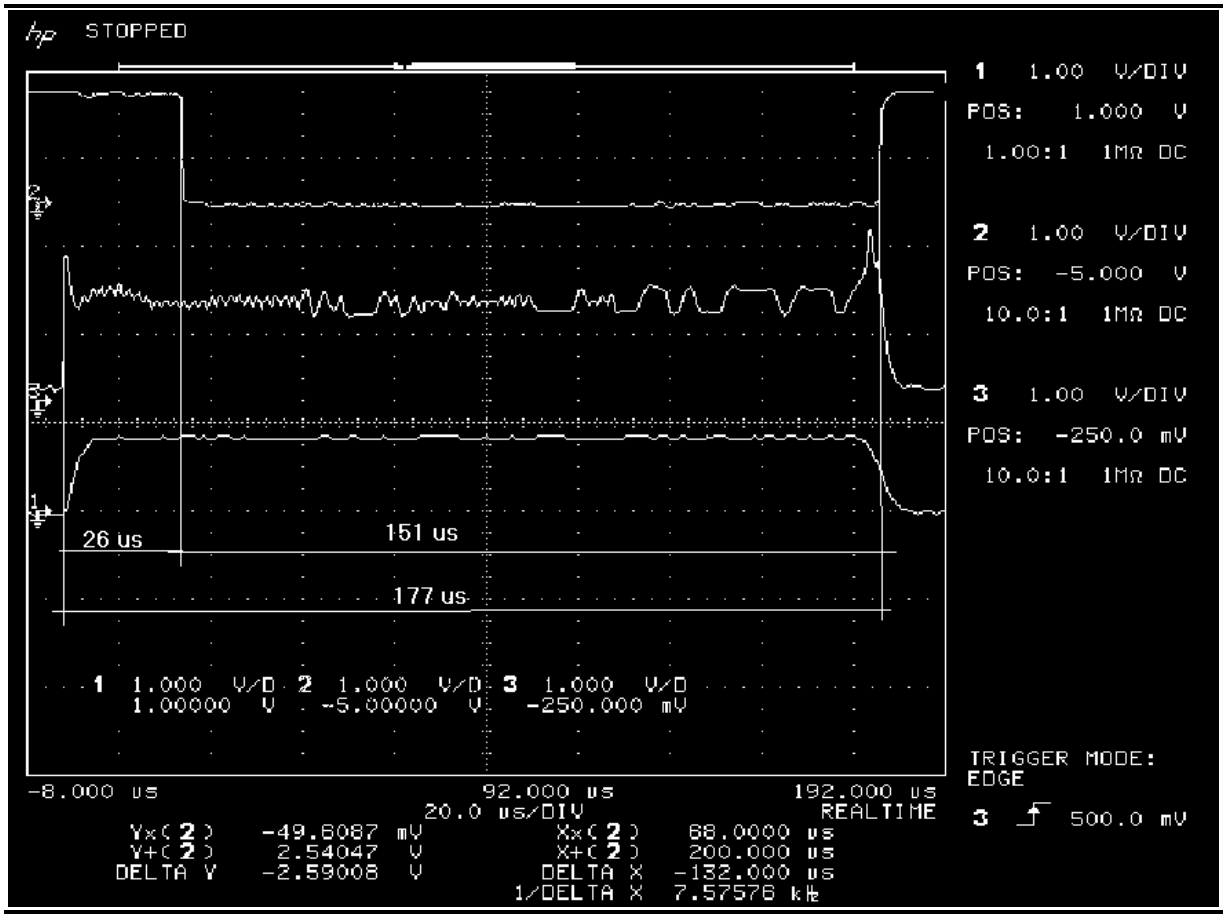


Figure 5-1 Mesure du signal I_CP_SW

Les signaux sur la Figure 5-1 sont, de haut en bas :

Nom	N° du canal
I_CP_SW	2
RX_data	3
RSSI	1

Cette mesure est faite sur le kit INDIEN, boîtier de l'esclave.

Nous avons supposé que le signal I_CP_SW, était un signal créé par le transceiver indiquant le début d'une trame. Malheureusement, ce n'est pas le cas. Le signal I_CP_SW est crée par la Baseband. Il permet à la PLL de recevoir plus de courant et ainsi accélère le changement de fréquence. Le signal est à 1 pendant le changement de fréquence, puis il vient à 0 lorsque la PLL s'est stabilisée. Sur la mesure, le I_CP_SW passe à 0 après le début de la réception d'une trame. On en déduit que le début d'un message ne contient pas d'informations, cela permet simplement de régler le récepteur et de laisser le temps à la PLL de se stabiliser. Si le début de la trame contenait des données utiles, elles seraient certainement perdues.

5.3.6. Mesures de la composition spectrale du signal reçu

Il est utile de connaître la composition spectrale du signal de données reçu. Cela permet de voir s'il y a une raie à la fréquence de l'horloge bit et de définir la largeur du filtre de mise en forme.

Le calcul du spectre du signal reçu a été fait sur MATLAB en utilisant une FFT. Le signal d'un paquet POLL a été récupéré depuis un oscilloscope numérique. Les mesures ont été effectuées sur le kit INDIEN puis sur le kit TEMIC. Les résultats n'ont pas montres de différence significative.

Le code source du script de commande pour le calcul du spectre du signal se trouve en annexe 7.8, page 129.

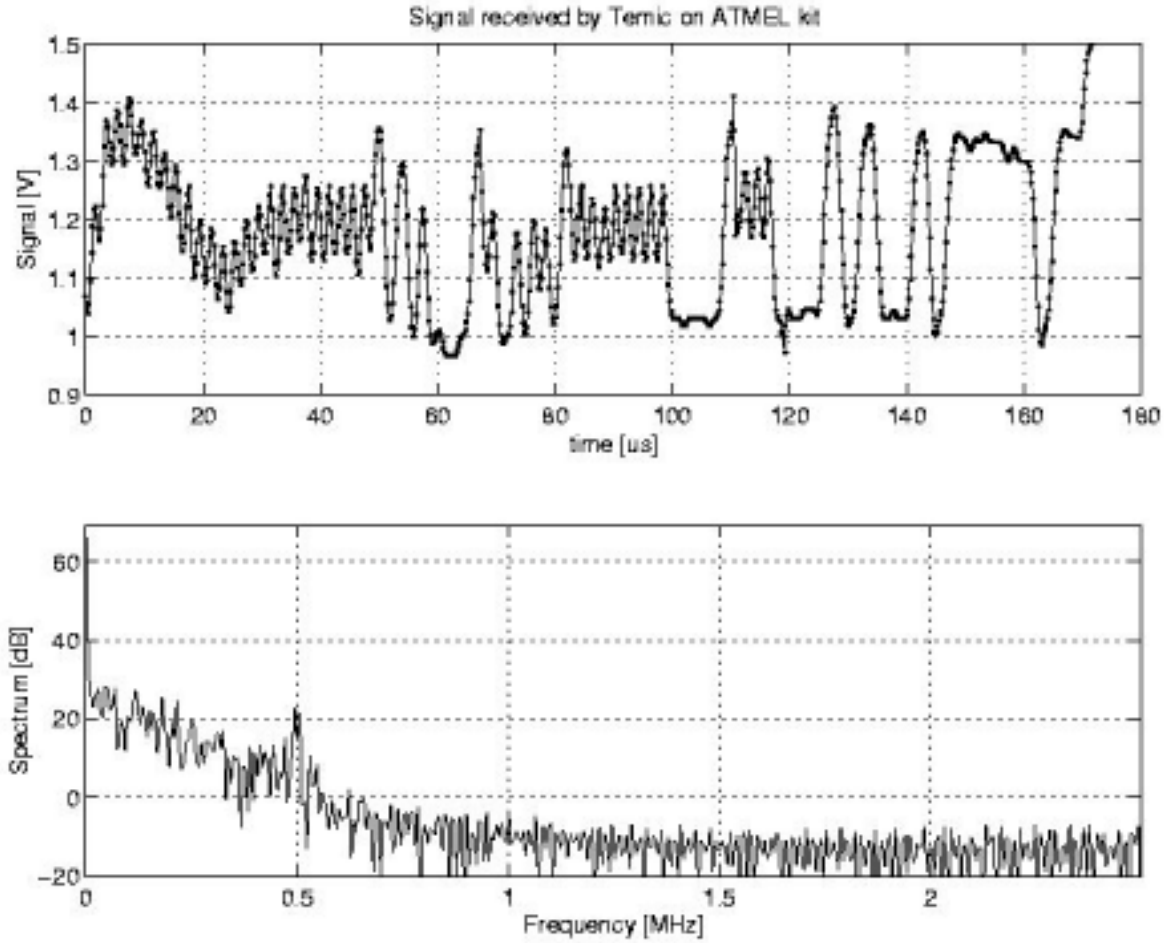


Figure 5-1 Spectre de fréquence du signal de donnée démodulé par le Frontend radio. La partie choisie pour ce calcul est l'*Access Code* d'un paquet POLL.

Le spectre de fréquence de Figure 5-1 comporte une raie à 0.5 MHz, la moitié de la fréquence de transmission ceci est dû au codage NRZ des données.

Ce signal a une importante composante continue et l'amplitude de son spectre diminue avec la fréquence.

5.3.7. Mesure des timings Rx/Tx d'un Inquiry

Le but de cette mesure est de comprendre de manière plus précise les *timings* utilisés par l'initiateur d'un *Inquiry* car la norme n'est pas très explicite a ce sujet, comme expliqué au chap. 2.6 Inquiry, page 31.

Les questions suivantes se posent sur le *timing* de *l'Inquiry*:

- Dans le *slot* Rx de 625 µs, la Baseband doit recevoir deux paquets FHS de 366 bits (366µs). La réception de ces deux paquets n'est pas possible (2*366=732>625).

- La norme [3] indique, chap. 10.7.3 Inquiry, page 110, que les *timings* de l'*Inquiry* sont très semblables à ceux du *Page* pour la réception du paquet FHS. Lors du *Page*, on émet deux paquets ID l'un après l'autre dans un slot Tx, mais dans le *slot* Rx, les paquets FHS sont toujours reçus au début du *slot*. On ne peut donc recevoir qu'un seul paquet FHS pour deux paquets ID émis. Cette façon permet d'avoir le temps de recevoir un paquet FHS entier, mais comment savoir quelle fréquence doit être écoutée puisqu'on peut nous répondre sur les fréquences f'(k) ou f'(k+1) ? De plus comment l'esclave sait qu'il doit envoyer son paquet FHS 312.5µs après la réception du paquet ID et non pas 625µs plus tard ?

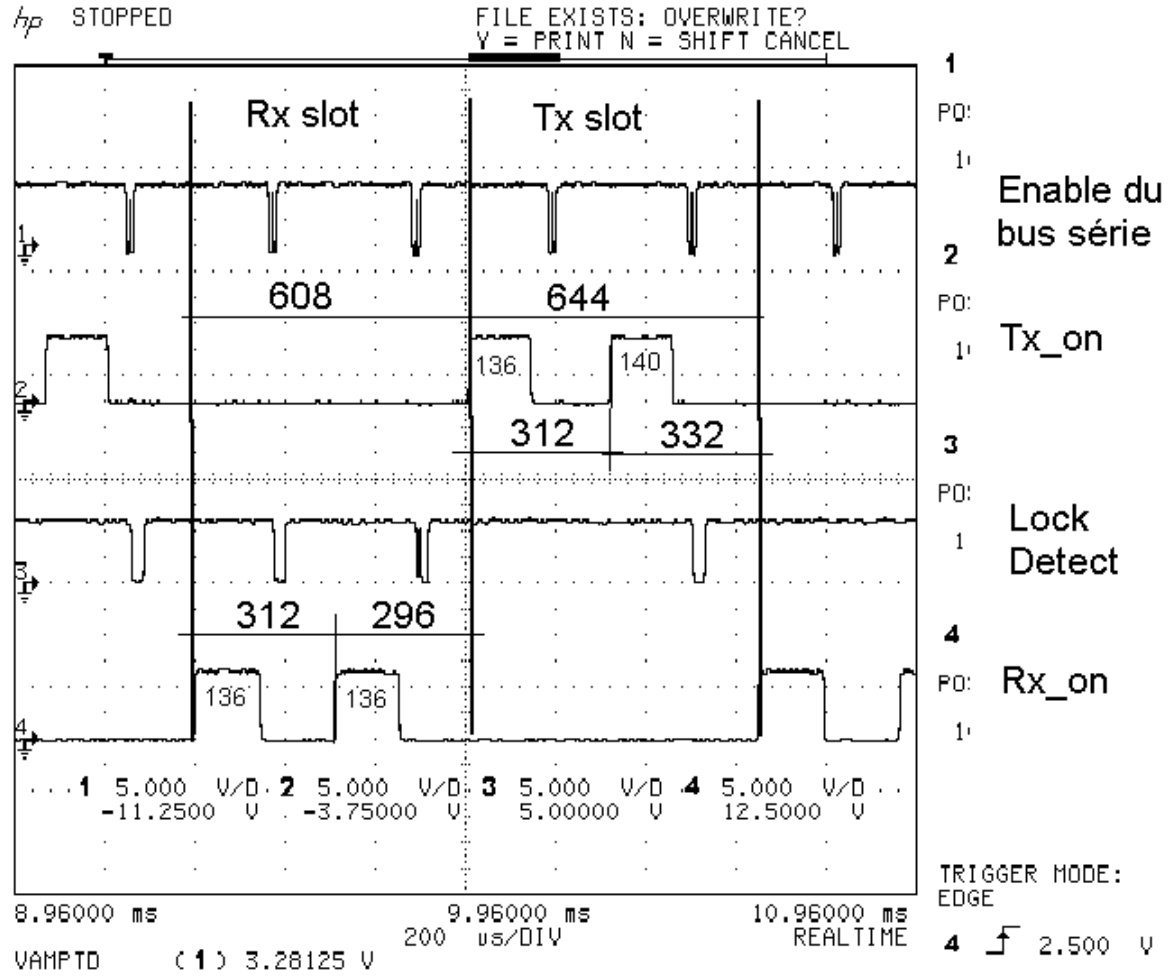


Figure 5-1 Mesure du timing Rx/Tx lors de l'Inquiry

La Figure 5-1 confirme que notre deuxième supposition est fausse, l'initiateur de *l'Inquiry* écoute sur deux fréquences différentes pendant le *slot* Rx à intervalles de 312,5 µs. Le changement de fréquence se remarque au saut du signal *Lock Detect*.

Par contre on remarque qu'il n'écoute pas assez longtemps pour recevoir un paquet FHS en entier (140µ<366µs)

On peut supposer que la Baseband d'INDIEN écoute pendant 136µs sur la fréquence f'_k. Si elle ne détecte pas de paquet FHS elle passe directement sur la fréquence $f'_{(k+1)}$. Mais comment se passe la réception lorsque le paquet FHS est reçu dans la deuxième moitié du slot.

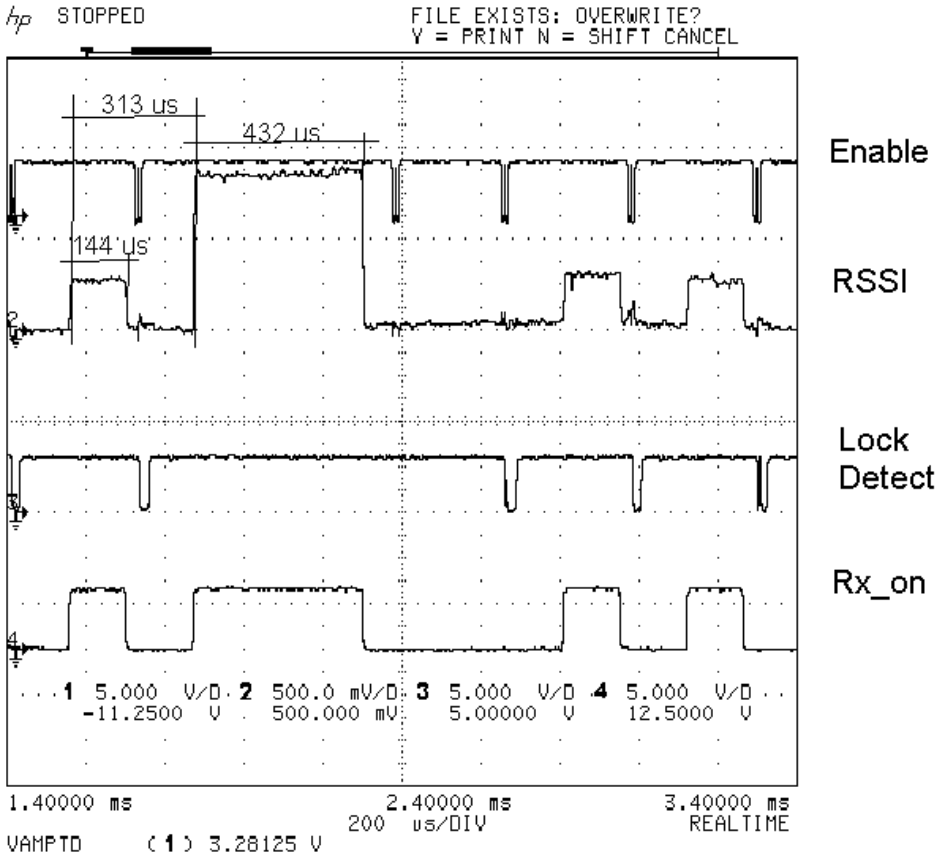


Figure 5-2 Mesure de la réception d'un paquet FHS dans la deuxième moitié d'un *slot* Rx.

La Figure 5-2 permet de comprendre ce qui se passe lors de la réception d'un paquet FHS dans la deuxième moitié d'un *slot* Rx. La réception du paquet FHS déborde sur le prochain *slot* Tx et supprime l'émission du paquet ID de la première moitié de ce *slot*. Sur cette figure, le signal RSSI confirme l'arrivé d'un paquet FHS. La durée de 432µs correspond à la durée du paquet FHS (366µs) additionné du temps de stabilisation de la PLL (70µs).

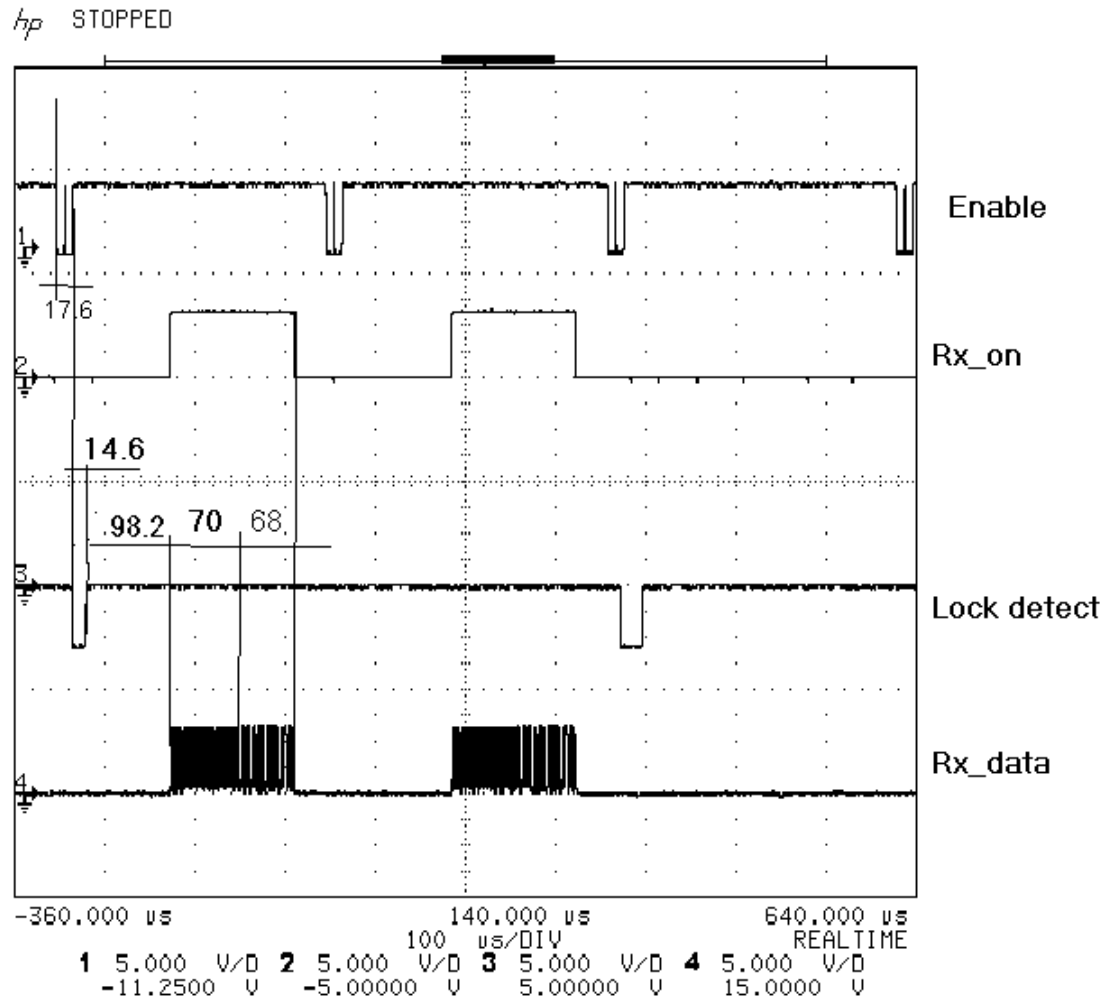


Figure 5-3 Mesure du temps de changement de fréquence du frontend radio. Les temps sont donnés en µs

La Figure 5-3 permet de connaître plus exactement les temps de commutation du frontend radio. L'envoi des deux mots de programmation sur le bus série dure 17.6 µs. Après cela, il y a 14.6 µs d'attente pour que la PLL se stabilise. Avant de démarrer la transmission, il y a une attente de 93.2 µs. Puis l'émission démarre. Le début de l'émission consiste en une suite de 101_b pendant 70 µs, puis le paquet ID est envoyé en 68 µs

Il aura fallut 200.4µs au frontend pour changer de fréquence, ce qui est moins que les 254µs donnés par le constructeur.

Conclusion

Cette mesure a permis de s'assurer de la compréhension des timings de l'*Inquiry*.

On remarque que le frontend reçoit bien les paquets FHS sur deux fréquences différentes, mais il coupe la réception à la première fréquence s'il ne reçoit pas de paquet FHS à ce moment.

5.3.8. Mesures des valeurs programmées sur le frontend radio du kit INDIEN

Le but de ces mesures est de connaître les valeurs programmées sur le frontend INDIEN pour pouvoir programmer le frontend de la carte TEMIC de la même manière et ainsi d'optimiser les transmissions

Ces mesures ont été faites à l'aide d'un analyseur logique placé sur le bus série du frontend du kit INDIEN.

Les chronogrammes des mesures se trouvent en annexe 7.6 Mesures des valeurs programmées sur le kit INDIEN, page 125.

Mesures en émission

Pour toutes les émissions, les valeurs programmées sont identiques, sauf les valeurs des compteurs (MC et SC) qui contrôlent la fréquence et le champ VCO-DAC qui dépend directement de la fréquence.

Emission d'un paquet ID lors d'un Inquiry

Mot 1

D22	D21	D20	D19	D18	D17	D16	D15	D14	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0	A0
MC				**SC**					**PS**	**PA**		**GF**	**MCC**	**GFCS**			**VCO-DAC**				**CPCS**		**1**
1	1	0	0	0	0	1	0	1	0	1	0	1	1	1	0	1	1	1	0	0	0	0	1

Mot 2

												E10	E9	E8	E7	E6	E5	E4	E3	E2	E1	E0	A0
												DEMODDAC					**MCCS**			**TEST**			**0**
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	0	0	0

Emission d'un paquet ACK lors d'un transfert de fichier

Mot 1

D22	D21	D20	D19	D18	D17	D16	D15	D14	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0	A0
MC				**SC**					**PS**	**PA**		**GF**	**MCC**	**GFCS**			**VCO-DAC**				**CPCS**		**1**
1	1	0	1	0	0	0	0	0	0	1	0	1	1	1	0	1	1	1	1	0	0	0	1

Mot 2

												E10	E9	E8	E7	E6	E5	E4	E3	E2	E1	E0	A0
												DEMODDAC					**MCCS**			**TEST**			**0**
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	0	0	0

Conclusion

INDIEN utilise les mêmes valeurs pour envoyer un paquet ID lors d'un *Inquiry* que lors de l'émission d'un paquet de données pendant une connexion entre le maître et l'esclave.

Les valeurs que l'on pouvait déduire de la norme comme l'utilisation du filtre Gaussien(GF=1), le réglage de l'amplificateur (PA) inférieur à 0dBm, le DEMODDAC inutilisé (DEMODDAC = 0) et les valeurs du champ TEST sont correctes.

On découvre par contre qu'il faut inverser la phase (PS=0), que le circuit *Modulation-Compensation* est enclenché (MCC =1) et on connaît maintenant les valeurs du filtre Gaussien à utiliser (GFCS) et du circuit de *Modulation-Compensation*(MCCS).

Mesures en réception

Réception d'un paquet FHS lors d'un Inquiry

Mot 1

D22	D21	D20	D19	D18	D17	D16	D15	D14	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0	A0
MC				**SC**					**PS**	**PA**		**GF**	**MCC**	**GFCS**			**VCO-DAC**				**CPCS**		**1**
1	0	0	1	1	1	0	1	1	0	1	0	1	1	0	0	0	1	0	0	1	0	0	1

Mot 2

												E10	E9	E8	E7	E6	E5	E4	E3	E2	E1	E0	A0
												DEMODDAC					**MCCS**			**TEST**			**0**
0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0

Réception d'un paquet DH1 lors d'un transfert de fichier

Mot 1

D22	D21	D20	D19	D18	D17	D16	D15	D14	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0	A0
MC				**SC**					**PS**	**PA**		**GF**	**MCC**	**GFCS**			**VCO-DAC**				**CPCS**		**1**
1	0	0	1	1	1	0	1	1	0	1	0	1	1	0	0	0	1	0	0	1	0	0	1

Mot 2

												E10	E9	E8	E7	E6	E5	E4	E3	E2	E1	E0	A0
												DEMODDAC					**MCCS**			**TEST**			**0**
0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0

Conclusion

En réception, seules le GFCS et le MCCS sont programmés avec des valeurs différentes que lors d'une émission. Les champs PS et Test restent inchangés. Les champs PA (*Power Amplifier*) et GF(*Gaussian Filter*) restent toujours enclenchés bien que le constructeur recommande dans la Table 5-1, page 77, de les arrêter lors de la réception.

Comme en émission, les champs MC, SC et VCO-DAC changent lors de chaque réception.

5.3.9. Conclusion

Ces mesures permettent de comprendre beaucoup de choses sur le fonctionnement du transceiver et du protocole Bluetooth.

- Les données transmises correspondent à la norme Bluetooth. La vitesse de transmission est de 1Mb/s.
- A part le signal RSSI, il n'y a aucun moyen de connaître le début d'un paquet. De plus le signal RSSI n'indique pas le début d'un paquet mais indique la présence d'une porteuse. Le début du paquet se trouve environ 50µs après l'apparition de la porteuse. Cette durée n'est pas constante.
- Le signal de donnée Rx_data, contient une composante continue qui varie avec le temps. Il faudra la supprimer avec le filtre, ceci est dû à la stabilité de la PLL de transmission.

- L'amplitude du signal Rx_data, n'est pas très grande, il faudra l'amplifier pour pouvoir l'exploiter.
- Le signal est codé en NRZ.
- La réception des deux paquets FHS d'un même *slot* Rx se fait à des moments différents et à deux fréquences différentes.
- Les valeurs à utiliser pour la programmation du frontend radio sont maintenant connues.

5.4. Circuit de mise en forme

Le circuit de mise en forme est constitué d'un filtre passe-bande suivi d'un comparateur à hystérèse et de deux fois trois portes inverseur trigger de Schmitt. Le filtre permet d'éliminer la composante continue et le bruit supérieur à 10Mhz. Le comparateur crée un signal carré à partir du signal de sortie du filtre et les portes trigger de Schmitt permettent d'isoler le circuit de mise en forme des parasites créés par le circuit de recouvrement de l'horloge.

5.4.1. Dimensionnement du filtre coupe-bande

Le but de ce filtre est de supprimer la composante continue ainsi que les bruits du signal et de ne laisser passer que les données.

La durée d'un bit est de 1µs à 1Mb/s. En supposant que les suites de bits à 1 ou à 0 dépasseront rarement 5 bits. On choisit comme fréquence de coupure basse f_{c1}, dix fois ce temps, c'est une règle empirique basée sur le temps de charge du condensateur de découplage.

$$f_{c1} = \frac{1}{10 \cdot 5 \cdot 10^{-6}} = 20kHz$$

La supposition que les suites de 1 ou de 0 seront rarement supérieures à 5 est fausse, comme on s'en apercevra plus tard dans le chap.5.4.4, page 90. La sélectivitée du filtre sera modifée en conséquence.

De plus, pour supprimer les parasites qui ont des fréquences supérieures au signal. On fixe donc f_{c2} à 10MHz de façon à laisser passer uniquement les harmoniques du signal inférieures à 10MHz.

$$f_{c2} = 10 \text{ MHZ.}$$

En utilisant la fonction de transfert du filtre donnée par :

$$A = -\frac{R2}{R1} \cdot \frac{j \cdot \omega \cdot R1 \cdot C1}{(1 + j \cdot \omega \cdot R1 \cdot C1) \cdot (1 + j \cdot \omega \cdot R2 \cdot C2)}$$

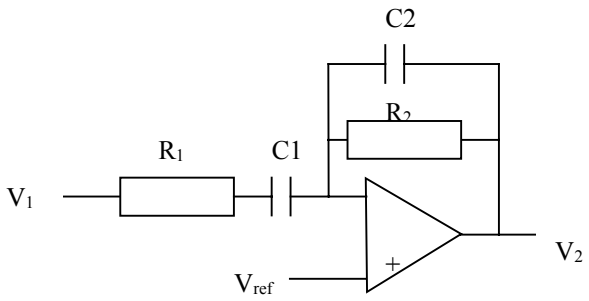


Figure 5-1 Schéma d'un filtre passe-bande

Les valeurs suivantes ont été déterminées par approximations successives :

R_1= 2 K C_1= 4.7 nF

R_2= 8.2 K C_2= 2.2 pF

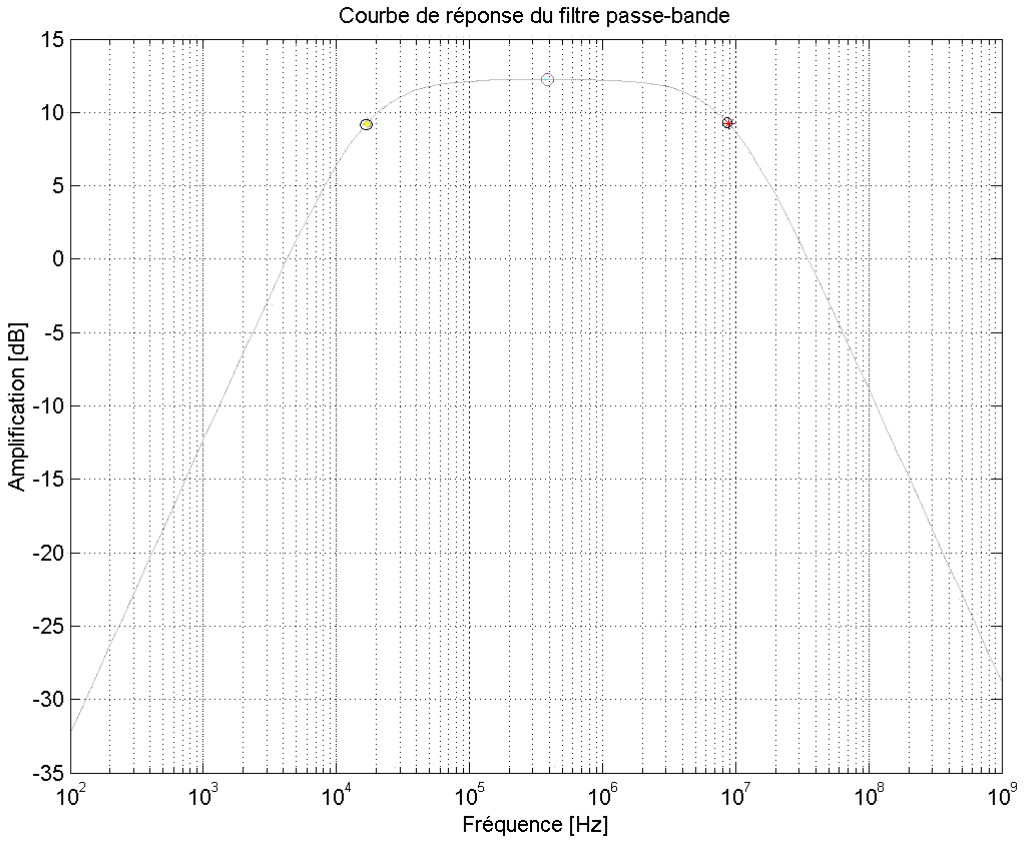


Figure 5-2 Courbe de réponse du filtre passe-bande.

Le scripte Matlab qui calcule la fonction de transfert du filtre se trouve en annexe 7.8

fc1=16.9 KHz f0= 388,6 KHz fc2= 8.2MHz

Av= - 4.1

Le codage utilisé est le codage NRZ, la fréquence du signal lors d'une suite de «1010 » sera de 500Khz, fréquence que notre filtre laisse passer sans atténuation.

5.4.2. Dimensionnement du comparateur à hystérèse

Une bascule de Schmitt permet de rendre le signal carré et de supprimer les petites oscillations du signal grâce à son seuil de basculement qui présente une hystérèse.

La largeur de l'hystérèse dépend de R_1 et R_2 selon la formule :

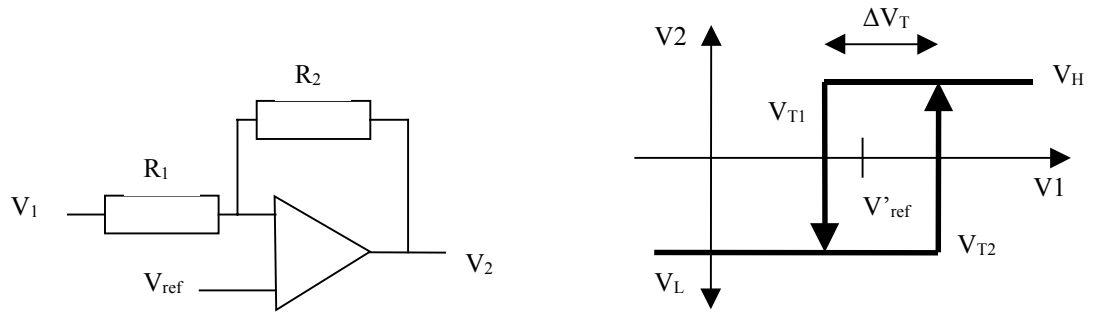


Figure 5-1 Schéma et fonction de transfert de la bascule de Schmitt

$$\Delta V_T = (V_H - V_L) \cdot \frac{R_1}{R_2}$$

Pour pouvoir modifier facilement ΔV_T, la résistance R2 sera un pot de 50kΩ en série avec une résistance de 10kΩ. Ainsi ΔV_T peut varier entre

$$\Delta V_{T\min} = (V_H - V_L) \cdot \frac{R_1}{R_2} = (9-0) \cdot \frac{1 \cdot 10^3}{10 \cdot 10^3 + 50 \cdot 10^3} = 150mV$$

$$\Delta V_{T\max} = (V_H - V_L) \cdot \frac{R_1}{R_2} = (9-0) \cdot \frac{1 \cdot 10^3}{10 \cdot 10^3} = 900mV$$

Le niveau de décision V_{ref} devra aussi pouvoir être modifié facilement, on utilisera un pot de 10kΩ branché entre +Vcc et la masse, ainsi V_{ref} peut varier entre 0 et 9V.

L'ampli-op utilisé est le CA3140, c'est un ampli-op à usage général fabriqué par Harris. Cet ampli-op ne se révélera pas assez rapide et sera échangé avec un LM311 par la suite. Voir chap.5.4.3 Protocole de mesures du circuit de mise en forme*,90.*

Le schéma électrique du filtre de mise en forme se trouve en annexe 7.10 Schéma électrique du filtre de mise en forme, page 133.

5.4.3. Protocole de mesures du circuit de mise en forme

Les mesures sur le circuit de mise en forme et celles du circuit de recouvrement de l'horloge ont étés faites en utilisant le transceiver radio T2901 du kit TEMIC comme récepteur radio. Ce transceiver n'a pas de contrôleur de Baseband qui lui permet de suivre les sauts de fréquences d'un canal Bluetooth. Pour résoudre ce problème et puisque cette carte radio est la même que celle utilisée par le kit INDIEN, le bus série qui programme le transceiver du kit INDIEN a été branché en dérivation sur le transceiver radio du kit TEMIC. Ainsi le transceiver du kit TEMIC était synchronisé avec le piconet créé entre les deux boîtiers INDIEN.

Le boîtier INDIEN avec l'adresse `0xAAAA 0000 AAAA` est le maître du piconet et le boîtier `0xBBBB 0000 BBBB` son esclave.
Pour créer du trafic sur le réseau, un fichier texte a été transféré depuis le maître sur l'esclave.
Le bus série du kit TEMIC était branché sur le bus série du boîtier INDIEN esclave `0xBBBB 0000 BBBB`.
La cartes TEMIC et les deux boîtiers INDIEN se trouvent dans un rayon de 30 cm les uns des autres.

5.4.4. Réglages et mesures du circuit de mise en forme

Les premières mesures faites sur le filtre ne sont pas très encourageantes. Le filtre n'est pas assez sélectif car il y a toujours des oscillations basse-fréquence sur le signal de données ce qui empêche le réglage correct du niveau de décision $V_{réf}$.

Figure 5-1 Problème de réglage du niveau de décision dû aux oscillations basse-fréquence (23Oct00print4)

Les signaux sur la Figure 5-1 sont, de haut en bas :

Nom	N° du canal
RSSI (trigger)	4
Pt de mesure B, sortie du comparateur.	1
Pt de mesure A, sortie du filtre passe-bande.	2
V_{ref}, entrée négative du comparateur U2, niveau	3

L'hystérèse du comparateur est réglée au minimum, P2 vissé au maximum dans le sens horaire.

On voit clairement sur cette mesure que la composante continue du signal de donnée au pt A varie continuellement d'environ 700mV. La plus faible amplitude du signal de données est de 500mV environ, il est donc impossible de régler le niveau de décision de façon correcte. De plus, la composante continue dépend du signal de sortie du comparateur. Lorsque le signal de sortie est à 1, la composante DC est supérieure au niveau de décision. Si le signal de sortie passe à 0, la composante continue passe en-dessous du niveau de décision.

Pour supprimer ces oscillations, on enlève la contre-réaction sur le comparateur ce qui empêche le signal au pt B de faire varier le signal au point A. On augmente aussi F_{c1} en diminuant C_1 à 560pF pour réduire encore plus les oscillations dues aux basses fréquences du signal de données.

En plus de ces modifications, des condensateurs de découplages ont été rajoutés sur l'alimentation des IC, sur l'entrée positive de U1 et sur l'entrée négative de U2. Ceci pour supprimer les parasites haute-fréquence se trouvant sur la tension de référence et sur le signal de données. De plus on diminue la fréquence de coupure supérieure du filtre (f_{c2}) en mettant un condensateur de 10pF, ceci aussi pour supprimer les parasite haute-fréquence.

Après ces modifications, les ondulations ont disparu. Malheureusement le slew-rate de l'ampli-op utilisé dans le comparateur n'est que de 9V/µs, ce qui est très insuffisant puisque la durée d'un bit est de 1 µs. L'ampli-op utilisé dorénavant sera le LM311 qui a un slew-rate de 260V/ µs.

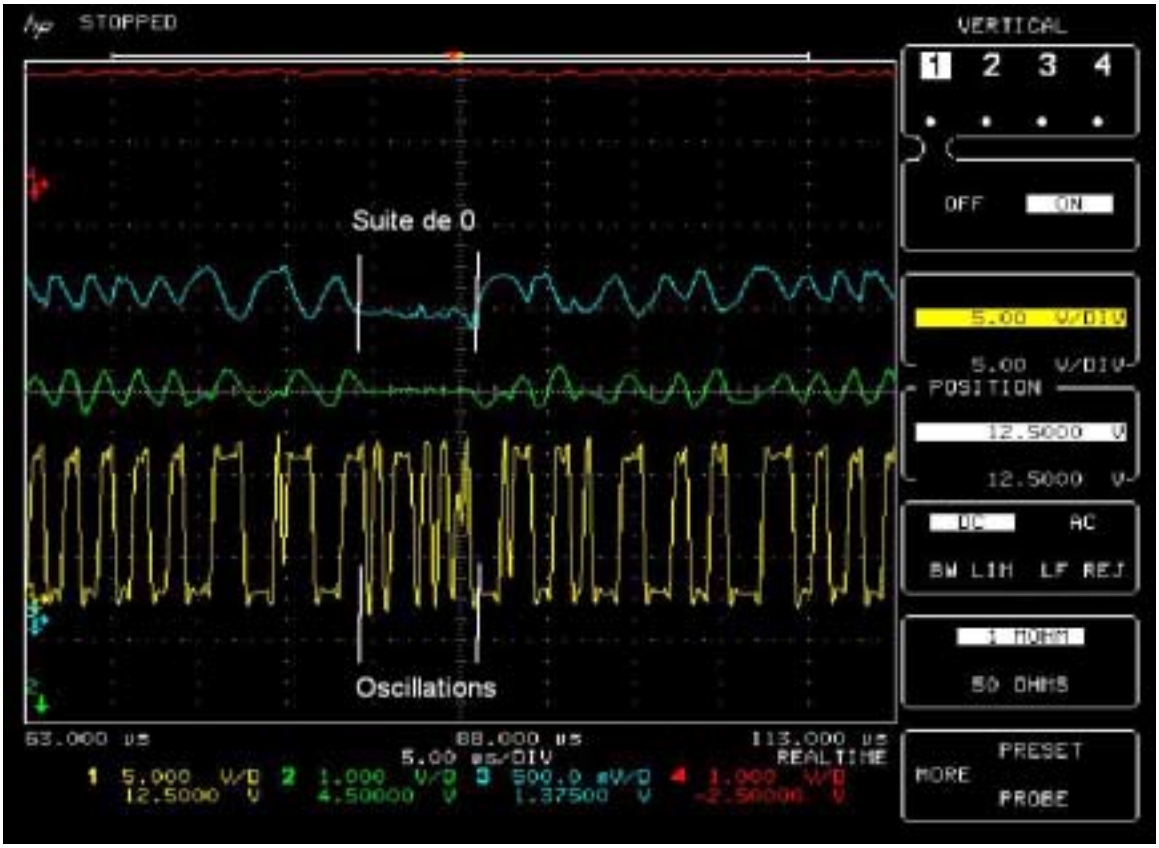


Figure 5-2 Problème lors de longue suite de 1 ou de 0 (27Oct001146)

Les signaux sur la Figure 5-2 sont, de haut en bas :

Nom	N° du canal
RSSI (trigger)	4
IN. entrée du circuit de mise en forme	3
Pt de mesure A, sortie du filtre passe-bande.	1
Pt de mesure B, sortie du comparateur	2

Cette mesure montre que la composante continue du signal à la sortie du filtre passe-bande est constante, il est donc possible de régler correctement le niveau de décision. Un autre problème apparaît maintenant, lors de longue suite de 0 ou de 1, le signal à la sortie du comparateur se met à osciller, ceci est dû à la trop grande sélectivité du filtre.

En calculant l'*Access Code* de la trame que l'on reçoit, voir annexe 7.14 Exemple complet de calcul du *Sync Word*, page 143, on s'apercoit que les suite de 1 ou de 0 peuvent atteindre 10 bits, alors que nous avions supposé qu'elle ne dépasseraient pas 5 bits.

Pour résoudre ce problème, on augmente C1. Après plusieurs essais, il s'avère que la valeur de 68nF pour C_1 élimine le problème d'oscillations.

Après ces dernières modifications, on obtient la mesure suivante :

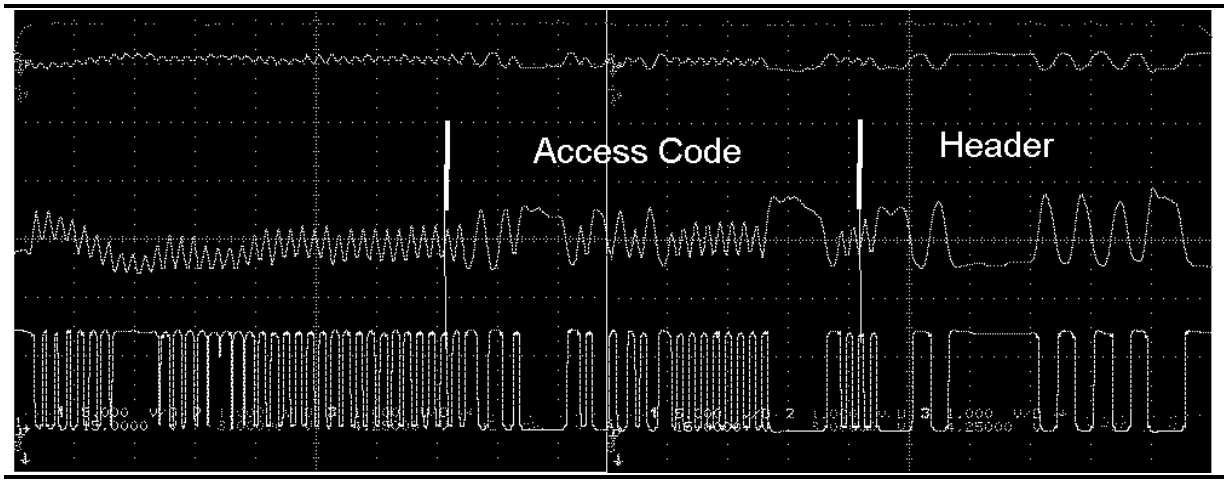


Figure 5-3 Mise en forme d'une trame complète (30Oct00).

L'Access Code de la Figure 5-3 est :

`1010 1100 1100 1000 0000 0110 1000 1001 0100 1101 0101 0101 0101 0100 0000 0000 1101 0101`$_b$

Les signaux sur la Figure 5-3 sont, de haut en bas :

Nom	N° du canal
RSSI (trigger)	4
IN. entrée du circuit de mise en forme	3
Pt de mesure A, sortie du filtre passe-bande.	1
Pt de mesure OUT2, sortie du circuit de mise en	2

Cette mesure nous montre le fonctionnement correct du circuit de mise en forme. Au début de la trame, les premiers bits `101`$_b$... ne sont pas tous correctement interprétés, cela n'est pas important car ils n'ont pas de significations particulières. Ils permettent de donner au filtre le temps de s'ajuster. A ce moment, les oscillations sur le signal IN sont dues à la PLL qui ne s'est pas encore stabilisée. Puis on reçoit l'*Access Code* et ensuite le *Header*.

L'*Access Code* reçu correspond exactement à l'*Access Code* calculé en annexe 7.14.

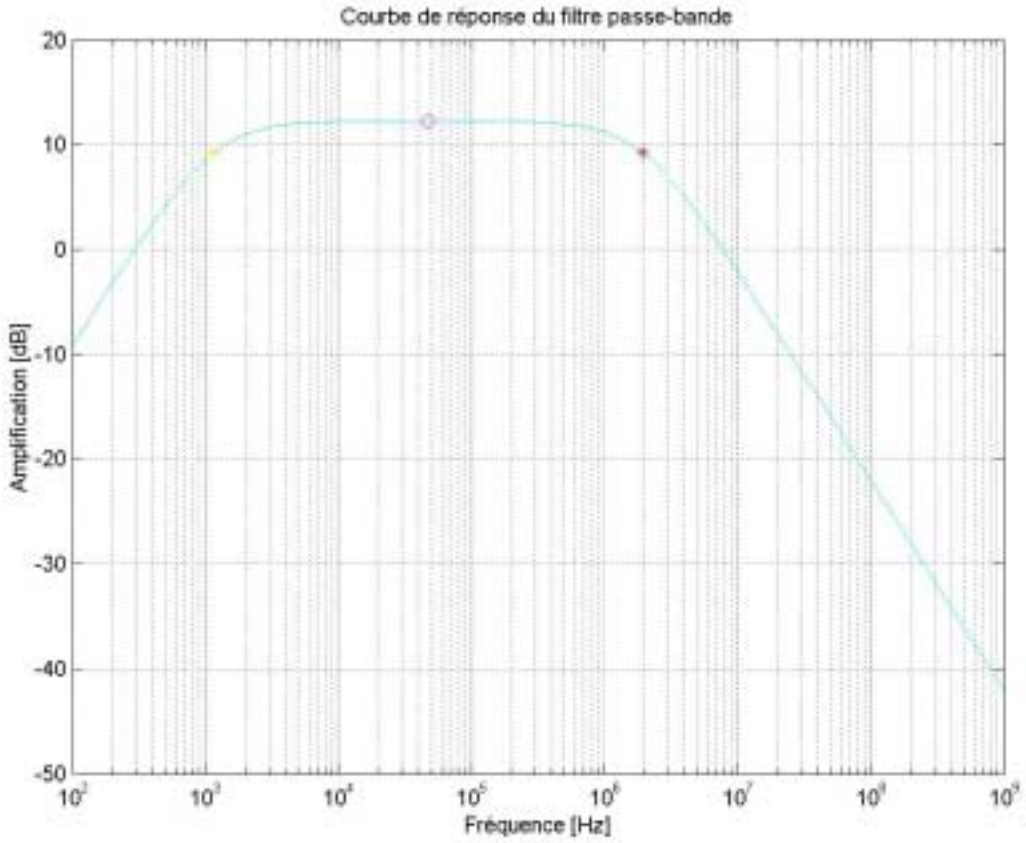


Figure 5-4 Courbe de réponse du filtre final

fc1=1.17 KHz f0= 47.7 KHz fc2= 1.94 MHz

Av= - 4.1

La Figure 5-4 représente la courbe de réponse du filtre final. Par rapport au filtre calculé au départ, la courbe de réponse du filtre final est moins sélective et elle a été déplacée à une fréquence 10 fois inférieure.

5.5. Recouvrement de l'horloge

5.5.1. Introduction

Le recouvrement de l'horloge permet de créer un signal d'horloge en phase avec le signal reçu. Pour ce faire, il a été implémenté une PLL numérique.

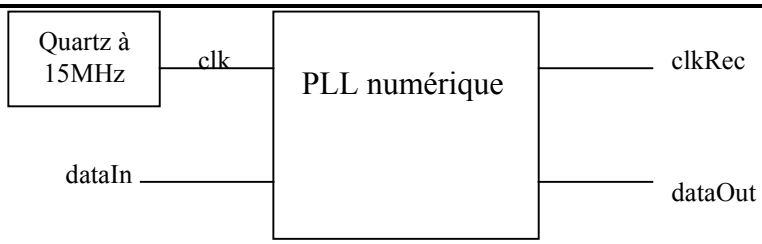


Figure 5-1 Schéma bloc de la PLL numérique

5.5.2. Implémentation

La PLL est constituée d'un compteur 4 bits de 0 à 14 et d'un détecteur de flanc qui détecte les flancs du signal dataIn. Le compteur divise le signal clk par 15 pour obtenir un signal à 1MHz. Puis à chaque changement d'état du signal dataIn, la PLL teste le compteur et le resynchronise avec le signal dataIn. Si au moment du changement d'états le compteur est en retard, la PLL va le faire avancer, alors que s'il est en avance, la PLL va le retarder. Le signal d'horloge retrouvé est la sortie MSB du compteur 4 bits.

Compteur	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

ClkRec

	Ok	Compteur en avance			Compteur en retard		
Action sur le compteur	+1	-1	-2	-	+3	+2	Cpt = 0

Figure 5-1 Synchronisation de la PLL avec le signal data

Si le compteur vaut 7 lors d'un changement d'états de data, on suppose qu'il y a une erreur dans le fonctionnement du système et on ne corrige pas.

On détecte les flancs du signal data en le faisant passer à travers deux portes inverseurs en séries. Ce système retarde le signal data. Le signal d'horloge est en phase avec le signal retardé Data2 et pas avec le signal dataIn reçu en entrée.

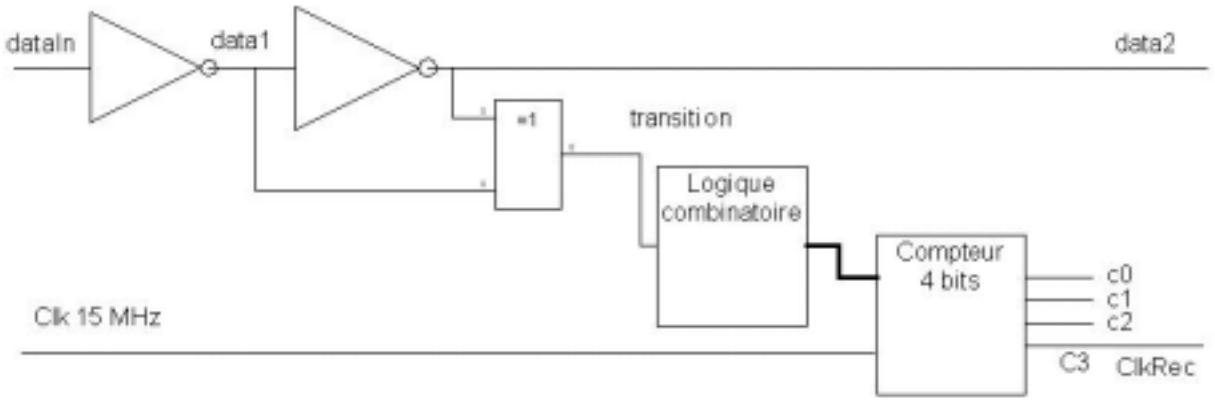


Figure 5-2 Implémentation du détecteur de flancs

Le code de la PLL numérique est écrit en ABEL. Il est ensuite programmé dans une CPLD XC9536 ou XC95108 de Xilinx.

Le programme se trouve en annexe 7.13 PLL numérique.

5.5.3. Mesures

Sur la mesure de la Figure 5-1 on voit que le signal d'horloge clkRec est bien en phase avec le signal de données data2.

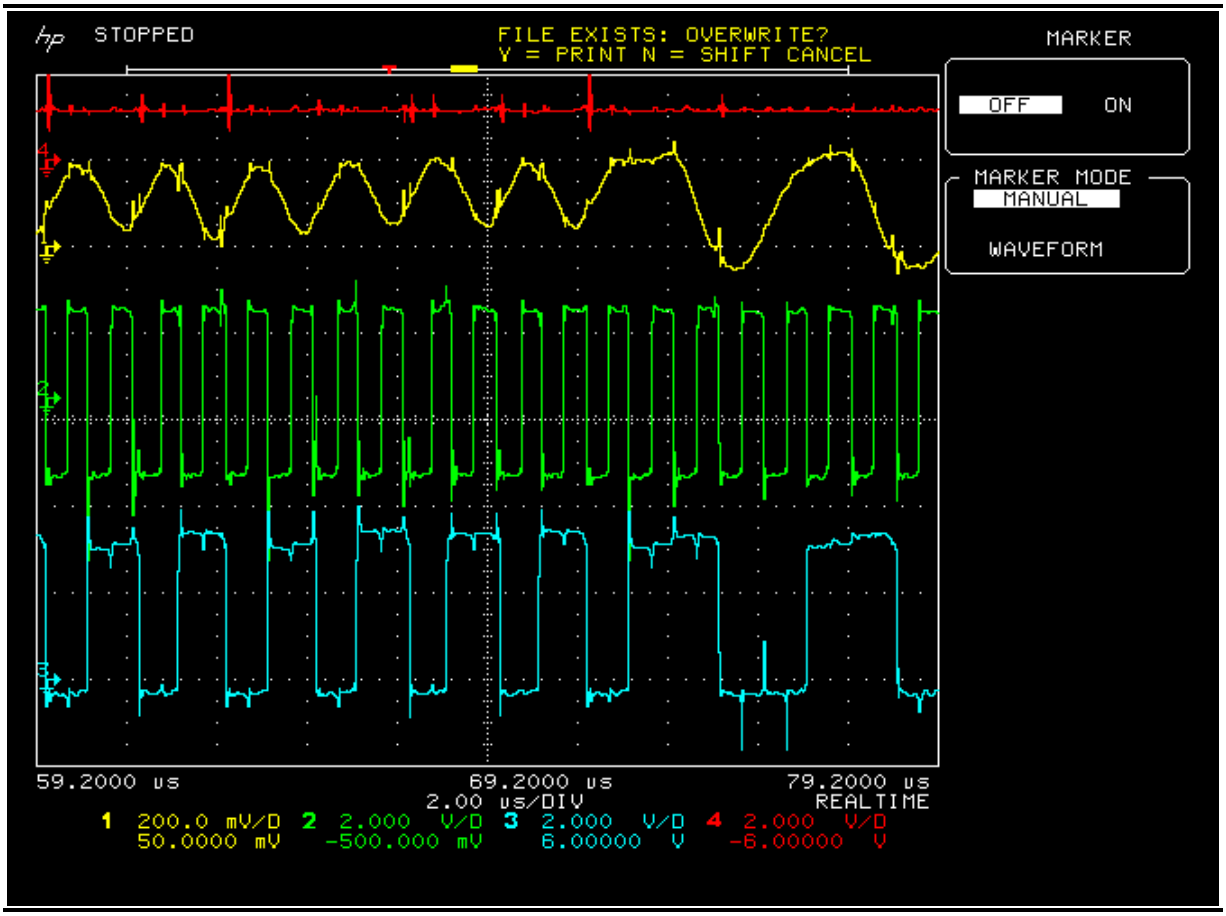


Figure 5-1 Fonctionnement du recouvrement de l'horloge

Les signaux sur la Figure 5-1 sont, de haut en bas :

Nom	N° du canal
RSSI (trigger)	4
RFRDATA	1
ClkRec	2
Data2	3

La mesure de la Figure 5-2 montre que même pour de longue suite de 0 sur le signal DataIn, le signal d'horloge clkRec reste en phase avec les données data2.

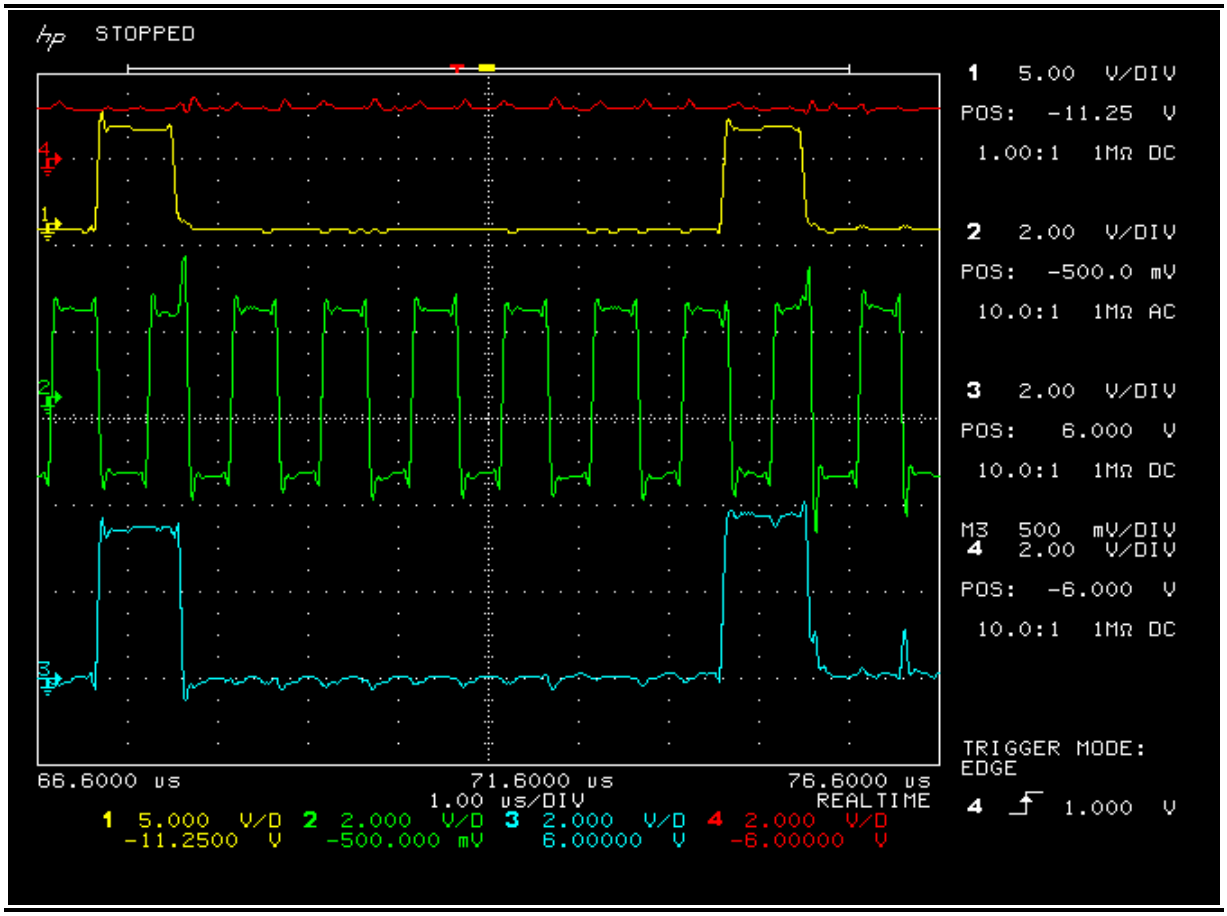


Figure 5-2 Fonctionnement du recouvrement de l'horloge lors de longue suite de 0

Les signaux sur la Figure 5-2 sont, de haut en bas :

Nom	N° du canal
RSSI (trigger)	4
DataIn	1
ClkRec	2
Data2	3

6. Implémentation du software

6.1. Contraintes

Les contraintes imposées au software sont des contraintes de temps. Les données arrive à un débit de 1Mb/s, pour ne pas en perdre, la lecture sur le port d'entrée doit se faire toutes les 1µs.

6.2. Solution proposée

6.2.1. Hardware

Pour respecter les contraintes, l'implantation se fera sur une carte DSP comportant deux processeurs. Un des processeurs s'occupe exclusivement de lire les trames reçues sur le port série et les places dans une partie de la mémoire partagée avec le second processeur. L'autre processeur traitera les trames reçues.

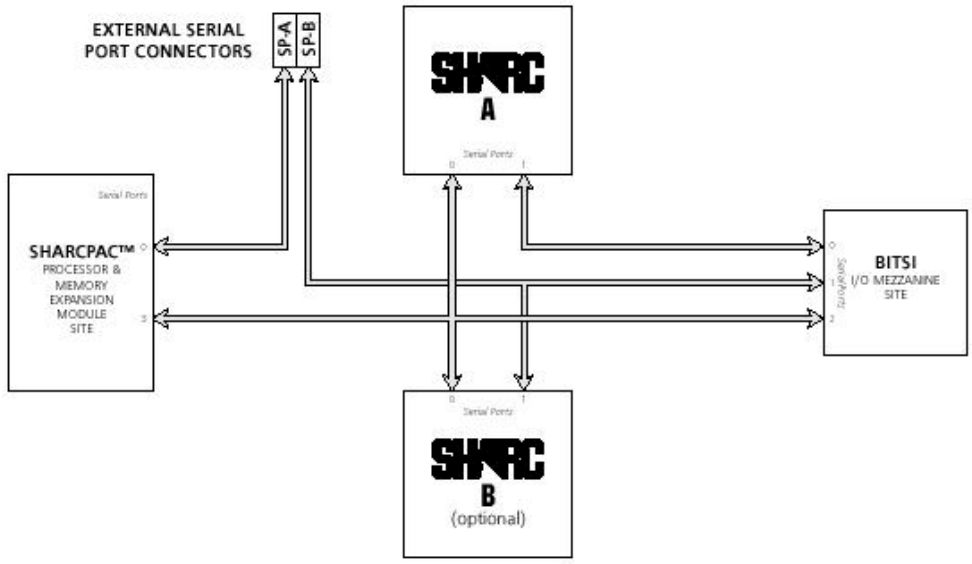


Figure 6-1 Connexions des ports séries

Carte utilisée

La carte utilisée est une carte DSP Snaggletooth-ISA de Bittware.

Cette carte a les caractéristiques principales suivantes :

- Deux processeurs SHARC ADSP-21062, 40 MHz, 32 bits.

- Les deux processeurs partagent le même bus processeur.
- Port JTAG pour le debuggage.
- 2 ports séries externes.

Pour compiler et debugger le code, on utilise le ToolKit ver. 5.1. de Bittware.

Les données seront introduites dans le DSP par un des ports séries disponibles sur la carte.

Cette carte comporte 4 ports séries (SPORT), dont deux comportent des connecteurs externes. Il s'agit des ports SP-A(0)et SP-B(1). La lecture des bits est effectuée sur le port SP-B(1). Ce port est directement relié au microprocesseur SHARC B, comme le montre la Figure 6-1.

Le processeur SHARC B mémorise les bits reçus par paquet de 32 bits dans un tampon circulaire.

La taille de 32 bits à été choisie pour deux raisons :

- Il n'est pas possible de lire des mots de 1 bit sur le port série. La taille minimum d'un mot est de 3 bits. Le traitement des bits se fera donc obligatoirement par paquet de plusieurs bits
- L'accès à la mémoire externe, où se trouve le tampon circulaire, est lent. Il faut 6 périodes d'horloge pour une lecture ou une écriture. Le processeur travaille sur 32 bits, la mémorisation des données se fera donc par paquet de 32 bits.

Port série

Les entrées du port série utilisé sont :

- Pin 1 : GND.
- Pin 7 : RCLK (horloge du signal, clkRec).
- Pin 15 : DR (signal de donnée).

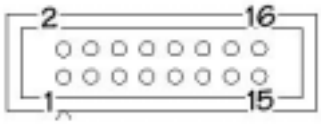


Figure 6-1 Vue de face du connecteur du port série

Initialisation du port série

Le port série est programmé pour lire des mots de 32 bits avec un signal d'horloge externe.

L'initialisation du port série se fait en écrivant dans le registre SRCTL1 (*Serial Control Register*).

6.2.2. Software

Les deux processus implantés sont la lecture des trames et le *parsing* des trames. Un tampon circulaire a aussi été implanté pour réguler le flux de données entre ces deux processus.

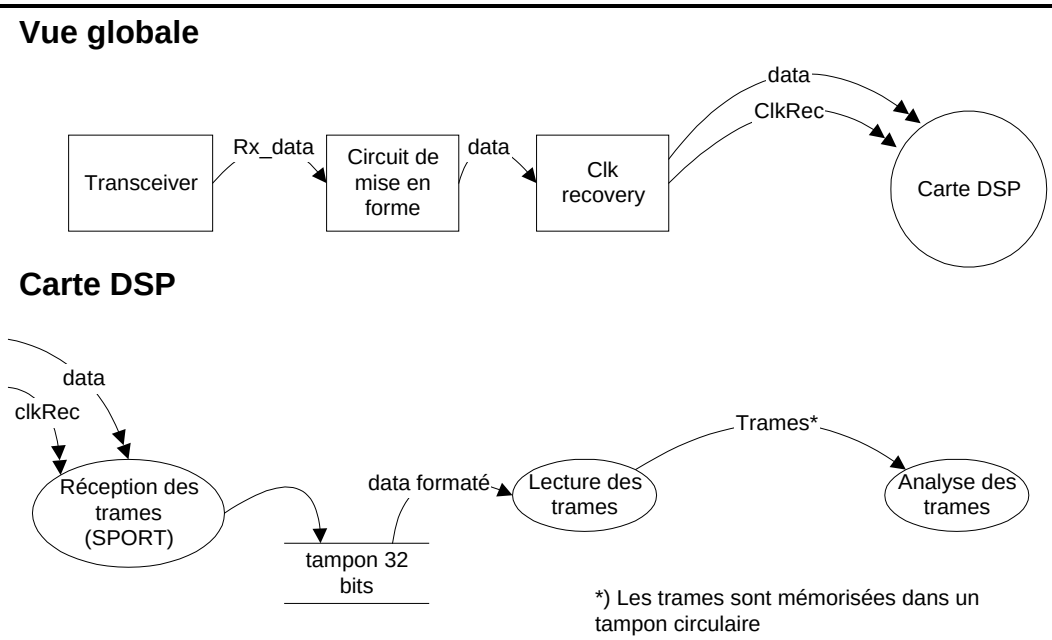


Figure 6-1Diagramme de flux

Tampon circulaire

La transmission entre les deux processeurs est faite à l'aide d'un tampon circulaire selon le paradigme "producteur-consommateur". Le tampon circulaire est implanté dans le fichier `ringbuf.c`.

Lecture des trames

Le code est implanté dans le fichier `lecteurSH_B.c`, il est exécuté par le processeur SHARC_B.

Le processus initialise le port série en lecture et les pointeurs du tampon circulaire. Puis il entre dans une boucle qui mémorise les mots de 32 bits reçus sur le port série dans le tampon circulaire.

Le champ `RXS` du registre de contrôle du port série (SRCTL1) permet de vérifier que le port série à reçu des données.

Analyse des trames

Ce processus recherche le début d'un paquet dans le flot de trames reçues. Il est exécuté sur le processeur SHARC_A.

Un début de paquet est reconnaissable aux 5 bits de *Preamble* appelé *beginning pattern* et aux 11 bits du *Trailer* et du Barker code réuni qui forment le *end pattern*.

Un début de paquet est trouvé lorsque :

- Le *beginning pattern* vaut 10101_b ou 01010_b.
- Le *end pattern* vaut 00011010101_b ou 11100101010_b.
- Les deux *patterns* (*end* et *beginning)* sont trouvés ;

- Le *beginning pattern* a été reçu avant le *end pattern ;*
- Il y a exactement 56 bits entre ces deux *patterns*.

La façon la plus concise d'écrire un algorithme pour la recherche de cette suite de bits, consiste à implanter un registre à décalage de 72 bits.

Pour chaque bit reçu, le processus vérifie si le registre contient la suite de bits recherchée. S'il ne la contient pas, il décale le registre vers la droite et concatène le bit suivant.

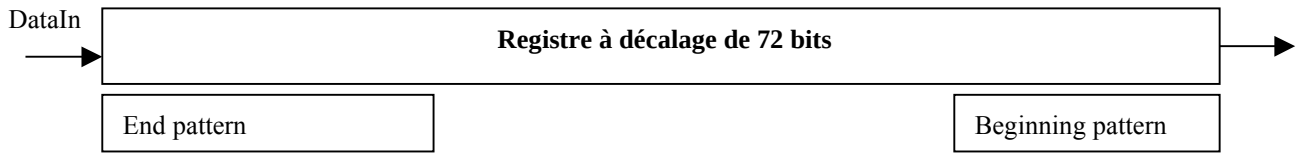


Figure 6-1 Recherche du début de trame à l'aide d'un registre à décalage

Cet algorithme doit aussi satisfaire une contrainte de temps. La recherche des *patterns*, le décalage du registre et la lecture du prochain bit doit se faire en moins d'1µs puisque l'on reçoit les données à un débit de 1Mb/s.

Le tampon circulaire étant en mémoire externe, le DSP à besoins de 6 cycles d'horloge pour y accéder, contre 1 cycle pour la mémoire interne. Il est donc plus intéressant de traiter les bits par paquets de 32 bits et d'accéder au tampon circulaire le moins souvent possible.

De plus pour éviter de devoir décaler le registre à chaque bit testé, on peut calculer à l'avance les 32 possibilités de début de trames. Il suffit ensuite de masquer les bits inintéressants.

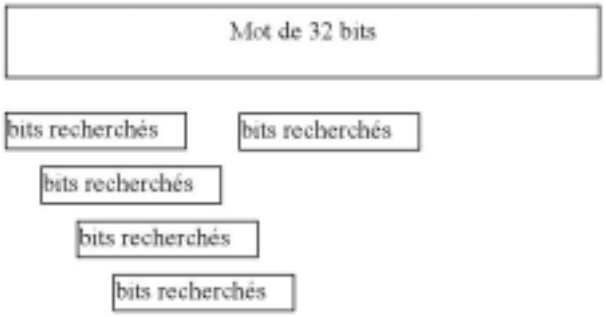


Figure 6-2 Recherche d'un pattern à l'aide de 32 masques différents

C'est cette solution qui a été implanté car elle est plus rapide.

Au début du fichier sont définis tous les masques et les *patterns* recherchés pour chacune des 32 positions possibles de début de trame.

Au démarrage de la recherche, un buffeur interne est chargé avec les 4 premiers mots du tampon circulaire.

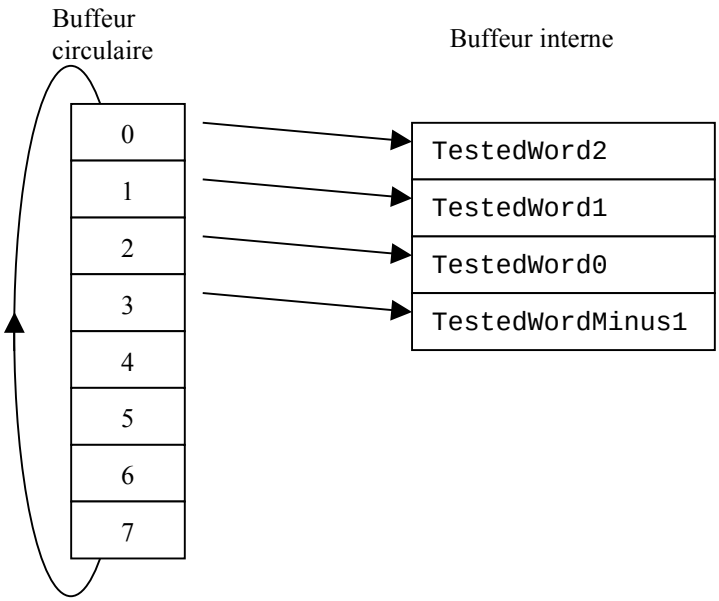


Figure 6-3 Chargement du buffeur interne

La recherche du début de la trame se fait à l'intérieur d'une boucle infinie qui recherche d'abord les bits du *end pattern*, puis s'il les détecte, vérifie si les bits du *beginning pattern* sont présents 71 (56+5) bits plus tôt.

Commencer la recherche par le *end pattern* est plus efficace car ce *pattern* a une plus faible probabilité d'être présent que le *beginning pattern*. Premièrement, il y a plus de bits dans le *end pattern* (11 bits) que dans le *beginning pattern* (5 bits) et deuxièmement, le *beginning pattern* qui est 10101_b ou 01010_b est facilement confondu avec le signal créé par la porteuse reçu avant le début de chaque paquet et qui est une suite de 101010_b.

La variable `trouve` indique que le *end pattern* et le *beginning pattern* ont été trouvés et qu'ils se trouvent à une distance de 57 bits l'un de l'autre.

La variable `position` indique lorsque `trouve = 1` l'emplacement du premier bit du *end pattern*. Ce bit est le 61ème bit de l'*Access Code.*

Dans l'exemple de la Figure 6-4 le *end pattern* se trouve entre les bits –10 à 0 et le *beginning pattern* se trouve entre les bits 54 à 61. Dans ce cas la détection du paquet doit se faire sur les deux lignes *testedWord0* et *testedWordMinus1.*

Lorsque le *end pattern* a été détecté, l'analyseur recule directement de 57 bits pour voir s'il trouve le *beginning pattern*. S'il le détecte, `trouve=1` et la position actuelle est mémorisée dans la variable `position`.

S'il ne détecte pas le *beginning pattern,* il recommence la recherche du *end pattern* à la prochaine position.

N° ligne	Numérotation des bits																															
TestedWord2	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79	80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95
TestedWord1	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
TestedWord0	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
TestedWord Minus1	-32	-31	-30	-29	-28	-27	-26	-25	-24	-23	-22	-21	-20	-19	-18	-17	-16	-15	-14	-13	-12	-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1

Figure 6-4 Représentation du buffeur interne avec un début de paquet à la position 0.

Lorsque les 32 possibilités ont été vérifiées, le buffeur interne est décalé.

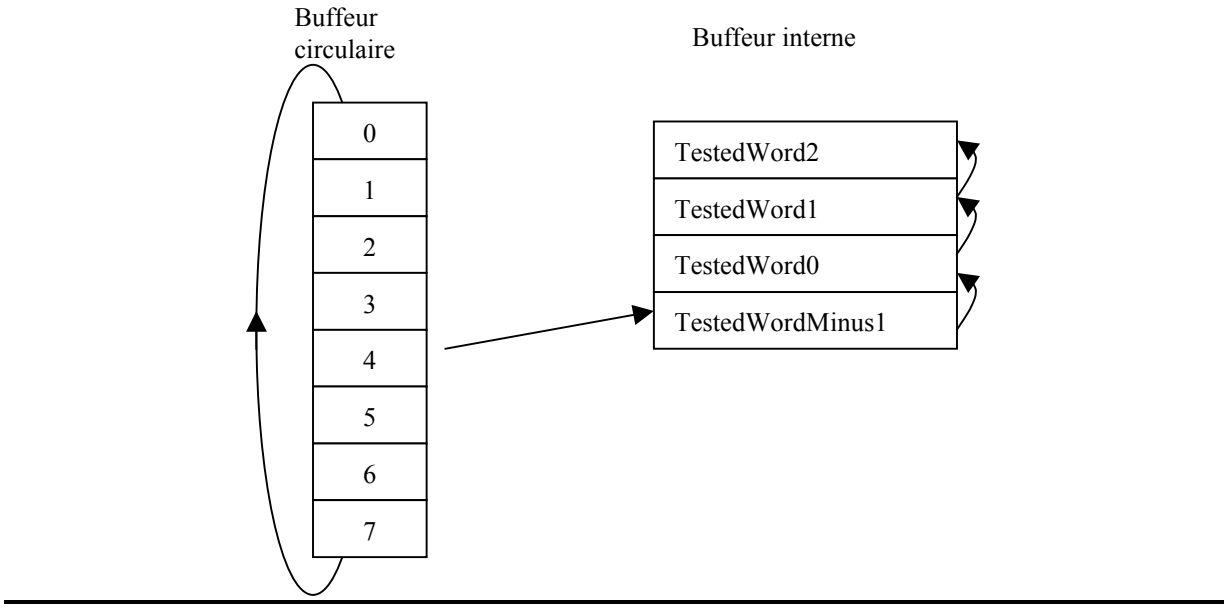


Figure 6-5 Décalage du buffeur interne

Et la vérification recommence à la prochaine ligne.

La liste de tous les masques et des *patterns* utilisés se trouve en annexe 7.18

Amélioration possible

Le programme mainSHARC_A.c n'est pas terminé. Les premiers tests effectués montrent qu'il trouve des trames. Mais la vérification du *Sync Word* n'existe pas encore. Lorsque l'analyseur trouve une trame, il fait clignoter une led du DSP, affiche la position du paquet trouvé et s'arrête. Ceci est voulu pour pouvoir tester le programme plus facilement. La suite de l'implantation devrait vérifier le *Sync Word* puis continuer la détection des paquets.

Conclusion

Mon travail de diplôme a porté sur l'analyse de la Baseband de Bluetooth pour réaliser un analyseur de trafic Bluetooth. Après trois mois de travail, une importante partie de l'analyse a été achevée, ce qui permettra la poursuite de ce projet dans des bonnes conditions. La réalisation de l'analyseur a aussi bien avancé, malgré les nombreuses difficultés identifiées initialement. Le signal reçu par le transceiver est mis en forme de façon à être compris par un microprocesseur et l'horloge du signal est exploitable. Les circuits de mise en forme et de recouvrement de l'horloge ont été testés et fonctionnent correctement. Un début de Baseband a aussi été implanté dans une carte DSP. Sur cette plate-forme, la réception des trames fonctionne correctement et la recherche de début de paquet est implantée mais doit encore être testé.

Ce travail de diplôme a été pour moi une expérience très intéressante, j'ai beaucoup appris sur le fonctionnement de la Baseband de Bluetooth ainsi que sur les transceiver radio. Pouvoir faire ce travail au sein d'une entreprise qui aborde de multiples domaine de haute technologie a facilité ce travail en mettant à ma disposition des équipements, des compétences et une infrastructure qu'il est rare de rencontrer dans les HES.

Sources

[1] http://www.bluetooth.com

[2] J.Stettler, Travail de Semestre Bluetooth – 802.11, 2000.

[3] Bluetooth specification, Version 1.0 B, 29.11.1999

[4] Bluetooth Demystified, Nathan J. Muller, MCGraw-Hill Telecom, 2000.

[5] Jim Geier , Wirless LANs, Macmillan Technical Publishing, 1998.

[6] TEMIC, T2901 Bluetooth Single-Chip Transceiver IC, Rev. A2, 22-feb-00.

[7] Brian W. Kernighan, The C Programming Language, 2nd edition, Prentice Hall, 1988.

[8] Shu Lin, Error Control Coding : Fundamentals and Applications, Prentice Hall, 1983.

Liste des abréviations

ACL	Asynchronous connection-less link
ALU	Arithmetic logic unit
AM_ADDR	Active membre address
ARQ	Automatic repeat request
BD_ADDR	Bluetooth device adresse
CAC	Channel access code
CAG	Control automatique du gain
CLK	Master clock
CLKN	Native clock
CPCS	Charge pump current settings
CRC	Cyclic redundancy code
DAC	Device access code
DAC	Digital-to-analog converter
DCI	Default check initalization
DEMODDAC	Demod DAC voltage
DH	Data high rate packet
DIAC	Device inquiry access code
DLL	Dynamic link library
DM	Data – medium rate packet
DSP	Digital signal processor
DS-SS	Direct sequence spread spectrum
FEC	Forward error correction
FFT	Fast fourier transform
FH-SS	Frequency hopping spread spectrum
GF	Gaussian filter
GFCS	Gaussian filtter current settings
GIAC	General inquiry access code
HCI	Host controller interface
HEC	Header error correction
HV	High quality voice packet
IAC	Inquiry access code
JTAG	Joint test action group
L_CH	Numéro du canal logique

L2CAP	Link control & adaptation protocol
LAP	Lower adress part
LSB	Least significant bit
MC	Main counter
MCC	Modulation compensation circuit
MCCS	Modulation compensation current settings
MIPS	Millions of instructions per second
MSB	Most significant bit
NAP	Non-significant adress part
NRZ	No return to zero
PA	Power amplifier
PCMCIA	Personal computer memory card International Association
PLL	Phase locked loop
PS	Phase select modulation-compensation
RSSI	Received signal strength indicator
Rx	Reception
SC	Swallow counter
SCO	Synchronous connection-oriented link
SHARC	Super Harward architecture computer
SNR	Signal to noise ratio
SPORT	Serial port
Tx	Transmission
UAP	Upper adresse part
VCO	Voltage controlled oscillator
VCO-DAC	Pretune DAC voltage

7. Annexe

7.1. Exemple d'utilisation des commandes HCI

7.1.1. Inquiry

Maître – 0xBB:BB:00:00:BB:BB			**Esclave - 0xAA:AA:00:00:AA:AA**		
Commandes ou Paramètres	**Valeurs**	**Code hex**	**Commandes ou Paramètres**	**Valeurs**	**Code hex**
			HCI_Write_Inquiry_Scan_Activity	0x1E	**1E**
					0C
					04
			- Inquiry_Scan_Interval	*0x10 00*	**00**
					10
			- Inquiry_Scan_window	*0x10 00*	**00**
					10
			--> HCI_Command_Complete	*0x0E*	**0E**
					04
			- Num_HCI_Command_Packets		**04**
			- Command_Opcode		**1E**
					0C
			- Return_Parameters		**00**
			HCI_Write_Scan_Enable	*0x1A*	**1A**
					0C
					01
			- Scan_Enable	*0x01*	**01**
HCI_Inquiry	0x01	**01**	--> HCI_Command_Complete	*0x0E*	**0E**
		04			**04**
		05	*- Num_HCI_Command_Packets*		**04**
- LAP	*0x9E 8B 33*	**33**	*- Command_Opcode*		**1A**
		8B			**0C**
		9E	*- Return_Parameters*		**00**
- Inquiry_Length	*0x09*	**09**			
- Num_Response	*0x00*	**00**			
--> HCI_Inquiry_Result		**02**			
		0F			
- Num_Responses		**01**			
- BD_ADDR		**AA**			
		AA			
		00			
		00			
		AA			
		AA			
- Page_Scan_Repetition_Mode		**01**			
- Page_Scan_Periode_Mode		**00**			
- Page_ScanMode		**00**			
- Class_Of_Device		**00**			
		00			
		00			
- Clock_Offset		**F6**			
		07			
--> HCI_Inquiry_Complete		**01**			

		02			
- Status		**00**			
- Num_Responses		**0B**			

F6	Code à envoyer au controleur HCI
02	Code retourné par le contrôleur HCI

7.1.2. Connect

Maître – 0xBB:BB:00:00:BB:BB			Esclave - 0xAA:AA:00:00:AA:AA		
Commandes ou Paramètres	**Valeurs**	**Code hex**	**Commandes ou Paramètres**	**Valeurs**	**Code hex**
HCI_Write_Page_Timeout	0x18	**18**	**HCI_Write_Page_Scan_Activity**	0x1C	**1C**
		0C	- Page_Scan_Interval	0x1000	**0C**
		02	- Page_Scan_Window	0x1000	**04**
- Page_Timeout	0xFFFF	**FF**			**00**
		FF			**10**
--> HCI_Command_Complete	0x0E	**0E**			**00**
		04			**10**
- *Num_HCI_Command_Packets*		**04**	--> HCI_Command_Complete	0x0E	**0E**
- *Command_Opcode*		**18**			**04**
		0C	- *Num_HCI_Command_Packets*		**04**
- *Return_Parameters*		**00**	- *Command_Opcode*		**1C**
					0C
			- *Return_Parameters*		**00**
			HCI_Write_Scan_Enable	*0x1A*	**1A**
					0C
					01
			- Scan_Enable	0x02	**02**
HCI_Create_Connection	0x05	**05**	--> HCI_Command_Complete	0x0E	**0E**
		04			**04**
		0D	- *Num_HCI_Command_Packets*		**04**
- BD_ADDR	0xAAAA0000 AAAA	**AA**	- *Command_Opcode*		**1A**
		AA			**0C**
		00	- *Return_Parameters*		**00**
		00			
		AA			
		AA			
- Packet_Type	0x0010	**10**			
		00			
- Page_Scan_Repetition_Mode	0x01	**01**			
- Page_Scan_Mode	0x00	**00**			
- Clock_Offset	0x0000	**00**			
		00			
- Allow_Role_Switch	0x00	**00**			
--> Commande_Status_Event		**0F**	--> HCI_Connection_Request		**04**
		04			**0A**
- Status		**00**			**BB**
- *Num_HCI_Command_Packets*		**04**			**BB**
- *Command_Opcode*		**05**			**00**
		04			**00**
					BB
					BB
					00
					EF
					CD
					01

			HCI_Accept_Connection_Request	0x09	**09**
					04
					07
			- BD_ADDR	0xBBBB0000BBBB	**BB**
					BB
					00
					00
					BB
					BB
			- Role	0x01	**01**
--> HCI_Connection_Complete		**03**	-- >HCI_Commande_Status_Event	0x0F	**0F**
		0B			**04**
- Status		**00**	- Status		**00**
- Connection_Handle		**07**	- *Num_HCI_Command_Packets*		**04**
		00	- *Command_Opcode*		**09**
- BD_ADDR		**AA**			**04**
		AA	--> HCI_Connection_Complete		**03**
		00			**0B**
		00	- Status		**00**
		AA	- Connection_Handle		**07**
		AA			**00**
- Link_Type		**01**	- BD_ADDR		**BB**
- Encryption_Mode		**00**			**BB**
					00
					00
					BB
			- Link_Type		**BB**
			- Encryption_Mode		**01**
					00

7.2. Organigramme de l'analyse du trafic

7.3. Mesure du signal de données en réception

7.4. Mesure de l'activité d'émission et de réception de l'esclave recevant un fichier

7.5. Mesure de la structure d'un paquet POLL

7.6. Mesures des valeurs programmées sur le kit INDIEN en réception

7.7. Mesures des valeurs programmées sur le kit INDIEN en émission

7.8. Calcul du spectre de fréquence d'un signal

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%
% plot_rx_Indien2.m
% Script used by Matlab
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%
% J.Stettler – eivd
% 5.12.00
%
% This Script calculates the frequency spectrum on a signal
%
% In this exemple, the signal is rx_data. Signal received by the the TEMIC
% radio frontend of the INDIEN kit.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%
%refsignal_INDIEN
close all;
load rx_Indien;
figure;
% Plot's the entire signal received
plot(rx_Indien(:,1),rx_Indien(:,2),'.-');
grid on;
xlabel('time [s]'); ylabel('rx_Indien');

% Extracts the portion to analyse
figure;
plot(rx_Indien(:,2),'.-');
axis([140 1200 0 5]);
x=rx_Indien([6528:7386],2);   %Here we can chose the portion to be analysed
dt=2*10^-7;
N=size(x,1);

figure;
subplot(2,1,1), plot([0:dt:(N-1)*dt]*10^6,x,'.-');
grid on;
title('Signal received by Temic on INDIEN kit');
xlabel('time [us]'); ylabel('Signal [V]');
freq_axis=[0:1/(N*dt):(N-1)/(N*dt)];


subplot(2,1,2), plot(freq_axis/10^6,10*log(F));
grid on;
axis([0 max(freq_axis)/10^6/2 -20 max(10*log(F))]);
xlabel('Frequency [MHz]');
ylabel('Spectrum [dB]');

figure; plot([0:dt:(N-1)*dt]*10^6,x,'.-');
title('Signal received by TEMIC on INDIEN kit');
grid on;
xlabel('time [us]'); ylabel('Signal [V]');
set(gca,'XTick',[0:180]);
axis([45 45+72 0.9 1.4]);

%Calculates the FFT
Fc=fft(x);
Fc2=Fc;
```

```
for i=1:25,
Fc2(i)=0; Fc2(859+1-i)=0;
end
x2=ifft(Fc2);

F=abs(fft(x));

figure;
% Plot's the portion choosen of the signal
subplot(2,1,1), plot([0:dt:(N-1)*dt]*10^6,x2,'.-');
grid on;
title('Signal received by TEMIC on INDIEN kit');
xlabel('time [us]'); ylabel('Signal [V]');
freq_axis=[0:1/(N*dt):(N-1)/(N*dt)];

% Plot's the frequency spectrum of the choosen portion
subplot(2,1,2), plot(freq_axis/10^6,10*log(abs(Fc2)));
grid on;
axis([0 max(freq_axis)/10^6/2 -20 max(10*log(abs(Fc2)))]);
xlabel('Frequency [MHz]');
ylabel('Spectrum [dB]');


%test
figure
plot([0:dt:(N-1)*dt]*10^6,x,'g.-');hold on;
plot([0:dt:(N-1)*dt]*10^6,x2,'.-');
set(gca,'XTick',[0:180]);
grid on;
bits=x2>0;
plot([0:dt:(N-1)*dt]*10^6,bits*0.2,'r');
axis([45 45+72 -0.2 0.25]);
```

7.9. Calculs de la fonction de transfert du filtre passe-bande

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% reponse_filtre.m
% Script used by Matlab
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%plot la courbe de reponse d'un filtre passe bande
%J.Stettler – eivd
%1.11.00

close all;
clear all;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Valeurs des éléments
R1=2000;  %R1<<R2
R2=8200;

C1=4.7*10^-9;  %C2<<C1
C2=2.2*10^-12;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Fréquence centrale
w0=1/sqrt(R1*R2*C1*C2);
f0=w0./(2*pi)

% calcul de l'amplification à f0
Amax=-1./((R1./R2)+(C2./C1))

%Calcul des frequences de coupures supérieurs et inférieurs
fc1=1/R1/C1/2/pi
fc2=1/R2/C2/2/pi

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Affichage de la fonction de transfert du filtre
freq_axis_exp =[2:0.1:9];
freq_axis = 10.^freq_axis_exp;
w=freq_axis*2*pi;
A=-(R2/R1)*(-i*w*C1*R1)./((1+i*w*R1*C1).*(1+i*w*R2*C2));

y0=20*log10(abs(Amax));
y1=y0-3;
figure
semilogx(freq_axis,20*log10(abs(A)),f0,y0,'o',fc1,y1,'*',fc2,y1,'*')
grid on; %Affiche la grille sur le graph
%axis([10^2 10^7 0 30]);
xlabel('Fréquence [Hz]')
ylabel('Amplification [dB]')
title('Courbe de réponse du filtre passe-bande')
```

7.10. Schéma électrique du filtre de mise en forme

7.11. Format des paquets

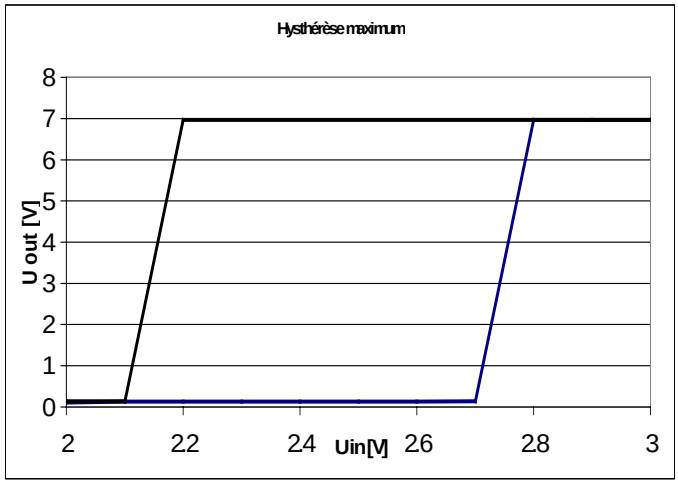
7.12. Tableau de mesure de la bascule de Schmitt

7.12.1. Mesure de l'hystérèse minimum

Le pot P2 est vissé au maximum dans le sens horaire.

$U_{réf}$=2.549V

IN, pt A [V]	OUT, pt B [V]	OUT, pt B [V]
0	0	0
1	0.07	0.14
2	0.11	0.14
2.1	0.13	0.14
2.2	0.13	6.97
2.3	0.13	6.97
2.4	0.13	6.97
2.5	0.13	6.97
2.6	0.13	6.97
2.7	0.14	6.97
2.8	6.97	6.97
2.9	6.97	6.97
3	6.97	6.97
5	6.97	6.97
7	6.97	6.97
9	6.97	6.97

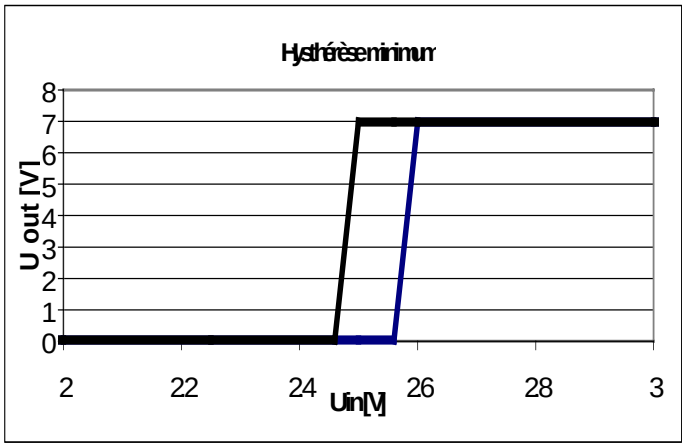


7.12.2. Mesure de l'hystérèse maximum

Le pot P2 est vissé au maximum dans le sens anti-horaire.

$U_{réf}$=2.549V

IN, pt A [V]	OUT, pt B [V]	OUT, pt B [V]
0	0.04	0.04
1	0.04	0.04
2	0.04	0.04
2.25	0.04	0.04
2.46	0.04	0.04
2.5	0.04	6.98
2.56	0.04	6.98
2.6	6.98	6.98
3	6.98	6.98
5	6.98	6.98
7	6.98	6.98
9	6.98	6.98



7.13. PLL numérique

```
"************************************************************************"
" Nom du fichier/circuit : clkrec00.abl                                   "
" Cree le                : 14.11.2000          par : J. Stettler          "
"                                                                         "
" Utilise dans           :Recouvrement d'un signal d'horloge a partir du  "
"                         signal de donne a 1MHz.                         "
"-----------------------------------------------------------------------"
"                                                                         "
"                                                                         "
" Entrees : dataIn      : Signal de donnee qui commande clkRec             "
"          reset       : Reset du circuit, RAZ du compteur                "
"                                                                         "
"          clk         : Horloge du circuit  16 MHz                       "
"                                                                         "
" Sorties : clkrec      : Signal d'horloge retrouve par le circuit         "
"          data2       : Signal de donnee, synchro avec clkRec            "
"                                                                         "
"-----------------------------------------------------------------------"
" Types de circuit utilisables :  Xilinx XC954108PC84  ou 9536VQ44        "
"************************************************************************"

module clkrec00
title 'clkrec00'

"************************************************************************"
"                              Declarations                               "
"************************************************************************"

declarations

" Entrees "
            dataIn  pin 47;" 26;"
            reset   pin 25;" 25;"
            clk60   pin 10;" 10;"

" Sorties "

            Q2..Q0       pin  istype 'reg';  "4 bits Counter"
            clkRec       pin 53 istype 'reg';
            data1        pin  istype 'reg';
            data2        pin 55 istype 'reg';  "Sortie des donnees"
            transition   pin  istype 'com';

            div0         pin  istype 'reg';  "Diviseur de frequence par 4"
            clk          pin  istype 'reg';  "Horloge a 15MHz";

" Listes "
            counter = [clkRec,Q2,Q1,Q0];
            div4 =[clk,div0];

"************************************************************************"
"                           Description logique                          "
"************************************************************************"
equations

"***************************** Generation l'horloge 15MHz ***************"

div4.clk=clk60;
```

```
div4:=div4+1;        "clk est a 15MHz."

"***************************** Recouvrement de l'horloge ****************"

" Fonctions auxiliaires "
  "Detecte les transitions 1->0 ou 0->1
  transition = data1$data2;

  "Assigne l'horloge 15MHz sur le compteur, data1 et data2
counter.clk = clk;
data1.clk = clk;
data2.clk = clk;

  "Serialisation des deux bascules D"
data2:=data1;
data1:=dataIn;


when (reset) then
{
  counter:=0;
  data1:=0;
  data2:=0;

}
else
{
  when (transition)  then {
    when ((counter>=1) & (counter<=4)) then
    {
      counter := counter-1;
    }
    else
    {
      when ((counter>=5) & (counter<=6)) then
      {
        counter := counter-2;
      }
      else
      {
        when ((counter>=8) & (counter<=10)) then
        {
          counter := counter+3;
        }
        else
        {
          when ((counter>=11) & (counter<=12)) then
          {
            counter := counter+2;
          }
          else
          {
            when (((counter>=13) & (counter<=14))) then
            {
              counter :=0;
            }
            else         "Rajoute cette ligne, sinon, que ce passe t'il a 7
et a 0 lors d'une transition"
            {
              counter:= counter+1;
            }
```

```
          }
        }
      }
    }
  }
  else
  {
    when (counter ==14) then
    {
      counter :=0;     "Compteur de 0  14.
    }
    else
    {        "Fonctionnement normal, lorsqu'il n'y a pas de transition"
    counter:=counter+1;
    }
  }
}

end clkrec00;
```

7.14. Exemple complet de calcul du *Sync Word*

7.15. Calcul du syndrome du FEC 2/3

```
/*****************************************************************************
 * Fichier :   FEC2_3.cpp                                                    *
 * Auteur  :   J.Stettler                                                    *
 * Date    :   22 novembre 2000                                              *
 *                                                                           *
 * But     :   Ce programme calcule les syndrome selon les bits recu.        *
 *             Utilisé pour le FEC2/3 de Bluetooth                           *
 *                                                                           *
 * Version :   1.0                                                           *
 *             On utilise deux int non signé pour mémoriser les data         *
 *****************************************************************************
 */
#include <stdio.h>
#include <math.h>      //pow


void main()
{
  unsigned short int registre =0;    //registre à décalage.
  const int taille=15;               //Taille des donnees recus
  unsigned short int dataRecu=0;     //Registre contenant
  unsigned short int dataIn=0;

  printf("erreur recu | syndrome calcule D0 - D4\n");
  printf("-------------------------------------------------\n");

  for (int k=0;k<taille;k++)
  {
    //Initialisation du registre de donnee ;
    registre=0;

    //Initialisation du mot de data;
    dataRecu = pow(2,k);

    //Affichage des data Recu
    printf("%6x",dataRecu);

    printf("      | ");

    //Décalage.
    for (int i=0;i<taille;i++)
      {
        //lit le prochain bit de donner
        if (dataRecu & 0x01)
          dataIn=0x8000;
        else
          dataIn = 0x0000;
        dataRecu=dataRecu>>1;

        //Calcule les valeurs dans le registre
        registre=(dataIn | (registre>>1))^((registre&0x0800)?0xA800:0);
      }
      //Affiche le résultat
      printf("%4x\n", registre&0xF800);
      }
}/*Fin du code FEC2_3.cpp*/
```

7.16. Mise en route du kit INDIEN

7.17. Code source pour le DSP

7.18. Table des Masques et des Patterns

Start bit = bit n° 61 de l'Access Code
Toutes les valeurs sont données en hexadecimales

	beginning pattern							
startBit Nb	**trame n°**	**preambleMask1**	**preambleTestA1**	**preambleTestB1**	**trame n°**	**preambleMask2**	**preambleTestA2**	**preambleTestB2**
31	2	F8	A8	50	-	-	-	-
30	2	1F0	150	A0	-	-	-	-
29	2	3E0	2A0	140	-	-	-	-
28	2	7C0	540	280	-	-	-	-
27	2	F80	A80	500	-	-	-	-
26	2	1F00	1500	A00	-	-	-	-
25	2	3E00	2A00	1400	-	-	-	-
24	2	7C00	5400	2800	-	-	-	-
23	2	F800	A800	5000	-	-	-	-
22	2	1F000	15000	A000	-	-	-	-
21	2	3E000	2A000	14000	-	-	-	-
20	2	7C000	54000	28000	-	-	-	-
19	2	F8000	A8000	50000	-	-	-	-
18	2	1F0000	150000	A0000	-	-	-	-
17	2	3E0000	2A0000	140000	-	-	-	-
16	2	7C0000	540000	280000	-	-	-	-
15	2	F80000	A80000	500000	-	-	-	-
14	2	1F00000	1500000	A00000	-	-	-	-
13	2	3E00000	2A00000	1400000	-	-	-	-
12	2	7C00000	5400000	2800000	-	-	-	-
11	2	F800000	A800000	5000000	-	-	-	-
10	2	1F000000	15000000	A000000	-	-	-	-
9	2	3E000000	2A000000	14000000	-	-	-	-
8	2	7C000000	54000000	28000000	-	-	-	-
7	2	F8000000	A8000000	50000000	-	-	-	-
6	1	1	1	0	2	F0000000	50000000	A0000000
5	1	3	2	1	2	E0000000	A0000000	40000000
4	1	7	5	2	2	C0000000	40000000	80000000
3	1	F	A	5	2	80000000	80000000	0
2	1	1F	15	A	-	-	-	-
1	1	3E	2A	14	-	-	-	-
0	1	7C	54	28	-	-	-	-

	end pattern							
startBit Nb	**trame n°**	**endMask1**	**endTestA**	**endTestB**	**trame n°**	**endMask2**	**endTestA2**	**endTestB2**
31	0	7FF	2A7	558	-	-	-	-
30	0	FFE	54E	AB0	-	-	-	-
29	0	1FFC	A9C	1560	-	-	-	-
28	0	3FF8	1538	2AC0	-	-	-	-
27	0	7FF0	2A70	5580	-	-	-	-
26	0	FFE0	54E0	AB00	-	-	-	-
25	0	1FFC0	A9C0	1560	-	-	-	-
24	0	3FF80	1538	2AC00	-	-	-	-
23	0	7FF00	2A700	55800	-	-	-	-
22	0	FFE00	54E00	AB000	-	-	-	-
21	0	1FFC00	A9C00	156000	-	-	-	-
20	0	3FF800	153800	2AC000	-	-	-	-
19	0	7FF000	2A7000	558000	-	-	-	-
18	0	FFE000	54E000	AB0000	-	-	-	-
17	0	1FFC000	A9C000	1560000	-	-	-	-
16	0	3FF8000	1538000	2AC0000	-	-	-	-
15	0	7FF0000	2A70000	5580000	-	-	-	-
14	0	FFE0000	54E0000	AB00000	-	-	-	-
13	0	1FFC0000	A9C0000	15600000	-	-	-	-
12	0	3FF80000	15380000	2AC00000	-	-	-	-
11	0	7FF00000	2A700000	55800000	-	-	-	-
10	0	FFE00000	54E00000	AB000000	-	-	-	-
9	0	FFC00000	A9C00000	56000000	-1	1	0	1
8	0	FF800000	53800000	AC000000	-1	3	1	2
7	0	FF000000	A7000000	58000000	-1	7	2	5
6	0	FE000000	4E000000	B0000000	-1	F	5	A
5	0	FC000000	9C000000	60000000	-1	1F	A	15

7.19. Installation du DSP Snaggletooth ISA