

Projet de diplôme
-
AUTOMNE 2003

**Adaptation de la couche MAC du
standard IEEE 802.15.4 à une couche
physique Ultra Wide Band**

Auteur :

VERNEZ Jérôme

Superviseurs :

**POLLINI Alexandre
HUTTER Andreas
ROBERT Stephan**



Table des matières :

INTRODUCTION.....	4
1. DESCRIPTION	7
1.1 Projet européen URSAFE.....	7
1.1.1 Couche physique de URSAFE	8
1.2 Couche PHY IEEE 802.15.4	9
1.2.1 Rôles et services	9
1.2.2 Caractéristiques générales	9
1.2.3 Structure générale d'un paquet PHY	10
1.3 Couche MAC IEEE 802.15.4	11
1.3.1 Rôles et services	11
1.3.2 Structure générale des paquets	11
1.3.2.1 Structure d'une trame de données	12
1.3.2.2 Structure d'une trame «Beacon»	15
1.3.2.3 Structure d'une trame d'acquiescement (ACK).....	18
1.3.2.4 Structure d'une trame de commande.....	19
1.3.3 Méthodes d'accès au canal	23
1.3.3.1 CSMA-CA.....	23
1.3.3.2 Structure d'une Superframe	27
1.3.4 Modèles de transmissions de données	28
1.3.4.1 Topologies réseau.....	28
1.3.4.2 Topologie étoile	29
1.3.4.3 Topologie «Peer to Peer».....	30
1.3.4.4 Temps entre les trames.....	31
1.4 Description détaillée des primitives	31
1.4.1 MCPS-DATA (request, confirm, indication)	33
1.4.1.1 Fonctionnement et utilisation	33
1.4.1.2 Construction de la trame MAC.....	34
1.4.1.3 Segmentation.....	35
1.4.1.4 Description par les statuts	36
1.4.2 MCPS-PURGE (request, confirm).....	37
1.4.2.1 Fonctionnement.....	38
1.4.2.2 Description par les statuts	38
1.4.3 MLME-ASSOCIATE.(request, confirm, indication, response)	38
1.4.3.1 Fonctionnement et utilisation	39
1.4.3.2 Description par les statuts	40
1.4.4 MLME-DISASSOCIATE (request, confirm, indication)	41
1.4.4.1 Fonctionnement et utilisation	41
1.4.4.2 Description par les statuts	42
1.4.5 MLME-BEACON-NOTIFY.indication.....	43
1.4.5.1 Fonctionnement.....	43
1.4.6 MLME-GET (request, confirm).....	44
1.4.6.1 Fonctionnement.....	44
1.4.6.2 Description par les statuts	44
1.4.7 MLME-GTS (request, confirm, indication).....	45
1.4.7.1 Fonctionnement et utilisation	45
1.4.7.2 Description par les statuts	46
1.4.8 MLME-ORPHAN (indication, response).....	47
1.4.8.1 Fonctionnement et utilisation	47
1.4.9 MLME-RESET (request, confirm)	48
1.4.9.1 Fonctionnement.....	49
1.4.9.2 Description par les statuts	49
1.4.10 MLME-RX-ENABLE (request, confirm).....	49
1.4.10.1 Fonctionnement	49

1.4.10.2	Description par les statuts	50
1.4.11	MLME-SCAN (request, confirm).....	50
1.4.11.1	Fonctionnement	50
1.4.11.2	Description par les statuts	51
1.4.12	MLME-COMM-STATUS.indication	52
1.4.12.1	Fonctionnement	52
1.4.13	MLME-SET (request, confirm)	52
1.4.13.1	Fonctionnement	52
1.4.13.2	Description par les statuts	53
1.4.14	MLME-START (request, confirm).....	53
1.4.14.1	Fonctionnement et utilisation.....	53
1.4.14.2	Description par les statuts	54
1.4.15	MLME-SYNC.request.....	55
1.4.15.1	Fonctionnement	55
1.4.16	MLME-SYNC-LOSS.indication.....	55
1.4.16.1	Fonctionnement	55
1.4.16.2	Description par les statuts	56
1.4.17	MLME-POLL (request, confirm).....	56
1.4.17.1	Fonctionnement et utilisation.....	56
1.4.17.2	Description par les statuts	57
2.	FONCTIONNEMENT ET SYNTHÈSE	58
2.1	Base de temps.....	58
2.2	MAC PAN Information Base (PIB).....	59
2.3	CSMA-CA.....	60
2.4	Communications	60
2.4.1	Transmission indirecte.....	60
2.4.2	Retransmission.....	61
2.4.3	Réception	61
2.4.4	Rejet	61
2.4.5	Récupération de données en attente	62
2.4.5.1	Réseau «Beacon Enabled».....	62
2.4.5.2	Réseau «Non Beacon Enabled».....	63
2.5	Types de balayages	63
2.5.1	Détection d'énergie.....	63
2.5.2	Balayage actif	64
2.5.3	Balayage passif	64
2.5.4	Balayage d'orphelin.....	65
2.6	Création du PAN.....	65
2.7	Association & Dissociation	66
2.7.1	Procédure d'association.....	66
2.7.2	Procédure de dissociation	67
2.7.3	Orphelin	67
2.8	Synchronisation.....	67
2.8.1	Réseau «Beacon Enabled»	67
2.8.2	Réseau «Non Beacon Enabled»	68
2.9	Réservation de temps (GTS)	68
3.	RÉALISATION	69
3.1	Logiciel IAR Embedded Workbench.....	69
3.1.1	L'éditeur et le compilateur	70
3.1.2	Le débogueur	72
3.2	Caractéristiques du MSP430F149.....	74
3.3	Présentation.....	74
3.3.1	Mise en route.....	75
3.3.2	Généralité de l'implémentation.....	75
3.3.3	Organisation mémoires.....	75
3.3.4	Réservation mémoires	76
3.4	Descriptions des fichiers du projet.....	78
3.4.1	Implémentation RFD et FFD.....	79

3.4.2	Variables et constantes globales.....	79
3.5	Conventions de l'écriture.....	82
3.6	Implémentation C des principales fonctionnalités.....	82
3.6.1	Type de données	82
3.6.2	CSMA-CA « <input type="checkbox"/> otted <input type="checkbox"/> et MLME-RX-ENABLE	83
3.6.2.1	CSMA-CA « <input type="checkbox"/> otted <input type="checkbox"/>	83
3.6.2.2	MLME-RX-ENABLE.....	83
3.6.3	CRC.....	84
3.6.4	Timers & Watchdog	84
3.6.4.1	Démarche vers la solution finale	84
3.6.4.2	Timer A	85
3.6.4.3	Timer B.....	86
3.6.4.4	Watchdog Timer.....	86
3.6.4.5	Priorité des interruptions.....	87
3.6.4.6	Optimisation des interruptions.....	87
3.6.5	Modifications de l'implémentation physique	88
3.6.6	Traitement sur les bits.....	89
3.7	Tests des primitives MAC de données.....	89
3.8	Applications	92
3.8.1	Cartes de test RXTX_BB V2	92
3.8.2	Procédure de programmation du MSP430.....	94
3.8.3	Application 1.....	95
3.8.3.1	Première partie <input type="checkbox"/> Envoi de données	95
3.8.3.2	2e partie <input type="checkbox"/> Envoi et réception de données.....	98
3.8.4	Application 2.....	99
3.8.5	Application 3.....	101
3.8.5.1	Première partie <input type="checkbox"/> Synchronisation	102
3.8.5.2	Deuxième partie <input type="checkbox"/> Association.....	104
3.8.5.3	Troisième partie <input type="checkbox"/> Association et réseau « <input type="checkbox"/> Beacon Enabled <input type="checkbox"/>	105
3.8.6	Application 4.....	105
4.	CONCLUSION.....	107
4.1	À propos du futur standard IEEE 802.15.4.....	107
4.2	À propos du logiciel IAR.....	107
4.3	Implémentations non réalisées	108
4.4	Conséquences de l'adaptation	109
4.5	Remarques personnelles	110
	LEXIQUE.....	112
	DOCUMENTS ET LIENS INTERNET.....	115
ANNEXE A<input type="checkbox"/>	Résumé de la norme	
ANNEXE B<input type="checkbox"/>	Code Applications	
ANNEXE C<input type="checkbox"/>	Code MAC	
ANNEXE D<input type="checkbox"/>	Code PHY	
ANNEXE E<input type="checkbox"/>	Code Autres	
ANNEXE F<input type="checkbox"/>	Code Applications Linux	
ANNEXE G<input type="checkbox"/>	Fichiers textes générés	

Introduction :

Durant ces dernières années, le monde des télécommunications sans fil a vécu une véritable révolution avec, entre autres, les succès du GSM, GPS, Wireless LAN (IEEE 802.11) et Bluetooth. Toutes ces technologies fonctionnent à des fréquences bien précises. Avec l'arrivée de nouvelles technologies, certaines bandes de fréquences deviennent surchargées, ce qui se traduit dans certains cas par des dysfonctionnements à cause des interférences mutuelles.

Le cas de IEEE 802.11b et Bluetooth est particulièrement probant. Ces technologies fonctionnent toutes deux sur la même bande ISM libre de droit (2.4 GHz). Les dernières versions de ces standards tentent d'améliorer une meilleure cohabitation, ce qui a pour effet de compliquer encore plus leurs fonctionnements et leurs implémentations.

La demande toujours grandissante de bande passante nécessite de trouver de nouvelles solutions techniques. À ce titre, la technologie **Ultra Wide Band (UWB)** apporte de nouvelles solutions par rapport à d'autres technologies. Elle a, en effet, un important potentiel de robustesse et de simplicité.

Un aspect de la technologie d'UWB est de fonctionner sur de très larges bandes de fréquences à des densités de puissance d'émission très faibles. Les technologies à bandes étroites percevront alors les transmissions UWB seulement comme un fiable bruit blanc additionnel au bruit ambiant. Un autre avantage de l'UWB est qu'il est nomade (utilisable partout dans le monde) au même titre des technologies à 27 MHz ou à 2.4 GHz.

C'est l'organisme de régulation américaine (FCC) qui a la première fois définit UWB. Ainsi, la dernière définition en date¹ spécifie qu'une diffusion UWB est reconnue comme étant un signal qui occupe une largeur de bande plus grande que 20% de sa fréquence centrale ou plus grande que 500 MHz si sa fréquence centrale est à plus de 2.5 GHz (Équ. 1). La largeur de bande est calculée entre la fréquence minimale et maximale qui ont un niveau de puissance inférieur de au moins 10 dB par rapport à celui de la fréquence centrale.

$$B \begin{cases} \geq 0.20 \cdot f_c & f_c \leq 2.5 \text{ GHz} \\ \geq 500 \text{ MHz} & f_c > 2.5 \text{ GHz} \end{cases}$$

Équ. 1 - Définition UWB

f_c : Fréquence centrale [Hz]

B : Largeur de bande [Hz]

C'est également la FCC qui a publié un masque de densité de puissance qui ne doit pas être dépassé par une émission UWB (Figure 1). La raison première de la forme de cette courbe a été de limiter au mieux les interférences avec le GPS (1.6 GHz) et les systèmes de la bande ISM 2.4 GHz. On note que la puissance de transmission autorisée est plus importante entre 3.1 GHz et 10.6 GHz, car la densité de puissance est maximale : -41.3 dBm/MHz . Cette valeur a été calculée sur la base des limites usuelles pour le champ électrique émis¹.

¹ Référence : Federal Communications Commission (FCC), "Revision of Part 15 of the Commission's Rules Regarding Ultra-Wideband Transmission Systems," First Report and Order, ET Docket 98-153, FCC 02-48; Adopted: February 14, 2002; Released: April 22, 2002.

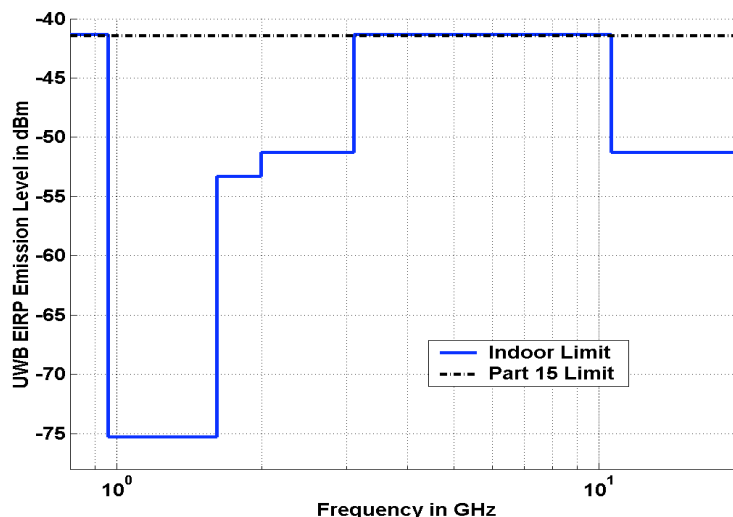


Figure 1 – Masque de densité de puissance pour UWB en intérieur (FCC)

Les systèmes utilisant des transmissions UWB ne sont qu'au début de leur développement, aucune intégration complète n'existant encore sur le marché à notre connaissance. La maturité commerciale de l'UWB n'est donc pas prévue avant plusieurs années. Les prédictions théoriques actuelles et les essais en laboratoire entendent notamment deux voies pour l'utilisation de l'UWB dans les réseaux WPAN (-10 m) ; une solution permettant de remplacer les câbles hauts débits actuel comme l'USB 2 (480 Mbit/s) ou les Firewire I et II (400 Mbit/s et 800 Mbit/s) et une solution plutôt pour les basses consommations et les bas débits venant directement concurrencer Bluetooth.

Dans le cadre de ce projet nous nous sommes concentré sur la solution du bas débit.

Dans les normes en développement qui concerne les applications à bas débit et à basse consommation, on trouve le standard **IEEE 802.15.4**. Ce standard favorise une très faible consommation de l'interface radio et offre des débits relativement faibles (<250kbps). Il offre également une bonne sécurité au niveau de la protection des données (cryptage des données au niveau MAC). Ce futur standard ouvert est d'ailleurs en train d'être complété par une association de plusieurs compagnies sous le nom de **Zigbee™**. Leur but étant de proposer un standard pour réseau sans fil à basse consommation et à bas coûts en normalisant les couches supérieures aux fonctionnalités MAC et Physique déjà décrites par le standard IEEE 802.15.4.

Les applications susceptibles d'utiliser Zigbee sont nombreuses :

- Réseau PAN sécurisé
- Commande à distance (lumière, appareil,...)
- Contrôle à distance (mesure, capteur)
- Clavier, souris sans fil
- Détecteur
- Domotique (building automation)
- Jeux

L'idée de base de ce travail de diplôme est de marier le standard IEEE 802.15.4 et la technologie UWB afin d'obtenir un ensemble innovant. Cet ensemble constitue un système de transmission avec un potentiel de grande autonomie, peu onéreux, facile à intégrer et fonctionnant dans une bande de fréquence avec une forte perspective de développement.

1. Description

L'**objectif général** de ce projet de diplôme est d'adapter et d'implémenter la **couche MAC** du standard **IEEE 802.15.4** à une couche physique **Ultra Wide Band (UWB)** développée au **CSEM** pour le projet européen **URSAFE** (Universal Remote Signal Acquisition for health). L'implémentation est réalisée avec un microprocesseur **MSP430**. Le fait de greffer une couche MAC IEEE 802.15.4 au système actuel de URSAFE va permettre d'étendre les possibilités (ajouts de nouveaux capteurs, transferts de données synchrones,...).

1.1 Projet européen URSAFE

L'idée générale du projet URSAFE est de pouvoir acquérir certaines caractéristiques physiologiques d'une personne à distance. Cela permet pour un médecin de réaliser de la surveillance à distance et d'intervenir rapidement s'il y a lieu.

Un premier réseau sans fil à faible portée est dédié à la zone environnante de la personne (**WPAN**). Il est basé sur une technologie UWB. Ce réseau regroupe 2 modules auxiliaires (**MU et WU**) qui contiennent des capteurs et une station de base portable (**PBS**) qui sert à récupérer les informations des capteurs et à les contrôler. Ce réseau WPAN est schématisé à la Figure 2. C'est également la PBS qui sert de pont vers l'extérieur, soit par le réseau GPRS, ou soit par un accès à un réseau local IEEE 802.11b. La PBS contient un système Linux embarqué.

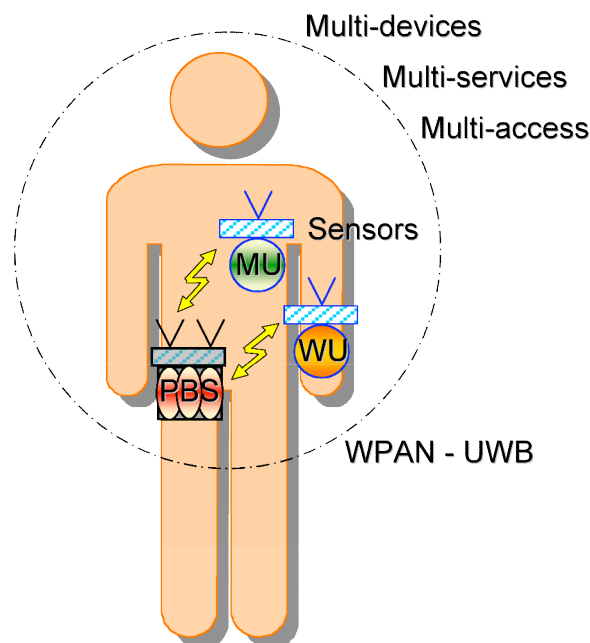


Figure 2 - Projet URSAFE (WPAN)

Le module MU (Monitoring Unit) qui est placé sur la poitrine contient un capteur de chute et un capteur ECG. Le module WU (Wrist Unit) qui est placé au poignet contient un capteur SpO2 servant à mesurer le taux d'oxygène dans le sang.

Pour ce projet, une couche physique, une couche MAC et un protocole de communications ont été spécifiquement développés. Mais, la couche MAC est réduite à sa plus simple expression et ce système est limité à l'utilisation des 3 capteurs prévus (pas d'extensibilité).

1.1.1 Couche physique de URSAFE

La couche physique de URSAFE a donc été spécialement conçue pour ce projet européen. Elle est propriétaire du CSEM. La technologie UWB est basée dans ce cas sur une double modulation FM à très large bande. Le système possède 3 canaux obtenus à l'aide de 3 sous-porteuses, chaque canal étant prévu pour accueillir un utilisateur. Le module physique **UWB-FM**® est composé d'une partie analogique pour la radio (hardware), d'une partie logique (« glue logic » avec une CPLD) et d'un microprocesseur MSP430 (firmware).

Le microprocesseur a deux tâches principales. Premièrement, il se charge de contrôler les communications radio et deuxièmement, il communique avec les capteurs des modules ou avec le contrôleur Linux dans la PBS. Le firmware est différent pour les deux cas.

La CPLD qui contient des portes logiques programmables est utilisée pour lier la partie radio analogique et le microprocesseur (glue logic). Elle va notamment remplir les fonctions suivantes :

- Découpage des données reçues du microprocesseur en trames pour la transmission
- Synchronisation des bits lors de la réception
- Codage en bande de base Manchester
- « Bit stuffing »
- Génération et contrôle de CRC
- Contrôle de la radio (transmission et réception ON/OFF)

Le « bit stuffing » correspond à l'insertion de certains bits supplémentaires. Les séquences de bits d'en-tête et de fin de trame qui sont présents dans les trames HDLC doivent être uniques. Pour pas que ces séquences de bits n'apparaissent dans les bits d'informations, il est indispensable de réaliser un « bit stuffing ».

Cette couche physique n'a pas toutes les fonctionnalités associées à une vraie couche physique IEEE 802.15.4. Elle a donc dû être adaptée pour que la compatibilité avec la couche MAC 802.15.4 puisse être au maximum satisfaite. L'adaptation est réalisée en encapsulant le firmware UWB dans un bloc aussi contenu dans le MSP 430. La Figure 3 montre cette adaptation.

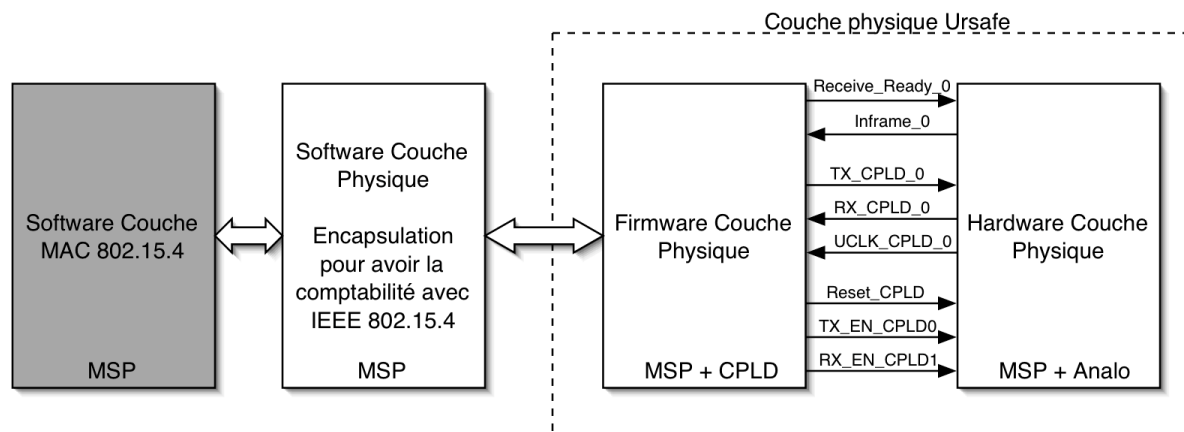


Figure 3 – Adaptation de la couche MAC 802.15.4 à la couche physique URSAFE

Cette adaptation partielle comporte quelques limitations, en particulier pour le débit des données. En effet, la couche PHY UWB communique avec des paquets fixes de 256 octets. Par contre la norme IEEE 802.15.4 ne tolère pas de paquet MAC (MPDU) plus grand que 127 octets. Les paquets PHY seront obligatoirement remplis de moitié par des zéros.

Le débit binaire de la transmission par UWB entre deux modules est fixé à 9.6 kb/s, mais le codage Manchester qui emploie 2 bits pour coder un symbole correspond à un débit à 4.8 kbaud.

1.2 Couche PHY IEEE 802.15.4

1.2.1 Rôles et services

La couche PHY a plusieurs rôles importants à réaliser :

- Activation et désactivation de la transmission radio
- Commutation des canaux
- Évaluation de la qualité du canal

La couche PHY fournit deux services à la couche supérieure :

- Service de données : PD-SAP (PHY Data service)
- Service de gestion : PLME-SAP (PHY Layer Management Entity)

1.2.2 Caractéristiques générales

La norme réunie en réalité deux couches physiques pratiquement identiques, chacune d'elles pouvant être combinées avec la couche MAC. La différence fondamentale entre les deux couches physiques est la bande de fréquence utilisée. L'une utilise la bande ISM 2.4 GHz et l'autre les bandes ISM 868 MHz (Europe) ou 915 MHz (Etats-Unis). La solution 2.4 GHz offre les meilleures performances. La solution 868/915 MHz est plutôt présente comme alternative si l'espace utilisé est déjà très encombré par d'autres appareils utilisant déjà la bande 2.4 GHz.

Les caractéristiques de chaque fréquence sont regroupées au Tableau 1.

Bande de fréquence [MHz]	Numéro du canal	Modulation	Bit rate [kb/s]	Symbol rate [kb/s]
868 - 868.6	0	BPSK	20	20
902 - 928	1,2,... 10	BPSK	40	40
2400 - 2483.5	11,12,... 26	O-QPSK	250	62.5

Tableau 1 - Caractéristiques de la couche PHY

Les débits maximaux ne sont pas les mêmes suivant la bande de fréquence utilisée, tout comme les deux modulations de phase. Au total, les 3 bandes de fréquence comportent 27 canaux différents. À 868 MHz, il n'y a qu'un seul canal présent. Les canaux à 900 Mhz sont espacés de 2 MHz, tandis que les canaux à 2.4 GHz sont espacés de 5 MHz.

Les deux couches physiques reposent sur le principe du DSSS (Direct Sequence Spread Spectrum). Le principe du DSSS consiste à étaler le spectre des données à l'aide d'une séquence pseudo aléatoire (SPA). Concrètement, la séquence de bits à transmettre est multipliée par une séquence pseudo aléatoire d'étalement. Cette opération a pour effet l'élargissement de la largeur de bande du signal original. À la réception, on multiplie le signal de bande large par la même séquence pseudo aléatoire et on récupère ainsi la séquence de bits originale.

La couche physique contient plusieurs fonctions de bas niveau permettant l'implémentation d'une sélection dynamique de canaux qui se fait aux couches supérieures :

- Détection d'énergie à la réception
- Indication de qualité du lien
- Gestion de la commutation de canal

Ces fonctions sont surtout utilisées lors de l'établissement de la connexion initiale au canal et pour les commutations de canaux.

1.2.3 Structure générale d'un paquet PHY

La structure des deux couches physiques est la même afin d'avoir une compatibilité unique avec la couche MAC, cette structure de trames est présente à la Figure 4.

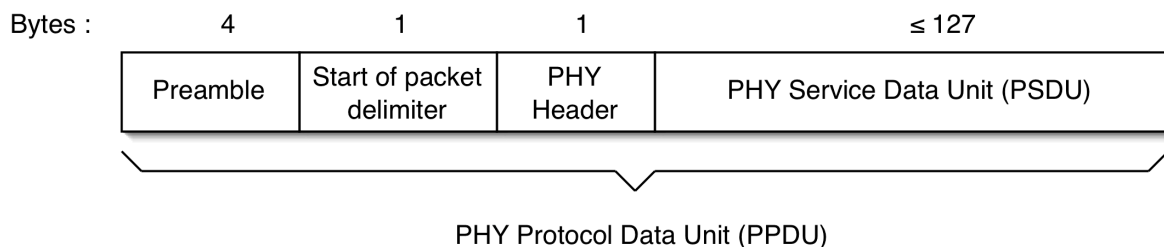


Figure 4 - Structure générale d'une trame PHY

L'ensemble de la trame est appelé PPDU (PHY Protocol Data Unit). Les 5 premiers bytes de la trame correspondent à un en-tête de synchronisation (Preamble + Start of packet delimiter). Le Byte « Start of packet delimiter » permet de spécifier la fin du préambule. Sinon, les 32 bits du préambule sont notamment prévus pour :

- Acquisition des symboles
- Synchronisation des « Chips »
- Ajustement de la fréquence (dans certain cas seulement)

Le champ « PHY Header » sert principalement à connaître la longueur de la cargaison avec un codage sur 7 bits des 8 disponibles. Le bit restant n'étant pas utilisé (réservé).

1.3 Couche MAC IEEE 802.15.4

1.3.1 Rôles et services

La couche MAC a plusieurs rôles importants à réaliser :

- Mécanisme d'accès au canal
- Ordonnancement des données
- Délivrement des trames d'acquiescement (ACK)
- Entretien des « time slot »
- Gestion des « Beacons » (signaux phares)
- Garantir l'intégrité des données

La couche MAC fournit deux services aux couches supérieures : SAP (Service Access Point) :

- Service de données : MCPS-SAP (MAC Common Part Sublayer)
- Service de gestion : MLME-SAP (MAC Layer Management Entity)

La couche MAC reçoit deux services de la couche physique qui est la couche inférieure.

- Service de données : PD-SAP (PHY Data service)
- Service de gestion : PLME-SAP (PHY Layer Management Entity)

La Figure 5 propose une illustration des interactions des services proposés et reçus par la couche MAC.

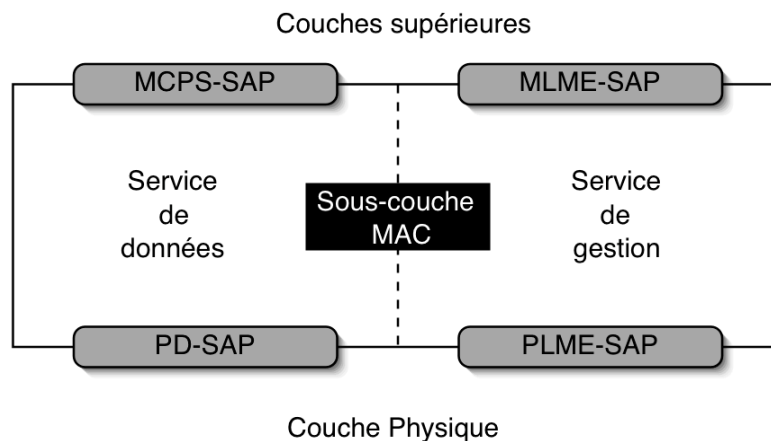


Figure 5 - Interactions des services de la couche MAC

1.3.2 Structure générale des paquets

Il existe 4 structures de paquet au niveau MAC :

- Trame de données
- Trame « Beacon » (trame phare)
- Trame d'acquiescement (ACK)
- Trame de commande

Les trames de données et de Beacon peuvent contenir des informations qui proviennent ou qui sont destinées aux couches supérieures. Les deux autres structures de trames sont générées et ne sont utilisées que par la couche MAC. Néanmoins, les trames Beacon peuvent ne contenir aucune information venant des couches supérieures et donc, n’être utilisées également que par la couche MAC.

1.3.2.1 Structure d’une trame de données

La structure générale d’une trame MAC de donnée est visible à la Figure 6.

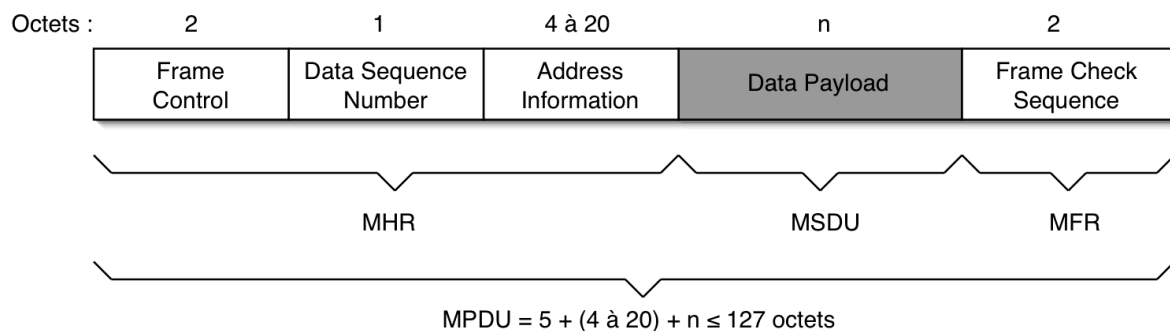


Figure 6 – Structure d’une trame MAC de donnée

La trame complète est appelée MPDU (MAC Protocol Data Unit) et ne peut posséder plus de 127 bytes. Elle contient un en-tête (MHR), des données provenant des couches supérieures (MSDU) et une fin de séquence (MFR). Voici la description des différents champs contenus dans l’en-tête et la fin de séquence.

- Frame Control (2 octets)

Ce champ de 16 bits d’informations est commun à tous les types de trames et sert à spécifier la structure et le contenu du reste de la trame. Son format est à la Figure 7.

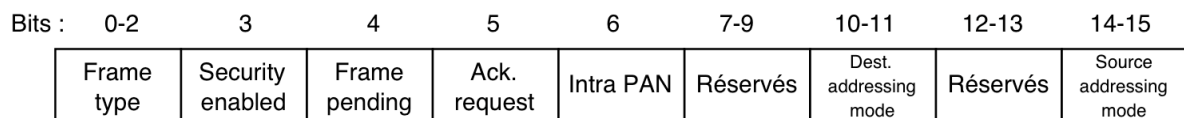


Figure 7 - Format de la trame de contrôle

- Frame type (3 bits)

Ce champ sert à définir le type de trame. Les valeurs définissant les différents types de trame se trouve au Tableau 2.

b2 b1 b0	Description
000	Beacon
001	Donnée
010	Ack
011	Commande
100-111	Réservé

Tableau 2 - Champ Frame type

- Security enabled (1 bit)

Une sécurité au niveau MAC peut être activée en mettant à 1 ce champ. La trame est alors protégée par cryptographie AES en utilisant une clé qui est stockée dans le MAC PIB.

- Frame pending (1 bit)

Quand ce bit est actif, cela indique au destinataire que des données sont encore présentes pour lui.

- Ack. Request (1 bit)

Ce bit permet de spécifier si le destinataire doit envoyer un acquittement (bit = 1) ou pas (bit = 0). Toute transmission Broadcast à ce champ à 0. Les trames Beacon et d'acquittement ont donc toujours ce champ à 0.

- Intra PAN (1 bit)

Ce bit permet d'indiquer si la trame doit être envoyée dans le même PAN (intra PAN) ou sur un autre PAN (inter PAN).

- Destination addressing mode (2 bits)

Ce champ permet de spécifier le type d'adressage de la destination. Les valeurs sont au Tableau 3.

- Source addressing mode (2 bits)

Ce champ permet de spécifier le type d'adressage de la source. Les valeurs sont au Tableau 3.

b1 b0	Description
00	L'identificateur PAN et le champ d'adresse ne sont pas présents
01	<i>Reservé</i>
10	Le champ d'adresse contient un adressage court (16 bits)
11	Le champ d'adresse contient un adressage étendu (64 bits)

Tableau 3 - Champs Addressing mode

- Data Sequence Number (1 octet)

Ce champ définit une numérotation de trames sur 8 bits, qui est notamment utilisée lors des acquittements afin de connaître quelles trames ont été acquittées. Sa valeur correspond à la variable PIB *macBSN*, qui est initialisée aléatoirement puis, pour chaque trame, elle est incrémentée d'une unité. Ce champ est utilisé pour les trames de type de données, de commande et d'acquittement.

- Address Info (4 à 20 octets)

Parmi toutes les configurations que peut prendre le champ « Address Info », il contient toujours au moins 4 octets pour une longueur maximale de 20 octets. La structure de ce champ est présentée à la Figure 8.

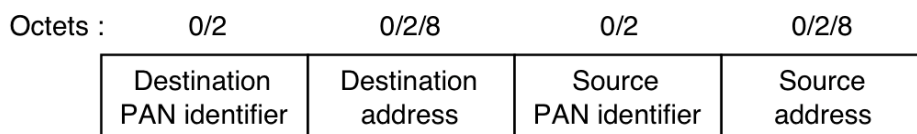


Figure 8 - Format du champ Address Info

- Destination PAN identifier (0 ou 2 octets)

L'identifiant PAN de destination utilise 16 bits et correspond au numéro du PAN qui reçoit les trames.

- Destination address (0, 2 ou 8 octets)

L'adresse de destination spécifie l'adresse à qui les trames sont envoyées. L'adresse peut être courte (16 bits) ou étendue (64 bits). L'adresse étendue est une adresse unique. L'adresse courte n'est utilisée qu'au sein d'un même PAN.

- Source PAN identifier (0 ou 2 octets)

L'identifiant PAN source utilise 16 bits et correspond au numéro du PAN qui envoie les trames.

- Source address (0, 2 ou 8 octets)

L'adresse source spécifie l'adresse de laquelle les trames sont envoyées. L'adresse peut être courte (16 bits) ou étendue (64 bits).

- Data Payload (cargaison)

Ce champ a une longueur variable. Néanmoins, une trame MAC total (MPDU) ne peut dépasser 127 octets de longueur. Le contenu de la cargaison est appelé MSDU et dans le cas d'une trame de données, celle-ci provient toujours des couches supérieures.

- Frame Check Sequence (2 octets)

Ce champ, qui s'ajoute après la cargaison, sert à contrôler l'intégrité des en-têtes et des données de la trame. Il est obligatoire pour tous les types de trames. Aucune correction n'est faite, si des bits erronés sont détectés, il doit y avoir retransmission. C'est un CRC (Cyclic Redundancy Check) de 16 bits qui est utilisé. Le taux d'erreurs non détectées par ce CRC est toujours le même quel que soit le nombre de données impliquées dans le calcul du CRC. Son taux d'erreur est calculé à l'Équation 2.

$$t_{err\%} = 2^{-n} = 2^{-16} = 0.00153\%$$

Équ. 2 - Taux d'erreur non détectée par le CRC

Le polynôme générateur spécifié par le standard est donné à l'Équation 3. Pour calculer la valeur hexadécimale, le bit de puissance 16 n'est pas pris en compte.

$$G_{16}(x) = x^{16} + x^{12} + x^5 + 1 \quad (0x1021)$$

Équ. 3 - Polynôme générateur du CRC

Le standard 802.15.4 spécifie encore que la valeur du CRC initial doit être de 0, ce qui implique que le CRC ne détecterait aucune erreur si tous les bits reçus valent 0. Il semble en effet étonnant que pour ce genre de transmission, le CRC ne soit pas plutôt initialisé avec des 1 (0xFFFF). Mais il est bien sûr impératif de suivre ce que propose le standard.

L'implémentation du CRC est décrit au paragraphe 3.6.3, tandis que la vérification du CRC implémenté est réalisé au paragraphe 3.7.

1.3.2.2 Structure d'une trame « Beacon »

La structure MAC d'une trame « Beacon » est visible à la Figure 9.

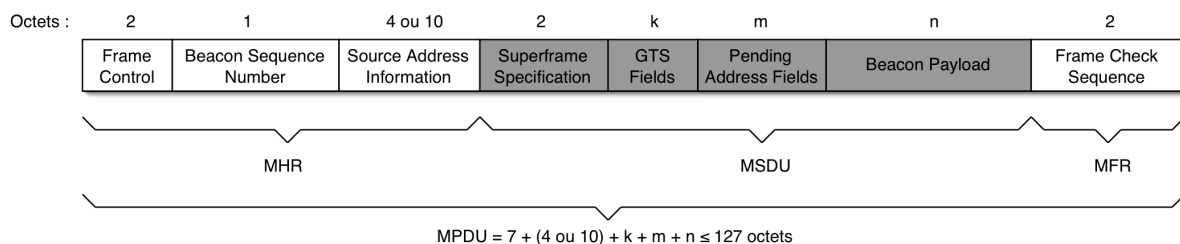


Figure 9 - Structure d'une trame MAC Beacon

Les trames « Beacon » ne peuvent être transmises que par un dispositif possédant toutes les fonctionnalités (FFD). Les informations qu'elles contiennent servent à la gestion du réseau en décrivant les caractéristiques du PAN. C'est à l'aide de Beacons que la définition d'un mode de transmission que l'on appelle Superframe (réseau « Beacon Enabled ») est possible, ce mode de transmission va servir à la synchronisation du réseau. C'est-à-dire que le coordinateur du PAN envoie régulièrement des Beacons et que les dispositifs utilisent la réception régulière des Beacons pour tenir une base temps commune. Une autre fonctionnalité des Beacons concerne les transmissions indirectes, c'est-à-dire que le Beacon peut contenir des adresses de dispositifs qui indique que des données sont en attente chez le coordinateur du PAN et qu'il faut aller les récupérer.

- Beacon Sequence Number (1 octet)

Ce champ contient le numéro de séquence du Beacon qui correspond à la variable *macBSN*. Au départ, il est initialisé à une valeur aléatoire, puis incrémenté d'une unité à chaque envoi de Beacon.

- Source Address information (4 ou 10 octets)

Ce champ indique l'adresse du dispositif source (2 ou 8 octets) qui envoie les trames Beacon et l'identifiant du PAN source (2 octets).

- Superframe Specification (2 octets)

Ce champ apporte plusieurs informations sur la configuration de la Superframe. Son format est présent à la Figure 10.

Bits :	0-3	4-7	8-11	12	13	14	15
	Beacon order	Superframe order	Final CAP slot	Battery life extension	Réservé	PAN coordinator	Association permit

Figure 10 - Format du champ Superframe Specification

- Beacon order (4 bits)

Le « Beacon Order » permet de déterminer par calcul (voir Équ. 4) l'intervalle des transmissions entre 2 Beacons (BI) en nombre de symboles. Si la valeur du BO est égale à 15, le réseau n'est alors pas synchronisé avec des Beacons (réseau « Non Beacon Enabled ») et il n'y a pas d'écart à calculer entre les Beacons.

- Superframe order (4 bits)

Le « Superframe Order » permet par calcul (voir Équ. 5) de connaître la durée du temps en nombre de symboles qu'une Superframe est active (SD) incluant le temps de la transmission de la trame Beacon. Si la valeur de SO est de 15, le mode de Superframe n'est pas utilisé puisque qu'aucun Beacon n'est transmis (réseau « Non Beacon Enabled »).

- Final CAP slot (3 bits)

Ce champ spécifie le dernier slot utilisé par la zone de contention (CAP). La durée du CAP est directement dépendante de ce champ. La plage valide est de 0 à 15.

- Battery life extension (1 bit)

Ce champ permet d'arrêter le récepteur après une période d'inter-trame (IFS) de la trame Beacon.

- PAN coordinator (1 bit)

Ce bit permet de spécifier que c'est effectivement un coordinateur PAN qui a envoyé la trame Beacon.

- Association permit (1 bit)

Ce champ permet d'indiquer au dispositif si le coordinateur accepte les demandes d'association au PAN.

- GTS Fields (k octets)

C'est ce champ qui permet de faire des réservations de temps pour des applications ayant besoin de faibles latences et des débits soutenus (meilleure qualité de service).

Il est séparé en 3 parties :

- GTS specification (1 octet)

On retrouve dans ce champ le « GTS descriptor count » (3 bits) qui sert à spécifier le nombre de GTS descriptor (de 3 octets) qui sont présents dans le champ « GTS list ». Si la valeur de ce champ « GTS descriptor count » est nulle, les champs « GTS directions » et « GTS list » ne sont pas présents.

L'autre champ présent est le « GTS permit » (1 bit) qui sert à savoir si une nouvelle demande de GTS est possible.

- GTS directions (0 ou 1 octet)

Si ce champ est utilisé, il se compose de « GTS directions mask » (7 bits). Le premier bit va définir la direction de transfert du premier GTS contenu dans la « GTS list » et ainsi de suite.

Un bit à 1 établit que le GTS est uniquement utilisé pour une réception et un bit à 0 établit que le GTS est uniquement utilisé pour une transmission.

- GTS list (octet variable)

Le nombre maximum de GTS Descriptor pouvant être contenu dans ce champ « GTS list » est limité à 7. Chaque GTS Descriptor occupe 24 bits.

Les 16 premiers bits sont utilisés pour l'adressage court. Ensuite, le champ « GTS starting slot » occupe 4 bits et contient le numéro du slot où le GTS doit débiter. Enfin, les 4 derniers bits sont utilisés par le champ « GTS length » qui donne le nombre de slots durant lequel le GTS est actif.

- Pending Address Fields (m octets)

Ce champ sert à connaître les adresses des dispositifs qui ont des données en attente chez le coordinateur, il est séparé en 2 parties :

- Pending address specification (1 octet)

Dans ce champ, on retrouve notamment le nombre d'adresse courte (3 bits) et le nombre d'adresse étendue (3 bits) qui sont présentes dans le champ « Address list ».

1.3.2.4 Structure d'une trame de commande

Les trames de commande sont utilisées pour réaliser des demandes des différentes fonctionnalités MAC. La structure d'une trame de commande est présente à la Figure 12.

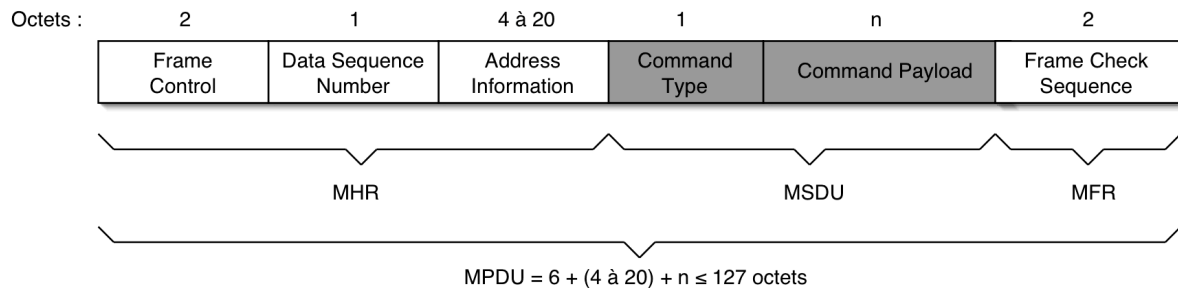


Figure 12 - Structure d'une trame MAC de commande

- Address Information (1 octet)

L'adressage va dépendre du type de commande utilisée. Pour savoir quel adressage est utilisé pour telle commande, il faut se reporter au Tableau 4.

- Command Type (1 octet)

C'est ce champ qui spécifie le type de la trame de commande. Il existe au total 9 types de trames de commande. Un FFD est capable d'envoyer et de recevoir toutes les trames de commande, contrairement au RFD. Le Tableau 4 indique les différents types de trames de commande et ce qu'est capable de réaliser un RFD.

Valeur	Nom de la commande	Command Payload [octet]	RFD		Adressage			
			Tx	Rx	Source	Dest.	PAN source	PAN dest.
0x01	Association Request	1	X		64 bits	16/64 bits	0xFFFF	16 bits
0x02	Association Response	3		X	64 bits	64 bits	16 bits	16 bits
0x03	Disassociation Notification	1	X	X	64 bits	64 bits	16 bits	16 bits
0x04	Data Request	-	X		16/64 bits	0/16 bits	16 bits	0/16 bits
0x05	PAN ID Conflict Notification	-	X		64 bits	64 bits	16 bits	16 bits
0x06	Orphan Notification	-	X		64 bits	16 bits	0xFFFF	0xFFFF
0x07	Beacon Request	-			0 bits	16 bits	0 bits	0xFFFF
0x08	Coordinator Realignment	7		X	64 bits	16/64 bits	16 bits	0xFFFF
0x09	GTS Request	1			16 bits	0 bits	16 bits	0 bits

Tableau 4 - Types de trame de commande

Le champ « Command Payload » va directement contenir les données des commandes MAC. Sa taille va donc varier en fonction des différentes commandes.

- Association Request

Cette commande permet à un dispositif de faire une demande d'association à un coordinateur.

La taille du champ « Address Information » est de 14 ou 20 octets.

Cette commande place le champ « **Capability Information** » (1 octet) dans le « Command Payload ». Ce champ se compose comme suit :

- Alternate PAN coordinator (bit 0)

Ce bit est actif si le dispositif est capable de devenir un coordinateur PAN.

- Device type (bit 1)

Ce bit est actif pour un FFD et inactif pour un RFD.

- Power source (bit 2)

Ce bit est actif si le dispositif reçoit de l'énergie provenant d'une autre source que la principale (source externe).

- Receiver on when idle (bit 3)

Ce bit est actif si le dispositif laisse enclencher son récepteur durant les périodes d'attente.

- Security capability (bit 6)

Ce bit est actif lorsque le dispositif est capable d'envoyer et de recevoir des trames MAC sécurisées.

- Allocate address (bit 7)

Ce bit est actif lorsque le dispositif souhaite que le coordinateur utilise un adressage court lors de la procédure d'association. Si le bit est inactif, l'adressage étendu doit être utilisé.

- Association Response

Cette commande permet au coordinateur de communiquer les résultats de l'association au dispositif qui a fait la demande.

La taille du champ « Address Information » est de 20 octets.

Le « Command Payload » contient 2 champs, « **Short Address** » (2 octets) et « **Association Status** » (1 octet).

Si le coordinateur PAN a associé un dispositif, le champ « Short Address » contient l'adresse courte que le dispositif doit utiliser pour communiquer avec son PAN. Si la valeur du champ est de 0xFFFF, le coordinateur n'a pas pu associer le dispositif au PAN. Si la valeur est de 0xFFFE l'association est réussie, mais il faut utiliser un adressage étendu de 64 bits.

Le champ « Association Status » permet de connaître l'état de l'association. Il existe 3 états différents :

- 0x00 : L'association est réussie.
- 0x01 : Le PAN n'a plus la capacité d'accueillir un nouveau dispositif.
- 0x02 : Le dispositif n'a pas les droits requis pour s'associer à ce PAN.

- Disassociation Notification

Cette commande permet de savoir pourquoi il y a eu une dissociation.

La taille du champ « Address Information » est de 20 octets.

Le « Command Payload » contient un seul champ, « **Disassociation Reason** » (1 octet), qui permet de connaître la raison de la dissociation :

- 0x01 : Le coordinateur a fait quitter le dispositif du PAN.
- 0x02 : Le dispositif a quitté de lui-même le PAN.

- Data Request

Cette commande est utilisée dans plusieurs cas différents. Tout d'abord dans un réseau « Beacon Enabled », elle est envoyée par un dispositif lorsqu'il a détecté par un Beacon reçu que des données sont en attente pour lui chez le coordinateur. Cette trame de commande est également utilisée lorsque le réseau est « Non Beacon Enabled » et que la couche supérieure du dispositif indique à la couche MAC de faire un « poll ». Et enfin, elle est employée durant la procédure d'association d'un dispositif au PAN.

La taille du champ « Address Information » est de 7, 11, 14 ou 17 octets.

Cette commande ne place pas de donnée dans le « Command Payload ».

- PAN ID Conflict Notification

Cette commande est envoyée par un dispositif au coordinateur lorsqu'un conflit d'identifiant PAN est détecté.

La taille du champ « Address Information » est de 20 octets.

Cette commande ne place pas de données dans le « Command Payload ».

- Orphan Notification

Cette commande est utilisée par un dispositif associé qui a perdu la synchronisation avec son coordinateur.

La taille du champ « Address Information » est de 14 octets.

Cette commande ne place pas de données dans le « Command Payload ».

- Beacon Request

Cette commande est utilisée par un dispositif FFD afin de localiser les coordinateurs dans le POS durant un scan actif.

La taille du champ « Address Information » est de 6 octets.

Cette commande ne place pas de données dans le « Command Payload ».

- Coordinator Realignment

Cette commande est envoyée par un coordinateur soit après avoir reçu une commande « Orphan notification » de la part d'un dispositif ou soit après un changement de plusieurs attributs de la configuration PAN (ordre donné par les couches supérieures). Cette commande permet au dispositif orphelin d'être de nouveau associé au coordinateur et à tous les dispositifs de réactualiser les informations à propos du PAN (envoi par Broadcast).

La taille du champ « Address Information » est de 14 ou 20 octets.

Le « Command Payload » contient 4 champs pour un total de 7 octets, « **PAN Identifier** » (2 octets), « **Coordinator Short Address** » (2 octets), « **Logical Channel** » (1 octet) et « **Short Address** » (2 octets).

Le champ « PAN Identifier » contient l'identifiant PAN de 16 bits.

Le champ « Coordinator Short Address » contient la variable *macShortAddress* (16 bits) que le dispositif utilisera pour toutes les futures communications.

Le champ « Logical Channel » (1 byte) contient le numéro du canal que le coordinateur à l'intention d'utiliser pour le PAN.

Si l'envoi est Broadcast, le champ « ShortAddress » (16 bits) vaut 0xFFFF. Si la trame de commande est envoyée pour un orphelin, ce champ contient la nouvelle adresse courte de l'orphelin qu'il utilise pour communiquer dans le PAN.

- GTS Request

Cette commande est utilisée par un dispositif associé qui fait une demande pour allouer un nouveau GTS ou pour arrêter d'allouer un GTS existant.

La taille du champ « Address Information » est de 6 octets.

Le « Command Payload » contient un seul champ, « **GTS Characteristics** » (1 octet) qui se compose comme suit :

- GTS length (bit 0-3)

Il contient la longueur du GTS en nombre de slots.

- GTS direction (bit 4)

Il contient le sens de la communication. Un bit actif équivaut uniquement à la réception de données durant le GTS. Un bit inactif correspondant à une réception de données durant le GTS.

- Characteristics type (bit 5)

Ce bit est activé afin d'allouer un nouveau GTS. Sinon quand le bit est à 0, cela permet d'arrêter d'allouer un GTS existant.

1.3.3 Méthodes d'accès au canal

L'une des tâches principales de la couche MAC est de proposer une méthode d'accès au canal. La norme contient deux méthodes d'accès, la principale est la méthode CSMA-CA qui est implémentée en deux versions, l'une non synchronisée (utilisé dans un réseau « Non Beacon Enabled ») et une synchronisée sur des Backoffs (utilisé dans un réseau « Beacon Enabled »). La deuxième méthode d'accès est une forme de polling par réservation de temps qui est obtenu grâce à un mode de transmission spécifique que l'on appelle Superframe. Cette dernière méthode ne peut-être obtenue que dans un réseau synchronisé par un coordinateur (réseau « Beacon Enabled »).

1.3.3.1 CSMA-CA

Le principe du CSMA-CA (Carrier Sense Multiple Access - Collision Avoidance) est de détecter l'activité du réseau avant de transmettre, afin d'éviter des collisions. Si le canal de transmission n'est pas libre, on attend qu'il se libère.

Il existe deux versions de l'algorithme du CSMA-CA dans la description du standard, la principale différence est que l'un est synchronisé pour l'accès au canal, tandis que l'autre ne l'est pas :

- **CSMA-CA slotted** (synchronisé sur des Backoffs) : Figure 14
- **CSMA-CA unslotted** : Figure 13

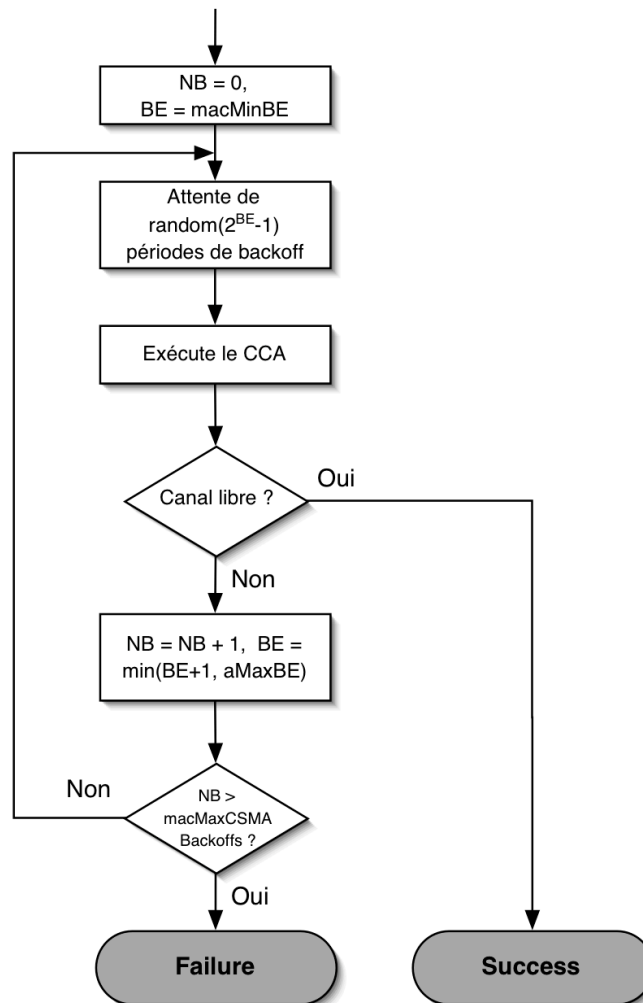


Figure 13 - Algorithme CSMA-CA "unslotted"

La version « slotted » est utilisée lorsque le réseau est « Beacon Enabled », tandis que la version « unslotted » est utilisée lorsque le réseau est « Non Beacon Enabled ». Dans les deux cas, le CSMA-CA est uniquement utilisé pour les transmissions de trames de **données** et de **commande**. La différence est que le CSMA-CA est dans tous les cas appliqué dans un réseau « Non Beacon Enabled », tandis que dans un réseau « Beacon Enabled », il est uniquement utilisé durant la période de contention de la Superframe. L'algorithme n'est par contre jamais utilisé pour transmettre des trames d'acquittement, de Beacon ou n'importe quelle trame durant la période de CFP de la Superframe.

Pour les deux algorithmes, l'unité principale de temps employée est le Backoff qui est fixée avec la variable *UnitBackoffPeriod*.

Avec la version du CSMA-CA « slotted », la couche MAC doit s'assurer que la couche PHY commence toutes ses transmissions pile sur le début d'une période de Backoff. Tous les dispositifs d'un même PAN sont donc exactement alignés entre eux lorsqu'ils doivent transmettre ou recevoir.

Avec la version du CSMA-CA « unslotted », les périodes de Backoff d'un dispositif n'ont aucun lien temporel avec les périodes de Backoff d'un autre dispositif associé au même PAN.

Les algorithmes CSMA-CA demandent à ce que chaque dispositif maintienne à jour 2 (« unslotted ») ou 3 (« slotted ») variables :

- NB : C'est le nombre de fois que l'algorithme CSMA-CA est demandé pour réaliser la transmission courante. Cette valeur est initialisée à 0 avant chaque nouvelle transmission. Si NB dépasse la variable PIB `macMaxCSMABackoffs`, alors l'accès au canal est considéré comme étant un échec.
- CW : C'est la longueur de la fenêtre de contention (rien à voir avec CAP). Elle permet de déterminer le nombre de période de Backoff avant que la transmission puisse commencer. Cette variable n'est utilisée que pour la version « slotted » du CSMA-CA.
- BE : C'est l'exponentielle du Backoff, il va permettre de calculer un nombre aléatoire de périodes de Backoff qu'un dispositif doit attendre avant d'essayer d'évaluer le canal.

Juste avant de réaliser l'évaluation du canal (CCA), l'algorithme du CSMA prévoit donc un temps aléatoire afin que les différents dispositifs qui sont en concurrence en même temps n'obtiennent pas tous un résultat positif du CCA et l'autorisation de transmettre.

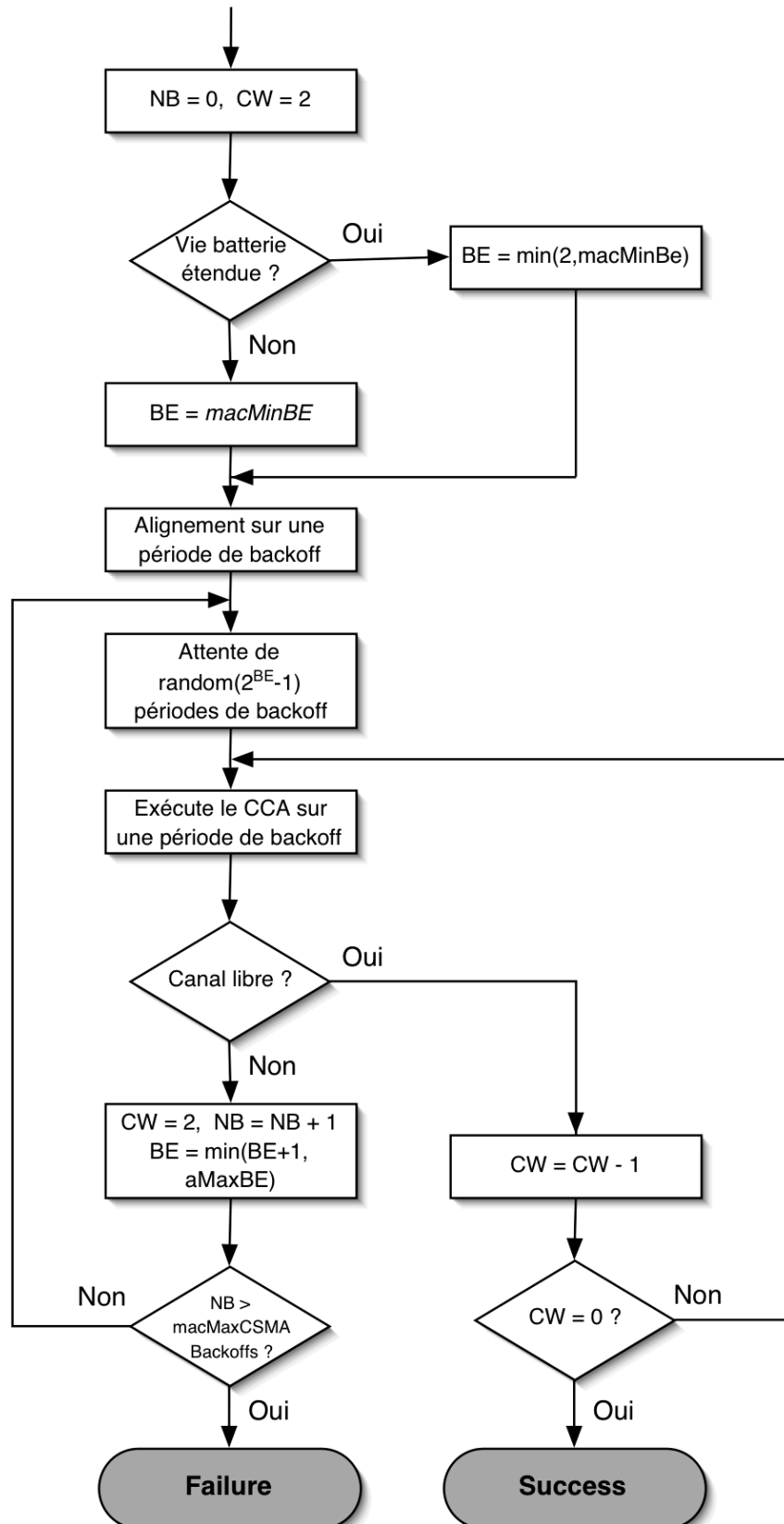


Figure 14 - Algorithme CSMA-CA "slotted"

1.3.3.2 Structure d'une Superframe

Une Superframe ne peut être générée qu'avec la topologie en étoile où un coordinateur PAN est présent. C'est ce dernier qui synchronise le PAN et gère les réservations de temps en envoyant des Beacons.

Une Superframe est délimitée par deux trames Beacon (signal phare). La partie active de la Superframe est toujours constituée de 16 slots de durée équivalente. Cette partie active est généralement divisée en deux, une période avec contention (CAP) et une période sans contention (CFP). La Superframe peut également contenir une partie inactive (non obligatoire) qui permet alors au coordinateur PAN d'entrer dans un mode de basse consommation. La structure générale d'une Superframe est à la Figure 15.

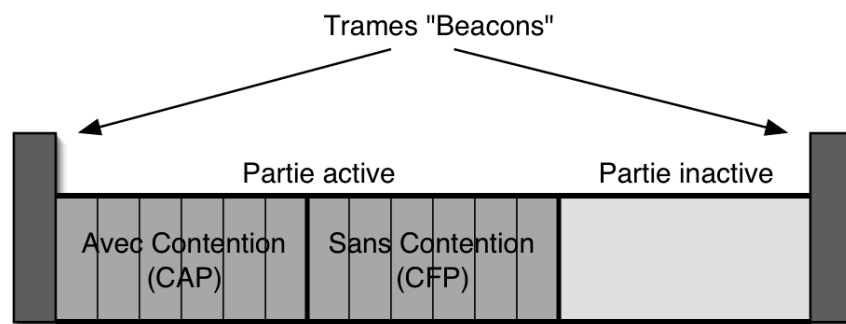


Figure 15 – Structure générale d'une Superframe

La période totale (Beacon + partie active + partie inactive) correspond à l'intervalle entre deux Beacons (BI), ce temps BI est déterminé par la variable PIB *macBeaconOrder* (BO). Le calcul est réalisé à partir de l'Équ. 4.

$$BI = aBaseSuperframeDuration \cdot 2^{BO} [\text{symboles}]$$

avec $0 \leq BO \leq 14$

Équ. 4 - Intervalle des Beacons (BI)

La longueur de la partie active (Beacon + CAP + CFP) appelé Superframe Duration (SD) se calcule de la même façon que pour le BI mais en utilisant la variable *macSuperframeOrder* (SO). La période active se calcule comme à l'Équ. 5.

$$SD = aBaseSuperframeDuration \cdot 2^{SO} [\text{symboles}]$$

avec $0 \leq SO \leq BO \leq 14$

Équ. 5 - Durée de la partie active d'une Superframe (SD)

Comme la partie active de la Superframe contient toujours un nombre de 16 Time slots, la durée d'un slot est directement liée à la valeur du « Superframe Order ».

Durant la période de contention, chaque nœud est en concurrence pour émettre, en utilisant le mécanisme CSMA-CA avec synchronisation sur les Backoffs (CSMA-CA « slotted »). Il est à remarquer qu'une Superframe sans CFP est tout à fait réalisable, elle ne contient alors que la période de contention. Durant la plage de CFP, le coordinateur PAN peut garantir des Time slots à un ou plusieurs dispositifs. Ces plages réservées sont appelées GTS (Guaranteed Time Slots). Un GTS peut s'étendre sur plusieurs slots.

Un exemple d'une structure typique de Superframe est présenté à la Figure 16. On remarque que les périodes active et inactive ont une durée équivalente, ce qui veut dire que le BO vaut une unité de plus que le SO. Dans cet exemple, la partie sans contention abrite deux réservations de temps (GTS 1 et GTS 2). Enfin, il faut souligner que temporellement la première trame Beacon fait partie du premier Time slot.

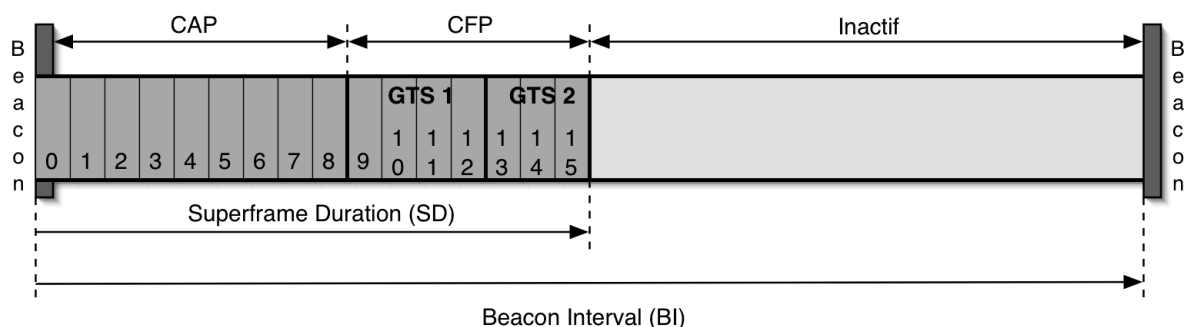


Figure 16 - Exemple d'une structure de Superframe

Pour obtenir un réseau « Non Beacon Enabled » (sans Superframe), il suffit de régler les variables *macBeaconOrder* et *macSuperframeOrder* à la valeur 15.

1.3.4 Modèles de transmissions de données

1.3.4.1 Topologies réseau

Au niveau de la couche réseau, la norme IEEE 802.15.4 permet la mise en place de deux topologies de réseau :

- Topologie en étoile
- Topologie « Peer to Peer » (particulier à particulier)

Ces deux topologies sont imagées à la Figure 17. On remarque que deux types de dispositifs peuvent constituer le réseau :

- **Full Function Device (FFD)**
- **Reduced Function Device (RFD)**

Un **FFD** est un dispositif qui possède toutes les fonctions MAC et PHY décrite par la norme. Les **RFD** sont quant à eux des dispositifs qui ne possèdent pas toutes les fonctions définies dans le standard.

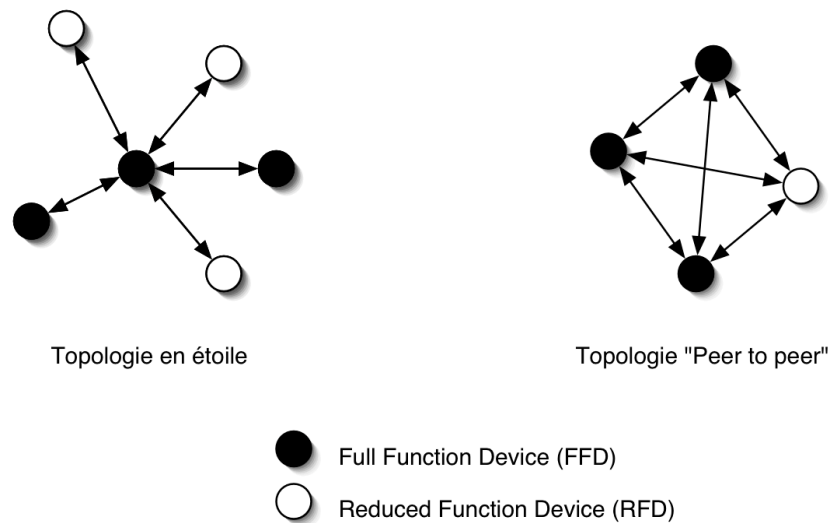


Figure 17 - Topologies proposées par la norme IEEE 802.15.4

1.3.4.2 Topologie étoile

Dans une topologie en étoile, il faut un coordinateur réseau qui contrôle la topologie du réseau et les transmissions de données, c'est le maître. Ce coordinateur réseau ne peut être qu'un FFD, le dispositif possédant toutes les fonctions. Les esclaves peuvent alors être des dispositifs aux fonctions réduites (RFD) ou même d'autres FFD.

Deux sens de transmissions sont possibles, la transmission ne se comportant pas de la même manière pour chaque cas :

- **Transmission uplink** (du dispositif esclave au coordinateur)
- **Transmission downlink** (du coordinateur au dispositif esclave)

Toutes les transmissions uplink se réalisent d'une façon directe, tandis que les transmissions downlink se font d'une manière indirecte. Une **transmission indirecte** permet au dispositif de rester dans un mode de veille, car c'est à lui de se manifester pour obtenir des données. Le coordinateur ne va jamais envoyer une trame de donnée ou une trame de commande d'une façon directe à un dispositif

Le réseau peut encore se comporter de deux manières différentes, en étant synchronisé ou en ne l'étant pas :

- « Beacon Enabled » (dispositifs synchronisés à l'aide de Beacons envoyés par le coordinateur)
- « Non Beacon Enabled » (dispositifs non synchronisés)

Cela donne finalement pour un réseau en étoile un nombre total de 5 possibilités de transmissions de données, car un dispositif esclave a encore la possibilité de réaliser des réservations de temps pour transmettre ses données dans un réseau « Beacon Enabled ». Le descriptif de ces 5 possibilités est résumé dans le Tableau 5.

	Uplink	Downlink
« Beacon Enabled » CAP	Le dispositif attend le Beacon en provenance du coordinateur qui spécifie le début de la Superframe. Le dispositif peut alors transmettre selon CSMA-CA « slotted » durant la période de contention (CAP).	Le coordinateur envoie régulièrement une trame Beacon spécifiant s'il y a des données en attente (pending data). Le dispositif ayant pris connaissance que des données sont en attente chez le coordinateur par la trame Beacon, va renvoyer une trame de commande « Data Request ». Le coordinateur envoie alors ses données selon CSMA-CA « slotted ». (transmission indirecte)
« Beacon Enabled » CFP (GTS)	Le dispositif a la possibilité de faire une demande de réservation de temps (Commande « GTS Request »). Le coordinateur confirme alors dans le prochain Beacon quand le dispositif aura le droit de transmettre dans la période de CFP.	-
« Non Beacon Enabled »	Le nœud peut à tout moment transmettre des données pour le coordinateur selon CSMA-CA « unslotted ».	Dans ce cas, le coordinateur doit conserver ses données tant que le dispositif n'a pas pris contact avec lui à l'aide d'un « poll » (consiste à réaliser une demande de données avec la trame de commande Data Request). Une fois fait, le coordinateur peut envoyer ses données au nœud selon CSMA-CA « unslotted ». (transmission indirecte)

Tableau 5 – Situations de transmission possibles dans une topologie en étoile

Les transmissions peuvent encore être acquittées par celui qui reçoit les données. Ces trames d'acquittement n'utilisent aucun mécanisme d'accès, car la réponse est directement renvoyée après la réception des données en attendant juste un temps très court qui permet au dispositif de commuter entre son mode de réception et de transmission.

1.3.4.3 Topologie « Peer to Peer »

Dans une topologie « Peer to Peer », il n'y a plus de coordination entre les dispositifs, car aucun coordinateur réseau n'est présent. Le réseau peut être constitué de FFD et de RFD qui peuvent tous communiquer entre eux à tout moment. Le réseau est donc obligatoirement « Non Beacon Enabled ». On se retrouve avec une seule situation de transmission possible. Elle est développée dans le Tableau 6.

	Uplink/ Downlink
Non « Beacon-enabled »	Un dispositif peut à tout moment transmettre des données pour un autre dispositif selon CSMA-CA « unslotted ».

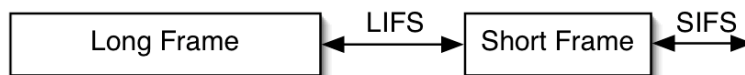
Tableau 6 – Situation de transmission possible dans une topologie peer to peer

1.3.4.4 Temps entre les trames

Entre chaque trame, la couche MAC qui reçoit les paquets a besoin d'un certain temps pour effectuer le traitement des données en provenance de la couche physique. C'est pourquoi deux constantes de temps ont été définies, SIFS et LIFS. C'est en fonction de la longueur de la trame que l'on va choisir quelle constante de temps appliquer après la trame. SIFS (Short Inter Frame Spacing) est la constante la plus petite, elle est utilisée lorsque la trame (MPDU) a une taille plus faible que la constante $aMaxSIFSFrameSize$, sinon on utilise LIFS (Long Inter Frame Spacing).

La durée de ces constantes correspond aux deux constantes $aMinSIFSPeriod$ et $aMinLIFSPeriod$. La Figure 18 résume l'utilisation des deux constantes de temps au niveau de la réception. Mais, c'est au niveau de la transmission que les attentes SIFS ou LIFS devront être réalisées. Dans un réseau avec acquittement, le récepteur doit attendre t_{ACK} avant de retransmettre l'acquittement afin que la couche physique ait le temps de commuter entre le mode de réception et le mode de transmission.

Transmission sans acquittement:



Transmission avec acquittement:

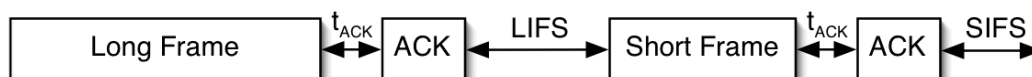


Figure 18 - Temps entre les trames (SIFS et LIFS) au niveau réception

Il est à noter que le temps t_{ACK} n'a pas besoin d'être attendu d'une manière passive, puisqu'en réalisant la demande de changement de mode de transmission du Transceiver, le temps minimum de la commutation est de tout façon attendue.

1.4 Description détaillée des primitives

Une primitive est un message servant à échanger des informations entre deux couches et qui est généralement composée de plusieurs paramètres.

La norme IEEE 802.15.4 utilise un total de 48 primitives au niveau de la couche PHY et de la couche MAC qui sont décomposées comme suit :

- Le service MAC de donnée (MCPS) possède 5 primitives
- Le service MAC de gestion (MLME) possède 30 primitives
- Le service PHY de donnée (PD) possède 3 primitives
- Le service PHY de gestion (PLME) possède 10 primitives

Toutes les primitives PHY vont s'échanger entre la couche physique et la couche MAC. Les primitives MAC de données vont s'échanger entre la couche MAC et le SSCS (Service Specific Convergence Sublayer), tandis que les primitives MAC de gestion vont s'échanger entre la couche MAC et les couches supérieures. La Figure 19 résume la situation.

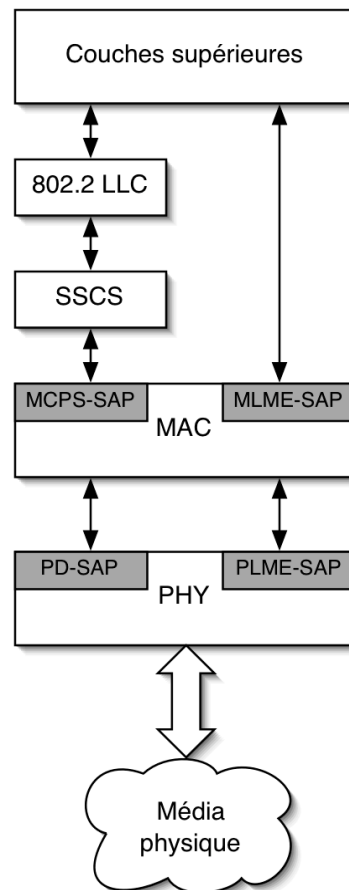


Figure 19 - Echange des primitives entre les services des couches

Sur cette Figure 19, on constate que les primitives de données passent par les couches LLC (Logical Link Control) et SSCS, tandis que les primitives de gestion communiquent directement avec les couches plus hautes.

La sous-couche SSCS sert à adapter les différentes couches MAC existantes dans les standards IEEE à une unique sous-couche LLC.

La sous-couche LLC est responsable des différentes transmissions de données possibles et doit également contrôler la taille des données transmises à la couche MAC. La sous-couche LLC est unique pour tous les réseaux IEEE et gère 3 types de transmissions de données :¹

- Type I : Service de datagrammes sans acquittements (supporte le point à point, le multipoint et les diffusions)
- Type II : Service de circuits virtuels (Contrôle de flux et corrections d'erreurs)
- Type III : Service de datagrammes avec acquittements (supporte uniquement le point à point)

C'est en partant de la définition de chaque primitive que le standard peut être passé entièrement en revue. Dans la partie qui suit, chaque primitive est décrite par ses paramètres, par les statuts que la primitive de type *indication* peut retourner et avec l'aide de diagramme en flèche résumant l'appel des primitives.

¹ Référence : TCP / IP, Karantjit S.Siyan, CampusPress, 706 pages, 2001

Cette partie 1.4 n'est pas conseillée à lire en continu, elle fait plutôt figure de référence tout comme l'**annexe A** qui regroupe un résumé de chaque primitive MAC avec une description de tous les paramètres. Pour une compréhension plus massive, il est préférable de se reporter directement à la deuxième partie du rapport : Fonctionnement et synthèse.

1.4.1 MCPS-DATA (request, confirm, indication)

MCPS-DATA.request :

[SrcAddrMode (integer), SrcPANid (integer), SrcAddr (device address), DstAddrMode (integer), DstPANid (integer), DstAddr (device address), msduLength (integer <= aMaxMACFrameSize), msdu (set of octets), msduHandle (integer), TxOptions (bitmap)]

De SSCS à MAC

MCPS-DATA.confirm :

[msduHandle (integer), Status (SUCCESS, TRANSACTION_OVERFLOW, TRANSACTION_EXPIRED, CHANNEL_ACCESS_FAILURE, INVALID_GTS, NO_ACK, UNAVAILABLE_KEY, FRAME_TOO_LONG, FAILED_SECURITY_CHECK or INVALID_PARAMETER)]

De MAC à SSCS

MCPS-DATA.indication :

[SrcAddrMode (integer), SrcPANid (integer), SrcAddr (device address), DstAddrMode (integer), DstPANid (integer), DstAddr (device address), msduLength (integer <= aMaxMACFrameSize), msdu (set of octets), mpduLinkQuality (integer), SecurityUse (boolean), ACLEntry (integer)]

De MAC à SSCS

1.4.1.1 Fonctionnement et utilisation

C'est par la primitive **MCPS-DATA.request** que tous les paquets de données sont envoyés. Elle fait appel aux primitives physiques PD-DATA.request et PD-DATA.confirm qui servent à mettre le Transceiver dans un mode d'envoi de données. Mais cette primitive de données MAC doit également être apte à recevoir des données dans le cas d'une transmission avec acquittement. Elle fait alors appel à la troisième primitive physique de données, PD-DATA.indication.

C'est également la primitive **MCPS-DATA.request** qui fait appel à l'algorithme du CSMA-CA pour l'accès au canal. Pour chaque envoi de paquets, il est impératif de connaître l'état du canal, c'est le CSMA-CA qui se charge de commander ces envois de paquets.

C'est par la primitive **MCPS-DATA.indication** que les paquets pourront être réceptionnés par un autre dispositif. En effet, cette primitive appelle la primitive PHY PD-DATA.indication qui sert à se mettre dans le mode de réception de données. Si la transmission doit être acquittée, la primitive MAC envoie un acquittement en se servant des primitives PD-DATA.request et PD-DATA.confirm.

La primitive **MCPS-DATA.confirm** sert alors à donner le résultat de la demande via un statut à la couche supérieure du dispositif qui a voulu envoyer les données.

L'utilisation et le séquençement général des trois primitives MCPS-DATA doit s'exécuter selon la Figure 20.

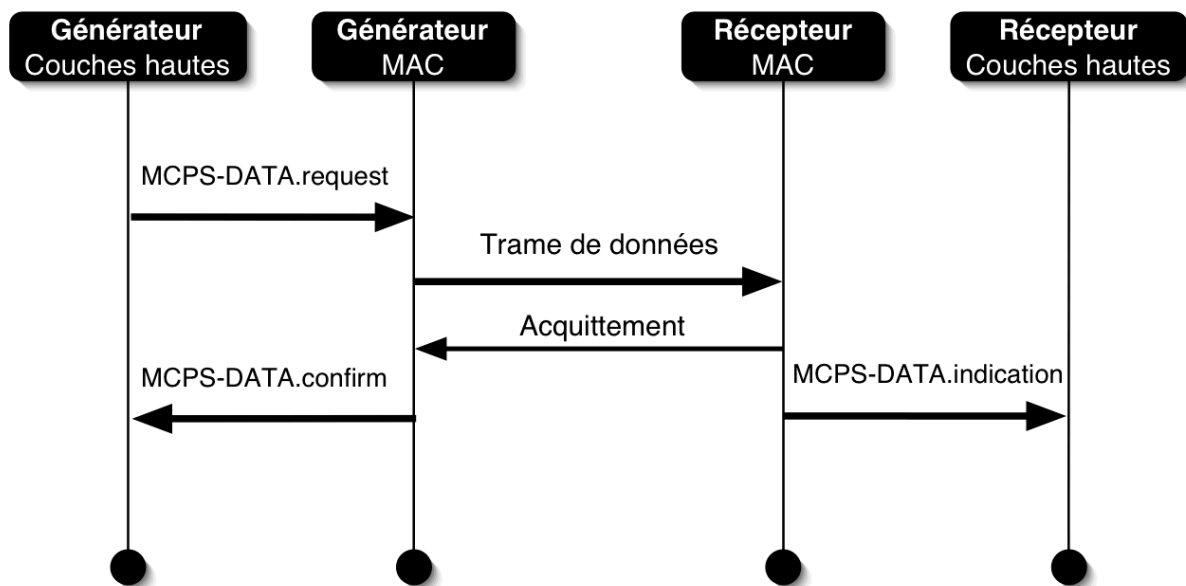


Figure 20 – Utilisation et séquençement des primitives MCPS-DATA

1.4.1.2 Construction de la trame MAC

Un des buts importants de la primitive **MCPS-DATA.request** est de construire une trame MAC de donnée à l'aide des données reçues par la couche supérieure et de l'envoyer à la couche physique. On reçoit de la couche supérieure un MSDU d'une longueur connue. Tous les transferts de données entre les couches se font en mémoire et par byte (8 bits). Les champs qui doivent être ajoutés peuvent être visualisés à la Figure 21 qui décrit la structure des trames MAC et PHY de données.

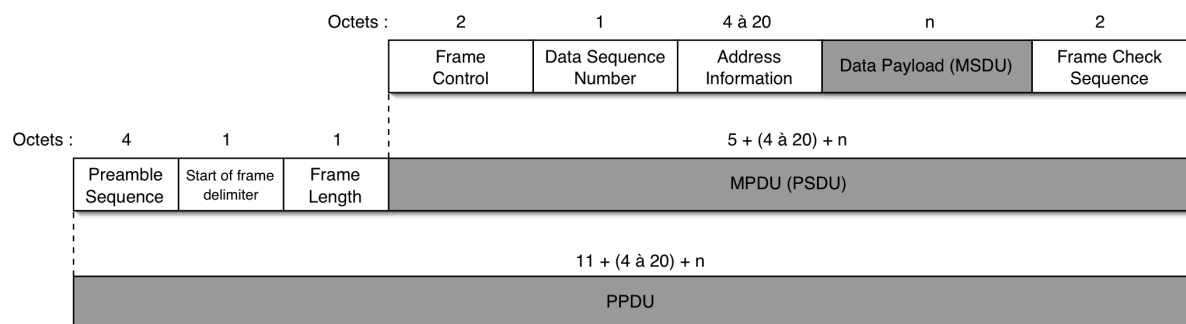


Figure 21 - Structure MAC et PHY d'une trame de données

La première action consiste à décaler les données du MSDU du nombre de bytes qu'utilise l'en-tête MAC (MHR) afin de pouvoir les y insérer. En effet, par principe, il ne faut pas que le pointeur change d'adresse mémoire, au risque d'effacer d'autres données situées avant en mémoire. Avant l'opération de construction de la trame, le pointeur se trouve au début du MSDU. À la fin de l'opération, il va pointer le début du MHR qui va correspondre au début du MPDU.

Le pointeur peut être décalé de 7 à 23 bits suivant les types d'adressages employés dans l'en-tête. Comme les adressages à utiliser sont connus, car indiqué par la couche supérieure, on peut effectuer le décalage. Le CRC doit également être ajouté, mais il n'y a pas besoin de réaliser de décalage, car il s'ajoute à la fin. Il faudra juste transmettre la bonne longueur du MPDU à la couche physique.

Le résumé de cette opération est à la Figure 22.

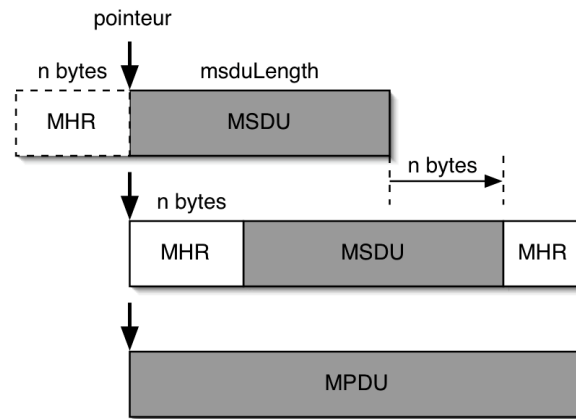


Figure 22 - Décalage du MSDU

1.4.1.3 Segmentation

Dans la norme IEEE 802.15.4, ce n'est pas à la couche MAC de segmenter les paquets, mais aux couches supérieures qui doivent s'assurer que la grandeur des données envoyées (MSDU) à la couche MAC ne va pas dépasser $aMaxMACFrameSize$, cette constante est définie à l'Équ. 6.

$$aMaxMACFrameSize = aMaxPHYPacketSize \square aMaxFrameOverhead = 127 \square 25 = 102 \text{ bytes}$$

Équ. 6 - Définition de $aMaxMACFrameSize$

La grandeur maximale d'une trame MAC complète est de 127 bytes, en déduisant la taille maximale utilisée par l'ensemble des en-têtes qui est de 25 bytes, on obtient la valeur de 102 bytes. La couche MAC ne peut donc pas recevoir des données de plus de 102 bytes provenant de la couche supérieure.

Étant donné que les couches supérieures ne sont pas implémentées dans ce projet, il peut être commode d'assurer une segmentation au niveau de l'application tournant au-dessus de la couche MAC. Cela permet alors de réaliser des applications où les données brutes provenant des couches supérieures pourront être de plus grande taille.

Le principal avantage de transmettre plusieurs petits paquets plutôt qu'un seul grand est que cela permet de diminuer le nombre de retransmissions de données lors d'éventuelles erreurs.

Générer des paquets trop petits n'est pas non plus optimal, car chaque en-tête de paquets représente un plus grand nombre d'informations à transmettre. Ce qui diminue le débit moyen de transmissions.

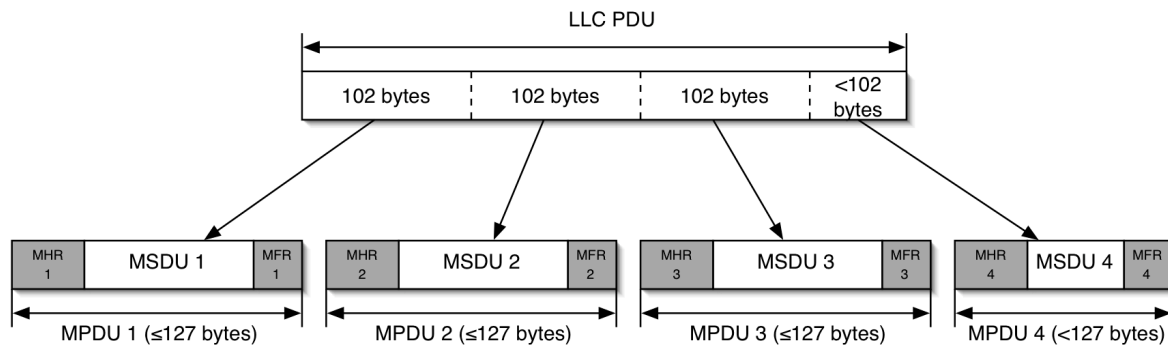


Figure 23 - Exemple d'une segmentation en 4

La Figure 23 présente un exemple de segmentation en 4 paquets. Dans ce cas, la couche supérieure (LLC) reçoit un trop grand nombre de données pour que la couche MAC puisse l'envoyer en une transmission. La couche supérieure doit alors segmenter les données en des longueurs maximales de *aMaxMACFrameSize* (102 bytes). Cela permet alors à la couche MAC de respecter la taille maximale d'une trame MAC, *aMaxPHYPacketSize* (127 bytes), pour n'importe quelle taille d'en-tête MAC. Dans cet exemple, on est donc contraint de créer 4 paquets de données MAC. Pour connaître combien de paquets segmentés l'opération nécessite, il suffit de connaître le quotient de la division entre la longueur du paquet venant des couches supérieures et de *aMaxMACFrameSize*.

1.4.1.4 Description par les statuts

La primitive **MCPS-DATA.confirm** sert à donner le résultat de la demande de la couche supérieure sous forme d'un statut. Voici les statuts possibles pour cette primitive :

SUCCESS : Le MSDU a été reçu correctement du SSCS et la trame est construite normalement avec ses en-têtes pour obtenir un MPDU. Une requête pour activer le mode de transmission est fait avec la primitive PLME-SET-TRX_STATE.request, si le résultat est SUCCESS ou TX_ON, la couche MAC peut envoyer son MPDU à la couche physique en se servant de la primitive PD_DATA.request. À la réception de PD-DATA.confirm, la couche MAC doit encore clôturer la communication en utilisant la primitive PLME-SET-TRX_STATE.request (mettre à TRX_OFF). Le statut SUCCESS est alors transmis à la sous-couche SSCS avec la primitive MCPS-Data.confirm. Si la transmission doit être acquittée, voir **NO_ACK**.

TRANSACTION_OVERFLOW : Le paramètre TxOption indique que la transmission est indirecte. Si le dispositif est un coordinateur PAN, l'information contenue dans la primitive est placée dans la liste d'attente des transactions. Si elle n'a plus la capacité de garder cette transaction, la couche MAC supprime le MSDU et le statut TRANSACTION_OVERFLOW est retourné.

TRANSACTION_EXPIRED : Si la trame de données placée dans la liste de transmissions du coordinateur n'est pas récupérée avant le temps *macTransactionPersistenceTime* par le dispositif concerné, la trame de données est supprimée de la liste et le statut TRANSACTION_EXPIRED est retourné.

CHANNEL_ACCESS_FAILURE : Si la transmission utilise CSMA-CA et que l'algorithme CSMA-CA échoue à l'accès du canal, toutes les informations (MSDU) de la transaction sont supprimées et le statut CHANNEL_ACCESS_FAILURE est retourné.

INVALID_GTS : Le paramètre TxOption indique que la transmission doit être faite lors d'un GTS. Si le dispositif est un coordinateur PAN, il va vérifier si l'adresse de destination correspond bien à un dispositif qui a fait une demande de GTS. Si aucune demande de GTS valide n'est trouvée, le statut INVALID_GTS est renvoyé avec la primitive MCPS-Data.confirm.

NO_ACK : Le paramètre TxOption indique que la transmission doit être acquittée. La trame est construite normalement et la couche MAC se met dans son mode réception dès que le MPDU est envoyé à la couche PHY. La couche MAC attend alors le temps de la variable *macAckWaitDuration* (en nombre de symboles). Si durant ce temps, la couche MAC n'a pas reçu d'acquittements, elle renouvelle un certain nombre de fois la transmission et l'attente de l'acquiescement en fonction de la constante *aMaxFrameRetries*. Si aucun ACK n'a été reçu par la couche MAC, le statut de MCPS-Data.confirm est NO_ACK et le MSDU est supprimé. (si un ACK est reçu, le statut est évidemment SUCCESS)

UNAVAILABLE_KEY : Le paramètre TxOption indique que la transmission doit être sécurisée. La couche MAC obtient alors une clé et des informations sécurisées qu'elle doit décoder avec la clé. Si aucune clé appropriée n'est trouvée dans le ACL, le statut de MCPS-Data.confirm est UNAVAILABLE_KEY et le MSDU est supprimé.

FRAME_TOO_LONG : Le paramètre TxOption indique que la transmission doit être sécurisée. La couche MAC a obtenu une clé valide, mais la longueur de la trame (MSDU) est plus grande que *aMaxMACFrameSize*, le MSDU est supprimé et le statut de MCPS-Data.confirm est FRAME_TOO_LONG. Ce statut est également utilisé lorsque la transaction est trop large pour tenir dans un CAP ou un GTS.

FAILED_SECURITY_CHECK : Si une erreur intervient durant le processus de sécurisation de la trame, le MSDU est supprimé et le statut de MCPS-Data.confirm est FAILED_SECURITY_CHECK.

INVALID_PARAMETER : Si un des paramètres de la primitive MCPS_DATA n'est pas du bon type ou est en dehors des plages autorisées, c'est le statut INVALID_PARAMETER qui est renvoyé avec MCPS-Data.confirm.

1.4.2 MCPS-PURGE (request, confirm)

MCPS-PURGE.request :

[msduHandle (integer)]

De SSCS à MAC (FFD seulement)

MCPS-PURGE.confirm :

[msduHandle (integer), Status (SUCCESS or INVALID_HANDLE)]

De MAC à SSCS (FFD seulement)

1.4.2.1 Fonctionnement

La primitive **MCPS-PURGE.request** permet à la couche SSCS de faire une demande pour supprimer un MSDU stocké dans la file d'attente de transmissions de la couche MAC d'un coordinateur PAN. La variable de 8 bits, *msduHandle*, est utilisée pour identifier les MSDU, cette variable correspond à une numérotation utilisée uniquement entre la couche SSCS et la couche MAC. Cette variable n'a rien à voir avec la numérotation de séquence utilisée par la couche MAC (DSN).

1.4.2.2 Description par les statuts

La primitive **MCPS-PURGE.confirm** va servir à donner le résultat de la demande de la suppression d'un MSDU sous forme d'un statut. Voici les statuts possibles pour cette primitive :

SUCCESS : La couche MAC a effectué la suppression du MSDU indiqué par la SSCS de sa file d'attente de transmissions.

INVALID_HANDLE : La couche MAC n'a pas réussi à retrouver le MSDU associé au Handle reçu. Aucun MSDU n'a été supprimé.

1.4.3 MLME-ASSOCIATE.(request, confirm, indication, response)

MLME-ASSOCIATE.request :

[LogicalChannel (integer), CoordAddrMode (integer), CoordPANId (integer), CoordAddress (device address), CapabilityInformation (bitmap), SecurityEnable (boolean)]

De couche supérieure à MAC

MLME-ASSOCIATE.confirm :

[AssocShortAddress (integer), Status (SUCCESS, CHANNEL_ACCESS_FAILURE, NO_ACK, NO_DATA, UNAVAILABLE_KEY, FAILED_SECURITY_CHECK or INVALID_PARAMETER)]

De MAC à couche supérieure

MLME-ASSOCIATE.indication :

[DeviceAddress (device address), CapabilityInformation (bitmap), SecurityUse (boolean), ACLEntry (integer)]

De MAC à couche supérieure (FFD seulement)

MLME-ASSOCIATE.response :

[DeviceAddress (device address), AssocShortAddress (integer), Status (enumeration), SecurityEnable (boolean)]

De couche supérieure à MAC (FFD seulement)

1.4.3.1 Fonctionnement et utilisation

La primitive **MLME-ASSOCIATE.request** permet à un dispositif isolé de s'associer à un PAN. La couche supérieure indique à la couche MAC notamment sur quel canal et avec quel PAN (*CoordPANId*) il faut s'associer. La couche supérieure connaît ces informations d'un précédent balayage (scan) obligatoire.

À la réception de la primitive **MLME-ASSOCIATE.request**, la couche MAC doit envoyer une trame de commande « Association Request » au coordinateur du PAN. La couche MAC du coordinateur PAN doit alors avertir la couche supérieure à l'aide de la primitive **MLME-ASSOCIATE.indication**. Cette dernière répond par la primitive **MLME-ASSOCIATE.response** en plaçant dans la liste d'attente de transactions du coordinateur une trame de commande « Association Response » (transmission indirecte). C'est alors à la couche MAC du dispositif qui a fait la demande d'association de faire une demande de données (Data Request) pour récupérer la trame de commande « Association Response ».

Le statut de la réponse peut donner 3 possibilités différentes :

- 0x00 : L'association est réussie.
- 0x01 : Le PAN n'a plus la capacité à accueillir un nouveau dispositif.
- 0x02 : Le dispositif n'a pas les droits requis pour s'associer à ce PAN.

Pour clore cette demande d'association, c'est à la couche MAC du dispositif à la base de la demande à devoir encore donner le résultat de l'association aux couches supérieures via la primitive **MLME-ASSOCIATE.confirm**.

L'utilisation et le séquençement général des quatre primitives MLME-ASSOCIATE doivent s'exécuter comme à la Figure 24.

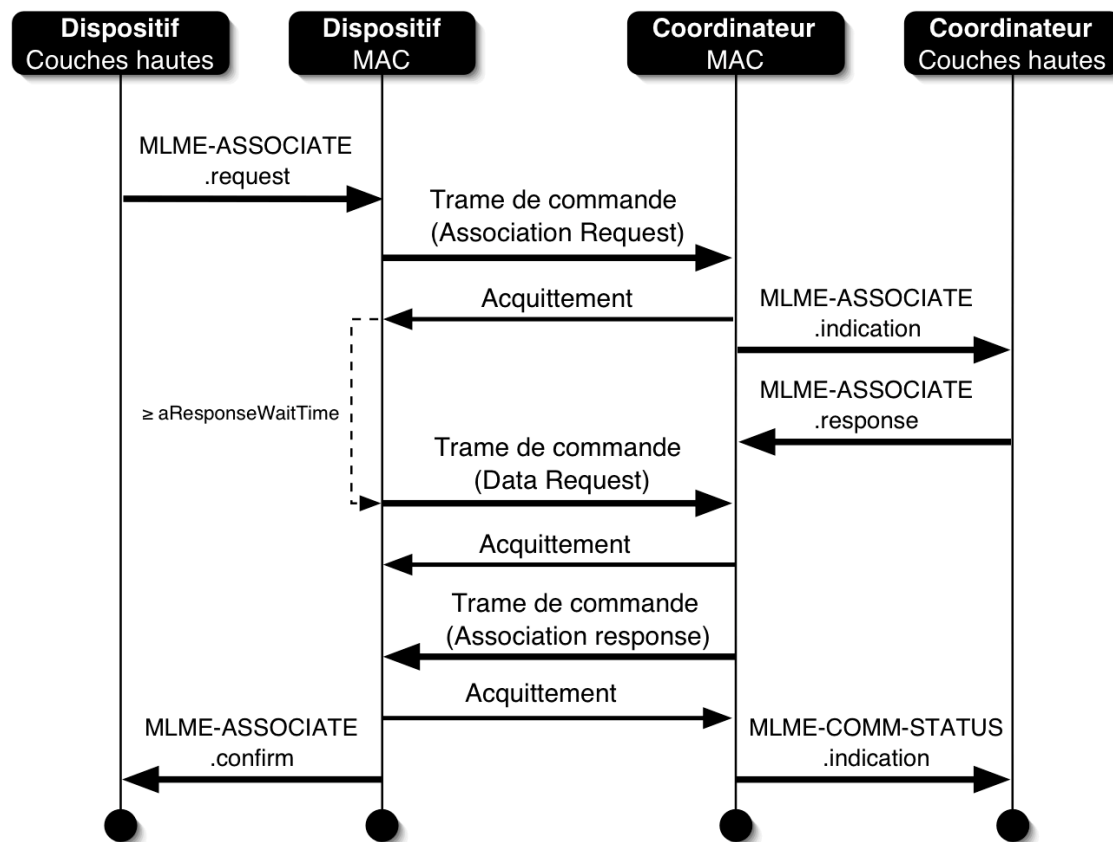


Figure 24 - Utilisation et séquençement des primitives MLME-ASSOCIATE

Il est à noter que la couche MAC du coordinateur réalise à la fin du processus un compte-rendu aux couches supérieures avec la primitive MLME-COMM-STATUS.indication. Cela permet notamment d'indiquer une éventuelle erreur qui s'est produite.

1.4.3.2 Description par les statuts

La primitive **MLME-ASSOCIATE.confirm** consiste à donner le résultat de la demande d'associations par la couche supérieure sous forme d'un statut. Voici les statuts possibles pour cette primitive :

SUCCESS : La trame de commandes « Association Request » a été reçue correctement par le coordinateur. Celui-ci envoie alors un acquittement au dispositif. À la réception de l'acquittement, la couche MAC du dispositif doit au moins attendre *aResponseWaitTime*. Puis c'est au dispositif de réaliser une récupération de données en attente chez le coordinateur (voir 2.4.5). Une fois effectué, la couche MAC du dispositif hérite d'une commande « Association Response ». La couche MAC doit encore acquitter cette trame de commande, puis peut confirmer à sa couche supérieure la réussite de l'association par le statut SUCCESS via la primitive MLME-ASSOCIATE.confirm.

CHANNEL_ACCESS_FAILURE : Si le CSMA-CA échoue durant l'accès au canal pour envoyer les trames de commande « Association Request » ou « Data Request ». Le statut CHANNEL_ACCESS_FAILURE est alors retourné.

NO_ACK : Après l'envoi des trames de commande, la couche MAC attend la venue d'acquittement durant le temps de la variable *macAckWaitDuration* (en nombre de symboles). Si durant ce temps, la couche MAC n'a pas reçu d'acquittement, elle renouvelle un certain nombre de fois la transmission et l'attente de l'acquittement en fonction de la constante *aMaxFrameRetries*. Si aucun ACK n'a été reçu par la couche MAC, le statut de MLME-ASSOCIATE.confirm est NO_ACK.

NO_DATA : Si durant le processus de récupération des données en attente chez le coordinateur, la couche MAC n'arrive pas récupérer la trame de commande « Association Response », le statut NO_DATA est renvoyé à la couche supérieure.

UNAVAILABLE_KEY : Si le paramètre « Security Enable » est actif, la couche MAC doit sécuriser les échanges. Si aucune clé appropriée n'est trouvée dans le ACL, le statut de MLME-ASSOCIATE.confirm est UNAVAILABLE_KEY.

FAILED_SECURITY_CHECK : Si une erreur intervient durant le processus de sécurisation d'une des trames de commande, le statut de MLME-ASSOCIATE.confirm est FAILED_SECURITY_CHECK.

INVALID_PARAMETER : Si un des paramètres de la primitive MLME-ASSOCIATE.request n'est pas du bon type ou est en dehors des plages autorisées, c'est le statut INVALID_PARAMETER qui est renvoyé avec MLME-ASSOCIATE.confirm.

1.4.4 MLME-DISASSOCIATE (request, confirm, indication)

MLME-DISASSOCIATE.request :

[DeviceAddress (device address), DisassociateReason (integer), SecurityEnable (boolean)]

De couche supérieure à MAC

MLME-DISASSOCIATE.confirm :

[Status (SUCCESS, TRANSACTION_OVERFLOW, TRANSACTION_EXPIRED, NO_ACK, CHANNEL_ACCESS_FAILURE, UNAVAILABLE_KEY, FAILED_SECURITY_CHECK or INVALID_PARAMETER)]

De MAC à couche supérieure

MLME-DISASSOCIATE.indication :

[DeviceAddress (device address), DisassociateReason (integer), SecurityUse (boolean), ACLEntry (integer)]

De MAC à couche supérieure

1.4.4.1 Fonctionnement et utilisation

La primitive de demande de dissociation **MLME-DISASSOCIATE.request** peut soit être utilisée par un dispositif associé qui veut se dissocier du PAN ou soit par le coordinateur du PAN qui veut dissocier un dispositif déjà associé au PAN. C'est le paramètre *DisassociationReason* qui différencie ces 2 possibilités :

- 0x01 : Le coordinateur veut qu'un dispositif quitte le PAN.
- 0x02 : Un dispositif veut quitter le PAN.

Un autre paramètre important transmis par la couche supérieure avec la primitive **MLME-DISASSOCIATE.request** est le paramètre *DeviceAddress*. Il correspond à l'adresse 64 bits où il faut envoyer la commande de dissociation (Disassociation Notification). Si ce paramètre est égal à *macCoordExtendedAddress*, alors c'est un dispositif qui envoie la commande de dissociation. S'il est d'une autre valeur, le coordinateur place la commande dans sa liste d'attente de transmissions (transmission indirecte).

La primitive **DISASSOCIATE.indication** sert à indiquer à la couche supérieure du dispositif ou du coordinateur qui reçoit la trame de commande « Disassociation Notification » qu'une dissociation a été réalisée.

L'utilisation et le séquençement général des trois primitives MLME-DISASSOCIATE doivent s'exécuter comme le montre le diagramme en flèche de la Figure 25.

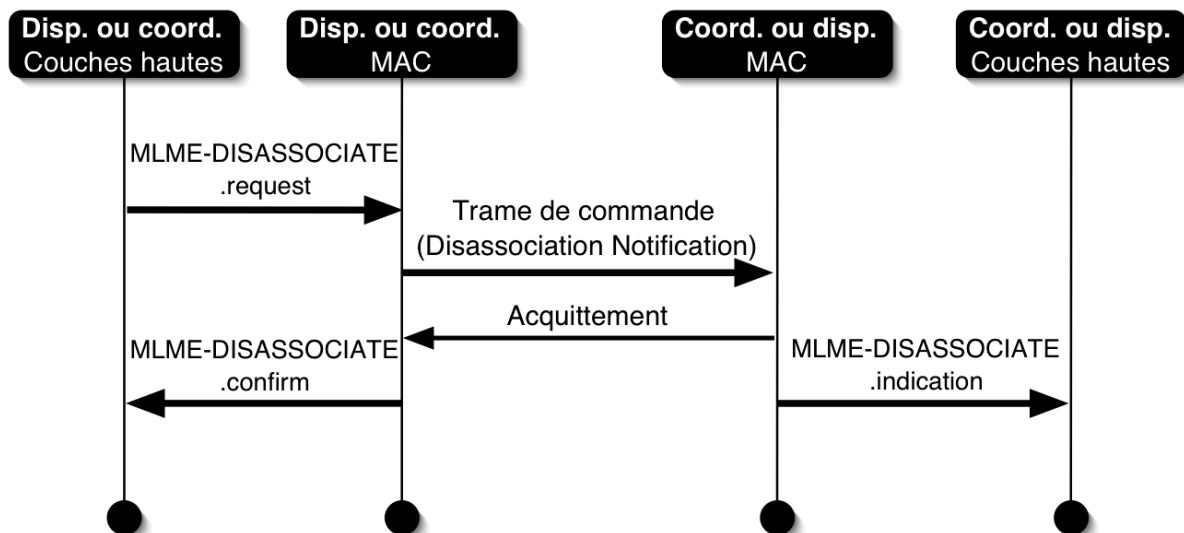


Figure 25 - Utilisation et séquençement des primitives MLME-DISASSOCIATE

1.4.4.2 Description par les statuts

La primitive **MLME-DISASSOCIATE.confirm** sert à donner le résultat de la demande de dissociation par la couche supérieure sous forme d'un statut. Voici les statuts possibles pour cette primitive :

SUCCESS : La trame de commande « Disassociation Notification » a été reçue correctement par le coordinateur ou le dispositif a pu récupérer cette commande chez le coordinateur. Dans les deux cas, la trame de commande est acquittée et l'on considère alors que la dissociation est réussie en envoyant le statut SUCCESS à la couche supérieure (du dispositif ou du coordinateur qui a fait la demande de dissociation).

TRANSACTION_OVERFLOW : Si le dispositif est un coordinateur PAN, l'information contenue dans la primitive va être placée dans la liste d'attente des transactions (transmission indirecte). S'il n'a plus la capacité de garder cette transaction, le statut TRANSACTION_OVERFLOW est retourné.

TRANSACTION_EXPIRED : Si la trame de commande placée dans la liste de transmissions du coordinateur n'est pas récupérée avant le temps *macTransactionPersistenceTime* par le dispositif, la trame de commande est supprimée de la liste et le statut TRANSACTION_EXPIRED est retourné.

NO_ACK : Après l'envoi de la trame de commande, la couche MAC attend la venue d'acquiescement durant le temps de la variable *macAckWaitDuration* (en nombre de symboles). Si durant ce temps, la couche MAC n'a pas reçu d'acquiescement, elle renouvelle un certain nombre de fois la transmission et l'attente de l'acquiescement en fonction de la constante *aMaxFrameRetries*. Si aucun ACK n'a été reçu par la couche MAC, le statut de MLME-DISASSOCIATE.confirm est NO_ACK.

CHANNEL_ACCESS_FAILURE : Si le CSMA-CA échoue durant l'accès au canal pour envoyer la trame de commande « Disassociation Notification » dans le cas où c'est le dispositif qui veut se dissocier, le statut CHANNEL_ACCESS_FAILURE est retourné à la couche supérieure.

UNAVAILABLE_KEY : Si le paramètre Security Enable est actif, la couche MAC doit sécuriser les échanges. Si aucune clé appropriée n'est trouvée dans le ACL, le statut de MLME-DISASSOCIATE.confirm est UNAVAILABLE_KEY.

FAILED_SECURITY_CHECK : Si une erreur intervient durant le processus de sécurisation de la trame, le MSDU est supprimé et le statut de MLME-DISASSOCIATE.confirm est FAILED_SECURITY_CHECK.

INVALID_PARAMETER : Si un des paramètres de la primitive MLME-DISASSOCIATE.request n'est pas du bon type ou est en dehors des plages autorisées, c'est le statut INVALID_PARAMETER qui est renvoyé avec MLME-DISASSOCIATE.confirm.

1.4.5 MLME-BEACON-NOTIFY.indication

MLME-BEACON-NOTIFY.indication :

[BSN (integer), PANDescriptor (enumeration), PendAddrSpec (bitmap), AddrList (list of device address), sduLength (integer), sdu (set of octets)]

De MAC à couche supérieure

1.4.5.1 Fonctionnement

Cette primitive **MLME-BEACON-NOTIFY.indication** permet d'indiquer à la couche supérieure d'un dispositif que les informations contenues dans une trame Beacon viennent d'être reçues. La majorité des informations est passée par la structure du PAN Descriptor qui est décrit au Tableau 7.

Nom	Type	Espace valide	Description
CoordAddrMode	integer	0x02-0x03	Mode d'adressage du coordinateur. 0x02 = 16 bits (adresse courte) 0x03 = 64 bits (adresse étendue)
CoordPANId	integer	0x0000-0xFFFF	Le numéro du PAN qu'utilise le coordinateur.
CoordAddress	device address	Dépend de CoordAddrMode	Adresse du coordinateur.
LogicalChannel	integer	Canal logique supporté par le la couche PHY.	Canal courant de transmission utilisé par le réseau.
SuperframeSpec	integer	Voir 1.3.2.2	Spécification de la Superframe. Correspond à un champ de la trame Beacon.
GTSPermit	boolean	TRUE (1) ou FALSE (0)	Autorisation qu'un coordinateur puisse faire des réservations de temps (GTS).
LinkQuality	integer	0x00 - 0xFF	Qualité du lien par lequel le Beacon a été reçu.
TimeStamp	integer	0x000000 - 0xFFFFFFFF (32 bits)	Temps en symbole lorsque le Beacon a été reçu.

SecurityUse	boolean	TRUE (1) ou FALSE (0)	TRUE s'il y a une erreur durant le processus de la sécurisation de la trame.
ACLEntry	integer	0x00-0x08	Entrée d'accès dans la liste de contrôle (0x08 si l'expéditeur de la trame n'est pas présent dans le ACL).
SecurityFailure	boolean	TRUE (1) ou FALSE (0)	Adresse étendue (64 bits) du coordinateur avec lequel le dispositif est associé.

Tableau 7 - PAN Descriptor

1.4.6 MLME-GET (request, confirm)

MLME-GET.request :

[PIBAttribute (enumeration)]

De couche supérieure à MAC

MLME-GET.confirm :

[Status (SUCCESS or UNSUPPORTED_ATTRIBUTE), PIBAttribute (enumeration),
PIBAttributeValue (various)]

De MAC à couche supérieure

1.4.6.1 Fonctionnement

La primitive **MLME-GET.request** permet à la couche supérieure de lire un attribut appartenant au PAN Information Base (PIB) de la couche MAC. La couche supérieure utilise le paramètre *PIBAttribute* pour spécifier l'attribut du PIB qui doit être lu. La primitive **MLME-GET.confirm** renvoie alors la valeur du paramètre demandé via le paramètre *PIBAttributeValue*.

1.4.6.2 Description par les statuts

La primitive **MLME-GET.confirm** retourne le statut correspondant à la demande de lecture d'un attribut PIB. Voici les statuts possibles :

SUCCESS : La valeur de l'attribut demandé a été correctement lue et placée dans le paramètre *PIBAttribute*.

UNSUPPORTED_ATTRIBUTE : L'attribut demandé n'a pas été trouvé dans le PIB.

1.4.7 MLME-GTS (request, confirm, indication)

MLME-GTS.request :

[GTSCharacteristics (enumeration), SecurityEnable (boolean)]

De couche supérieure à MAC (FFD seulement)

MLME-GTS.confirm :

[GTSCharacteristics (enumeration), Status (SUCCESS, DENIED, NO_SHORT_ADDRESS, CHANNEL_ACCESS_FAILURE, NO_ACK, NO_DATA, UNAVAILABLE_KEY, FAILED_SECURITY_CHECK or INVALID_PARAMETER)]

De MAC à couche supérieure (FFD seulement)

MLME-GTS.indication :

[DevAddress (device address), GTSCharacteristics (enumeration), SecurityUse (boolean), ACLEntry (integer)]

De MAC à couche supérieure (FFD seulement)

1.4.7.1 Fonctionnement et utilisation

Ces primitives permettent la mise en place d'une gestion de réservation de temps. La primitive **MLME-GTS.request** permet à un dispositif associé à un coordinateur PAN de faire une demande de réservation de temps par une allocation d'un nouveau GTS ou d'annuler une réservation de temps existante en arrêtant d'allouer un GTS.

Toutes les informations concernant l'allocation d'un GTS sont présentes dans le paramètre *GTSCharacteristics* qui est transmis par la couche supérieure. Ce paramètre correspond directement au champ « GTS Characteristics » présent dans les trames Beacons. Il se compose dans l'ordre du champ « GTS length » (4 bits), « GTS direction » (1 bit) et « Characteristics Type » (1 bit).

Si le « Characteristics Type » vaut la valeur 1, c'est un nouveau GTS qui est alloué, sinon c'est un GTS existant qui est arrêté d'être alloué. Le « GTS direction » permet de choisir si on alloue un temps durant lequel des données sont transmises (à 0) ou reçues (à 1).

Plusieurs cas sont alors possibles, le plus courant est celui d'un dispositif faisant une demande de GTS. Il doit alors faire parvenir au coordinateur une trame de commande « GTS Request ». Le coordinateur acquitte la trame et indique à sa couche supérieure la demande de GTS avec la primitive **MLME-GTS.indication**. Puis, la couche MAC du dispositif attend la venue du prochain Beacon broadcast pour savoir si sa demande de GTS a réussi. Il peut attendre jusqu'à *aGTSDescPersistenceTime* nombre de beacon.

Un autre cas stipule que le dispositif fasse une demande d'arrêter l'allocation d'un GTS. Le dispositif n'a pas besoin de lire le prochain Beacon pour confirmer l'arrêt de l'allocation, l'acquittement du coordinateur suffit.

Sinon, un troisième cas est possible : c'est le coordinateur qui décide si une allocation de GTS doit être arrêtée. Dans ce cas-là, c'est dans un Beacon qui est envoyé indiquant que le GTS du dispositif est supprimé du « GTS Descriptor ». Le dispositif s'en rend alors compte et renvoie à sa couche supérieure la primitive **MLME-GTS.indication**.

Le résumé de l'utilisation et le séquençement de ces primitives MLME-GTS sont à la Figure 26. Le cas illustré correspond à l'allocation d'un GTS par un dispositif.

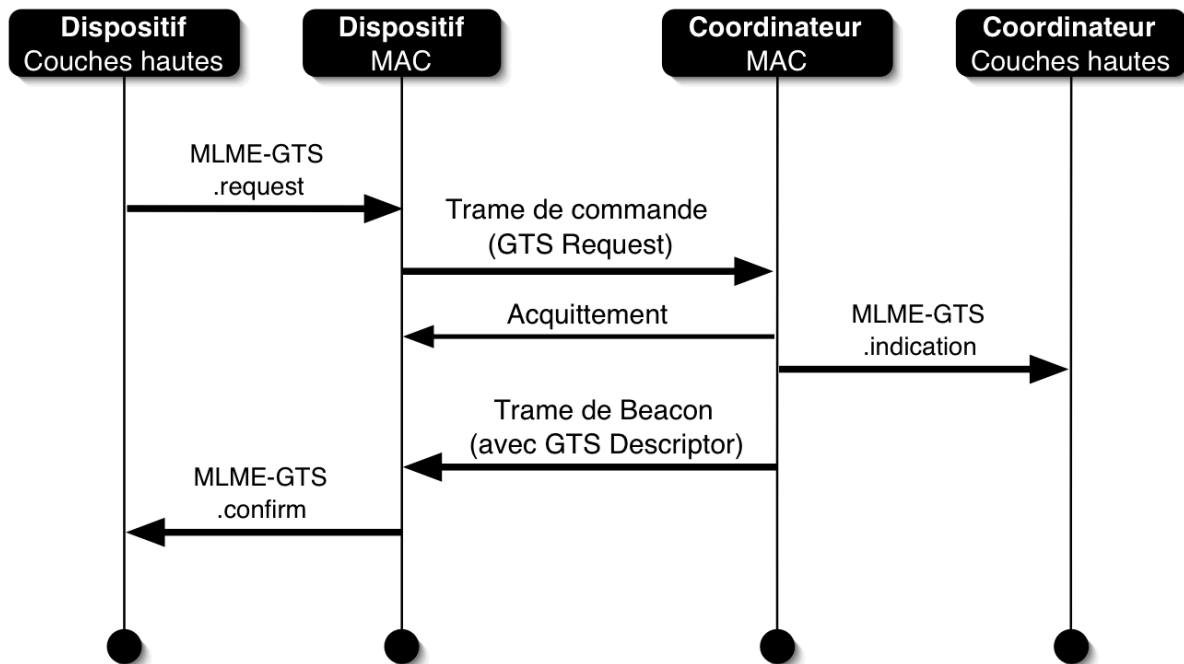


Figure 26 - Utilisation et séquençement des primitives MLME-GTS (cas allocation)

1.4.7.2 Description par les statuts

La primitive **MLME-GTS.confirm** sert à donner le résultat de la demande d'allocation ou l'arrêt d'allocation d'un GTS à la couche supérieure sous forme d'un statut. Voici les statuts possibles pour cette primitive :

SUCCESS : La trame de commande « GTS Request » a été reçue correctement par le coordinateur. La trame de commande a été acquittée. Puis si la lecture du « GTS Descriptor » du Beacon reçue avec l'adresse *macShortAddress* confirme que la même allocation de GTS que celle demandée a été faite, la couche MAC du dispositif peut envoyer le statut SUCCESS à sa couche supérieure. Le Beacon doit juste être reçu avant le temps *aGTSDescPersistenceTime* qui correspond à 4 beacons. Le paramètre SUCCESS est également retourné si la couche MAC du dispositif reçoit un acquittement du coordinateur après sa demande d'arrêter d'allouer un GTS.

DENIED : Si dans le Beacon reçu, le GTS Descriptor indique que le slot de départ du GTS est le zéro, le coordinateur a refusé l'allocation de GTS. Le statut DENIED est donc retourné à la couche supérieure.

NO_SHORT_ADDRESS : Si la variable *macShortAddress* vaut 0xffff ou 0xffffe, le dispositif n'a pas le droit de réaliser des demandes de GTS, c'est donc le statut NO_SHORT_ADDRESS qui est renvoyé.

CHANNEL_ACCESS_FAILURE : Si le CSMA-CA échoue durant l'accès au canal pour envoyer la trame de commande « GTS Request », le statut CHANNEL_ACCESS_FAILURE est retourné à la couche supérieure.

NO_ACK : Après l'envoi de la trame de commande « GTS Request », la couche MAC attend la venue d'un acquittement durant le temps de la variable *macAckWaitDuration* (en nombre de symbole). Si durant ce temps, la couche MAC n'a pas reçu d'acquittement, elle renouvelle un certain nombre de fois la transmission et l'attente de l'acquittement en fonction de la constante *aMaxFrameRetries*. Si aucun ACK n'a été reçu par la couche MAC, le statut de MLME-GTS.confirm est NO_ACK.

NO_DATA : Si le Beacon n'est pas reçu durant le temps *aGTSDescPersistenceTime*, alors c'est le statut NO_DATA qui est renvoyé à la couche supérieure du dispositif.

UNAVAILABLE_KEY : Si le paramètre *Security Enable* est actif, la couche MAC doit sécuriser la trame de commande « GTS Request ». Si aucune clé appropriée n'est trouvée dans le ACL, le statut de MLME-GTS.confirm est UNAVAILABLE_KEY.

FAILED_SECURITY_CHECK : Si une erreur intervient durant le processus de sécurisation de la trame, le MSDU est supprimé et le statut de MLME-DISASSOCIATE.confirm est FAILED_SECURITY_CHECK.

INVALID_PARAMATER : Si un des paramètres de la primitive MLME-GTS.request n'est pas du bon type ou est en dehors des plages autorisées, c'est le statut INVALID_PARAMETER qui est renvoyé avec MLME-GTS.confirm.

1.4.8 MLME-ORPHAN (indication, response)

MLME-ORPHAN.indication :

[OrphanAddress (device address), SecurityUse (boolean), ACLEntry (integer)]

De MAC à couche supérieure (FFD seulement)

MLME-ORPHAN.response :

[OrphanAddress (device address), ShortAddress (integer), AssociatedMember (boolean), SecurityEnable (boolean)]

De couche supérieure à MAC (FFD seulement)

1.4.8.1 Fonctionnement et utilisation

La primitive **MLME-ORPHAN.indication** permet à la couche MAC du coordinateur d'indiquer la présence d'un dispositif orphelin à sa couche supérieure. Cette primitive est envoyée lorsque la couche MAC reçoit la commande « Orphan Notification ».

La couche supérieure répond avec la primitive **MLME-ORPHAN.response** notamment en indiquant si le dispositif orphelin était un ancien membre du PAN avec le paramètre *AssociatedMember*. Si c'est le cas, la couche supérieure retourne également l'adresse courte du dispositif (*ShortAddress*) qui lui avait été allouée. Le paramètre *ShortAddress* peut alors valoir 0xFFFE si le dispositif doit utiliser uniquement son adresse étendue de 64 bits. La couche MAC du coordinateur doit envoyer une trame de commande « Coordinator Realignment ».

Si le dispositif n'appartenait pas au PAN, la valeur du paramètre *ShortAddress* est de 0xffff. La couche MAC du coordinateur va alors ignorer cette primitive de réponse. Si le dispositif qui a envoyé la commande « Orphan Notification » ne reçoit pas de trame de commande « Coordinator Realignment » après le temps *aResponseWaitTime*, il considère qu'il n'est plus associé avec de coordinateur.

Dans le cas où le dispositif est réassocié, la couche MAC du coordinateur doit encore faire un contre-rendu de ce qui s'est passé à la couche supérieure à l'aide de la primitive *MLME-COMM-STATUS.indication*.

Le résumé de l'utilisation et le séquençement de ces primitives *MLME-ORPHAN* sont à la Figure 27.

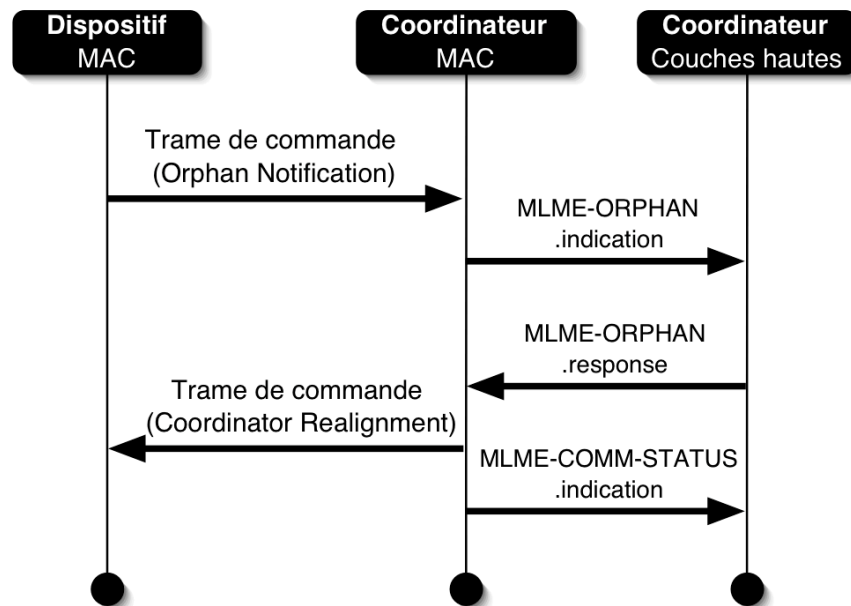


Figure 27 - Utilisation et séquençement des primitives *MLME-ORPHAN*

1.4.9 MLME-RESET (request, confirm)

MLME-RESET.request :

[SetDefaultPIB (boolean)]

De couche supérieure à MAC

MLME-RESET.confirm :

[Status (SUCCESS or DISABLE_TRX_FAILURE)]

De MAC à couche supérieure

1.4.9.1 Fonctionnement

La primitive **MLME-RESET.request** permet à la couche supérieure de faire une demande de réinitialisation de la couche MAC. C'est le paramètre *SetDefaultPIB* qui spécifie comment doit être réalisé la réinitialisation. À TRUE, la couche MAC est réinitialisée et toutes les valeurs PIB par défaut sont réinsérées. À FALSE, les valeurs PIB actuelles sont conservées, mais la couche MAC est tout de même remplacée à son état initial. Pour retrouver son état initial, le Transceiver doit notamment être arrêté en utilisant la primitive physique PLME-SET-TRX-STATE.

1.4.9.2 Description par les statuts

La primitive **MLME-RESET.confirm** peut retourner les deux statuts suivants :

SUCCESS : La couche MAC a été initialisée et le PLME-SET-TRX-STATE a été mis à TRX_OFF.

DISABLE_TRX_FAILURE: La tentative d'arrêt du Transceiver a échoué.

1.4.10 MLME-RX-ENABLE (request, confirm)

MLME-RX-ENABLE.request :

[DeferPermit (boolean), RxOnTime (integer), RxOnDuration (integer)]

De couche supérieure à MAC

MLME-RX-ENABLE.confirm :

[Status (SUCCESS, TX_ACTIVE, OUT_OF_CAP or INVALID_PARAMETER)]

De MAC à couche supérieure

1.4.10.1 Fonctionnement

La primitive **MLME-RX-ENABLE.request** permet à la couche supérieure de rendre actif ou inactif le récepteur pendant un certain laps de temps.

Pour un réseau « Beacon Enabled », le paramètre *DeferPermit* indique s'il est possible (TRUE) de reporter l'activation du récepteur à la prochaine Superframe lorsque le temps demandé par le paramètre *RxOnDuration* ne peut plus être tenu. Sinon (FALSE), l'activation du récepteur est de toute façon réalisée dans la Superframe courante.

Le paramètre *RxOnTime* permet quant à lui de décider quand depuis le début de la Superframe le récepteur doit être activé.

Dans un réseau « Non Beacon Enabled » les paramètres *DeferPermit* et *RxOnTime* sont ignorés et le récepteur est activé immédiatement pendant le temps indiqué par *RxOnDuration*.

Si le paramètre *RxOnDuration* vaut la valeur 0, le récepteur est rendu inactif.

1.4.10.2 Description par les statuts

La primitive **MLME-RX-ENABLE.confirm** sert à donner le résultat de la demande d'activation du récepteur. Voici les statuts possibles pour cette primitive :

SUCCESS : La couche MAC fait une demande d'activation du récepteur via la primitive physique PLME-SET-TRX-STATE.request avec le paramètre RX_ON. Si la primitive de confirmation PLME-SET-TRX-STATE.confirm retourne la valeur RX_ON ou SUCCESS, la primitive MAC MLME-RX-ENABLE.confirm retourne le statut SUCCESS.

TX_ACTIVE : Si la primitive physique PLME-SET-TRX-STATE.confirm délivre le statut TX_ON, la primitive MAC MLME-RX-ENABLE.confirm retourne le statut TX_ACTIVE.

OUT_OF_CAP : lorsque le paramètre *DeferPermit* est à FALSE et que la Superframe est déjà écoulée depuis plus de $(RxOnTime - aTurnaround)$, c'est le statut OUT_OF_CAP qui est retourné. Ce statut est également retourné lorsque $(RxOnTime - aTurnaround)$ est plus grand que la zone de contention (CAP).

INVALID_PARAMETER : Si en additionnant les deux paramètres *RxOnTime* et *RxOnDuration*, le résultat est plus grand que l'intervalle entre deux Beacons (BI), le statut INVALID_PARAMETER est retourné.

1.4.11 MLME-SCAN (request, confirm)

MLME-SCAN.request :

[ScanType (integer), ScanChannels (bitmap), ScanDuration (integer)]

De couche supérieure à MAC

MLME-SCAN.confirm :

[Status (SUCCESS, NO_BEACON or INVALID_PARAMETER), ScanType (integer),
UncannedChannels (bitmap), ResultListSize (integer), EnergyDetectList (list of integers),
PANDescriptorList (enumeration)]

De MAC à couche supérieure

1.4.11.1 Fonctionnement

La primitive **MLME-SCAN.request** sert à réaliser un balayage (scan) du POS afin de faire des mesures d'énergie et de déterminer si des PAN sont présents ou pas. Le paramètre *ScanChannels* permet notamment de spécifier les canaux à scanner. On utilise les 27 bits de poids faibles d'un type de 32 bits pour indiquer si le canal doit être scanné ou non (1 = à scanner, 0 = ne pas scanner).

Le balayage peut être de 4 types différents. La couche supérieure l'indique via le paramètre *ScanType* :

- 0x00: Détection d'énergie (FFD seulement)
- 0x01 : Scan actif (FFD seulement)
- 0x02 : Scan passif
- 0x03 : Scan d'orphelin

La **détection d'énergie** permet de déterminer le taux d'utilisation des canaux. La couche MAC utilise la primitive PLME-ED.request durant le temps ($aBaseSuperframeDuration * 2^n - 1$) pour chaque canal. La variable n correspondant au paramètre *ScanDuration*. Chaque canal est associé à un maximum d'énergie mesurée.

Les **balayages actif et passif** permettent de localiser des Beacons afin de lister les PAN existants et de pouvoir s'associer avec l'un deux.

Le **balayage actif** consiste à envoyer une trame de commande « Beacon Request » sur chaque canal. La couche MAC stocke alors une liste de PAN Descriptor déterminée par la lecture des Beacons reçus.

La différence avec un **balayage passif** est que l'on ne transmet pas de commande « Beacon Request », le transmetteur n'étant pas utilisé. Le principe est juste d'écouter sur les canaux pendant ($aBaseSuperframeDuration * 2^n - 1$) afin de détecter des Beacons et d'en regrouper les informations dans une liste comme dans le cas du balayage actif.

Le **balayage d'orphelin** sert à localiser le PAN duquel le dispositif a perdu la communication. Le principe est d'envoyer une trame de commande « Orphan Notification » sur tous les canaux indiqués par le paramètre *ScanChannel*. L'attente maximale après l'envoi de la trame de commande est de *aResponseWaitTime*. Le balayage prend fin si une trame de commande « Coordinator Realignment » est reçue ou si tous les canaux ont été parcourus sans succès.

Durant n'importe quel balayage, la couche MAC doit arrêter toute transmission de Beacon, de plus elle accepte de la couche physique seulement les résultats provenant du balayage.

C'est la primitive MLME-SCAN.confirm qui sert à transmettre les résultats à la couche supérieure par plusieurs paramètres sous forme de listes dépendantes du type de balayage.

1.4.11.2 Description par les statuts

La primitive **MLME-SCAN.confirm** sert à donner le résultat de la demande de balayage du POS. Voici les statuts possibles pour cette primitive :

SUCCESS : Le type de balayage réalisé a correspondu au paramètre *ScanType* et les canaux spécifiés par le paramètre *ScanChannels* ont tous été scannés durant le temps défini à partir du paramètre *ScanDuration*. Dans les cas d'un balayage actif ou passif, il peut finir si le nombre de PAN détectés a atteint le maximum prévu (7).

NO_BEACON : Lors d'un balayage actif ou passif, si aucun Beacon n'est trouvé à travers les différents canaux, le statut NO_BEACON est retourné.

INVALID_PARAMETER : Si un des paramètres de la primitive MLME-SCAN.request n'est pas du bon type ou est en dehors des plages autorisées, c'est le statut INVALID_PARAMETER qui est renvoyé.

1.4.12 MLME-COMM-STATUS.indication

MLME-COMM-STATUS.indication :

[PANid (integer), SrcAddrMode (integer), SrcAddr(device address), DstAddrMode (integer), DstAddr (device address), Status (SUCCESS, TRANSACTION_OVERFLOW, TRANSACTION_EXPIRED, CHANNEL_ACCESS_FAILURE, NO_ACK, UNAVAILABLE_KEY, FRAME_TOO_LONG, FAILED_SECURITY_CHECK or INVALID_PARAMETER)]

De MAC à couche supérieure

1.4.12.1 Fonctionnement

La primitive **MLME-COMM-STATUS.indication** est utilisée pour communiquer à la couche supérieure les informations à propos de la communication qui a eu lieu. Cette primitive est utilisée lorsqu'un échange d'informations a été initié sans l'aide d'une primitive de type « Request ». La primitive **MLME-COMM-STATUS.indication** remplace en quelque sorte les primitives de type « Confirm ». Elle est utilisée après les primitives MLME-ASSOCIATE.response et MLME-ORPHAN.response.

La primitive **MLME-COMM-STATUS.indication** peut retourner les statuts rencontrés pour les primitives de type « Confirm ».

1.4.13 MLME-SET (request, confirm)

MLME-SET.request :

[PIBAttribute (enumeration), PIBAttributeValue (various)]

De couche supérieure à MAC

MLME-SET.confirm :

[Status (SUCCESS, UNSUPPORTED_ATTRIBUTE, INVALID_PARAMETER), PIBAttribute (enumeration)]

De MAC à couche supérieure

1.4.13.1 Fonctionnement

La primitive **MLME-SET.request** permet à la couche supérieure d'écrire un attribut appartenant au PAN Information Base (PIB) de la couche MAC. La couche supérieure utilise le paramètre *PIBAttribute* pour spécifier l'attribut du PIB qui doit être lu et le paramètre *PIBAttributeValue* pour donner la valeur d'écriture.

1.4.13.2 Description par les statuts

La primitive **MLME-SET.confirm** retourne le statut correspondant à la demande d'écriture d'un attribut PIB. Voici les statuts possibles :

SUCCESS : La valeur de l'attribut demandé a été correctement remplacée par la valeur du paramètre *PIBAttributeValue*.

UNSUPPORTED_ATTRIBUTE : L'attribut qui doit être écrit (paramètre *PIBAttribute*) n'a pas été trouvé dans le PIB.

INVALID_PARAMETER : Le paramètre *PIBAttributeValue* de la primitive MLME-SET.request n'est pas du bon type ou est en dehors des plages autorisées, c'est donc le statut INVALID_PARAMETER qui est renvoyé.

1.4.14 MLME-START (request, confirm)

MLME-START.request :

[PANid (integer), LogicalChannel (integer), BeaconOrder (integer), SuperframeOrder (integer),
PANCoordinator(boolean), BatteryLifeExtension (boolean), CoordRealignment (boolean),
SecurityEnable (boolean)]

De couche supérieure à MAC (FFD seulement)

MLME-START.confirm :

[Status (SUCCESS, NO_SHORT_ADDRESS, UNAVAILABLE_KEY, FRAME_TOO_LONG,
FAILED_SECURITY_CHECK or INVALID_PARAMETER)]

De MAC à couche supérieure (FFD seulement)

1.4.14.1 Fonctionnement et utilisation

La primitive **MLME-START.request** permet d'initialiser le début d'une nouvelle Superframe avec l'envoi de Beacons (« Beacon Enabled »). Les différents paramètres reçus par la primitive MLME-START.request permettent principalement de mettre à jour les valeurs PIB correspondantes. Si le paramètre *BeaconOrder* vaut 15, le PAN devient alors « Non Beacon Enabled ».

Si le paramètre *PANCoordinator* est à TRUE, la couche MAC met à jour *macPANId* et *phyCurrentChannel* avec les paramètres correspondants. À FALSE, ces paramètres sont ignorés. Si le paramètre *CoordRealignment* est à TRUE, la couche MAC envoie par broadcast une commande de « Coordinator Realignment » contenant le nouveau PANid et le numéro du canal logique à utiliser.

Dans le cas où le dispositif envoyait déjà des Beacons, la nouvelle configuration de la Superframe prend effet dès le prochain Beacon programmé. Sinon, la nouvelle configuration prend place immédiatement.

Pour transmettre des Beacons, il faut mettre le transmetteur en marche (TX_ON) à l'aide de la primitive physique PLME-SET-TRX-STATE.request. Le statut de retour doit être TX_ON ou SUCCESS. Le premier Beacon est alors envoyé à l'aide de PD-DATA.request. Si la partie active de la Superframe est encore présente après l'envoi de la trame Beacon, la couche MAC donne l'ordre de se mettre en mode réception (RX_ON), sinon le récepteur n'est pas activé.

La Figure 28 montre comment doivent être utilisées les primitives MLME-START.

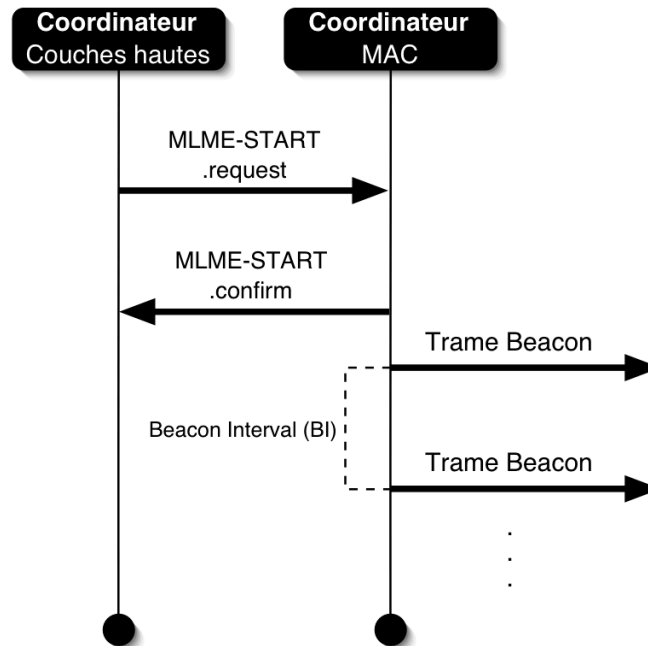


Figure 28 - Utilisation et séquençement des primitives MLME-START

1.4.14.2 Description par les statuts

La primitive **MLME-START.confirm** sert à donner le résultat de la demande de la nouvelle configuration de Superframe. Voici les statuts possibles pour cette primitive :

SUCCESS : Si la nouvelle configuration de Superframe a pu être réalisée correctement, alors c'est le statut SUCCESS qui est retourné via la primitive MLME-START.confirm.

NO_SHORT_ADDRESS: Ce statut est retourné lorsque le dispositif n'est pas associé à un PAN (*macShortAddress* = 0xffff).

UNAVAILABLE_KEY: Si dans la liste ACL, il n'y a pas de clé appropriée, ce statut est retourné.

FAILED_SECURITY_CHECK: Ce statut est renvoyé lorsqu'une erreur survient lors de la sécurisation de données de la trame.

FRAME_TOO_LONG: Si la longueur de la trame résultante est plus grande que *aMaxMACFrameSize*, ce résultat est renvoyé.

INVALID_PARAMETER: Si n'importe quel paramètre est en dehors de sa plage de données supportées, ce statut est retourné.

1.4.15 MLME-SYNC.request

MLME-SYNC.request :

[LogicalChannel (integer), TrackBeacon (boolean)]

De couche supérieure à MAC

1.4.15.1 Fonctionnement

Cette primitive **MLME-SYNC.request** sert à se synchroniser avec le coordinateur PAN via les Beacons qu'il génère périodiquement.

Deux types de synchronisation sont possibles, via le paramètre *TrackBeacon*. Si ce paramètre est à TRUE, la synchronisation se fait sur tous les Beacons qui surviennent. Si ce paramètre est à FALSE, la synchronisation se fait uniquement par la recherche du prochain Beacon. Dans ce mode, c'est donc la couche supérieure qui va dicter la lecture des Beacons. Si le dispositif détecte sa propre adresse dans un des Beacons, il peut alors procéder à une récupération de données en attente chez le coordinateur (voir 2.4.5).

Lorsque la couche MAC doit percevoir tous les Beacons, elle doit activer son récepteur juste avant le temps prévu de la venue du Beacon. C'est en ayant réalisé un balayage préalable (obligatoire) que l'on peut calculer l'intervalle entre les Beacons à l'aide d'un timer.

1.4.16 MLME-SYNC-LOSS.indication

MLME-SYNC-LOSS.indication :

[LossReason (PAN_ID_CONFLICT, REALIGNEMENT or BEACON_LOST)]

De MAC à couche supérieure

1.4.16.1 Fonctionnement

La primitive sert à indiquer à la couche supérieure la perte de synchronisation avec le coordinateur.

1.4.16.2 Description par les statuts

Cette primitive permet d'indiquer par trois raisons la perte de synchronisation :

PAN_ID_CONFLICT : Lorsqu'un dispositif a détecté un conflit au niveau du numéro du PAN ou qu'un coordinateur reçoive la commande « Conflict Notification », le statut retourné est PAN_ID_CONFLICT.

REALIGNEMENT : Si le dispositif a reçu une commande « Coordinator Realignment » en provenance du coordinateur avec lequel il est associé et que le dispositif n'est pas en train d'effectuer un balayage d'orphelin, alors le paramètre retourné dû à la perte de synchronisation est REALIGNMENT.

BEACON_LOST : En considérant que la primitive MLME-SYNC.request a été utilisée avant, il se peut que la recherche du premier Beacon échoue ou que des Beacons ne sont plus reçus durant les temps prévus, c'est alors la primitive MLME-SYNC-LOSS.indication qui est envoyée à la couche supérieure avec comme argument de la perte de synchronisation, BEACON_LOST.

1.4.17 MLME-POLL (request, confirm)

MLME-POLL.request :

[CoordAddrMode (integer), CoordPANid (integer), CoordAddress (device address), SecurityEnable (boolean)]

De couche supérieure à MAC

MLME-POLL.confirm :

[Status (SUCCESS, CHANNEL_ACCESS_FAILURE, NO_ACK, NO_DATA, UNAVAILABLE_KEY, FAILED_SECURITY_CHECK or INVALID_PARAMETER)]

De MAC à couche supérieure

1.4.17.1 Fonctionnement et utilisation

La primitive **MLME-POLL.request** permet de réaliser une demande de données chez un coordinateur. Pour cela, la couche MAC doit envoyer une trame de commande « Data Request » au coordinateur. Si la destination de la commande est le PAN coordinateur, aucun paramètre d'adresse de destination n'est présent. Sinon, les paramètres de destinations doivent correspondre à *CoordPANId* et *CoordAddress*.

Le coordinateur répond à la commande par un acquittement. Si le champ « Frame Pending » est actif, cela signifie que le dispositif possède des données en attente chez le coordinateur. Il s'attend donc à recevoir des données de la part du coordinateur. Une fois les données reçues, elles peuvent être acquittées, puis la primitive **MLME-POLL.confirm** est retournée à la couche supérieure suivi de la primitive MAC de donnée MCPS-DATA.indication pour transmettre les données.

L'utilisation des primitive MLME-POLL à la Figure 29 correspond au cas où des données en attente étaient présentes chez le coordinateur.

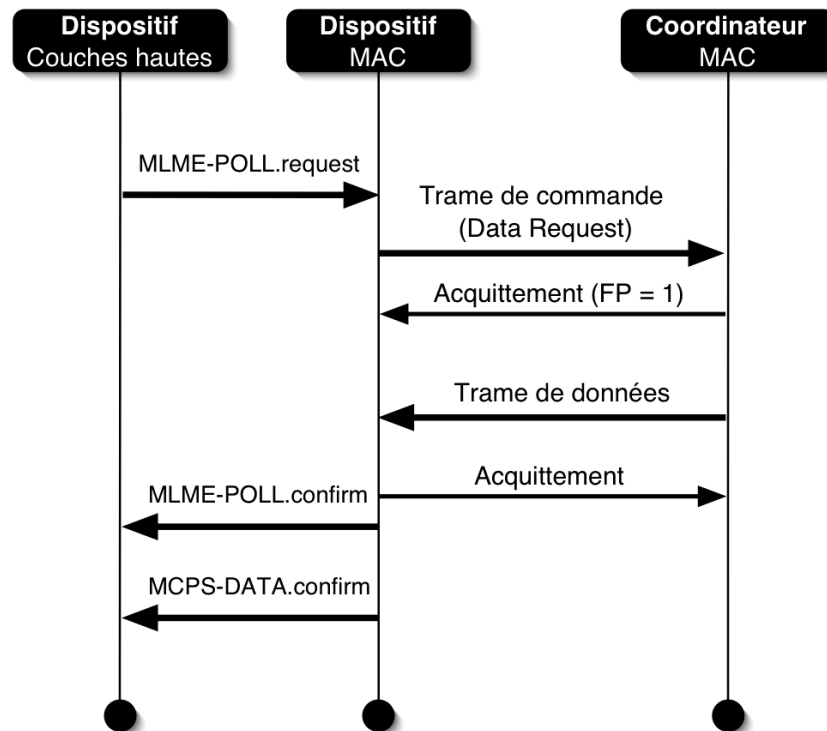


Figure 29 - Utilisation et séquençement des primitives MLME-POLL

1.4.17.2 Description par les statuts

La primitive **MLME-POLL.confirm** sert à donner le résultat de la demande de données sous forme d'un statut. Voici les statuts possibles pour cette primitive :

SUCCESS : L'acquittement reçu par le coordinateur après que la commande « Data Request » ait été envoyée, indique la présence de données par le champ « Frame Pending ». Si les données sont alors correctement récupérées par la couche MAC du dispositif, c'est le statut SUCCESS qui est retourné à la couche supérieure par la primitive MLME-POLL.confirm.

CHANNEL_ACCESS_FAILURE : Si le CSMA-CA échoue durant l'accès au canal pour envoyer la trame de commande « Data Request », le statut CHANNEL_ACCESS_FAILURE est retourné à la couche supérieure.

NO_ACK : Si après l'envoi de la trame de commande « Data Request », la couche MAC attend la venue d'acquittement durant le temps de la variable *macAckWaitDuration* (en nombre de symboles) et qu'aucun acquittement n'est reçu après *aMaxFrameRetries* de fois cette même opération, le statut de MLME-POLL.confirm est NO_ACK.

NO_DATA : Si par l'acquittement reçu du coordinateur, la couche MAC s'aperçoit qu'il n'y a pas de données en attente chez le coordinateur (champ Frame Pending à 0), c'est le statut NO_DATA qui est renvoyé à la couche supérieure du dispositif. Ce même paramètre est retourné si la trame reçue après l'acquittement est une trame de commande ou une trame de données avec un chargement de 0 de longueur ou encore si, après le temps *aMaxFrameResponseTime*, aucune trame n'est reçue.

UNAVAILABLE_KEY : Si le paramètre « Security Enable » est actif, la couche MAC doit sécuriser les échanges. Si aucune clé appropriée n'est trouvée dans ACL, le statut de MLME-POLL.confirm est UNAVAILABLE_KEY.

FAILED_SECURITY_CHECK : Si une erreur intervient durant le processus de sécurisation de la trame, le statut de MLME-POLL.confirm est FAILED_SECURITY_CHECK.

INVALID_PARAMETER : Si un des paramètres de la primitive MLME-POLL.request n'est pas du bon type ou est en dehors des plages autorisées, c'est le statut INVALID_PARAMETER qui est renvoyé avec MLME-POLL.confirm.

2. Fonctionnement et synthèse

Le but de ce chapitre est d'essayer de souligner les principales fonctionnalités du standard afin d'en faciliter la programmation pour la suite. Des discussions sont également entreprises vis-à-vis des choix d'implémentations.

Les choix d'implémentations sont motivés par ce qui est prévu et possible à réaliser au niveau des applications de test. Tout d'abord, il est important de préciser que dans le cadre de ce projet l'implémentation n'est prévue qu'en **bande de base** pour se soustraire à des problèmes éventuels relatifs à la couche physique. Le nombre de modules disponibles pour les applications de test est de deux, ce qui limite tout de suite à l'utilisation d'applications mettant en jeu qu'**un seul PAN**. La partie **sécurité** décrite dans la norme a été complètement mise de côté, car ce n'est pas une priorité dans le fonctionnement global et c'est une partie conséquente qui mérite une étude et une implémentation à part entière. La sécurité n'est d'ailleurs nulle part abordée durant ce projet. Enfin, par rapport aux capacités du microprocesseur MSP430 imposé, c'est une **programmation de type séquentielle** qui est réalisée. Le programme principal (l'application) correspond aux couches OSI au-dessus de la couche MAC.

2.1 Base de temps

L'unité de temps de base utilisée dans le standard est le **symbole**. La durée d'un symbole est directement liée au débit de donnée fixé par la couche physique. Comme la couche physique de URSAFE possède un débit fixe de 9.6 kb/s, il est facile de déterminer le temps que prend un symbole. En sachant que le codage Manchester utilise 2 bits pour coder un symbole d'information, on obtient un débit de symbole (D) de moitié moins que le débit binaire : **D=4.8 kBaud**. La durée d'un symbole correspond alors au calcul de l'équation 7.

$$1 \text{ symbol} = \frac{1}{D} = \frac{1}{4800} = 208 \mu s$$

Équ. 7 - Durée d'un symbole

Au moyen du résultat de l'équation 7, on peut déterminer les temps des constantes MAC les plus importantes, car celles-ci sont définies par leur nombre de symboles. Les résultats temporels sont résumés au Tableau 8.

Constante	Valeur [symbole]	Valeur temporelle	Description
aBaseSlotDuration	60	12.5 ms	Durée d'un slot si SO = 0.
aBaseSuperframeDuration	aBaseSlotDuration * aNumSuperframeSlot = 60 * 16 = 960	200 ms	Nombre de symboles (temps) qu'utilise une Superframe complète si SO = 0.
aMaxFrameResponseTime	1220	254 ms	Nombre maximum de symboles que doit attendre un dispositif entre l'envoi et la réponse des données.
aMinCAPLength	440	91.5 ms	Durée minimale du CAP (exception: maintenance de GTS).
aMinLIFSPeriod	40	8.3 ms	Durée LIFS minimum
aMinSIFSPeriod	12	2.5 ms	Durée SIFS minimum
aResponseWaitTime	32 * aBaseSuperframeDuration	6.39 s	Nombre maximum de symboles que doit attendre un dispositif entre une demande et une réponse.
aUnitBackoffPeriod	20	4.2 ms	Temps utilisé comme période de base pour l'algorithme CSMA-CA.

Tableau 8 - Valeurs temporelles des constantes MAC

Il est également intéressant de connaître l'intervalle maximal et minimal entre deux Beacons lorsque le réseau est « Beacon Enabled ». En reprenant l'Équ. 4, on obtient les résultats de l'Équ. 8.

$$\begin{aligned}
 BO = 0 : \quad BI_{\min} &= 960 \cdot 2^0 = 200ms \\
 BO = 14 : \quad BI_{\max} &= 960 \cdot 2^{14} = 54 \text{ min } 30s
 \end{aligned}$$

Équ. 8 - Interval max et min des Beacons

2.2 MAC PAN Information Base (PIB)

Le MAC PIB est un regroupement de plusieurs attributs qui, globalement, correspondent à une certaine configuration de la couche MAC. Les attributs MAC PIB peuvent être lus et écrits par la couche supérieure avec les primitives MLME-GET et MLME-SET. À l'initialisation chaque attribut PIB est mis à une valeur par défaut.

Certains attributs PIB ne sont pas utilisés pour une implémentation RFD, notamment les attributs concernant la génération des trames de Beacon où seul un FFD est autorisé à le faire. Il existe également un certain nombre d'attributs PIB concernant la sécurité, ces attributs ne seront pas implémentés. Mais sinon, comme les attributs MAC PIB constituent une fonctionnalité fondamentale de la couche MAC, tous les attributs seront implémentés comme le prévoit le standard.

Les attributs MAC PIB sont regroupés dans la partie résumée de la norme (**Annexe A**).

2.3 CSMA-CA

D'une façon générale, les deux implémentations de CSMA-CA que définit la norme reposent sur le Clear Channel Assessment (CCA). C'est-à-dire que, par une mesure d'énergie, la couche physique va décider par rapport à un seuil si le canal est libre ou occupé. Cette mesure et cette décision se font via les primitives physiques PLME-CCA.request et PLME-CCA.confirm. Une telle mesure d'énergie n'est pas possible à réaliser avec l'implémentation d'URSAFE, mais par contre en allant lire sur des pins de l'interface radio, on peut déterminer si le canal est libre ou pas. Si la radio n'est pas utilisée, la primitive PLME-CCA.request peut, par exemple, renvoyer une variable aléatoire de Bernoulli pour simuler l'occupation du canal (0 ou 1).

La division en slots de la partie active de la Superframe sert uniquement à indiquer la limite entre le CAP et le CFP. Le CSMA-CA « slotted » n'utilise que la limite des périodes de Backoff pour transmettre. De base ($SO = 0$), un slot contient 3 Backoffs, car la constante *aUnitBackoffperiod* vaut 20 et la constante *aBaseSlotDuration* vaut 60. C'est pour cette raison qu'une limite de slot correspond toujours à une limite de Backoff.

L'attente aléatoire réalisée durant l'algorithme du CSMA augmente toujours plus son maximum (BE) si le canal n'est pas libre, et recommence à chaque fois au moins deux CCA. À la première réalisation du CSMA, BE peut valoir au maximum 3, ce qui correspond à une attente aléatoire maximale de 140 symboles (7 backoffs). Après plusieurs essais du canal, BE peut valoir au pire 5 (*aMaxBE*). Ce qui peut donner une attente pouvant atteindre 620 symboles (31 backoffs).

C'est la variable CW de l'algorithme du CSMA-CA « slotted » qui permet de savoir combien de périodes de Backoff vont être réservés pour que le CCA s'exécute avant que la transmission puisse commencer. Un CCA doit également toujours démarrer de façon synchrone à une période de Backoff. Si $macMinBE = 0$, le CCA est désactivé durant la première partie de l'algorithme.

L'algorithme CSMA-CA « slotted » doit également être capable de connaître à tout moment combien de périodes de Backoff il restent d'ici la fin du CAP, afin de savoir si la transmission utilise moins de périodes de Backoff que celles qui restent. Sinon, la transmission est remise à la prochaine Superframe.

Lorsque $CW=0$, la transmission peut être réalisée, mais elle doit commencer seulement sur la prochaine période de Backoff.

L'implémentation du CSMA réalisée est décrite dans la troisième partie du rapport concernant la réalisation au point 3.6.2.1.

2.4 Communications

2.4.1 Transmission indirecte

Une transmission indirecte est réalisée pour toutes les communications de données ou de commandes uplink (d'un dispositif à un coordinateur). Le principe est que le coordinateur stocke la transmission en attendant que le dispositif réalise une demande de données. Cela permet au dispositif de pouvoir fonctionner en mode basse consommation.

Le coordinateur PAN possède une pile mémoire servant à stocker plusieurs transmissions qui sont destinées aux dispositifs, qui est appelée **liste d'attente de transmissions**. Toute implémentation de FFD doit posséder une telle pile. Le standard conseille de pouvoir stocker jusqu'à 7 MSDU, valant 889 bytes (7×127).

Chaque transmission en attente demeure au moins *macTransactionPersistenceTime* dans la file d'attente du coordinateur. Après que ce temps se soit écoulé, le statut TRANSACTION_EXPIRED est renvoyé à la couche supérieure.

Si une transmission ne peut être stockée dans la liste d'attente par manque de place, c'est le statut TRANSACTION_OVERFLOW qui est renvoyé à la couche supérieure.

Pour qu'une liste d'attente de transmissions soit indispensable, il faut au moins que le coordinateur possède 2 dispositifs associés. Comme cette configuration n'est pas sûre d'être réalisée, l'implémentation de la liste d'attente de transmissions n'est pas une priorité. D'autant plus qu'une transmission de données indirectes peut se réaliser sans liste d'attente de transmissions qui sert en premier lieu à faire des économies d'énergie.

2.4.2 Retransmission

Une retransmission ne peut se faire que dans une communication acquittée. Une transmission peut échouer de deux manières. La trame de données n'arrive pas à son destinataire ou l'acquittement n'arrive pas à l'expéditeur de la trame de données. Dans les deux cas, il doit y avoir retransmission. C'est l'expéditeur de données qui va décider si la transmission a échoué en réalisant une mesure de temps. Dès que la variable PIB *macAckWaitDuration* est écoulée, l'expéditeur tente de renvoyer une trame de données. La retransmission peut se faire au maximum *aMaxFrameRetries* de fois. Par défaut, cette variable est initialisée à 3.

Cette fonctionnalité de retransmission qui est appliquée pour toutes trames de données et de commandes sera implémentée exactement comme le décrit le standard.

2.4.3 Réception

Chaque dispositif a la possibilité, par la variable *macRxOnWhenIdle*, que sa couche MAC active ou non le récepteur durant les périodes d'attentes. Si *macRxOnWhenIdle* est à TRUE, la couche MAC ordonne à la couche physique d'activer le récepteur. À FALSE, le récepteur est éteint durant les périodes d'attente. Durant ces périodes d'attente, la couche MAC doit aussi être apte à recevoir des demandes de la couche supérieure.

Cette fonctionnalité ne peut être que difficilement implémentée, car la couche MAC doit être réveillée par la couche physique, tandis que pour l'implémentation séquentielle prévue, la couche MAC n'est contrôlée que par la couche Application. C'est donc à la couche Application de savoir quand elle doit mettre le dispositif dans un mode de réception.

2.4.4 Rejet

La couche MAC doit être capable de rejeter des trames ne correspondant pas au standard 802.15.4. À la réception d'une trame, un premier niveau de filtrage est de toute façon réalisé, c'est la vérification du CRC. Si le CRC n'est pas exact, la trame est directement rejetée.

La variable *macPromiscuousMode* (implémentation FFD seulement) décide si un deuxième niveau de filtrage est réalisé à la réception de la trame. Si cette variable est à TRUE, les données peuvent être directement envoyés aux couches supérieures. Si la variable PIB est à FALSE, le deuxième niveau de filtrage consiste à accepter les trames qui satisfont les 5 conditions suivantes :

- 1) Le champ Frame type contient un type de trame légal.
- 2) Dans le cas d'une trame Beacon, le champ *Source PAN id* doit correspondre à *macPANId* pour autant que celui-ci ne soit pas à 0xFFFF (pas associé).
- 3) Dans le cas d'une trame de données ou de commande, si le champ *Destination PAN id* est présent, il doit correspondre à la valeur de *macPANId*.

- 4) Dans le cas d'une trame de données ou de commande, la valeur de l'adresse destination doit correspondre à *macShortAddress* (adresse courte du dispositif) si elle est de 16 bits ou à *aExtendedAddress* (adresse étendue unique du dispositif) si elle est de 64 bits.
- 5) Dans le cas d'une trame de données ou de commande, si l'en-tête ne contient que des champs d'adresse source, la trame ne peut être acceptée que par un coordinateur PAN.

Si toutes ces conditions sont remplies, la trame est considérée comme valide et peut être envoyée à la couche supérieure. Pour une implémentation RFD, ces tests sont de toute façon réalisés.

Il est à remarquer que si la variable *macPromiscuousMode* est à TRUE, la variable *macRxOnWhenIdle* doit automatiquement se retrouver à TRUE également. De la même façon, si *macPromiscuousMode* est à FALSE, *macRxOnWhenIdle* est à FALSE.

Comme aucune limite d'adaptation ne pose problème, cette fonctionnalité de rejet de trames est implémentée comme venant d'être décrite.

2.4.5 Récupération de données en attente

Pour toutes transmissions considérées comme **indirectes**, c'est au dispositif de s'occuper de récupérer chez le coordinateur des données lui appartenant. Le coordinateur peut stocker dans une liste d'attente jusqu'à 7 trames MAC de maximum 127 octets.

2.4.5.1 Réseau « Beacon Enabled »

N'importe quel dispositif peut déterminer si des données sont en attente pour lui chez le coordinateur, car les Beacons broadcast qui sont envoyés régulièrement par le coordinateur contiennent les adresses des dispositifs qui ont des données en attente. Le dispositif peut alors envoyer une trame de commande « Data Request » durant la période de contention (CAP). Cette demande doit être acquittée par le coordinateur.

Si le coordinateur a assez de temps pour contrôler que le dispositif a effectivement des données en attente avant que le temps *macAckWaitDuration* soit écoulé, il met le champ « Frame Pending » de l'acquittement à la valeur correspondante (0 ou 1). Si ce n'est pas possible, le coordinateur place ce champ à 1.

À la réception de l'acquittement, si le champ « Frame Pending » est à 0, le dispositif conclut qu'il n'y a pas de données en attente pour lui. Si le champ est à 1, le dispositif active le mode de réception durant au moins *aMaxFrameResponseTime* symboles afin de pouvoir recevoir les données venant du coordinateur. Si le coordinateur n'avait en fait pas de données pour le dispositif, le coordinateur envoie une trame de données avec un chargement de 0 octet pour faire comprendre qu'il n'y avait pas de données. Si des données étaient présentes, elles sont évidemment envoyées par le coordinateur.

Le mécanisme utilisé pour cet envoi de données est le CSMA-CA « slotted ».

Si le champ Frame Pending de la trame reçue par le dispositif est à 1, le dispositif recommence toute la procédure depuis le début pour obtenir la prochaine trame de données.

Le principe de la transmission indirecte faite par le coordinateur et de la récupération des données en attentes par le dispositif est imagée à la Figure 30.

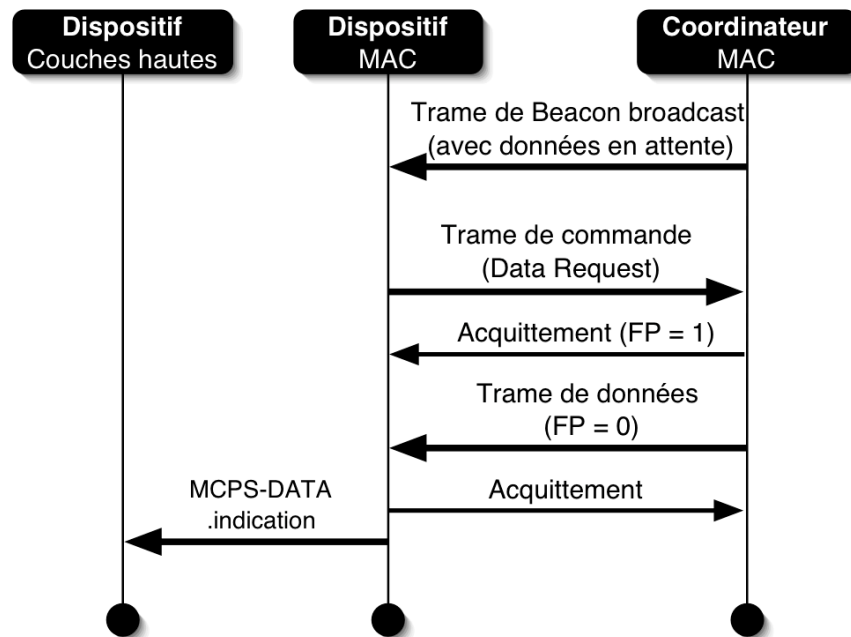


Figure 30 - Transmission indirecte et récupération de données en attente

La récupération des données en attente sera implémentée comme le prévoit le standard.

2.4.5.2 Réseau « Non Beacon Enabled »

La couche MAC reçoit une primitive MLME-Poll.request de la couche supérieure lorsqu'il faut essayer de récupérer des données chez le coordinateur. Ensuite, l'extraction des données se réalise de la même façon que dans un réseau « Beacon Enabled ».

2.5 Types de balayages

Tous les dispositifs sont capables de réaliser des balayages passifs et d'orphelins. Un dispositif avec fonctionnalité complète (FFD) peut également réaliser des balayages actifs et à détection d'énergie. La couche supérieure précise la liste des canaux où les balayages doivent être réalisés et le temps du balayage par canal.

2.5.1 Détection d'énergie

Un balayage à détection d'énergie permet à un FFD de mesurer l'énergie maximale dans chaque canal indiqué par la couche supérieure. Ce balayage sert lors de la création d'un nouveau PAN, afin que le futur coordinateur choisisse le canal le plus clair.

La durée du balayage par canal est de $(aBaseSuperframeDuration * 2^n - 1)$ nombre de symboles où la variable n (0 à 14) correspond au paramètre *ScanDuration* de la primitive MLME-SCAN.request. La primitive physique qui réalise ce balayage est PLME-ED.request.

Ce type de balayage n'a pas à être implémenté, car premièrement le matériel de URSAFE ne permet pas de réaliser ce genre de balayage et deuxièmement, URSAFE ne fonctionne que sur trois canaux seulement et qu'un seul est utilisé pour les applications qui sont de toute manières réalisées en bande de base.

2.5.2 Balayage actif

Un balayage actif de canal permet à un FFD de localiser un coordinateur qui transmet des Beacons. Ce type de balayage peut servir à un futur coordinateur pour créer un nouveau PAN avec un autre PAN Id. Il peut aussi servir à un dispositif FFD qui veut s'associer avec un PAN.

Pour réaliser ce balayage, la couche MAC doit sauvegarder son PAN Id et s'identifier comme un dispositif non associé (0xffff). Le PAN Id est restauré à la fin du balayage.

La durée du balayage par canal est de $(aBaseSuperframeDuration * 2^n - 1)$ nombre de symboles où la variable n (0 à 14) correspond au paramètre *ScanDuration* de la primitive MLME-SCAN.request.

Ce balayage actif consiste à envoyer une trame de commande « Beacon Request » sur chaque canal. Si le coordinateur PAN du canal sélectionné est dans un mode de « Beacon Enabled », il doit ignorer la trame de commande. Par contre, si le coordinateur est dans un mode de « Non Beacon Enabled », alors il répond avec l'envoi d'un Beacon en CSMA-CA « unslotted ». La couche MAC du dispositif qui a fait le balayage actif crée et stocke une liste de PAN Descriptor pour chaque canal où un Beacon a été reçu. Le standard ne prévoit pas de limite maximale de stockage, cela dépend alors de l'implémentation du dispositif.

Ce type de balayage ne sera pas implémenté, car la gestion de plusieurs PAN n'est pas prévue. L'avantage de ce balayage n'apporte pas grand chose par rapport au balayage passif qui lui sera implémenté.

2.5.3 Balayage passif

Un balayage passif de canal permet à n'importe quel dispositif de localiser un coordinateur transmettant des Beacons. Ce type de balayage sert à tous les dispositifs qui prévoient de s'associer avec un PAN.

Pour réaliser ce balayage, la couche MAC doit sauvegarder son PAN Id et s'identifier comme un dispositif non associé (0xffff). Le PAN Id est restauré à la fin du balayage.

La différence avec un balayage passif est que l'on ne va pas transmettre de trame de commande « Beacon Request ». Le principe est juste d'écouter sur les canaux pendant $(aBaseSuperframeDuration * 2^n - 1)$ afin de détecter des Beacons et d'en récupérer les informations dans une liste comme dans le cas du balayage actif.

Ce balayage est implémenté, car il est obligatoire pour tous dispositifs qui veulent s'associer. La seule différence est que le balayage ne se déroulera que sur un seul canal.

2.5.4 Balayage d'orphelin

Un balayage d'orphelin permet à un dispositif orphelin de retrouver le coordinateur avec lequel il était associé avant la perte de la synchronisation.

Le principe est d'envoyer une trame de commande « Orphan Notification » sur tous les canaux indiqués par le paramètre *ScanChannel*. L'attente maximale, après l'envoi de la trame de commande est de *aResponseWaitTime*. Le balayage prend fin si une trame de commande « Coordinator Realignment » est reçue ou si tous les canaux ont été parcouru sans succès. La trame de commande « Coordinator Realignment » contient le numéro du PAN et généralement l'ancienne adresse courte du dispositif.

La trame de commande « Coordinator Realignment » est seulement envoyée par le coordinateur si celui-ci a retrouvé la trace de l'association du dispositif orphelin dans sa base de données.

Comme l'interface radio n'est pas prévue d'être intégrée dans les applications réalisées, cette implémentation de balayage d'orphelin n'est pas une priorité. De plus, en bande de base, ce type de balayage est logiquement inutile.

2.6 Création du PAN

Un PAN ne peut être créé que par un FFD, ce dernier doit obligatoirement réaliser un balayage à détection d'énergie afin de déterminer le canal optimal à utilisé. Il faut également choisir un numéro de PAN Id. Le standard prévoit que la procédure du balayage de canal renvoie une liste de descriptions et que d'après celle-ci un algorithme va permettre de choisir une adresse PAN appropriée.

Pour cette adaptation, comme un seul PAN est prévu, le balayage à détection d'énergie devient inutile et il suffit de se contenter de fixer une valeur pour l'adresse du PAN Id.

Le réseau peut devenir « Beacon Enabled » en utilisant la primitive MLME-START.request. Seuls les dispositifs dont la variable *macShortAddress* n'est pas à 0xFFFF (dispositifs déjà associés) peuvent transmettre des trames Beacons. Le temps qui est écoulé depuis la transmission du dernier Beacon est stocké dans la variable PIB *macBeaconTxTime* (en nombre de symboles). Ce temps est toujours initialisé depuis la même limite de symboles pour chaque Beacon.

Toutes les trames Beacons sont transmises au début de chaque Superframe. Pour rappel, le taux de répétition des Beacons (BI) est de $aBaseSuperframeDuration * 2^n$ symboles. La lettre n correspondant à la variable *macBeaconOrder*.

Un FFD qui n'est pas le coordinateur du PAN n'a pas le droit d'envoyer de Beacons avant qu'il soit associé au PAN. C'est également la primitive MLME-START.request qui permet de transmettre des Beacons. Le paramètre PANCoordinator doit être à FALSE. Le champ Source PAN Id doit correspondre à l'adresse du PAN où le dispositif est associé (*macPANId*). Le réglage de l'intervalle des Beacons est le même principe que pour le coordinateur.

Les trames Beacons servent donc premièrement à indiquer aux dispositifs associés au PAN la présence d'informations en attente pour eux, et Deuxièmement à la synchronisation entre tous les dispositifs du PAN. Les transmissions de Beacons ont la priorité sur toutes autres opérations de transmission ou de réception.

2.7 Association & Dissociation

2.7.1 Procédure d'association

Un dispositif ne peut s'associer à un PAN que si primo une réinitialisation de sa couche MAC a été faite (MLME_RESET.request) et deuxio qu'un balayage de canal a été réalisé (Active Channel Scan ou Passive Channel Scan). Le résultat du balayage sert à choisir le PAN auquel il est possible de s'associer et de déjà connaître quelques informations sur le réseau « Beacon Enabled » du PAN.

Le coordinateur peut seulement autoriser une association d'un dispositif si Sa variable *macAssociationPermit* est à TRUE. Suivant le PAN auquel le dispositif veut s'associer, la couche MAC doit configurer les PIB PHY et MAC nécessaire à l'association :

- *phyCurrentChannel* : Canal utilisé pour l'association
- *macPANId* : Identifiant PAN auquel on s'associe
- *macCoordShortAddress* ou *macCoordExtendedAddress* : adresse (courte ou étendue) du coordinateur, elle s'obtient par la réception d'une trame Beacon.

Dans un réseau Beacon-Enabled, le dispositif peut commencer à se synchroniser sur les Beacons du coordinateur avec la primitive MLME_SYNC.request afin d'optimiser l'association. Le paramètre *TrackBeacon* de la primitive est alors à TRUE.

La primitive MLME_ASSOCIATE.request permet alors de s'associer au PAN.

Un dispositif encore non associé peut initier la procédure d'association par l'envoi d'une trame de commande « Associate Request » au coordinateur du PAN. Ce dernier renvoie de toute façon un acquittement si la commande a été reçue correctement. La réponse de l'association n'est pas contenue dans cet acquittement. Le coordinateur doit avoir du temps pour déterminer si les ressources actuelles du PAN sont suffisantes pour accueillir un nouveau dispositif. Le coordinateur prend la décision en au moins *aResponseWaitTime* symboles. La décision du coordinateur est renvoyée au dispositif via la trame de commande « Association Response ».

Cette réponse d'association de commande est considérée comme une transmission indirecte. C'est-à-dire que le coordinateur va placer cette trame dans sa liste d'attente. C'est au dispositif de faire une demande afin de récupérer l'information (voir 2.4.5 - Récupération de données en attente).

Si le champ « Allocate Address » de la commande « Association Request » est à 1, le coordinateur alloue une adresse courte de 16 bits, cette adresse devient la nouvelle adresse du dispositif dans le PAN. Si le champ est à 0, l'adresse allouée par le coordinateur est de 0xffff. Le dispositif utilise alors uniquement son adresse étendue de 64 bits.

Si durant l'association le dispositif est synchronisé avec le coordinateur, dès que la trame Beacon indique que des données sont en attente pour le dispositif, celui-ci peut extraire les données du coordinateur (voir 2.4.5 - Récupération de données en attente).

Si le dispositif n'est pas synchronisé avec le coordinateur (réseau « Non Beacon Enabled »), le dispositif essaye d'extraire la réponse du coordinateur après le temps *aResponseWaitTime* symboles. S'il n'y a pas de donnée en attente, le statut NO_DATA est retourné aux couches supérieures du dispositif par la primitive MLME_ASSOCIATE.confirm. Dans ce cas, toute synchronisation avec le coordinateur est arrêtée par un MLME_SYNC.request où le paramètre *TrackBeacon* est à FALSE.

À la réception de la commande « Association Response », le dispositif doit encore envoyer un acquittement au coordinateur. Si l'association est acceptée, le dispositif fait les changements d'adresses nécessaires. Si l'association échoue, le dispositif met la variable *macPANId* à 0xffff.

La procédure d'association peut être implémentée comme le standard la définit sans aucune restriction.

2.7.2 Procédure de dissociation

Une dissociation se réalise par la primitive `MLME_DISASSOCIATE.request` venant de la couche supérieure.

Un premier cas possible de dissociation est que c'est le coordinateur qui veut que l'un des dispositifs actuellement associés quitte le PAN. Le coordinateur utilise la commande « Disassociation Notification » en transmission indirecte. Une fois que le dispositif a récupéré la commande, il doit encore envoyer un acquittement. Si le coordinateur ne reçoit pas d'acquiescement, il considère tout de même que le dispositif a été dissocié.

Le deuxième cas possible de dissociation correspond au cas où le dispositif lui-même veut quitter le PAN. Il doit envoyer au coordinateur une commande de « Disassociation Notification ». Le coordinateur acquiesce alors sa demande. Si le dispositif ne reçoit pas d'acquiescement, il se considère tout de même dissocié du PAN.

La procédure de dissociation peut être implémentée comme le standard la définit sans aucune restriction.

2.7.3 Orphelin

Si la couche supérieure reçoit plusieurs fois de suite des échecs de transmission de données, elle conclut que le dispositif est orphelin. Une première solution est de faire un réalignement avec le coordinateur en utilisant la commande « Coordinator Realignment » afin d'être à nouveau synchronisé. Comme deuxième solution, la possibilité de carrément réinitialiser la couche MAC (`MLME_RESET.request`) existe, puis de s'associer à nouveau avec le coordinateur (`MLME_ASSOCIATE.request`).

2.8 Synchronisation

2.8.1 Réseau « Beacon Enabled »

Dans un réseau Beacon Enabled (*macBeaconOrder* < 15) tout dispositif a la possibilité de se synchroniser sur les Beacons reçus par le coordinateur. Pour vérifier la validité du Beacon, le champ « PAN Identifier » doit correspondre à la variable PIB *macPANId*. La synchronisation est effectuée par la primitive `MLME_SYNC.request`. Si le paramètre *Tracking* est activé dans la primitive, le dispositif doit activer régulièrement son récepteur pour accueillir les Beacons. Si le paramètre *Tracking* n'est pas activé, le dispositif ne accueille que le prochain Beacon, par la suite la synchronisation ne se fera plus.

Si un Beacon n'est pas reçu à l'instant prévu durant un temps maximal de *aBaseSuperframeDuration* * 2^{n+1} , la recherche est répétée pour le prochain Beacon. Une fois que le nombre de Beacons manqués dépassent *aMaxLostBeacon*, la couche MAC envoie à la couche supérieure la raison de la perte de la synchronisation en utilisant la primitive `MLME_SYNC-LOSS.indication` en envoyant le paramètre `BEACON_LOST`.

Si un Beacon valide est reçu et que *macAutoRequest* est à FALSE, la couche MAC indique les paramètres du Beacon à la couche supérieure en utilisant la primitive `MLME_BEACON_NOTIFY.indication`. Si *macAutoRequest* est à TRUE, la couche MAC va également envoyer les paramètres du Beacon à la couche supérieure, mais sans contrôler si la trame est valide ou non.

Lorsqu'un dispositif est associé à un PAN et que le réseau est « Beacon Enabled », le dispositif place toutes les informations des Beacons reçus dans la structure du PAN Descriptor. Précédemment un balayage (actif ou passif) doit avoir été réalisé avant d'activer la synchronisation afin de connaître le PAN Descriptor (le scan récupère un Beacon).

L'implémentation de la synchronisation par réseau « Beacon Enabled » sera la plus conforme à la description du standard. Les seules différences vont se situer au niveau des primitives de type « indication ». À la place d'être générées par la couche MAC, elles devront être générées par la couche applicatif supérieure qui gère le séquençement du code.

2.8.2 Réseau « Non Beacon Enabled »

Dans un réseau « Non Beacon Enabled », il ne peut y avoir une synchronisation, mais la couche MAC reçoit un MLME-Poll.request de la couche supérieure lorsqu'il faut essayer de récupérer des données du coordinateur. La couche MAC entreprend alors une extraction de données (voir 2.4.5 - Récupération de données en attente).

2.9 Réserve de temps (GTS)

Une réserve de temps peut être demandée par la couche supérieure lors d'un transfert de données (paramètre *TxOption*). Cela permet de garantir une meilleure qualité de services pour la transmission en obtenant des temps de latence bas et un débit fixe.

C'est en utilisant la primitive MLME-GTS.request qu'un dispositif associé peut réaliser une réserve de temps. Le dispositif doit être une implémentation FFD, un RFD n'intégrant pas la gestion de GTS. La primitive MLME-GTS.request réalise une demande d'allocations d'un nouveau GTS en utilisant une trame de commande « GTS Request ». Le coordinateur doit alors décider si la demande est possible à effectuer en examinant la capacité actuelle du réseau. Afin que la demande de GTS soit acceptée, la longueur du nouveau GTS à allouer doit faire en sorte que le nombre de symboles minimums (*aMinCAPLength*) pour la période de contention (CAP) soit respectée. L'allocation de GTS est donc basée sur le principe du premier arrivé, premier servi.

Les informations correspondant à la nouvelle allocation de GTS sont présentes dans le « GTS Descriptor » du prochain Beacon broadcast qui est envoyé par le coordinateur. Le coordinateur a le droit d'attendre jusqu'à *aGTSDescPersistenceTime* qui coïncide à un temps de 4 Superframe avant de mettre la valeur du « Final CAP Slot » dans le champ « Superframe Specification » et les champs GTS du Beacon. Tous les Beacons broadcast envoyés ultérieurement possèdent cette configuration de GTS et cela tant qu'elle n'a pas été arrêtée par le dispositif concerné ou par le coordinateur.

Pour toutes transmissions réalisées durant la période du GTS alloué, le dispositif doit garantir que le temps total comprenant acquittement et les temps SIFS - LIFS ne dépassent en aucun cas la limite de fin du GTS.

L'arrêt d'allocation d'un GTS se réalise aussi avec la primitive MLME-GTS.request. C'est la variable « Characteristics field » du paramètre « GTS Characteristics » qui est mis inactif pour préciser l'arrêt de l'allocation. Pour spécifier qu'un arrêt d'allocation de GTS a été fait ou qu'une allocation a été refusée, le GTS Descriptor du Beacon contient une absurdité en indiquant que l'allocation du GTS commence au slot 0.

Une ou plusieurs demandes d'arrêts d'allocations peuvent amener une fragmentation au niveau des GTS alloués encore présents. Le CFP doit être défragmenté afin de garantir une période contention (CAP) maximum. Pour cela, le coordinateur met à jour automatiquement tout GTS qui commence à un slot plus petit que le GTS dont l'allocation vient d'être arrêtée.

Le coordinateur doit avoir la capacité de se rendre compte qu'un dispositif n'utilise plus son allocation de GTS pour une réception ou un envoi. Le coordinateur prend la décision d'arrêter d'allouer un GTS lorsque qu'un certain nombre de Superframe est passé sans que l'un des GTS n'ait été utilisé.

Même si cette fonctionnalité des réservations de temps est importante, elle n'est pas primordiale au fonctionnement général. Son implémentation sera donc réalisée seulement si le temps l'autorise.

3. Réalisation

La réalisation consiste à décrire le standard en langage C ANSI à partir d'une forme principalement descriptive faite de textes et de schémas. Une autre solution d'étude de la norme est également possible en déchiffrant le langage SDL (Specification and Description Language). En effet, la norme IEEE 802.15.4 est entièrement décrite sur environ 350 pages en langage SDL. Cette voie demanderait l'apprentissage de SDL en plus, elle ne sera donc pas suivie.

Hormis l'implémentation C directe, une deuxième solution de réalisation a été légèrement discutée. Il se trouve qu'à l'interne même du CSEM, un traducteur SDL fait en Java est en développement. Pour suivre cette voie aisée d'apparence, il aurait tout d'abord fallu obtenir la description SDL sous forme de fichier texte. Comme la norme est toujours au stade de projet, cette description n'est pas disponible publiquement. De plus, le traducteur bien que proche d'une version finalisée, possède tout de même encore quelques restrictions au niveau de l'interprétation SDL. Par contre, de choisir cette solution d'implémentation aurait permis d'arriver à une description très proche du standard et de garantir une meilleure compatibilité.

L'idée de base de l'implémentation directe en C, est certainement plus intéressante à réaliser dans le cadre d'un projet de diplôme. La norme ne peut de toute façon être portée telle quelle, car le fonctionnement physique de URASFE est différent et plus limité de ce que prévoit le standard. L'adaptation peut être réalisée d'une manière plus facile avec un code que l'on a entièrement écrit que s'il est généré par un traducteur. Par contre, on ne peut éviter un plus grand risque d'éloignement à cause de l'interprétation de la norme écrite et schématique.

En résumé, pour implémenter le standard MAC IEEE 802.15.4 dans le microprocesseur MSP430, le standard doit être décrit en langage C ANSI. La compilation du langage C en assembleur et la programmation de L'EEPROM se font à l'aide du logiciel IAR par le port JTAG du MSP430.

3.1 Logiciel IAR Embedded Workbench

La principale plateforme de travail utilisée pour ce projet est le logiciel **IAR Embedded Workbench** (Figure 31) qui permet de faire la compilation du code C en assembleur MSP430. La société **IAR System**, qui développe ce logiciel, est spécialisée dans les outils de développements pour systèmes embarqués. Leurs logiciels sont conçus pour programmer les microprocesseurs de très nombreux fabricants. La version du logiciel IAR Embedded Workbench qui est employée pour cette implémentation est la **2.20.1.7**, le programme installé ne peut que compiler la famille MSP430 de Texas Instruments.

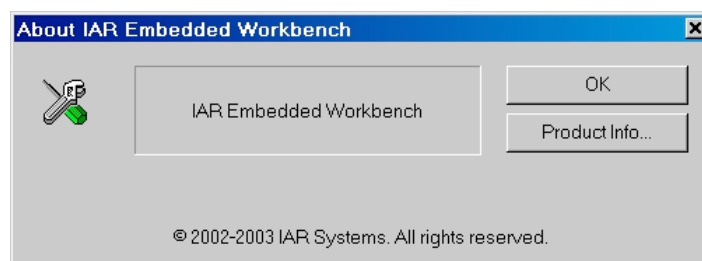


Figure 31 - Logiciel IAR Embedded Workbench (About)

IAR est un logiciel de développement typique dans sa forme et dans sa prise en main dont la qualité n'est pas à reprocher. L'interface et l'utilisation sont simples, les options peu nombreuses, mais généralement suffisantes et les bugs ne sont heureusement que très rares.

3.1.1 L'éditeur et le compilateur

Une fois le logiciel lancé, voici les quelques étapes fondamentales liées à la création d'un nouveau projet depuis la barre de menu.

Nouveau projet :

- File - New - Workspace
- Project - Create New Project
- Project - Add files (*.c)

Une fois le projet créé, les fichiers ajoutés peuvent être ouverts depuis la fenêtre d'exploration en double cliquant dessus. Chaque fichier correspond à une fenêtre indépendante, lesquelles peuvent être réorganisées comme on le souhaite. L'interface de travail peut par exemple ressembler à la capture de la Figure 32.

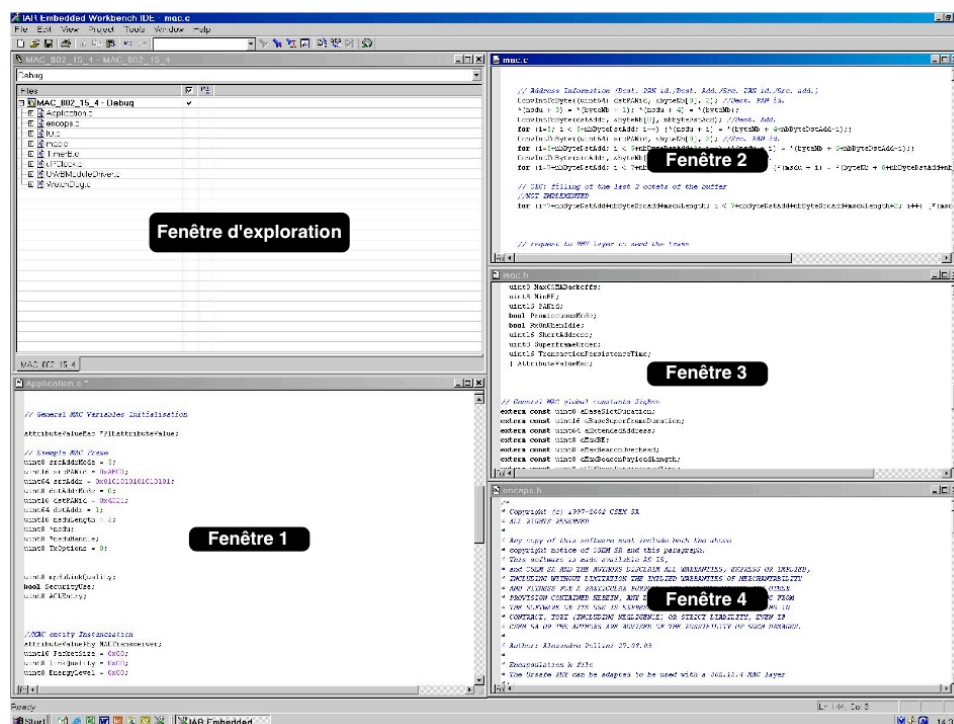


Figure 32 - Environnement de travail de l'éditeur IAR

Les menus ne sont pas trop chargés et n'ont pas à être souvent utilisés. Les principales commandes se retrouvent d'ailleurs dans la barre des raccourcis sous formes d'icônes. Hormis les quelques icônes classiques, comme par exemple la création d'un nouveau fichier ou l'enregistrement, on retrouve quelques icônes propre à l'application (Figure 33).

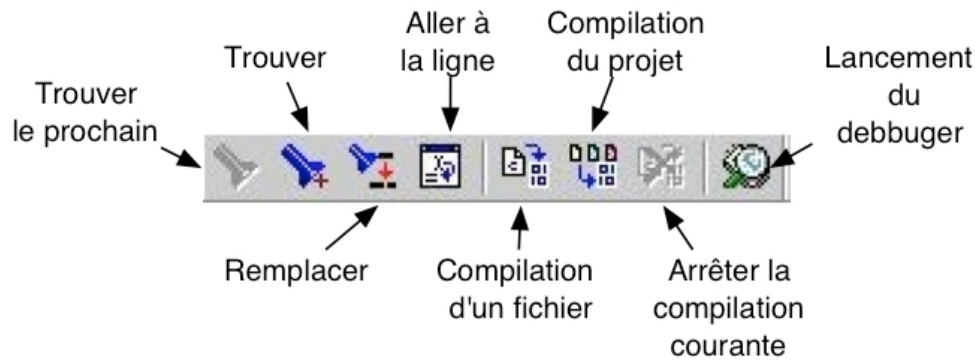


Figure 33 – Icônes de raccourcis de l'éditeur IAR

Les commandes de compilation peuvent également être appelées à l'aide des raccourcis claviers suivants :

- Compilation d'un fichier : CTRL-F7
- Compilation du projet : F7
- Lancement du débbugeur : CTRL-D

Lors d'une compilation, les fichiers concernés sont automatiquement sauvegardés, de même que lors du lancement du débbugeur, le projet est automatiquement compilé s'il ne l'a pas été fait auparavant.

Lors de l'édition des fichiers, il est essentiel de bien configurer les tabulations, afin que les fichiers puissent être lus sans dérangement sous d'autres éditeurs. Cette configuration se fait via le chemin *Tools – Option - Editor*. Voici ce qui a été choisi :

- Indent Size : 4
- Tab Key Function : Indent with Spaces

Les fenêtres n'ont rien de particulier sauf la présence dans le coin en bas à gauche d'un bouton *f()* qui permet de lister toutes les fonctions présentes dans la fenêtre. En cliquant sur une fonction, le curseur se déplace automatiquement au début de la fonction choisie.

Avant de compiler la première fois, il faut configurer quelques options propres au projet, dont le choix du microprocesseur MSP430. Ces options sont atteignables en faisant un clic droit sur le nom du projet dans la fenêtre d'exploration ou par le raccourci clavier ALT-F7. La Figure 34 montre la fenêtre de ces options. Dans *General – Target - Device* le microprocesseur **MSP430F149** doit être sélectionné. Il faut également cocher la case *Hardware multiplier* afin d'accélérer les multiplications.

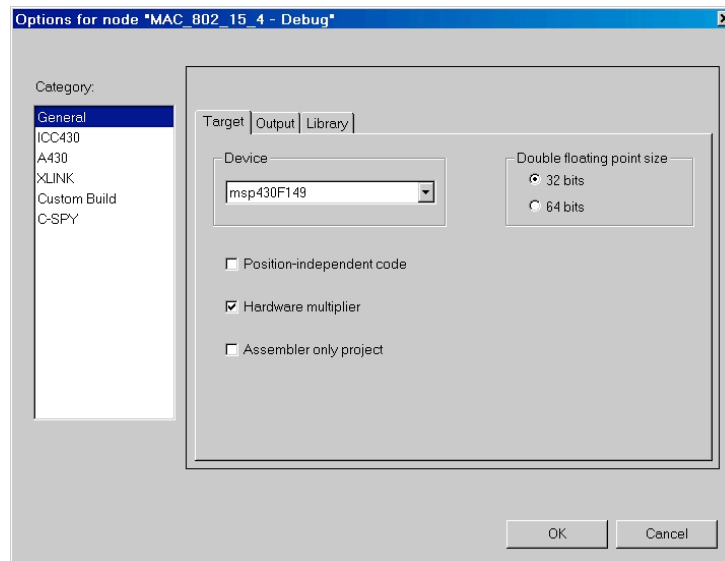


Figure 34 - Options d'un projet IAR

Toujours dans ces options, sous *C-SPY - Setup-Driver*, on peut y régler la façon de déboguer le programme. Il y a deux manières principales de faire :

- **Flash Emulation Tool** : Le programme est directement placé dans le microprocesseur et le débogage se déroule directement via le matériel.
- **Simulator** : On réalise un débogage à l'écran par simulation Windows.

Une fois la compilation réalisée, dans la fenêtre d'exploration, en dessous des programmes compilés, s'affichent des liens notamment vers les header (*.h). Une fenêtre *Messages* s'affiche également afin de signaler les éventuelles erreurs et warnings. En cliquant sur ceux-ci, le logiciel nous amène directement à la ligne de code où le problème de syntaxe a été détecté.

3.1.2 Le débogueur

L'environnement de travail du débogueur se présente comme à la Figure 35. Seules les 4 fenêtres les plus utilisées ont été représentées :

- **Fenêtre Programme** : C'est dans une fenêtre de programme que l'on peut placer des breakpoints et suivre les étapes de l'exécution du code C.
- **Fenêtre Dissassembly** : C'est le correspondant à la fenêtre Programme, mais le suivi du code est directement proposé en assembleur. C'est aussi par cette fenêtre que l'on peut atteindre des adresses mémoires personnalisées.
- **Fenêtre Watch** : On peut suivre ici l'état de toutes les variables de notre implémentation. Il suffit d'entrer son nom manuellement. On peut connaître sa valeur, son type et sa position mémoires. Ces variables peuvent être séparées dans 3 onglets différents (Watch1, Watch2 et Watch3).
- **Fenêtre Locals** : C'est le même principe de visualisation que la fenêtre Watch, mais il n'y a que les variables locales qui sont visibles dans cette fenêtre. Celles-ci s'inscrivent et disparaissent automatiquement.

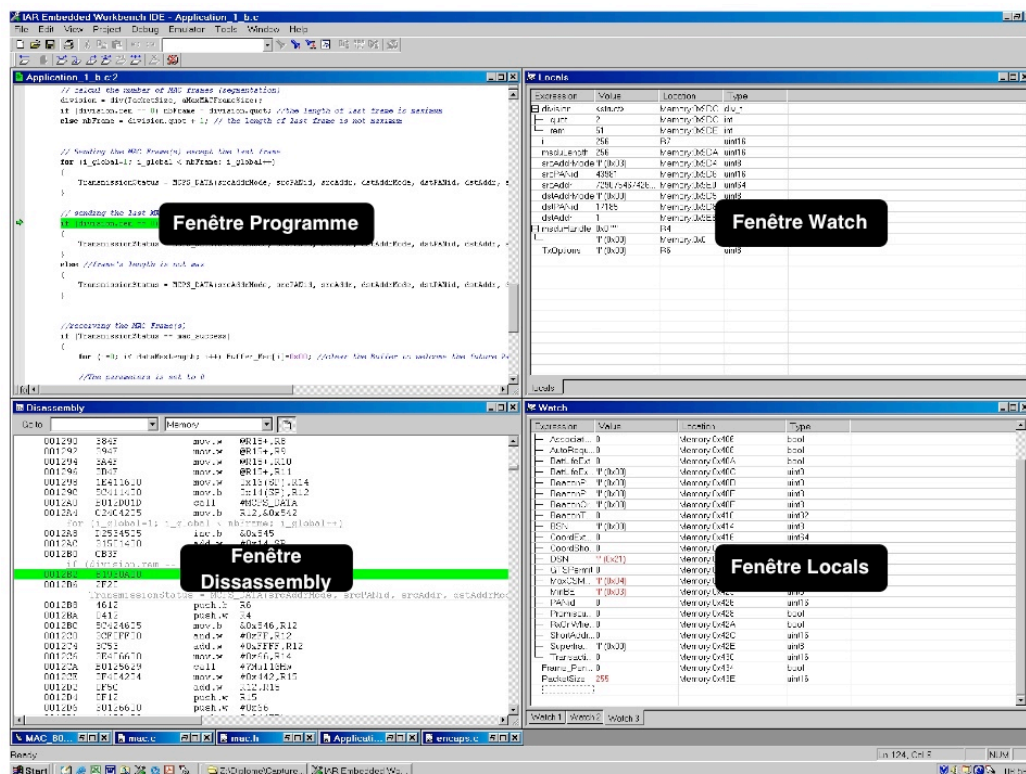


Figure 35 - Environnement de travail du débogueur IAR

Une autre fenêtre qui peut s'avérer importante à consulter lors de test, est la fenêtre des registres qui est atteignable par le menu *View*. Dans cette fenêtre, qui est illustrée à la Figure 36, on retrouve tous les registres du processeur dont le plus important est le registre d'état (Status Register) qui permet d'avoir des informations actuelles sur le masquage des interruptions (GIE), sur les horloges et les modes basses consommations et sur les bits d'opérations V N Z C.

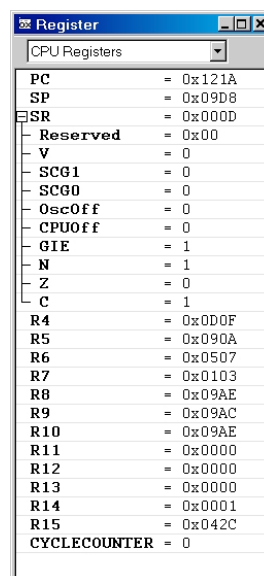


Figure 36 - Fenêtre des registres

Comme pour l'éditeur, il existe pour le débogueur quelques raccourcis sous forme d'icône qui sont très utilisés et qu'il est pratique de connaître. Leur description est faite à la Figure 37.

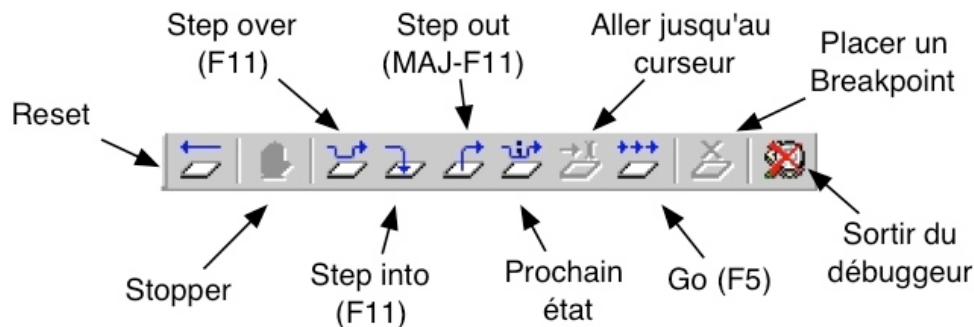


Figure 37 - Icônes de raccourcis du débogueur IAR

3.2 Caractéristiques du MSP430F149

Voici la liste des caractéristiques principales du type de microprocesseur utilisé, le **MSP430F149** :

- Architecture **RISC** Von Neumann de **16 bits**
- **60 KB ROM** (mémoire Flash)
- **2 KB RAM**
- Temps d'un cycle minimum d'instruction : **250 ns** (4 Mhz)
- Alimentation **1.8 V - 3.6 V**
- **Consommation ultra faible** :
 - Mode actif (1 MHz, 2.2 V) : 160 uA
 - Mode standby : 0.7 uA
 - Mode Off (Arrêt de la RAM) : 0.1 uA
- Réveil depuis le mode stand-by en moins de 6 μ s
- **Horloges configurables** :
 - Résistances variables internes ou une externe
 - Quartz de 32 kHz et plus
 - Source d'horloge externe
- **3 Compteurs de 16 bits** :
 - Timer_A
 - Timer_B
 - Watchdog Timer
- **ADC** : Convertisseur A/D 8 canaux 12 bits
- **Comparateur analogique**
- **2 USART** en Hardware avec deux modes de communication :
 - UART : Interface série asynchrone (RS232)
 - SPI : Interface série synchrone
- **Multiplieur câblé**
- **Port JTAG** pour programmation directe et débogueur
- **Boîtier** 64LQFP de **48 pins**
- **Prix** : ~6\$

Pour plus de détail sur cette famille de microprocesseur, il faut se reporter au **Projet de Semestre** réalisé avant le diplôme.

3.3 Présentation

3.3.1 Mise en route

La mise en route a été de reprendre le projet (Workspace) de la couche physique adaptée. Cela a surtout permis d'avoir un premier contact avec le logiciel IAR en ouvrant ce projet, en le compilant et en programmant l'EEPROM MSP430. Pour la programmation, c'est la carte de test **RXTX_BB_V2** déjà utilisée pour URSAFE qui est employée. Cette première étape est suivie directement par des simulations de l'implémentation. L'envoi et la réception de données sont simulés par un **PC Linux**. Ces simulations fonctionnent sans interface radio (bande de base). Les seules fonctionnalités testées sont les primitives physiques touchant à l'envoi de données. Le test s'est déroulé avec succès, le PC envoi puis reçoit les données correctement via ses deux ports série RS-232.

La suite a été de créer un nouveau projet en reprenant les fichiers du projet de l'adaptation physique. La seule adaptation qui a dû être apportée concernent les chemins d'accès de l'appel des modules C (`#include`). De pouvoir reprendre le projet de l'adaptation physique donne la possibilité de se baser sur certaines structures de programme *.c qui existent déjà ce qui apporte un gain de temps pour la familiarisation de l'environnement et du code.

3.3.2 Généralité de l'implémentation

La voie suivie pour l'écriture du code est la suivante : les primitives MAC sont d'abord écrites sous forme de fonctions C. Au début, seule une ossature principale de chacune des fonctions est réalisée, puis celles-ci sont développées lorsqu'elles ont été testées par des applications.

On part du principe que le type de primitive *request* correspond à une réception de paramètres provenant d'une fonction et que le type *confirm* correspond à l'envoi d'arguments d'une fonction. En résumé, on se rend compte que l'on peut utiliser une seule fonction C pour les types de primitives *request* et *confirm*. Exemple : la couche supérieure fait une demande d'envoi de données à l'aide de certains paramètres, et une fois toute la procédure d'envoi réalisée (qui est contenue dans la fonction), un ou plusieurs arguments peuvent être retournés correspondant aux paramètres de la primitive de type *indication*. Généralement, c'est un statut qui est retourné à la couche supérieure pour l'informer du travail effectué.

Le programme principal (*main*) correspond à la couche supérieure de la couche MAC. L'implémentation, en général, se base sur une programmation séquentielle au niveau de cette couche applicative. Toutes les primitives doivent être exécutées depuis cette couche d'application, y compris les primitives qui devraient être envoyées par la couche MAC. De ne pas s'en tenir à une programmation de ce style, demande à ce que la couche d'application doive se tenir prête à toutes réceptions de données à tout moment. Il faudrait alors se diriger vers une programmation concurrente, dont il n'est pas envisageable à réaliser avec les ressources du système embarqué qui sont imposées. Le MSP430 n'est en effet pas prévu pour réaliser du multi-tâches.

Il y a néanmoins une fonctionnalité qui n'est pas réalisée d'une manière séquentielle. Il s'agit de l'envoi et la réception des trames Beacons (voir la partie du Watchdog Timer : 3.6.4.4). On est obligé d'utiliser les interruptions pour garantir que ces trames de synchronisations soient envoyées ou reçues à des temps précis.

La nécessité d'employer une programmation séquentielle demande de prévoir, au niveau du programme d'application, des périodes de temps pour la réception des primitives provenant de la couche MAC (exemple : association, arrivée de données,...). Ces cas correspondent aux primitives de type *indication*.

3.3.3 Organisation mémoires

Les microprocesseurs modernes proposent maintenant le choix entre une normalisation Little Endian (le byte de poids fort est en bas de la pile) ou Big Endian (le byte de poids fort est en haut de la pile). Il est donc important de mentionner que l'écriture du code (notamment pour le stockage et la transmission d'informations) a été pensée pour un fonctionnement **Big Endian** qui est le mode par défaut du microprocesseur.

Il est encore utile de remarquer que dans un byte, le bit de poids fort est à gauche et le bit de poids faible à droite. Un exemple pour un type de 16 bits résume cette normalisation des bits à la Figure 38.

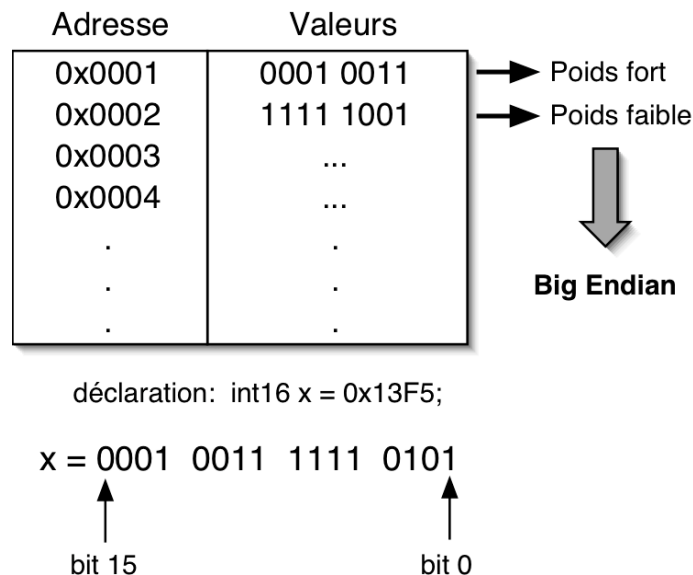


Figure 38 – Organisation mémoires (exemple d'un type de 16 bits)

3.3.4 Réserveation mémoires

Comme les trames MAC et PHY sont constituées directement en mémoire RAM, il est nécessaire de réserver de la mémoire à l'avance. La solution adoptée consiste à utiliser trois variables « Buffer » qui sont générées à l'initialisation. Leur taille correspond directement aux nombres de données maximum que la couche à le droit de transférer à la couche inférieure, les tailles des Buffers sont donc fixes. Les deux premiers Buffers servent à stocker les trames des couches MAC et PHY, tandis que le troisième Buffer est utilisé pour accueillir les données provenant des couches supérieures. L'avantage d'avoir trois réservations mémoires séparées pour chaque couche est que cela permet de retrouver plus rapidement des erreurs de programmations lors de debugage. L'utilisation d'un seul Buffer serait évidemment recommandée dans le cas d'une implémentation finale afin de réduire l'utilisation en mémoire.

Le « **Buffer UP** » récupère les données déjà segmentées par les couches supérieures. Sa taille correspond à la valeur maximum que peut accepter un MSDU, c'est à dire 102 bytes (constante *aMaxMACFrameSize*). Le « **Buffer MAC** » occupe 127 bytes en mémoire, correspondant à la constante *aMaxPHYPacketSize*. Enfin, le « **Buffer PHY** » doit s'adapter au paquet physique envoyé avec l'interface radio de URSafe qui utilise des paquets d'une longueur de 256 bytes (constante *RFPacketMacLength*). Les réservations mémoires des trois Buffers se trouvent à la Figure 39.

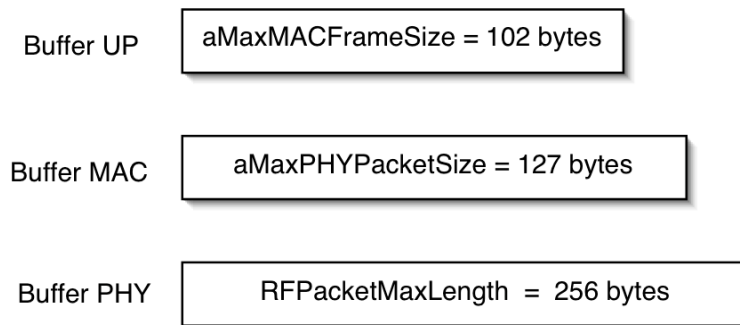


Figure 39 - Réservations mémoires avec Buffer UP, Buffer MAC et Buffer PHY

Une autre réservation mémoires importante devrait être réalisée dans le cas d'une implémentation d'un FFD, la liste d'attente des transmissions. Celle-ci doit idéalement contenir jusqu'à 7 trames de données ou de commande MAC, on obtient donc un total de $7 * 127 = 889$ bytes (Figure 40).

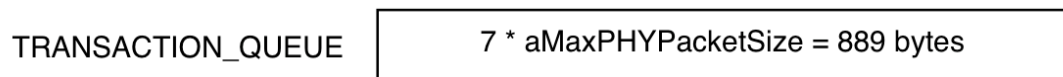


Figure 40 - Réservation mémoire supplémentaire pour l'implémentation FFD

Malgré le grand espace qu'occuperait cette dernière variable, il ne devrait pas y avoir de problème en ce qui concerne un éventuel dépassement des 2048 bytes disponibles en mémoire RAM.

Il s'est avéré que la création d'une liste de transmissions est inutile pour les applications réalisées par la suite. Cette réservation mémoires n'est donc pas implémentée. Par contre, le principe de la liste de transmissions est tout même utilisé, mais une seule place mémoire correspondant au Buffer MAC est réservée. La différence avec le standard se traduit alors que ce n'est pas le MSDU qui est placé dans la liste d'attente, mais directement la trame de données ou de commande déjà construite. L'attente de l'envoi de la trame se fait dans la primitive de données MCPS-DATA.

3.4 Descriptions des fichiers du projet

- **Application_XXX.c** : C'est le fichier qui contient toutes les définitions de **variables et constantes globales** utilisées par les couches MAC et PHY. C'est aussi le fichier qui contient le programme principal (**main**) servant à faire appel aux primitives MAC d'une manière séquentielle.
- **crc1.c & crc2.c** : Comme il existe deux implémentations de CRC, ceux-ci sont laissés sous forme de fichiers afin de pouvoir choisir rapidement le type d'implémentation de CRC qu'il faut. Le fichier à utiliser doit être renommé en **crc.c** (même principe pour les fichiers Headers).
- **encaps.c** : Ce fichier regroupe les **primitives PHY 802.15.4** qui ont directement été adaptées par rapport à la couche physique d'URSAFE sous forme de fonctions.
- **encaps.h** : C'est dans ce fichier qu'est décrit les **énumération PHY** servant à spécifier les statuts, les **constantes PHY** et les **variables globales** qui sont déclarées comme externes. Elles ont du être déclarées dans le fichier d'application avec le *main*.
- **IO.c (IO.h)** : Ces fichiers définissent principalement les deux fonctions **Read_Data_On_UART1** et **Send_Data_On_UART1** qui vont permettre de lire ou d'écrire sur les ports asynchrone RS-232.
- **mac.c**: Ce fichier regroupe toutes les **primitives MAC 802.15.4** décrites dans le standard sous forme de fonctions.
- **mac.h**: C'est dans ce fichier qu'est décrit les **énumérations MAC** servant à spécifier les statuts, l'**attribut MAC PIB**, le **PAN Descriptor**, les **constantes MAC** et les **variables globales** qui sont déclarées comme externes. Elles ont du être déclarées dans le fichier d'application avec le *main*.
- **macFunctions.c** : Ce fichier regroupe plusieurs fonctions qui sont utilisées au niveau MAC, notamment pour la construction des trames, la lecture des trames et pour l'envoi et la réception de diverses trames.
- **macFunctions.h** : Ce fichier regroupe principalement les paramètres TIB servant à faire passer des informations entre fonctions par variable globale sous forme d'une structure appelée Transfer Information Base (TIB).
- **msp430x14x.h** : Ce fichier contient des constantes et des fonctions du microprocesseur MSP430. À la place d'aller de devoir spécifier certaines adresses mémoires pour des registres d'états, des noms correspondant à ces accès mémoires sont créés dans ce fichier.
- **TimerA.c (TimerA.h) / TimerB.c (TimerB.h)** : Ces fichiers définissent les fonctions permettant de contrôler les Timers A et B. Les routines d'interruption des Timers sont présent dans les fichiers .c.
- **typedefs.h** : C'est le fichier contenant toutes les définitions de type.
- **UPClock.c (UPClock.h)** : Ces fichiers initialisent les horloges du microprocesseur.

- **UWBModuleDriver.c (UWBModuleDriver.h)** : Ces fichiers gèrent l'activation du CPLD, du récepteur et de l'émetteur via plusieurs fonctions. C'est ici qu'est décrit les deux fonctions d'envoi et de réception de donnée par UWB : *SendDataOverUWB* et *ReceiveDataOverUWB*.
- **Watchdog.c (Watchdog.h)** : Les fichiers du Watchdog décrivent principalement la routine d'interruption du Watchdog qui est employé pour l'envoi et la réception des Beacons.

3.4.1 Implémentation RFD et FFD

Comme plusieurs variables et primitives ne doivent pas être implémentées dans un RFD par rapport au FFD, il a fallu trouver une solution simple pour passer rapidement d'une implémentation à l'autre. La solution choisie a été d'avoir un code unique qui est utilisé pour les deux implémentations. Par exemple, lorsqu'une fonctionnalité (fonction ou variable) ne doit pas être implémentée pour un RFD, on entoure la fonctionnalité d'une paire de *#ifdef FFD ... #endif*.

Cette façon de procéder permet de ne pas réaliser des erreurs d'implémentations pour les dispositifs qui ne possèdent pas toutes les fonctionnalités. Cela sert également de savoir à tout moment, si on est en présence d'une implémentation RFD ou FFD. Lors de la programmation d'un module, il suffit d'indiquer au début du fichier *mac.h* un *#define RFD* ou un *#define FFD* pour choisir l'implémentation.

Par contre, de connaître que l'implémentation est un FFD ne détermine pas si c'est le coordinateur PAN. Pour le savoir, il faut réaliser le test de comparer l'adresse étendue unique de l'implémentation (*aExtendedAddress*) et la variable PIB *macCoordExtendedAddress*. Si le test est positif, alors nous sommes en présence du coordinateur PAN.

3.4.2 Variables et constantes globales

Toutes les variables et constantes globales du projet ont dû être déclarés dans le fichier *Application_xxx.c*. La raison est que le compilateur refuse de compiler si les variables et constantes globales ne sont pas dans le même fichier que le programme principale (*main*). Voici la liste de toutes les variables et constantes globales avec une brève description de leur fonctionnalité :

Variables globales des couche hautes :

- **Buffer_Up** : Cette réservation mémoire permet d'accueillir le MSDU de 102 octets provenant ou en destination des couches supérieures.

Variables et constantes globales de la couche MAC :

Variables générales :

- **Buffer_Mac** : Cette réservation mémoire permet de réaliser la construction et la lecture des trames MAC. Pour un coordinateur, il peut également être utilisé comme liste d'attente de transmissions.

- **TransmissionStatus** : Cette variables permet d'accueillir les statuts utilisés au niveau de la couche MAC. Elle est souvent égale à une primitive de type *.confirm*.
- **MAC_PIB** : Cette variable correspond à la structure unique du PAN Information Base de la couche MAC. Elle regroupe plusieurs informations à propos de l'implémentation MAC.
- **PAN_DES** : Cette variable correspond à la structure du PAN Descriptor qui est renvoyé à la couche supérieure lors d'un balayage actif ou passif. Il est également utilisé au niveau de la couche MAC pour garder à jour plusieurs variables décrivant l'état du PAN.
- **MAC_TIB** : Cette variable correspond à une structure (Transfer Information Base) permettant de faire passer des paramètres entre les fonctions de la couche MAC. Cela permet de ne pas déclarer trop d'argument pour ces fonctions et d'avoir une meilleure clarté dans le code.

Variables et constantes concernant le temps :

- **Time** : Cette variable 32 bits représente la référence du temps pour toute l'implémentation, et cela à tous moment. Elle est exprimée en nombre de symbole.
- **TimeMax** : Cette constantes indique tout simplement la valeur la plus haute que la variable *Time* puisse atteindre. Elle permet de gérer le rebouclage de la variable *Time* qui, il est vrai n'arrive que tous les 248 heures en utilisant la durée d'un symbole de l'interface physique employée (208 μ s).
- **AdaptSymbolTime** : Cette variable permet de faire le lien temporel entre le temps que dure un symbole et le temps mesuré par le comptage des Timers A et B. La référence de base utilisée pour les Timers est définie par le fait que la valeur de comptage 0x5000 est atteinte après environ 1 seconde.
- **StartChrono, StopChrono** : Ces variables permettent en premier lieu à réaliser des différences de temps. Elles sont surtout utilisées pour déterminer le temps que prend la réception et l'envoi d'un Beacon pour réaliser une correction à l'initialisation de la base de temps de la Superframe.

Constantes concernant des marges de sécurité temporelles :

- **MaxDurationOfExtraction** : Cette constante indique le nombre de symboles qu'il faut pour réaliser une récupération de données en attente. Cette valeur va influencer sur la valeur minimum possible de la partie active de la Superframe (SO), car l'implémentation limite à ce qu'une récupération de donnée doit se réaliser entièrement avant la fin de la partie active. Par défaut *MaxDurationOfExtraction* vaut 0x4000, ce qui limite le SO minimale à 5.
- **EnableReceiverBeforeBeacon** : Cette constante indique le nombre de symboles dont le récepteur est activé avant la venue théorique du Beacon à recevoir. Par défaut *EnableReceiverBeforeBeacon* est mis à 50.
- **SecurityTransmission** : Cette constante correspond au maximum de symboles que peut durer une transmission de donnée (y compris retransmissions et acquittements) après avoir effectué l'algorithme de CSMA-CA « slotted ». Par défaut *SecurityTransmission* est mis à 960 (*aBaseSuperframeDuration*).
- **SecurityEndOfCAP** : Cette constante correspond à la marge de symboles qu'il y a jusqu'à la fin du CAP de la partie active d'une Superframe. Si cette marge ne peut être tenue, la transmission ou la réception ne se fait pas. Par défaut *SecurityEndOfCAP* est mis à 960 (*aBaseSuperframeDuration*).

Variables pour la transmission indirecte :

- **FrameInTransactionQueue** : Cette variable permet à un coordinateur de savoir si des données sont présentes dans sa liste d'attente de transmissions. Cela permet à l'envoi de Beacons de spécifier l'adresse à qui est destinée la transmission en attente. Dans le cas d'un dispositif, cette variable est également utilisée. Elle permet d'indiquer qu'il faut réaliser une récupération de donnée en attente.
- **AddressShortFramePending** : Cette variable aide le coordinateur à savoir à qui appartient les données en attente dans la liste de transmissions en indiquant l'adresse courte du dispositif.

Sinon, il a été pratique de déclarer d'une manière globale les paramètres de retour de la fonction MCPS-DATA.indication afin de pouvoir les utiliser dans la routine d'exception du Watchdog et dans le programme d'application.

Variables globales de la couche PHY :

Variables générales :

- **Buffer_Phy** : Cette réservation mémoire sert à construire les trames physiques juste avant l'envoi. Il permet aussi de réceptionner les informations reçues par d'autres modules.
- **TimerA_TimeUp, TimerB_TimeUp** : Ces variables permettent de savoir si les Timer A ou B ont fini de compter. Elles peuvent être activées lors de l'exception des Timers.
- **TransceiverStatus** : Cette variable permet d'accueillir les statuts utilisés au niveau de la couche PHY. Elle est souvent égalée à une primitive de type *.confirm*.
- **PHY_PIB** : Cette variable correspond à la structure unique du PAN Information Base de la couche PHY. Elle regroupe plusieurs informations à propos de l'implémentation physique.

Variables permettant de tester le CSMA

- **AttemptChannel** : Cette variable permet d'avoir un comptage global du nombre d'accès au canal afin de parcourir un tableau de valeur simulant l'état du canal.

Il y a également quelques constantes qui ont été déclarées par *#define* dans le fichier *mac.h* afin que ces valeurs soient disponibles les programmes qui utilisent *mac.h* et qu'elles puissent être modifiées à un seul endroit.

- **Symbol** : Cette constante correspond directement à la durée d'un symbole en milliseconde. Avec l'interface physique employée, elle a dû être fixée à 208 ms.
- **SymbolPrecision** : Cette constante permet de réduire le nombre d'interruptions par une perte de précision concernant les mesures de temps. Sa valeur correspond au nombre de symboles qui sont incrémentés par interruption.
- **RFPacketMaxLength** : Cette constante correspond à la taille fixe des paquets qui sont échangés physiquement. C'est imposé par l'interface physique employée.

3.5 Conventions de l'écriture

Quelques conventions d'écriture ont été réalisées afin d'obtenir un code cohérent et facile à lire. Voici la manière de reconnaître les différents types de déclaration :

- **Variables locales, énumération de constantes et paramètres de fonctions** : Ces types de données débutent toujours leur nom par une miniscule suivie d'une majuscule pour chaque nouveau mot si le tout étant collé. Si les mots sont séparés par des underscores, le mot suivant est tout en minuscule.

Exemple : *crc, out_of_cap, commandType, aUnitBackoffPeriod*

- **Variables globales, #define, énumération de type de commande ou de trame et champs composant une structure** : Ces types de données débutent toujours par une majuscule comme tous les mots composant leur nom. Les mots peuvent être collés ou séparés par des underscores. Plusieurs majuscules peuvent être suivies.

Exemple : *MAC_PIB, Symbol, SuperframeOrder, Data_Request, Ack*

- **Fonctions des primitives** : Le nom de la fonction est tout en majuscule et séparé par des underscores. Seul le type de la primitive, s'il y en a un, est en minuscule.

Exemple : *MCPS_DATA, MLME_ASSOCIATE_indication*

- **Fonctions diverses** : Le nom de la fonction a toujours la première lettre en majuscule, puis les mots sont collés avec toujours la première lettre du mot en majuscule.

Exemple : *Wait(),ConvIntToByte(),SendAck(),ReceiveBeacon()*

Afin de retrouver rapidement les fonctionnalités qui ne sont pas encore implémentées, le commentaire NOT IMPLEMENTED est placé dans le code à la place de la fonctionnalité.

3.6 Implémentation C des principales fonctionnalités

3.6.1 Type de données

Voici la définition des types utilisés pour toutes l'implémentation, ils sont décrits dans le fichier *typedef.h* :

```
typedef unsigned char bool;  
typedef unsigned int uint;  
typedef unsigned char uint8;  
typedef signed char int8;  
typedef unsigned short uint16;  
typedef signed short int16;  
typedef unsigned long uint32;  
typedef signed long int32;  
typedef unsigned long long uint64;
```

Généralement ce sont uniquement les types non-signés (unsigned) qui sont employés, car il n'y a pas d'utilité à se servir des nombres négatifs. Le type *bool* est défini pour qu'un 0 corresponde à FALSE et un 1 à TRUE.

3.6.2 CSMA-CA « slotted » et MLME-RX-ENABLE

Ces deux fonctionnalités sont très importantes, car ce sont elles qui doivent garantir si une émission ou une réception ont le droit d'être réalisées à un instant précis. Cela permet de certifier que le Transceiver n'est ni dans un mode d'émission, ni dans un mode de réception et cela durant toute partie inactive de Superframe et à l'approche d'un envoi ou d'une réception de Beacons. Plusieurs constantes concernant des marges de sécurité temporelles sont utilisées pour ces deux fonctionnalités, il faut se reporter au point 3.4.2 pour avoir des détails sur ces constantes globales.

3.6.2.1 CSMA-CA « slotted »

L'implémentation est légèrement modifiée comparée au standard. Comme le but est de savoir si l'envoi de données peut être oui ou non réalisé durant la Superframe courante, une constante déterminant la limite de l'envoi de données est définie ; *SecurityEndOfCAP*. Elle indique le nombre de périodes de Backoff avant la fin du CAP. Si la valeur actuelle du temps est plus grande que cette constante, la transmission ne se réalise pas, sinon elle peut se faire. Par défaut, la marge définie par cette variable est de *aBaseSuperframeDuration* (960 symboles).

Autrement, une fonctionnalité supplémentaire a été ajoutée au CSMA-CA « slotted » afin qu'il n'y ait pas d'échec à l'accès au canal durant une période inactive de Superframe. Lorsque l'algorithme du CSMA doit, pour la première fois, retrouver une période de Backoff est que celle-ci se trouve dans la partie inactive de la Superframe, une attente passive se fait dans le CSMA jusqu'au début de la prochaine période active. Le défaut de cette méthode est son manque d'optimisation pour une basse consommation si la partie inactive est longue.

3.6.2.2 MLME-RX-ENABLE

Sa fonctionnalité a dû être modifiée par rapport à l'attente de l'activation du récepteur qui ne pouvait pas se réaliser dans la primitive, puisque plusieurs cas de réception sont possible (trame de commande ou trame de données). Cette primitive permet surtout de calculer par rapport à l'instant que l'on se trouve, combien de temps au maximum le récepteur pourra être activé. Donc, si les paramètres *RxOnTime* et *RxOnDuration* font en sorte que le temps dépasse la période active d'une Superframe, le statut *INVALID_PARAMETER* n'est pas renvoyé. À la place, le Timer B est initialisé à la valeur qui correspond au maximum possible de l'activation du récepteur jusqu'à la fin de la partie active de la Superframe. Pour signaler que le temps d'attente spécifié par le paramètre *RxOnDuration* ne pourra être attendu entièrement, c'est le statut *OUT_OF_CAP* qui est renvoyé. L'attente est de toute façon réalisée après l'exécution de cette primitive au niveau physique. Il faut néanmoins que la primitive physique *PD-DATA.indication* soit appelée après la primitive *MLME-RX-ENABLE*.

Lorsque le paramètre *RxOnTime* vaut 0, le mode de réception est enclenché le plus rapidement possible après le Beacon du début de la Superframe. Sinon, *RxOnTime* doit valoir au moins le temps que dure le Beacon. Si le paramètre *Deferpermit* est actif, l'enclenchement du récepteur se fait dans tous les cas au début de la prochaine Superframe.

Elle emploie également la constante *SecurityEndOfCAP* servant à avoir une marge temporelle des opérations qui doivent encore se dérouler après cette primitive.

3.6.3 CRC

L'implémentation du CRC a été réalisée à partir de codes C proposés librement sur Internet. Au final, deux choix d'implémentations ont été rendus possibles par quelques recherches.

La première d'entre elles correspond à une implémentation qui calcule le CRC à partir d'un certain polynôme générateur et d'une initialisation des valeurs du CRC en indiquant évidemment l'adresse mémoire de départ des valeurs à coder et de la longueur de ces valeurs. Cette implémentation est très pratique, car elle intègre directement plusieurs algorithmes de CRC (16 ou 32 bits) dont le XMODEM correspond par sa description directement à ce que le standard 802.15.4 exige. Cette implémentation utilise donc peu de mémoire, mais plus de ressources processeur puisque tout l'algorithme doit être calculé. Le code de cette implémentation correspond à *crc1.c* et *crc1.h* (**Annexe E**).

La deuxième implémentation repose sur un tableau déjà calculé qui correspond aux coefficients du polynôme générateur. Ce dernier concorde évidemment à celui requis par le standard 802.15.4. La valeur d'initialisation du CRC peut quant à elle être choisie, elle a donc été mis à 0 comme l'exige le standard. Au final, cette implémentation demande un peu plus de ressources mémoires, mais sollicite moins de ressources processeur. Le code de cette deuxième implémentation correspond à *crc2.c* et *crc2.h* (**Annexe E**).

Le résultat des deux implémentations de CRC est exactement identique. Comme le but n'est pas de réaliser des tests de performance, le choix de l'implémentation utilisée est arbitraire. Pour certaines applications, il a néanmoins été préféré d'utiliser la deuxième implémentation pour une raison de rapidité afin de respecter les constantes de temps introduites par le standard.

3.6.4 Timers & Watchdog

La bonne gestion des trois Timers à disposition est capitale, car dans un réseau « Beacon Enabled », plusieurs variables sont dépendantes de la mesure du temps. L'utilisation de variables temporelles demande des précautions au niveau de la programmation. Encore plus au niveau des tests de condition dont l'opérateur d'égalité ne doit jamais être employé, seuls les opérateurs < ou > doivent être utilisés afin d'être certain de ne pas manquer de condition.

Au niveau du microprocesseur, il n'est pas possible de garantir des temps exacts, car les mesures de temps se font d'une manière logicielle (il faudrait tenir compte du temps qu'utilise certaines instructions assembleurs). Ce sont toujours les temps théoriques calculés exacts et les valeurs des constantes exactes qui sont employés, aucun arrondissement n'a jamais été réalisé.

3.6.4.1 Démarche vers la solution finale

La première idée a été d'utiliser un Timer par couche, le Timer B étant utilisé au niveau physique, le Timer A au niveau de la couche MAC et le Watchdog servant à tenir une base de temps et également de gérer l'envoi et la réception des Beacons dans un réseau « Beacon Enabled ».

Il s'est vite avéré que cette solution possédait un gros désavantage, le temps n'est plus mesuré lors de la construction et l'envoi de Beacon ou durant la réception et la lecture du Beacon. La raison est que la gestion des Beacons se fait dans la même exception du Watchdog que le calcul de la base de temps. La difficulté est que l'on est incapable de savoir combien de temps s'écoule durant cette gestion des Beacons, Il faudrait alors réaliser des estimations à l'avance qui ne sont jamais faciles à faire. Cette idée a été définitivement abandonnée, lorsque que l'on se rend compte qu'une extraction de données est envisageable à réaliser par le Watchdog si le Beacon reçu indique la présence de données chez le coordinateur. Ce qui implique que durant tout le processus, il n'y aurait pas de base de temps pour l'algorithme du CS MS-CA « slotted », ce qui n'est pas toléré.

La piste suivante a été de sacrifier un Timer pour incrémenter les variables temporelles, le Timer A en l'occurrence. Le Watchdog est alors uniquement utilisé pour la gestion des Beacons dans un réseau « Beacon Enabled ». Cette solution permet de ne jamais arrêter de compter les valeurs temporelles et donc de connaître exactement quand survient le temps 0. Pour un coordinateur, cela correspond exactement à l'instant où le Beacon commence à être envoyé. Pour un dispositif, cela correspond au moment où le premier bit du Beacon est détecté à la réception. Cette solution suppose tout d'abord que les exceptions du Timer A puissent être levées durant la routine d'exception du Watchdog Timer. Cette solution implique également que toutes les mesures de temps au niveau MAC qui ont été réalisées avec le Timer A doivent être repensées par rapport à sa nouvelle fonctionnalité. Le problème qui survient alors, est que pour un réseau « Non Beacon Enabled », le Timer A n'est pas enclenché puisque les variables temporelles à incrémenter ne servent qu'à connaître quand le dernier Beacon a été envoyé ou reçu. Il est alors impossible de calculer les mesures de temps qu'a besoin la couche MAC (attentes d'ACK,...).

L'idée finale est d'utiliser une variable globale (*Time*) servant à connaître le temps en symboles écoulés à tout moment. Cette variable est incrémentée par le Timer A en réalisant une exception à chaque symbole. Le Timer A fonctionne donc dès l'initialisation de l'implémentation tant pour un coordinateur PAN que pour un dispositif réduit. La variable sera de 32 bits afin de pouvoir garantir les plus longs temps et est rebouclée sur elle-même quand elle arrive à son maximum.

3.6.4.2 Timer A

Le Timer A sert à gérer les variables indiquant le temps. Idéalement, l'exception du Timer A doit être levée tous les symboles, c'est-à-dire tout les 208 μ s. Trois variables ont la possibilité d'être incrémentées à chaque exception :

- **Time:** Cette variable globale de 32 bits correspond à la base de temps pour l'implémentation en général. Elle est incrémentée en tout temps.
- **macBeaconTxTime:** Cette variable PIB de 32 bits (24 suffisent) est employée lorsque le réseau est « Beacon Enabled » par le coordinateur PAN. Elle permet de connaître le temps qui s'est écoulé en nombre de symboles depuis le dernier envoi de Beacon.
- **TimeStamp:** Cette variable du PAN Descriptor de 32 bits (24 suffisent) est employée lorsque le réseau est « Beacon Enabled » par les dispositifs associés au PAN. Elle permet de connaître le temps qui s'est écoulé en nombre de symbole depuis la réception du dernier Beacon.

À propos de la taille des variables *macBeaconTxTime* et *TimeStamp*, 24 bits suffisent car cela représente une durée maximum de 58 minutes, tandis que l'intervalle maximum possible entre deux Beacon (BO = 14) est de 56 minutes. Le type utilisé est tout de même de 32 bits, car on ne peut pas déclarer de type de 24 bits.

Les mesures de temps à l'aide du Timer A se déroule de la façon suivante. Lorsque la mesure doit commencer, on égalise la variable globale *StartChrono* au temps actuel de la variable *Time*. Lorsque la mesure doit être arrêtée, c'est la variable globale *StopChrono* qui est égale au temps actuel. En comparant les deux variables *StartChrono* et *StopChrono*, on connaît alors le temps qui s'est écoulé en nombre de symboles.

Sur ce principe de mesure de temps, deux fonctions ont été réalisées dans le fichier *MacFunctions.c* :

- **Wait(uint16 n)**: Cette fonction réalise tout simplement une attente passive de n symboles.
- **bool UpChrono(uint16 timeDuration)**: Cette fonction permet de savoir si le temps *timeDuration* en nombre de symboles a été dépassé ou non.

3.6.4.3 Timer B

Le Timer B est utilisé au niveau de la couche physique, où il sert à lever une exception lorsqu'un envoi ou une attente de données a pris trop de temps. Il peut parfois être initialisé au niveau de la couche MAC afin de jouer avec différentes attentes dépendant des envois ou réceptions des types de paquets. Le compteur est incrémenté par une horloge de 32 kHz, mais comme l'incrémentation est faite de manière software, le temps n'est pas exact. Voici, d'après ce qui a été mesuré, la correspondance pour l'adaptation temporelle : **0x5000 = 1 seconde**. Cette adaptation se fait via la variable *adaptSymbolTime*. L'imprécision des mesures de temps réalisée avec ce Timer B n'est pas grave, car celui-ci n'est jamais employé pour réaliser des mesures de constantes ou de variables MAC.

3.6.4.4 Watchdog Timer

Le Watchdog Timer remplit la fonction de gestion de Beacons au niveau envoi (coordinateur PAN) et réception (dispositif du PAN) lorsque le réseau est « Beacon Enabled ». Sa fonction peut aller plus loin lorsque des données sont à récupérer chez le coordinateur. Son programme consiste en premier lieu à tester si les variables *macBeaconTxTime* (pour un coordinateur) et *TimeStamp* (pour un dispositif) correspondent à l'intervalle des Beacons (calcul de BI) défini dans le PAN. Ces tests doivent être réalisés au moins aussi rapidement que l'intervalle des symboles, afin de ne pas dépasser la condition de plus d'un symbole. Comme le Watchdog ne peut pas lever une exception avec un temps personnalisé, il a fallu choisir la valeur la plus proche en dessous d'une valeur de symboles (208 us). C'est 125 us qui a dû être choisi. L'exception du Watchdog se lève donc tous les 125 us.

Les valeurs de temps dont le Watchdog peut utiliser pour ses interruptions se trouvent dans le fichier *mcp430x14x.h*. La constante choisie est *WDT_MDLY_0_5* qui veut dire que l'interruption survient tous les 500 us (avec une horloge de 1 MHz). Comme c'est une horloge de 4 MHz qui est employée, l'interruption interviendra tous les 125 us.

Lorsque le réseau est « Beacon Enabled », le Beacon doit être compris dans le temps d'une Superframe. Une correction de temps doit être faite, car il faut tenir compte du temps que prend l'envoi ou la réception du Beacon. Cette correction ne peut se faire qu'après l'envoi ou la réception du Beacon. Pour l'envoi, on initialise la variable *macBeaconTxTime* avec le nombre de symbole qui se sont écoulés durant l'envoi du Beacon grâce au calcul du temps par la variable globale *Time* du Timer A. Il en est du même principe pour la réception, mais pour la variable *TimeStamp*.

Si une extraction de données doit être réalisée à la récupération de trames de données, il faut également que le MSDU soit transmis aux couches plus hautes via la fonction *MCPS-DATA.indication* qui elle fait appel à la fonction *Send_Data_On_UART1*.

Il ne faut pas oublier de sauvegarder toutes les variables globales utilisées pour l'envoi ou réception des Beacons au début de l'interruption afin de pouvoir les restaurer à la fin de l'interruption.

Lorsque le réseau est « Non Beacon Enabled », le Watchdog est désactivé.

3.6.4.5 Priorité des interruptions

La solution d'implémentation décrite précédemment demande que, durant la routine d'exception du Watchdog, la routine d'exception du Timer A puisse s'exécuter. En feuilletant le « User Guide » du MSP430 de Texas, il semble que lorsque le Watchdog est utilisé en mode Timer, il a justement la même priorité que les autres Timers. Mais cela ne suffit pas pour que l'opération fonctionne car, par défaut durant une routine d'exception, toutes les interruptions de même priorité sont interdites. Pour connaître l'état d'autorisation des interruptions à se lever, il suffit de visualiser le registre d'état du microprocesseur. Si le bit GIE (General Interrupt Enable) est à 1, les interruptions sont autorisées et à 0, les interruptions ne sont pas autorisées.

L'opération à réaliser est donc de rendre actif les exceptions durant la routine d'interruption du Watchdog afin que l'interruption du Timer A puisse se lever. Pour cela, il existe deux fonctions rendant actif ou inactif les interruptions de priorité « général » :

- `_EINT()` : Enable Interrupt
- `_DINT()` : Disable Interrupt

La subtilité est qu'en activant les interruptions à l'intérieur d'une routine d'interruption avec la fonction `_EINT()`, l'interruption peut elle-même se rappeler si la routine d'interruption n'a pas finie d'être exécutée. Il a donc encore fallu spécifier durant l'exécution de la routine d'interruption du Watchdog la désactivation de celle-ci avec la fonction *DisableWatchDogInterrupt*. Il faut également ne pas oublier de réactiver les interruptions du Watchdog en sortant de la routine d'interruption avec la fonction *EnableWatchDogInterrupt*.

3.6.4.6 Optimisation des interruptions

Durant certaines applications, le microprocesseur se bloquait lorsqu'en plus des interruptions du Timer A et du Watchdog, il réalisait des autres tâches. Il s'est donc avéré que la cause provenait d'une apparition trop rapide des exceptions. Il a été nécessaire d'optimiser la gestion des interruptions afin de libérer des ressources processeur.

Une solution de facilité est d'augmenter la constante globale *symbol* qui correspond au temps que dure un symbole. Les exceptions du Timer A surviennent alors moins souvent, mais il se trouve que tout le système fonctionne au ralenti, même si cela se déroule d'une façon correcte. Les temps ne sont donc plus respectés. Cette solution n'a pas été gardée.

Il a donc fallu trouver une autre méthode pour diminuer le nombre d'interruptions du Timer A. Comme le symbole est l'unité temporelle de base pour toutes les constantes et variables temporelles, il est possible de trouver des diviseurs communs à leurs valeurs. Il s'est avéré que toutes ces valeurs sont toujours paires, ce qui implique tout simplement que cela ne gêne pas que l'exception du Timer A soit levée seulement tous les deux symboles. Le principe peut même être poussé jusqu'à plus de symboles, le compromis étant que l'on gagne des ressources processeurs par cette augmentation, mais que l'on perd en précision pour les mesures de temps.

Afin de réaliser cette optimisation, la constante globale *SymbolPrecision* a été définie. Le principe est que le Timer A lève la valeur de *SymbolPrecision* fois moins ces interruptions. Les variables temporelles dans la routine d'interruption du Timer A (Time, BeaconTxTime, Timestamp) sont donc incrémentées de la valeur de *SymbolPrecision*.

Par défaut *SymbolPrecision* a été mis à 2, l'interruption du Timer A est donc levée deux fois moins souvent et les variables de temps de la routine d'interruption sont incrémentées par multiple de 2. Si *SymbolPrecision* est augmenté, toutes les constantes ou variables de temps du standard vont alors s'arrondir au multiple le plus proche de *SymbolPrecision*.

Pour gagner encore plus de ressources processeurs, lorsque le réseau est « Beacon Enabled », il est également possible de modifier la cadence des interruptions du Watchdog. À la place d'être appelé tous les 125 µs avec la constante WDT_MDLY_0_5 (horloge externe de 4 MHz), elle pourrait l'être tous les 2 ms avec la constante WDT_MDLY_8 qui est la valeur la plus proche au-dessus. L'envoi des Beacons est alors réalisé d'une façon beaucoup moins précise, puisque la base de temps si *SymbolPrecision* = 2 vaut théoriquement 416 µs. L'envoi des Beacons deviendrait donc précis à seulement 2 ms près.

Dans le code final, le choix a été de privilégier un meilleur fonctionnement plutôt qu'une bonne précision. Les intervalles des interruptions ne sont donc pas au minimum ; *SymbolPrecision* = 4 et c'est la constante WDT_MDLY_8 qui employée. Cela permet d'avoir un nombre d'interruption restreint, mais en conservant une précision de 2 ms tout à fait valable.

3.6.5 Modifications de l'implémentation physique

Quelques modifications ont dû être apportées au programme « *encaps.c* », qui regroupe les primitives physiques, soit pour une question d'adaptation par rapport aux variables globales ou soit par corrections de quelques égarements dans le codage (heureusement peu nombreux).

- Tout d'abord la fonction PD-DATA contenait un petit Bug qui provoquait l'effacement systématique des 5 derniers bits du paquet physique. La raison est que la boucle qui remplissait le reste du paquet par des zéros ne tenait pas compte des 5 bytes de l'en-tête physique 802.15.4. Il a donc fallu décaler le comptage de la boucle de 5 unités.
- La plus grosse modification concerne la fonction PD-DATA.indication qui a pratiquement dû être entièrement réécrite. En effet, cette primitive qui sert à récupérer les trames physiques réalisait l'adaptation des en-têtes 802.15.4 dans le mauvais sens. C'est-à-dire qu'à la place de supprimer les en-tête physiques, elle les ajoutait. En fait, tout est relatif à ce que l'on va réceptionner comme trame, si c'est une trame URSAFE qui arrive, aucune modification ne doit être apportée. Mais dans le cas d'échange de paquets 802.15.4, comme cela a été réalisé dans toutes les applications réceptionnant des données, la fonction PD-DATA doit écarter les en-tête physiques afin de transmettre uniquement le MPDU à la couche MAC.
- Les fonctions PLME_GET et PLME-SET ont dû être retouchées afin que la couche MAC puisse correctement lire ou écrire des données appartenant au PIB physique.
- Finalement la fonction PLME-CCA a dû être modifiée afin de convenir aux procédures de test. Le code étant en place dans cette fonction sert à déterminer si le canal est libre ou pas lorsque l'interface radio est utilisée. Comme tous les tests sont réalisés en bande de base, le code a été remplacé par un tableau de valeurs simulant les états du canal.

3.6.6 Traitement sur les bits

Beaucoup de traitement sur les bits doivent être réalisés pour la construction et la lecture des trames. Voici les opérateurs les plus utilisés :

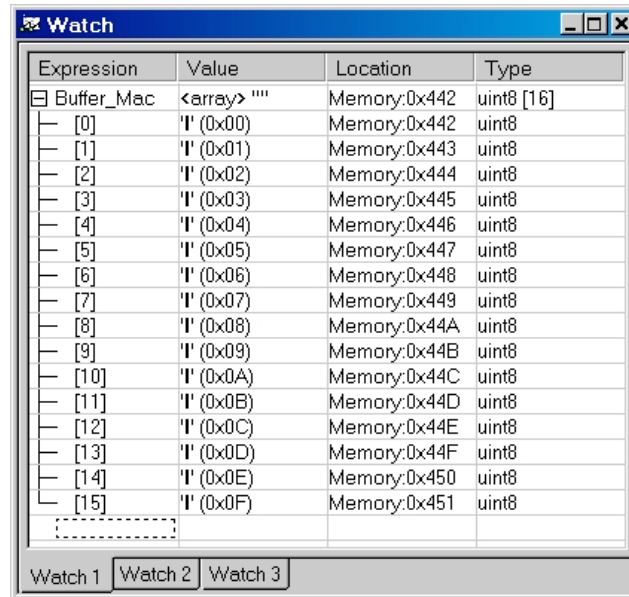
- **&** : Cet opérateur est très utilisé pour la lecture des trames puisque qu'il permet de masquer certains bits. Exemple : **&15** met à zéro tous les bits sauf les 4 bits de poids faible ($15 = 1+2+4+8$).
- **|** : Cet opérateur est très utilisé pour la construction des trames puisque qu'il permet de mettre actif les bits indiqués. Exemple : **|15** met à un les 4 bits de poids faible ($15 = 1+2+4+8$).
- **<< ou >>** : Ces opérateurs permettent de décaler des bits à gauche (<<) ou à droite (>>) un certain nombres de fois. L'opérateur << est également utile pour réaliser les puissances de 2.

3.7 Tests des primitives MAC de données

Ce test a pour principal but de vérifier la fonction C correspondant aux primitives MCPS_DATA.request et MCPS_DATA.confirm. Il est réalisé dans le mode Simulator (simulation sous Windows). Dans le fichier *application.c*, sont définis les paramètres essentiels à la construction de la trame MAC. Ces données proviennent normalement de la couche supérieure, ici elles sont personnalisées de toute pièce.

- `srcAddrMode = 3 ; //64 bits`
- `srcPANid = 0xABCD ;`
- `srcAddr = 0x01030507090AoCoF`
- `dstAddrMode = 0 ; //0 bits`
- `dstPANid = 0x4321 ;`
- `dstAddr = 0 ;`
- `msduLength = 15 ;`
- `TxOption = 0 ;`
- Données (msdu) : `for (i=0 ; i<= msduLength ; i++) *(msdu +i) = i ;`

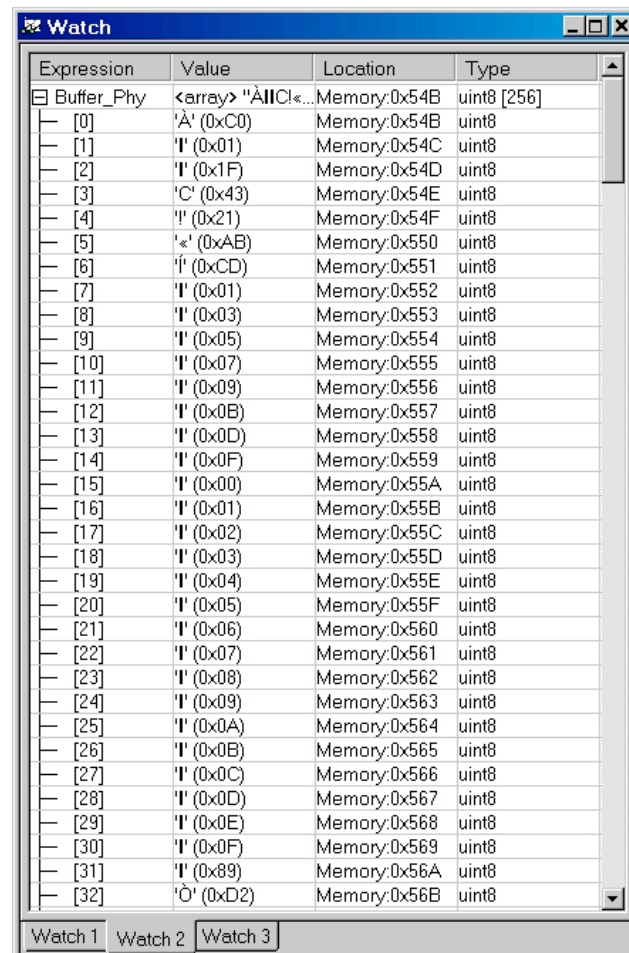
Les résultats obtenus en mémoire sont directement tirés de la fenêtre Watch du débogueur. Tout d'abord le MSDU qui correspond aux données reçues de la couche supérieure est stocké dans le Buffer_Mac, les valeurs stockées peuvent être visualisées à la Figure 41. Ces données correspondent aux valeurs engendrées par la boucle *for* utilisée.



Expression	Value	Location	Type
Buffer_Mac	<array> ""	Memory:0x442	uint8 [16]
[0]	'I' (0x00)	Memory:0x442	uint8
[1]	'I' (0x01)	Memory:0x443	uint8
[2]	'I' (0x02)	Memory:0x444	uint8
[3]	'I' (0x03)	Memory:0x445	uint8
[4]	'I' (0x04)	Memory:0x446	uint8
[5]	'I' (0x05)	Memory:0x447	uint8
[6]	'I' (0x06)	Memory:0x448	uint8
[7]	'I' (0x07)	Memory:0x449	uint8
[8]	'I' (0x08)	Memory:0x44A	uint8
[9]	'I' (0x09)	Memory:0x44B	uint8
[10]	'I' (0x0A)	Memory:0x44C	uint8
[11]	'I' (0x0B)	Memory:0x44D	uint8
[12]	'I' (0x0C)	Memory:0x44E	uint8
[13]	'I' (0x0D)	Memory:0x44F	uint8
[14]	'I' (0x0E)	Memory:0x450	uint8
[15]	'I' (0x0F)	Memory:0x451	uint8

Figure 41 - Buffer_Mac (test)

C'est autour du MSDU que les champs de la couche MAC sont construits. Le Buffer_Phy contient alors, après construction, les valeurs de la Figure 42. Seuls 33 bytes du Buffer sont utilisés sur les 256 de prévus, le reste des bytes étant mis à 0.



Expression	Value	Location	Type
Buffer_Phy	<array> "AIIII<..."	Memory:0x54B	uint8 [256]
[0]	'A' (0xC0)	Memory:0x54B	uint8
[1]	'I' (0x01)	Memory:0x54C	uint8
[2]	'I' (0x1F)	Memory:0x54D	uint8
[3]	'C' (0x43)	Memory:0x54E	uint8
[4]	'I' (0x21)	Memory:0x54F	uint8
[5]	'<' (0xAB)	Memory:0x550	uint8
[6]	'I' (0xCD)	Memory:0x551	uint8
[7]	'I' (0x01)	Memory:0x552	uint8
[8]	'I' (0x03)	Memory:0x553	uint8
[9]	'I' (0x05)	Memory:0x554	uint8
[10]	'I' (0x07)	Memory:0x555	uint8
[11]	'I' (0x09)	Memory:0x556	uint8
[12]	'I' (0x0B)	Memory:0x557	uint8
[13]	'I' (0x0D)	Memory:0x558	uint8
[14]	'I' (0x0F)	Memory:0x559	uint8
[15]	'I' (0x00)	Memory:0x55A	uint8
[16]	'I' (0x01)	Memory:0x55B	uint8
[17]	'I' (0x02)	Memory:0x55C	uint8
[18]	'I' (0x03)	Memory:0x55D	uint8
[19]	'I' (0x04)	Memory:0x55E	uint8
[20]	'I' (0x05)	Memory:0x55F	uint8
[21]	'I' (0x06)	Memory:0x560	uint8
[22]	'I' (0x07)	Memory:0x561	uint8
[23]	'I' (0x08)	Memory:0x562	uint8
[24]	'I' (0x09)	Memory:0x563	uint8
[25]	'I' (0x0A)	Memory:0x564	uint8
[26]	'I' (0x0B)	Memory:0x565	uint8
[27]	'I' (0x0C)	Memory:0x566	uint8
[28]	'I' (0x0D)	Memory:0x567	uint8
[29]	'I' (0x0E)	Memory:0x568	uint8
[30]	'I' (0x0F)	Memory:0x569	uint8
[31]	'I' (0x89)	Memory:0x56A	uint8
[32]	'O' (0xD2)	Memory:0x56B	uint8

Figure 42 - Buffer_Phy (test)

Afin de vérifier si la construction de la trame est correcte, on parcourt les valeurs hexadécimales dans l'ordre croissant afin de les analyser pour découvrir à quoi correspondent les valeurs :

- **Frame Control (co 01)**

Les deux premières valeurs hexadécimales correspondent donc aux 16 premiers bits de l'en-tête MAC qui se trouve être le champ du Frame Control. Le détail du champ est présenté à la Figure 43.

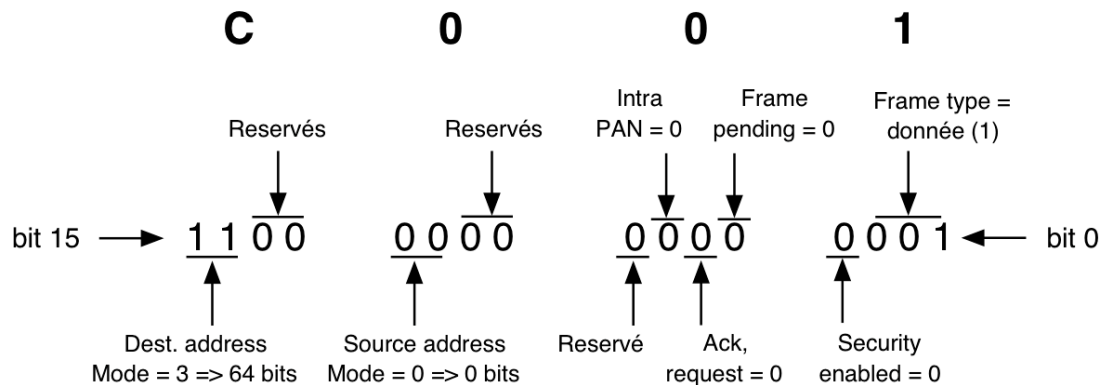


Figure 43 - Valeurs Hexadécimales du Frame Control

- **Data Sequence Number (1F)**

Ce nombre est généré aléatoirement entre les valeurs décimales 0 et 255 . 1F est donc une valeur hexadécimale aléatoire.

- **Address Info**

Les valeurs obtenues correspondent effectivement à celles qui ont été choisies dans le programme d'application :

- Destination PAN identification : 0x4321 (2 bytes)
- Destination Address : - (0 bytes)
- Source PAN identification : 0xABCD (2 bytes)
- Source Address : 0x01030507090AoCoF (8 bytes)

- **MSDU (données)**

On retrouve bien les 15 octets de données (MSDU) qui ont été générés par la boucle *for* présentée précédemment.

- **CRC (89D2)**

Les derniers deux bytes du Buffer correspondent au 16 bits du CRC. Pour vérifier rapidement ce résultat, l'utilisation d'un site internet (voir Liens Internet) a été requis. Il a suffi d'indiquer le polynôme générateur (0x1021), la valeur d'initialisation (0) et la séquence de 31 bytes à coder (Figure 42). Le résultat obtenu (0x89D2) correspond effectivement aux résultats de l'implémentation comme le prouve la capture d'écran de la Figure 44.

CRC parameters

CRC order (1..64)	16	
CRC polynom (hex)	1021	reverse!
Initial value (hex)	0	convert!
Final XOR value (hex)	0	

☐ nondirect ☒ direct

☐ reverse data bytes ☐ reverse CRC result before Final XOR

clear CRC-CCITT CRC-16 CRC-32

Data sequence

%C0%01%1F%43%21%AB%CD%01%03% clear

Result

89D2 (hex), 31 data bytes compute!

Figure 44 - Résultat de la simulation de CRC (capture Web)

Ce premier test a permis de prendre en main le débogueur et à bien assimiler la façon dont les bits doivent être organisés en mémoire. Avant l'obtention des résultats adéquats, plusieurs erreurs provenant principalement des opérateurs sur les bits et des bornes dans les boucles *for* ont dû être corrigées.

Le même genre de test est également effectué pour la fonction `MCPS_DATA_Indication`, où il a fallu s'assurer que les trames reçues sont correctement interprétées par la couche MAC en renvoyant les bonnes informations sous forme de paramètres à la couche supérieure.

3.8 Applications

Les applications réalisées tout au long du projet servent en premier lieu à vérifier les fonctionnalités importantes et à corriger les éventuelles erreurs de programmation. C'est en implémentant également le programme d'application dans le microprocesseur que l'on va pouvoir tester les fonctionnalités MAC de notre choix.

3.8.1 Cartes de test RXTX_BB V2

Pour ces applications, ce sont directement les cartes de test RXTX_BB V2 déjà employées pour URSAFE qui sont utilisées. Le microprocesseur est donc programmé via ces cartes. Celles-ci servent encore à d'autre chose, notamment à pouvoir transmettre les informations entre cartes via une transmission série câblée. L'avantage de pouvoir faire de la transmission en bande de base est que l'on va éviter les nombreux problèmes que peut poser la transmission radio, notamment en ce qui concerne les interférences. La carte contient également deux ports RS-232 qui permettent alors la réception et l'envoi de données à partir d'un PC. Enfin, des Leds permettent d'entrevoir le sens des communications. Lors de débogage, elles sont très souvent employées pour connaître le mode actuel de transmission (rx_on, tx_on ou trx_off)

Voici un petit descriptif de la carte qui est présentée à la Figure 45.

- | | |
|----------------------|---|
| 1 : Port RX0 - TX433 | 9 : LED TX433 (UWB) |
| 2 : Port CPLD JTAG | 10 : Port RS-232-1 |
| 3 : Port RX1 | 11 : Connecteur power (Batt) |
| 4 : Port RS-232-0 | 12 : Port TX |
| 5 : LED Power | 13 : Port Reserved (transmissions entre cartes) |
| 6 : LED TX | 14 : Bouton Reset |
| 7 : LED RX1 | 15 : Port JTAG |
| 8 : LED RX0 | 16 : MSP 430 |

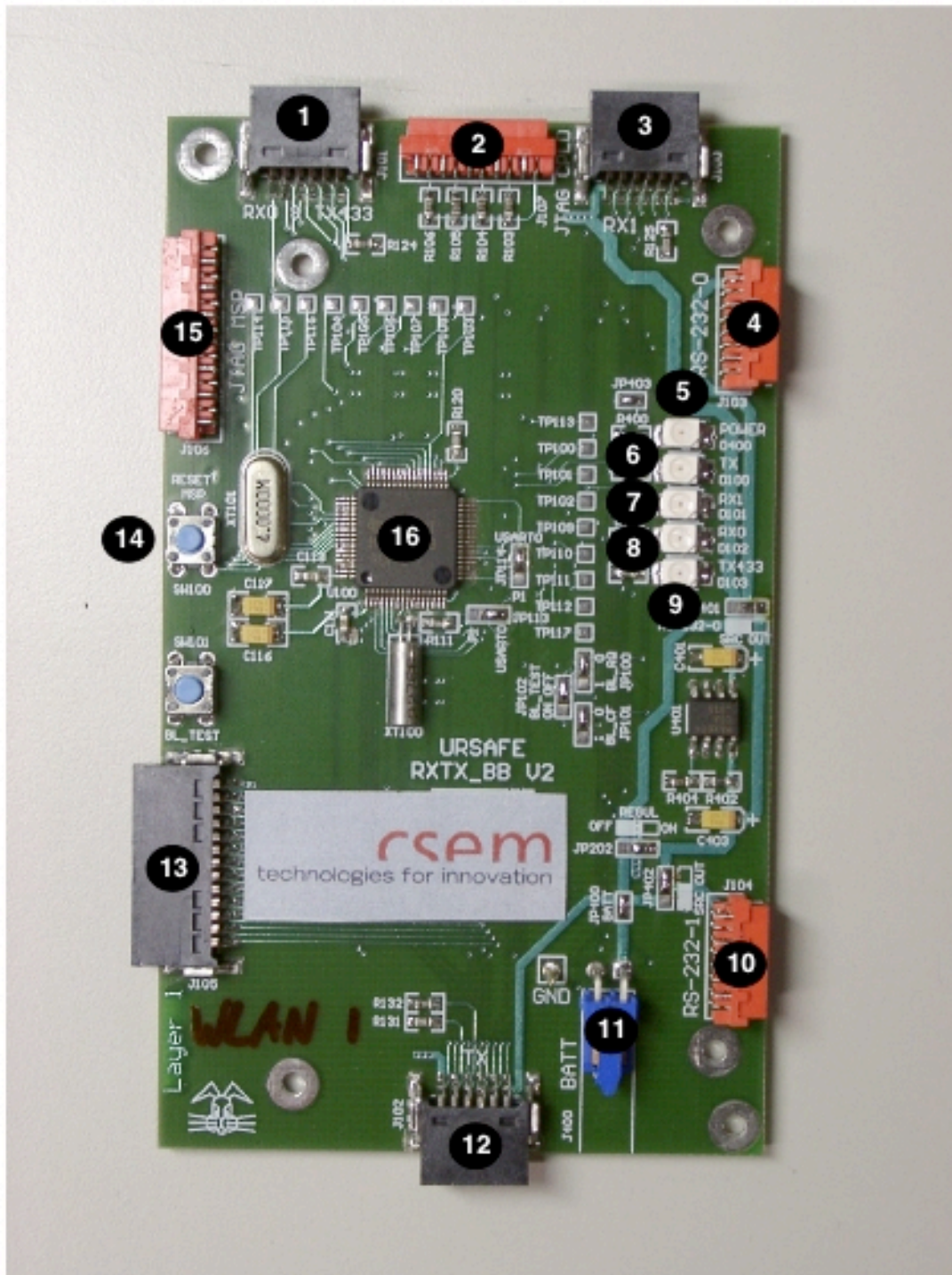


Figure 45 - Carte RXTX_BB_V2

Un total de 4 applications principales a été réalisé. Elles n'ont malheureusement pas pu être toutes compatibles entre elles (utilisation du même code), car l'implémentation a passablement évolué entre la première et la dernière application. En fait, les applications 1 et 2 fonctionnent avec un code qui a été laissé de côté en cours d'implémentation, tandis que les applications 3 et 4 fonctionnent avec le code qui est listé en annexe.

La principale raison de l'abandon du code des 2 premières applications est dû à la segmentation qui au début du projet a été réalisé à la couche MAC. Il a seulement été constaté par la suite que la segmentation se fait au-dessus de la couche MAC.

Sinon, la transition entre les deux implémentations s'est surtout traduite par des changements au niveau des variables globales et de la réservation mémoire. En effet, le Buffer MAC servait à contenir les données non segmentées provenant des couches supérieures. Puis une fois la segmentation réalisée, les primitives MAC recevaient des données segmentées qui étaient directement stockées dans le Buffer PHY. La construction des trames MAC était donc réalisée dans le Buffer PHY, puis transmise à la couche physique. Les tailles des deux Buffers étaient de 256 chacuns. Pour rappel, la réservation mémoire qui a été faite pour les applications 3 et 4 est celle décrite au point 3.3.4.

3.8.2 Procédure de programmation du MSP430

Voici la démarche globale qu'il faut réaliser pour programmer une des cartes RXTX_BB_V2 :

- 1) Alimenter la carte RXTX_BB_V2 avec une tension de 5 V et relier le port JTAG de la carte au port parallèle du PC Windows IAR. Une clé Hardware doit être insérée à l'une des extrémités de la connexion.
- 2) Dans IAR, ouvrir le projet *MAC_802_15_4.eww* puis choisir l'application qui doit être implémentée en ajoutant le fichier dans la fenêtre d'exploration de IAR. Ce fichier d'application doit contenir le *main*.
- 3) Apporter des éventuelles modifications aux variables ou constantes globales qui sont déclarées dans le fichier d'application.
- 4) Choisir le type d'implémentation au début du fichier *mac.h* : RFD ou FFD. Choisir éventuellement l'adresse étendue unique du dispositif par la constante *aExtendedAddress* (toujours dans le fichier *mac.h*).
- 5) Lancer le débogueur en faisant CTRL_D, le module se programme. En quittant le débogueur le module reste programmé même si l'implémentation est de type Debug (et non Release).

3.8.3 Application 1

Le but de cette première application est de tester la transmission et la réception de données au niveau MAC. En fait, on utilise que les trois principales primitives MAC de données ; **MCPS_DATA.request**, **MCPS_DATA.confirm** et **MCPS_DATA.indication**. La construction des trames, la lecture des trames, la segmentation et le réassemblage sont donc les points qui pourront être vérifiés. Les acquittements ne sont pas utilisés.

Le schéma de montage de cette première application est à la Figure 46.

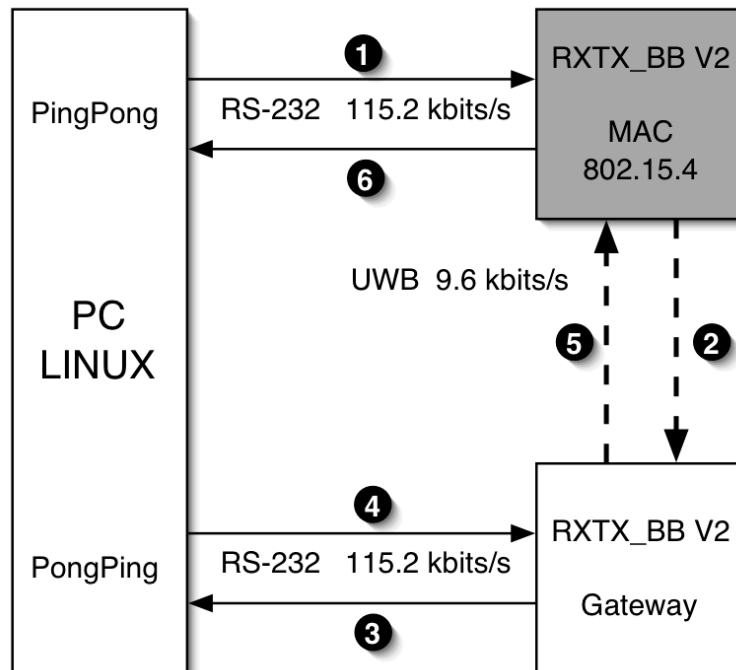


Figure 46 – Schéma de montage de l'application 1

Cette application va se dérouler en deux parties. Dans la première partie, seule la transmission au niveau MAC est testée, la réception étant encore traitée par la couche physique. Dans la deuxième partie, les programmes de test sont modifiés afin de pouvoir réaliser une transmission segmentée et une réception avec un réassemblage en utilisant les 3 primitives MAC cette fois-ci. Il faudra alors également modifier le fonctionnement du module de Gateway. Le même schéma est utilisé pour les deux parties, ce sont uniquement les implémentations et les programmes de contrôles qui vont changer.

3.8.3.1 Première partie : Envoi de données

L'implémentation MAC qui correspond à Application_1_a.c (**Annexe B**) est chargée dans un des 2 modules RXTX_BB V2 utilisés. En envoi, ce sont les primitives MAC MCPS-DATA.request (et .confirm) qui vont intervenir, tandis qu'en réception, c'est la primitive physique PD-DATA.indication qui est utilisée.

Le deuxième module correspond à la Gateway du projet URSAFE, ses fonctionnalités ne sont pour l'instant pas modifiées. Ce module se charge de transmettre de UWB vers RS-232 (mode réception) puis RS-232 vers UWB (mode transmission) d'une manière alternative. Comme une segmentation est réalisée et que ce module doit recevoir trois paquets consécutifs en mode réception, on joue sur le fait qu'environ toutes les deux secondes le module recommence son programme (par une exception du timer B) pour contourner le problème. On doit donc espacer les paquets segmentés de 2 secondes, ce qui est fait dans le premier module en utilisant la fonction *wait*. Pour cette application et les suivantes, les deux cartes sont reliées entre elles par leur port RX et TX (bande de base).

Le PC Linux sert à simuler les couches plus hautes en générant des suites de bits qui sont proposés aux cartes RXTX_BB V2 via les interfaces RS_232. Ces interfaces ont d'ailleurs dû être reconfigurés pour le déroulement de cette application. Voici les commandes du Shell permettant de voir les propriétés des 2 ports RS-232 :

port série 0 : `stty -F /dev/ttySo -a` ou `stty -F /dev/tts/o -a`

port série 1 : `stty -F /dev/ttyS1 -a` ou `stty -F /dev/tts/1 -a`

Les variables doivent être configurées de la manière suivante (exemple pour le port 0) :

```
stty -F /dev/ttySo speed 115200 min 255 -parenb -parodd cs8 -hupcl -cstopb cread -clocal  
crtsets -ignbrk -brkint -ignpar -parmrk -inpck -istrip -inlcr -igncr -icrnl -ixon -ixoff -iucL -  
ixany -imaxbel -opost -olcuc -ocrnl -onlcr -onocr -onlret -ofill -ofdel nlo cro tabo bso vto ffo  
-isig -icanon -iexten -echo -echoe -echok -echonl -noflsh -xcase -tostop -echoprtr -echoctl -  
echoke
```

Les paramètres ayant un signe négatif (-) devant eux ont leur fonctionnalité désactivée. Parmi les fonctionnalités activées, on retrouve :

cs8: Initialisation de la taille d'un caractère à 8 bits.

cread: Autorise la réception de données sur l'entrée.

crtsets: Autorise des acquittements RTS/CTS.

Pour le projet URSAFE, deux programmes de test avaient été créés, **PingPong** et **PongPing**. Le premier sert à transmettre 256 bits de données par le premier port série (RS-232), puis d'attendre un retour d'informations. Le deuxième attend des informations sur le deuxième port série (RS-232), puis une fois reçues, transmet à son tour par le même port 256 bits de données. Les deux programmes s'exécutent chacun dans un Shell différent. Ils vont également générer des fichiers .txt contenant les données hexadécimales qui leur sont parvenues.

Ces deux programmes de test que sont *PingPong* et *PongPing* sont donc repris et légèrement adaptés (codes à l'**Annexe F**) afin de pouvoir tester cette première application. L'adaptation concerne la modification des répertoires des fichiers générés, le nombre de paquets échangés entre les deux programmes et surtout la modification de l'instant d'envoi des paquets de *PongPing*. Cette dernière modification a dû être réalisée, car les données générées sont toujours de 256 bits, même si la couche UWB travaille avec des paquets de 256 bits, la couche MAC ne peut recevoir plus de 102 bits de données. Il y a donc segmentation (de 3 trames) et c'est pourquoi le programme *PongPing* ne doit plus envoyer des données à chaque réception de paquets, mais faire seulement un envoi de données tous les trois paquets reçus.

Les séquences de tests envoyées par *PingPong* et *PongPing* sont construites de la façon suivante. Le premier byte prend la valeur de 00 si c'est le port série So (PingPong) qui a émis la séquence ou 01 si c'est le port S1 (PongPing). Le deuxième byte est un compteur qui part de 00. Les autres bytes sont alors une suite de nombres partant de 02, 03, 04, ... jusqu'à FF.

La photo de la Figure 47 montre les modules, la connexion en bande de base et le branchement au PC via les ports RS-232. Cette configuration est également employée pour toutes les applications futures.

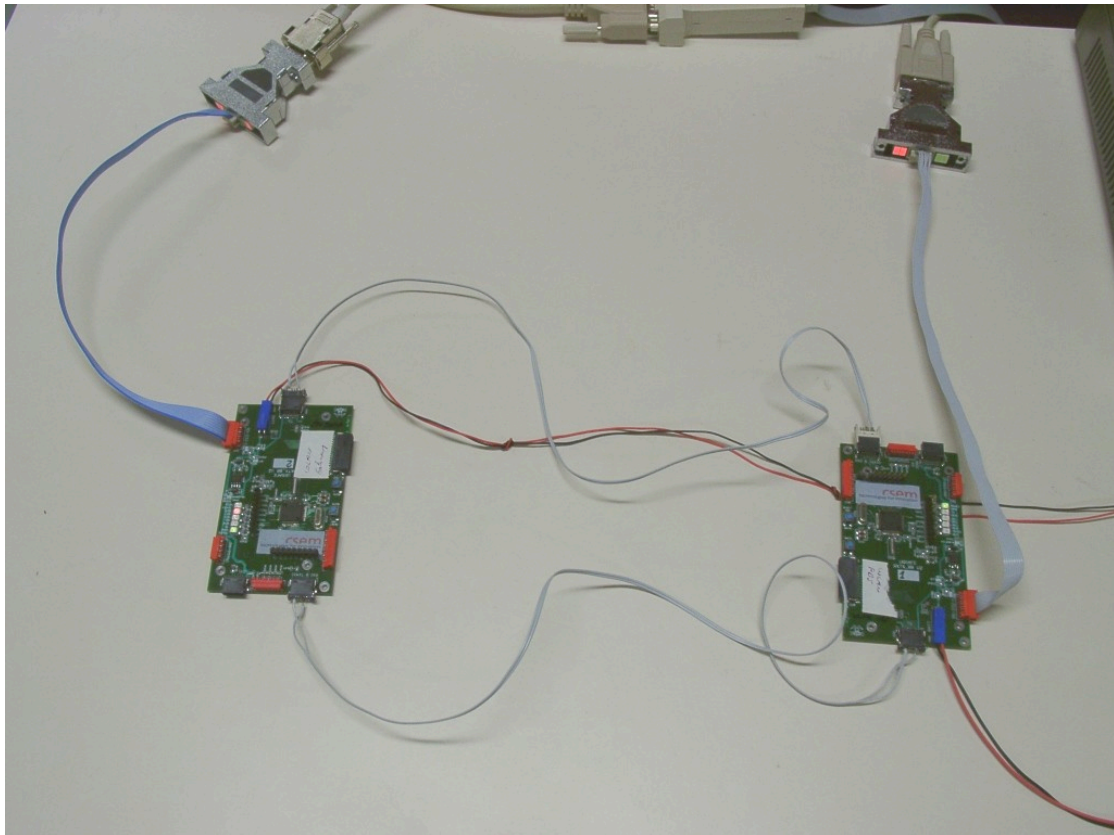


Figure 47 - Photos des connexions entre cartes

En reprenant la Figure 46, on peut alors suivre le chemin que prend les informations. En (1), le programme *PingPong* émet une séquence test de 256 bytes qui est transmis par RS-232 au module où l'implémentation MAC est présente. Le programme *PingPong* se met alors dans un mode de réception. Au niveau du module MAC, les données sont segmentées en trois trames MAC de données que le module va s'empresse de faire parvenir au module de Gateway (2) en les espaçant d'environ 2 secondes. Théoriquement cette transmission devrait se faire en UWB, mais durant les applications du projet, les deux modules sont reliés par des fils séries (bande de base). Le module de Gateway récupère donc les trois trames une après l'autre. Entre temps, en (3), le module de Gateway transmet les trames par RS-232 au programme *PongPing* qui est dans un mode de réception. Une fois les trois trames en sa possession (qui sont écrites en hexa dans un fichier .txt), le programme *PongPing* se met dans un mode de transmission. Il transmet alors une séquence test (4) en destination du module URSAFE par RS-232. Une fois la transmission de cette trame URSAFE de 256 bytes faite, le programme revient en mode réception. En (5), le module de Gateway fait parvenir la trame de 256 bytes au module MAC. Comme ce n'est pas une trame MAC qui est reçue, l'implémentation travaille uniquement au niveau physique en faisant parvenir les données au programme *PongPing* toujours par la liaison RS-232 (6). Ce programme écrit alors les données reçues dans un fichier .txt puis se remet en mode transmission pour envoyer sa deuxième séquence test. Au final, trois séquences de tests par programme seront envoyées.

Les codes des programmes se trouvent tous en **Annexe B, C, D, E**. Les code des applications Linux sont en **Annexe F** et les fichiers textes générés en **Annexe G**. Les en-têtes de la trame MAC correspondent aux mêmes que ceux qui sont utilisés pour le test de primitive MAC de données (voir 3.7)

3.8.3.2 2e partie : Envoi et réception de données

Pour cette deuxième partie, le montage reste exactement le même. Le premier changement concerne le programme d'application de l'implémentation MAC qui va utiliser la primitive MAC de réception de trame (MCPS-DATA.indication). Sinon, l'envoi de trame se fait toujours de la même manière. Le programme *PingPong* (inchangé) fait parvenir par RS-232 les 256 bytes au module où l'implémentation MAC est réalisée. Celui-ci les traite en réalisant une construction des en-têtes MAC et une segmentation en 3 trames MAC. Le deuxième module réceptionne les trames et les renvoie au programme PongPing par le deuxième port série.

C'est le programme *PongPing* (code en **Annexe F**) qui a passablement été modifié. À la place de renvoyer des données brutes, il va cette fois-ci récupérer les données reçues et les renvoyer à l'identique de ce qui a été reçu. Il attend donc la réception des trois trames puis renvoie les trois à la suite sans aucune modification.

Cette manière de faire a contraint également la modification de l'implémentation du module de Gateway. Son programme d'application *Rec_PBS_main.c* est donc retouché afin qu'il se mette en mode réception trois fois de suite, puis qu'il se mette en mode transmission également trois fois de suite. Comme le code de cette implémentation a été réalisé avec une version plus ancienne de IAR, il a fallu apporter quelques modifications pour pouvoir compiler avec la nouvelle version de IAR, notamment au niveau de la syntaxe des interruptions. L'exception du timer a été désactivée momentanément afin que le module ne puisse se réinitialiser tout seul lors d'attente de données trop longues. Cela permet surtout d'utiliser le mode pas à pas dans le débogueur. Cette modification impose une pression du bouton reset avant chaque nouvelle simulation afin de redémarrer le programme.

Finalement le module de la couche MAC implémentée récupère les trois trames MAC afin de reconstituer le chargement qu'elles contiennent pour le faire parvenir à la couche supérieure qui n'est rien d'autre que le programme *PingPong*, la séquence de test qu'elle avait initialement envoyée.

Afin que le module MAC ait le temps de reconstituer la séquence test, il a fallu espacer les trames segmentées d'environ une seconde, ce qui a été réalisé à l'envoi dans le programme *PongPing* à l'aide de la fonction *sleep*. Une seconde d'attente est largement assez, mais la fonction *sleep* ne permet pas d'attendre moins qu'une seconde.

Un résumé des paquets échangés entre les programmes et les modules est présent à la Figure 48. Lors de la communication par UWB, il y a toujours 256 bytes qui sont échangés entre les deux couches physiques des modules, c'est la réalisation URSAFE qui le veut. Dans notre cas, on transmet toujours moins que 127 bytes au niveau MAC et 6 bytes au niveau physique, les bytes restants sont donc mis à 0 par l'adaptation. Le programme *PongPing* reçoit des trames physiques (avec les 6 bytes), car le module de Gateway ne contient pas l'adaptation 802.15.4, les bytes physiques ne sont donc pas supprimés. Tous les résultats des fichiers textes générés sont en **Annexe G**.

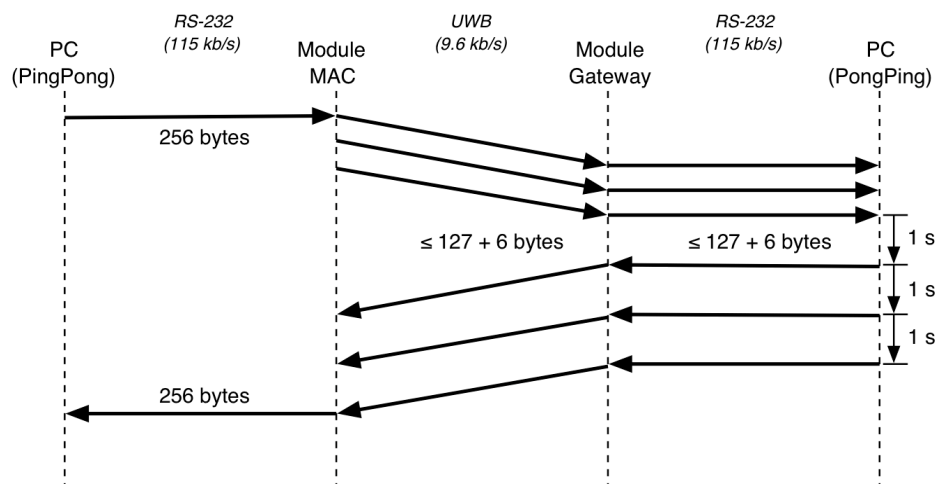


Figure 48 - Échange des paquets dans l'application 1b

3.8.4 Application 2

L'objectif de cette deuxième application est de pouvoir simuler un échange d'informations avec acquittement tout en étant soumis au CSMA-CA « unslotted » pour l'accès au canal. Le schéma d'ensemble de la première application est conservé, mais cette fois-ci les deux modules contiennent la même implémentation de la couche MAC. C'est au niveau de leur couche d'application que la différence se fait. Le premier module a plutôt un rôle de transmettre des données (TX), tandis que le deuxième module possède le rôle de recevoir ces données (RX). La difficulté de cette simulation est que les deux implémentations ne peuvent être testées que séparément, malgré le fait qu'elles doivent fonctionner ensemble pour obtenir un séquençement correct.

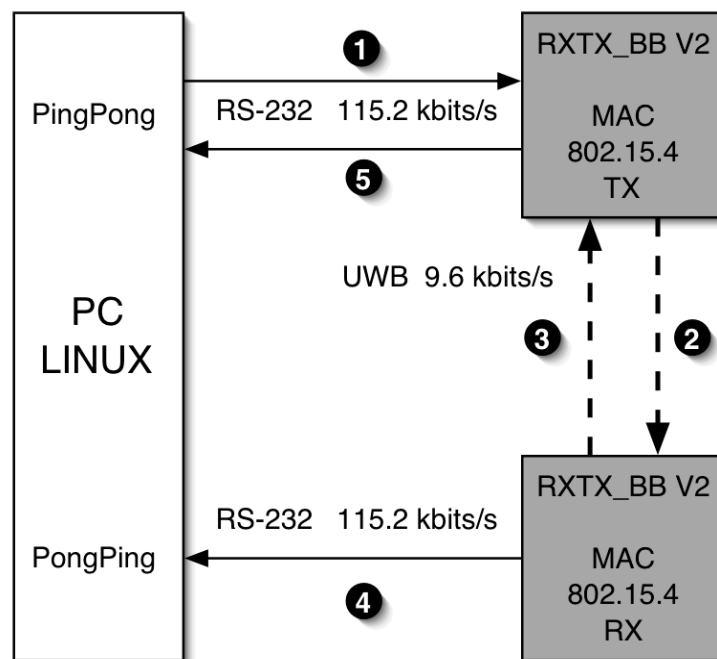


Figure 49 - Schéma de montage de l'application 2

Les programmes *PingPong* et *PongPing* sont conservés afin qu'ils puissent simuler les couches supérieures en réalisant l'envoi et la réception de données. Ils ont néanmoins été légèrement retouchés pour satisfaire cette nouvelle transmission (voir code en **Annexe F**).

Les échanges de paquets de cette application sont présents à la Figure 50. On reçoit donc 256 bytes bruts provenant du PC qui simule les couches supérieures (1). La couche MAC du module MAC TX segmente ces bytes en trois paquets de 102, 102 et 52 bytes. La couche MAC ajoute également des entêtes et un CRC représentant 17 bytes. On obtient alors au niveau Mac, trois trames de 119, 119 et 69 bytes. La couche physique ajoute encore 6 bytes. Le module MAC RX réceptionne les paquets (2) en enlevant tout d'abord l'en-tête physique et en renvoyant un acquittement pour chaque trame (3). Puis les données sont réassemblées afin de transmettre les données brutes à la couche supérieure (4) qui se trouve être l'application *PongPing*.

Pour rappel, le temps LIFS sert à garantir que le récepteur (module RX) arrive à traiter les données lors de ce temps, c'est pour cela que le transmetteur (module TX) doit attendre LIFS.

Le dernier acquittement envoyé par le module RX est également transféré à la couche supérieure (*PingPong*) du module MAC TX (5) afin de donner un coup d'envoi pour les prochaines informations à transmettre (prochain 256 bytes bruts).

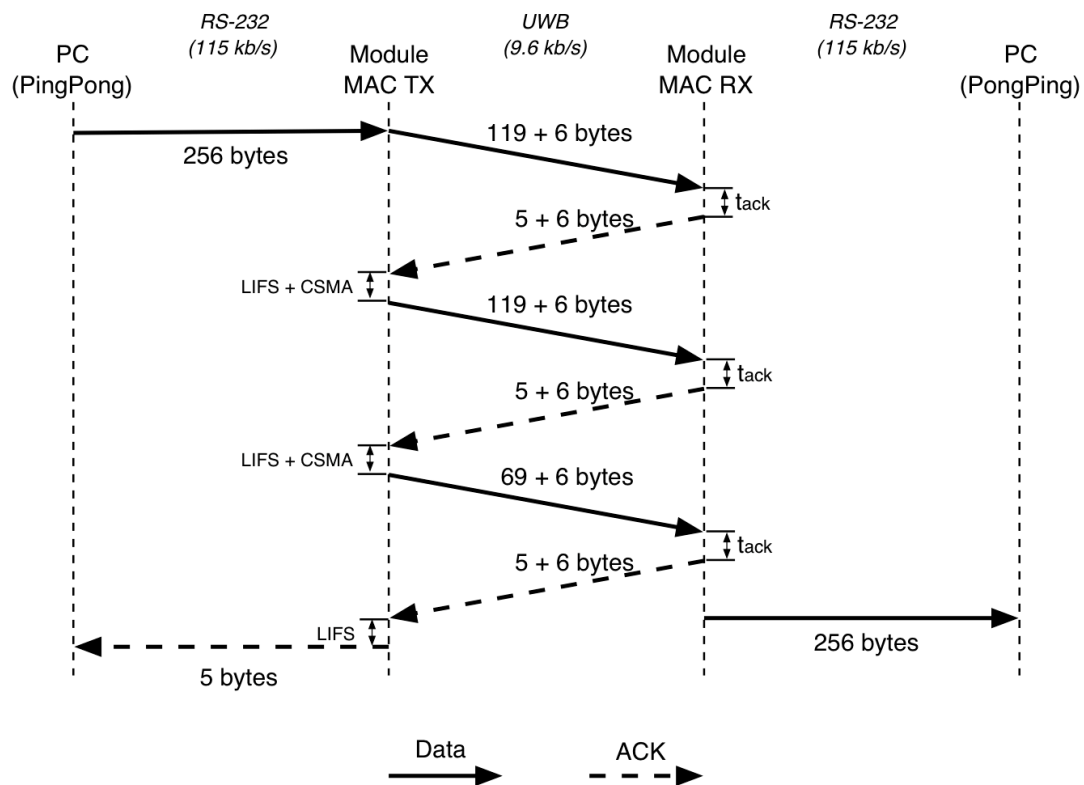


Figure 50 - Échange des paquets dans l'application 2

Afin de pouvoir tester le rôle du CSMA, un tableau de l'état de canal a été défini pour remplacer la fonction PLME-CCA :

$Idle_channel[15] = [f, f, f, t, f, t, f, t, t, t, f, f, f, f, t]$

Les lettres t et f correspondent à FALSE et à TRUE. Pour qu'un accès canal puisse toujours se faire, le nombre de FALSE de suite n'est jamais de plus de 4. En effet, le CSMA fait un *channel_access_failure* lorsque la variable PIB *macMaxCSMABacxkoffs* est dépassée, celle-ci est initialisée par défaut à 4.

Pour sélectionner les états les uns après les autres, la variable globale *attempt_global* est créée. Elle est incrémentée à chaque tentative d'accès au canal.

Pour tester le CSMA-CA, on peut éventuellement modifier le tableau afin de vérifier que le statut de la transmission sera *channel_access_failure* à partir de 4 essais d'accès. Pour cela il faut donc avoir dans le tableau une suite d'au moins 5 FALSE.

Cette application a également demandé l'intégration des retransmissions. Celles-ci ne concernent que les communications avec acquittements. Sans acquittement, il n'y a jamais de retransmission. Si l'envoyeur de données ne reçoit pas d'acquittement après le temps *macAckWaitDuration*, il y a des retransmissions *aMaxFrameRetries* de fois.

Une variable Buffer à d'ailleurs dû être ajoutée à cause des retransmissions : *Buffer_Re*. En effet, le *Buffer_Phy* devait contenir en même temps les données des trames à retransmettre et l'arrivée des acquittements. Avant la boucle des retransmissions, le *Buffer_Phy* est copié dans le *Buffer_Re* afin que le *Buffer_Phy* puisse être restauré avant l'éventuelle retransmission. Le *Buffer_Phy* peut alors accueillir les acquittements sans effacer les données qui pourraient être retransmises.

Afin de pouvoir mesurer des temps au niveau de la couche MAC, il a été décidé d'utiliser le Timer A. Le Timer B étant laissé exclusivement pour la couche physique. Les fichiers *TimerA.c* et *TimerA.h* sont donc créés à l'identique des fichiers déjà existants; *TimerB.c* et *TimerB.h* (voir Annexe E). Les variables du microprocesseur ont juste du être adaptées. Il a également fallu enlever l'interruption qui désactivait le Timer A dans le fichier *IO.c*.

Certaines interruptions levées avec le Timer B n'ont d'ailleurs plus lieu d'être avec ce début d'implémentation de la couche MAC, car c'est aux temps définis par la norme de rythmer les attentes.

Une première exception du Timer B est supprimée dans la fonction *ReceiveDataOverUWB*. À la place de l'exception, c'est le Timer A qui est utilisé, celui-ci est initialisé à l'extérieur de la fonction, car la valeur d'initialisation dépend si c'est un acquittement ou des données qui sont attendues. Dans la fonction, le Timer A est arrêté dès que les informations arrivent. Si le Timer A a terminé de compter avant la venue d'informations, on sort immédiatement de la fonction et on teste à l'extérieur si le timer a effectivement atteint 0. Si c'est le cas, il y a retransmission pour l'expéditeur de données, car cela veut dire qu'il n'a pas reçu d'acquiescement. Ou dans le cas du destinataire, une nouvelle attente de données est réalisée avec la primitive *MCPS-DATA.indication*.

Pour pouvoir tester cette deuxième application, le programme d'application TX contient délibérément une ligne de code de blocage qui permet de connaître le statut de la transmission lors d'un comportement anormal. On peut notamment savoir si le CSMA a échoué (*channel_access_failure*), car seule une transmission réussie (*success*) permet de passer cette ligne de code. Cela permet également de tester si aucun acquiescement n'est parvenu dans le temps *macAckWaitDuration*, la transmission aura le statut de *no_ack*. Pour obtenir ce statut, il suffit par exemple d'abaisser le temps d'attente des acquiescements (variable *macAckWaitDuration*).

3.8.5 Application 3

Cette troisième application va servir à mettre en place un réseau avec un coordinateur. Deux fonctionnalités importantes sont intégrées pour cette troisième application. La première concerne la **synchronisation** des dispositifs sur le coordinateur avec l'envoi de Beacons. Tandis que la seconde correspond à l'**association** des dispositifs au réseau PAN. Cette application est décomposée en trois parties. Les deux premières testant les nouvelles fonctionnalités séparément, tandis que la troisième correspond à une association de dispositif lorsque le réseau est « Beacon Enabled ».

Pour cette application, la segmentation est laissée pour les couches supérieures et n'est pour l'instant pas utilisée. Ceci implique que les échanges de donnée par la communication RS232 entre le PC et les modules n'excèdent jamais 102 Bytes puisque c'est la valeur maximum que la couche MAC accepte comme MSDU.

Au niveau de la configuration des ports séries sur Le PC Linux, il a donc fallu modifier la taille du tampon en changeant le **paramètre MIN** de 255 à 101 bytes. Puis, au niveau de l'implémentation, des modifications ont été nécessaires dans les programmes *Read_Data_On_UART1* et *Send_Data_On_UART1* dans le fichier *IO.c*. À la place de recevoir et d'envoyer des trames de 256 bytes, ils ont été modifiés afin que la fonction *Read_Data_On_UART1* accepte des données de 102 bytes et que la fonction *Send_Data_On_UART1* envoie des données par trame de 102 bytes.

Depuis cette troisième application, c'est l'implémentation du CRC reposant sur les coefficients déjà calculés qui est utilisée. La raison est que l'autre implémentation de CRC prend beaucoup trop de temps de calcul. Certaines variables ou constantes définies par le standard deviennent alors trop courtes. D'ailleurs dans l'application 3b, la variable PIB *macAckWaitDuration* a dû être doublée (de 120 à 240) pour que le dispositif attendant la trame d'acquittement s'impatiente moins. Le problème vient peut-être de la vitesse de commutation du système de URSAFE qui est moins rapide que celui prévu par le standard. Dans le standard, la constante *aTurnaround* (utilisé pour t_{ack}) correspondant au temps de commutation maximum pour passer d'un mode RX à TX ou TX à RX n'est que de 12 symboles (2.4 ms). Pour URSAFE, le temps de commutation est d'environ 10 ms, mais au niveau de l'implémentation c'est même 20 ms qui sont attendus au niveau de la couche physique (fonction *sDataRX_Valid* et *sDataTX_Valid* dans le fichier *UWBModuleDriver.c*).

Le problème qu'occasionne les retransmissions à propos de la réservation mémoire est toujours présent pour cette application. Mais l'obstacle est résolu autrement que dans l'application 2 dont une autre variable Buffer avait été créée. La solution est simplement d'utiliser une variable locale (*bufferMacSave*) qui sauve le Buffer_MAC avant un envoi afin de prévoir l'éventuel effacement du Buffer MAC par la venue d'un acquittement erroné.

3.8.5.1 Première partie : Synchronisation

Comme le but de cette partie est de vérifier si le dispositif se synchronise avec les Beacons envoyés par le coordinateur, lorsque les Beacons sont reçus par le dispositif, ils sont transmis au PC Linux par le port RS232 (programme *PongPing*). Dans ce dessein, il a suffi d'utiliser la fonction *Send_Data_On_UART1* après chaque réception de Beacon dans le programme d'interruption du Watchdog pour pouvoir visualiser les valeurs hexa des Beacons (fichier texte). La Figure 51 présente le schéma de cette première partie d'application.

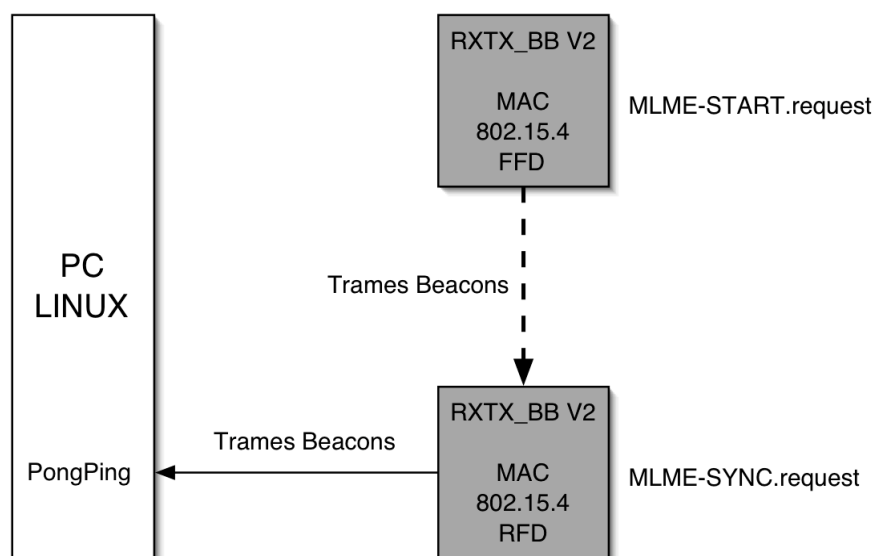


Figure 51 - Schéma de montage de l'application 3a

Le programme d'application du coordinateur se comporte de la manière suivante. Il procède tout d'abord à une réinitialisation de sa couche MAC en utilisant la primitive MLME-RESET.request. Les variables PIB sont donc mises à leurs valeurs par défaut. Puis, il se proclame coordinateur du PAN en choisissant un numéro 16 bits du PAN (0xABCD) et un numéro pour son adresse courte (0x1234) qui est utilisée dans le PAN. Enfin, il se met dans le mode de réseau « Beacon Enabled » le reste du temps par la primitive MLME-START.request.. L'ordre des Beacons choisi est de BO = 3, ce qui correspond à environ 1.6 secondes entre chaque Beacon (BI). Les Beacons sont donc générés indéfiniment tous les 1.6 s, car la couche d'application finit par une boucle infinie.

Le programme d'application du dispositif RFD procède également à une initialisation de sa couche MAC avec la primitive MLME-RESET.request. Puis, on simule, via un balayage, que le dispositif arrive à récupérer les valeurs contenues dans le PAN Descriptor. Il tente alors de se synchroniser avec les Beacons en provenance du coordinateur PAN en utilisant la primitive MLME-SYNC.request. La synchronisation se fait alors indéfiniment tout comme l'envoi du Beacon reçu par le port RS232 vers le PC Linux.

Un exemple de Beacon reçu par le programme *PongPing* est le suivant (voir **Annexe G**)

Co 00 5F AB CD 01 03 05 07 09 0A 0D 0F 4F 23 00 00 57 8D

Ces valeurs hexadécimales correspondent aux informations suivantes :

- | | |
|-----------------------------------|-------------------------|
| • Frame Control : | Co 00 |
| • Beacon Sequence Number (BSN) : | 5F |
| • Source PAN Id : | AB CD |
| • Source Address : | 01 03 05 07 09 0B 0D 0F |
| • Superframe Specification : | 4F 23 |
| • GTS Specification : | 00 |
| • Pending Address Specification : | 00 |
| • CRC : | 57 8D |

Dans le champ de spécification de la Superframe (voir Figure 10), on apprend principalement que l'ordre du Beacon (BO) est de 3 et que l'ordre de la Superframe (SO) est de 2. La Superframe est donc composée d'une partie active autant grande que la partie inactive. On voit aussi que c'est un coordinateur PAN qui a généré le Beacon, mais qu'il n'accepte pas d'association au PAN. On apprend également que le dernier slot du CAP est le 15, c'est à dire le dernier de la partie active, cela permet de déduire qu'il n'y a pas de partie de contention libre (CFP). Cela se confirme avec le champ suivant puisque qu'il n'y a aucun GTS. Enfin, on remarque qu'il n'y a pas de données en attente chez le coordinateur, puisque le champ « Pending Address Specification » ne précise aucune adresse de 16 bits ni de 64 bits. Finalement, aucun chargement (payload) de Beacon est présent. Le chargement de Beacon n'a d'ailleurs pas été implémenté, car il n'est utilisé que par les couches supérieures.

3.8.5.2 Deuxième partie : Association

Le but avec cette deuxième partie d'application est de pouvoir faire des demandes d'associations avec un dispositif et de les accepter ou de les refuser avec un coordinateur PAN. Comme la principale nouveauté de cette application est l'introduction des trames de commande, celles-ci sont bien sûr récupérées par la couche MAC, mais aussi transmises au PC Linux via la liaison RS232, afin de pouvoir les analyser.

Pour l'instant l'association se fait dans un réseau « Non Beacon Enabled », la transmission est indirecte. Le schéma de l'application est présenté à la Figure 52.

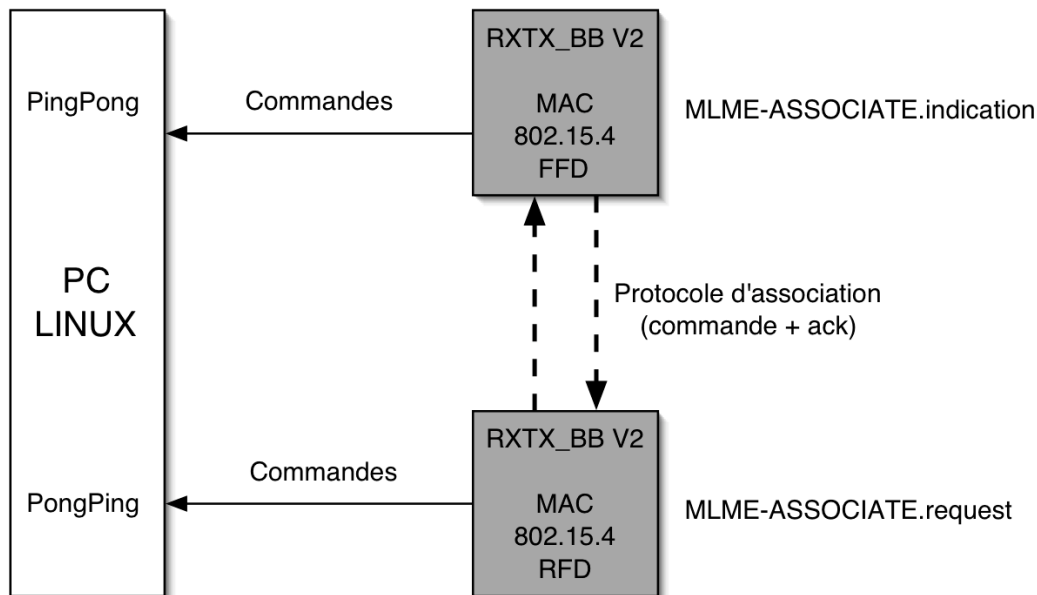


Figure 52 - Schéma de montage de l'application 3b

Dans le programme d'application du FFD (coordinateur), l'attente d'association se fait tant que la primitive `MLME-COMM-STATUS.indication` ne signale pas que l'association est réussie (avec une boucle *while*). C'est aussi avec une boucle *while* que l'on attend le début du protocole d'association via la primitive `MLME-ASSOCIATE.indication`.

Du côté du dispositif qui veut s'associer au PAN, c'est le même principe. En effet, tant que la primitive `MLME-ASSOCIATE.confirm` ne retourne pas un statut de réussite, la primitive `MLME-ASSOCIATE.request` est exécutée.

Le cycle complet du protocole d'association se déroule dans un temps d'environ 9 secondes, ce qui peut sembler relativement long. La raison est dû au fait que le dispositif n'a pas le droit de faire la demande de données avant le temps *aResponseWaitTime* qui dure plus de 6 secondes. Ce temps est censé permettre au coordinateur de déterminer si les ressources du réseau permettent une nouvelle association. Ce n'est qu'une fois ce temps passé que le coordinateur active son récepteur et qu'il peut recevoir la trame de commande « Data Request » en provenance du dispositif. Le coordinateur envoie alors le résultat de l'association sous forme de la commande « Association Response ».

Les valeurs des trames échangées peuvent être visualisées à l'**Annexe G** où sont regroupés les fichiers textes générés par les programmes *PingPong* et *PongPing*.

3.8.5.3 Troisième partie : Association et réseau « Beacon Enabled »

Le but de cette troisième partie est de réaliser une association dans un réseau « Beacon Enabled ». En théorie, il suffit de fusionner les deux premières parties d'applications venant d'être réalisées, mais en pratique cela s'est révélé beaucoup plus difficile, car beaucoup de nouveaux aspects apparaissent. Par exemple, lorsque le réseau est « Beacon Enabled », tous les envois ainsi que les attentes de trames ne peuvent se faire que lors de la période du CAP et doivent donc être arrêtés durant le dernier slot du CAP.

Pour cette application, il a également fallu implémenter la primitive MLME-SCAN, car avant une synchronisation, il est obligatoire pour un dispositif de réaliser un balayage passif. D'ailleurs, seul le balayage passif a été implémenté avec la primitive MLME-SCAN. Le fait de réaliser ce balayage passif permet au dispositif de connaître les principales informations (sous forme d'un PAN Descriptor) du réseau PAN où il doit se synchroniser puis s'associer.

Puis, la principale difficulté a été de garantir que l'instant où un Beacon doit être reçu ou envoyé, le Transceiver soit dans un mode de repos (ni Tx ON, ni Rx ON). Au niveau de l'émission, c'est l'algorithme du CSMA-CA « slotted » qui permet de ne pas activer le mode d'émission n'importe quand. Tandis qu'au niveau réception, c'est la primitive MLME-RX-ENABLE qui permet de ne pas activer le mode de réception au mauvais moment. Donc à l'aide du CSMA-CA « slotted » et à la primitive MLME-RX-ENABLE, la couche MAC n'autorise aucune émission ou réception durant toute partie inactive de la Superframe et durant la période où un Beacon broadcast doit être envoyé ou reçu.

C'est pour cela que le CSMA-CA « slotted » et la primitive MLME-RX-ENABLE sont employés avant tout envoi et respectivement réception de trames de données ou de commandes. Pour une transmission d'acquiescement, le moment de l'émission ou de la réception n'est pas testé, car cela se déroule de toute façon après un envoi ou une réception de trames qui elles ont dû être validées. Par contre, il faut alors prévoir des temps de sécurité dans le CSMA-CA « slotted » et la primitive MLME-RX-ENABLE pour garantir que la transmission sera effectivement terminée lorsque l'instant d'interdiction d'émission ou de réception surviendra.

3.8.6 Application 4

L'objectif de cette dernière application est de réaliser une transmission de donnée indirecte. La synchronisation et l'association de l'application 3 sont conservées. Le schéma de principe correspond toujours à celui des applications précédentes.

Une fois le dispositif synchronisé et associé au coordinateur PAN, le programme « PingPong » permet de proposer un MSDU de 102 octets au coordinateur. Cette arrivée de MSDU peut se faire au moment souhaité en exécutant le programme « PingPong ». L'arrivée de cet MSDU correspond à une demande d'envoi de données de la part des couches supérieures. Le programme d'application exécute donc la primitive MCPS-DATA.request. La transmission doit se faire d'une manière indirecte et acquiescée.

La transmission indirecte qui a été implémentée consiste, à qu'une fois le trame de donnée construite, le coordinateur stocke la trame de donnée dans la liste d'attente des transmissions. La liste implémentée n'a qu'une place mémoire qui correspond à la variable Buffer MAC. La transmission de données est gardée dans la liste d'attente pendant au moins le temps de *macTransactionPersistenceTime* qui est exprimé en nombre de Superframe. Cette variable est mise à 5 durant l'application. Les données sont donc conservées durant 5 temps de Superframe dans la liste de transmissions.

Dès que le MSDU est reçu, le coordinateur enclenche son récepteur durant les cinq prochaines parties actives de Superframe. Les Beacons envoyés durant ce laps de temps contiennent alors l'adresse du dispositif associé, à qui les données sont destinées. Le dispositif démarre une récupération de donnée en attente dès qu'un Beacon reçu le lui fait comprendre. Une fois les données récupérées, il les communique au programme « PongPing ».

L'opération peut recommencer avec une prochaine trame de donnée lorsque le coordinateur envoie la dernière trame d'acquittement envoyée par le dispositif au programme « PingPong ». Ce dernier réalise alors le transfert d'un nouveau MSDU.

L'échange de données réalisé durant cette application correspond aux diagrammes en flèches de la Figure 24 et de la Figure 30.

Les valeurs des données échangées peuvent être visualisées à l'**Annexe G** où sont regroupés les fichiers textes générés par les programmes *PingPong* et *PongPing*.

4. Conclusion

4.1 À propos du futur standard IEEE 802.15.4

La documentation du projet de norme est en général de très bonne facture. Sa structure est faite d'une manière que la plupart des fonctionnalités sont expliquées plusieurs fois, ce qui aide à mieux comprendre. Il est vrai que la plupart des fonctionnalités ne sont décrites qu'avec du texte, ce qui rend souvent l'interprétation vaseuse. Mais généralement il est toujours possible de retomber sur une logique, car l'ensemble des fonctionnalités doit être cohérent.

Très peu d'erreurs ont été repérées, les seules concernent quelquefois des nombres de bytes qui sont faux sur une figure, car contredisant le texte. Par contre, un point concernant les adressages n'a pu être éclairci. Aucune solution n'a pu être trouvée à ce problème et cela en lisant plusieurs fois les parties de la norme le concernant. Voici la description du problème. Un coordinateur qui envoie des Beacons, peut, soit indiqué son adresse courte, ou soit son adresse étendue dans l'en-tête du Beacon broadcast. Si l'on considère un dispositif qui veut s'associer à un PAN et qu'il se synchronise déjà sur les Beacons. Il connaît alors soit l'adresse courte ou soit l'adresse étendue de celui-ci, cela dépend du mode d'adressage qui est utilisé dans les Beacons. La trame de commande « Association Request » qui est envoyé par le dispositif va donc contenir soit l'adresse courte ou soit l'adresse étendue du coordinateur PAN comme adresse de destination (en fonction de celle qui est connue). La prochaine étape pour le dispositif est d'envoyer une trame « Data Request », ici le standard prévoit que l'adresse de destination est obligatoirement une adresse courte. Le problème est que si le coordinateur communique son adresse étendue dans les trames Beacons, il n'est pas possible de s'associer comme le prévoit le standard. Durant l'application 3, le coordinateur a donc été contraint de placer son adresse courte dans les Beacons. Après une association, l'adresse étendue du coordinateur est de toute façon connue par le dispositif, car la trame de commande « Association Response » envoyée par le coordinateur contient obligatoirement une adresse source de 64 bits.

Est-ce un problème de compréhension ou est-ce vraiment une faille dans la norme ? La question reste ouverte. De tout façon, ce genre de standard fait généralement assez long pour arriver dans une version finale, certainement à cause de correction de ce genre de problème. Cette version 18 de Draft ne sera donc certainement pas la dernière avant la version finale.

4.2 À propos du logiciel IAR

Un gros problème rencontré fut lors d'un débogage, le programme s'arrêtait à des endroits où aucun Breakpoint n'était présent et qu'il n'y avait pas de raison dû au code que le programme se stoppe à ce moment. La solution a été d'aller effacer le dossier *settings* se trouvant dans le répertoire de travail du projet. Une fois fait, il est encore conseillé de réaliser un *Rebuilt all* à la compilation (menu *Project*) afin de recompiler entièrement le projet.

Sinon, c'est surtout des problèmes mineurs concernant la mise en place des Breakpoints qui surviennent quelquefois. Souvent, il n'est plus possible de placer des Breakpoints à un endroit autorisé. Il suffit alors de quitter le débogueur et de recompiler le projet pour que le débogueur fonctionne à nouveau correctement.

Hormis ces quelques problèmes, le débogueur est très stable. Ces principaux défauts se situent plutôt du côté de fonctionnalités manquantes. Il n'est, par exemple, pas possible d'arrêter le programme lorsqu'une variable atteint une certaine valeur ou encore lorsqu'un Breakpoint a été rencontré un certain nombre de fois. Il faut donc souvent rusé en ajoutant des lignes de codes. Enfin, il n'y a pas non plus d'outils d'optimisation comme des graphes ou des pourcentages de temps passés dans les routines.

4.3 Implémentations non réalisées

Voici les nombreux points du standard qui n'ont pu être implémentés, soit pour une raison de temps ou soit par le manque de ressources offerts par le MSP430.

- **Sécurisation des données**

La sécurisation des données dans un réseau sans fil est primordiale et le standard 802.15.4 consacre une très large partie à l'aspect sécurité avec cryptage AES. Malheureusement, pour une implémentation faite en laboratoire et dans un contexte de recherche, la sécurité n'est pas une priorité d'implémentation, il a d'abord fallu privilégier le fonctionnement global des transmissions.

- **Multi PAN**

Le standard est prévu pour fonctionner avec plusieurs PAN environnants. Cette voie d'implémentation a rapidement été mise de côté, car il aurait été difficile d'obtenir encore plus de modules que ceux déjà obtenus pour un seul PAN. De plus, cela aurait rendu la procédure de test vraiment compliquée et longue à réaliser, puisque tout est réalisé en bande de base, il aurait donc été peu pratique de relier chaque module d'une manière physique.

- **Implémentation multi-tâches**

Pour obtenir une implémentation complète du standard, il aurait fallu dédier plusieurs tâches à plusieurs fonctionnalités. Malheureusement comme déjà relevé, le microprocesseur MSP430 n'a pas les ressources nécessaires pour réaliser du multi-tâches. Une solution CoolRISC aurait semble-t-il été une meilleure solution, car ce microprocesseur permet le multi-tâches.

Le multi-tâches aurait par exemple permis dans le cas d'une attente de données de voir une exception qui se déclenche quand des données arrivent. Le programme d'interruption placerait juste les données reçues dans un Buffer. Cela serait au programme principal de vérifier si de nouvelles données sont apparues dans le Buffer et pourrait les traiter comme il se doit. Si chaque fonctionnement annexe est traitée par une autre tâche, le programme principal a juste à lancer ou stopper les tâches et à s'occuper du traitement des données reçues.

Dans notre cas, le Watchdog a été considéré comme une tâche indépendante du programme principal, seulement la conception de celui-ci le rend pas vraiment utilisable comme étant une vraie tâche qui est commutée avec le programme principal puisque que la routine d'interruption du Watchdog doit se terminer afin que le programme principale puisse continuer à s'exécuter.

- **Primitives MAC**

Toutes les primitives MAC n'ont pu être implémentées. La seule barrière à leur non implémentation a été le temps. Néanmoins, les primitives non implémentées correspondent aux fonctionnalités considérées comme les moins prioritaires. Même si l'implémentation d'une primitive n'est pas très longue, c'est l'application et les corrections qu' ils faut réalisés en conséquence qui coûte du temps. Voici la liste des primitives MAC qui n'ont pas été implémentée :

- MCPS-PURGE
- MLME-DISASSOCIATE
- MLME-BEACON-NOTIFY
- MLME-GTS
- MLME-ORPHAN
- MLME-POLL

On peut encore y ajouter la primitive MLME-SCAN qui n'a été que partiellement implémentée, puisque seul le balayage passif est présent.

4.4 Conséquences de l'adaptation

Comme plusieurs libertés ont été prises par rapport à la définition du standard, il peut être utile de résumer les principales différences entre le standard et ce qui a été implémenté. Ces différences d'implémentation peuvent avoir plusieurs origines. Les principales proviennent des adaptations qui ont dû être réalisées par rapport aux limitations des ressources du microprocesseur MSP430. Mais d'autres origines proviennent du fait qu'il fallait obtenir rapidement une implémentation de base fonctionnelle. Il faut tout de même souligner que ces différences ne mettent pas en question la compatibilité avec une implémentation IEEE 802.15.4 étrangère.

- Comme le Buffer MAC simule la **fil d'attente de transmission**, ce n'est pas le MSDU qui est stocké en attente, mais directement la trame de données ou de commande déjà construite. De plus la programmation séquentielle oblige à réaliser une attente passive avant que le dispositif se manifeste pour obtenir sa trame. Cette attente est réalisée dans la primitive de données MCPS-DATA.
- Comme le **chargement (payload) des Beacons** n'a pas été implémenté, aucune transmission de Beacons n'est réalisée hormis celle du coordinateur qui sert à synchroniser le PAN et à avertir les dispositifs associés de la présence de données en attente chez le coordinateur.
- Comme les réservations de temps par **GTS** n'ont pas été intégrées, la période de contention (CAP) correspond toujours au maximum (16 slots) de la partie active d'une Superframe.
- Comme la programmation est séquentielle, toutes les **primitives de type indication** sont exécutées de la couche applicative plutôt que de la couche MAC.
- Par souci d'un meilleur fonctionnement, le **CSMA-CA « slotted »** n'échoue pas lorsque l'on se trouve dans une partie inactive de Superframe, mais attend le début de la prochaine période CAP.
- La primitive **MLME-RX-ENABLE** n'active pas le récepteur durant son exécution, mais seulement après en faisant appel à la primitive PD-DATA.indication. Cette modification provient du fait que plusieurs type de trames peuvent arriver, ce qui amène des comportements différents qui n'ont pas à être gérés par la primitive MLME-RX-ENABLE.
- Les ressources processeurs ne permettent pas une utilisation trop fréquente des **interruptions**. C'est pourquoi la précision des envois de Beacon est de 2 ms (WDT_MDLY_8) et que les mesures de temps au niveau MAC d'environ 0.8 ms ($SymbolPrecision = 4$). La précision générale imposée par le standard est d'une unité symbole (208 μ s).
- La manière dont l'implémentation a été réalisée ne permet pas que la durée de la partie active d'une Superframe soit plus courte que la durée d'une récupération de données en attente si des transmissions indirectes doivent être réalisées. C'est pour cette raison que le **Superframe Order (SO)** n'est pas plus faible que 5.
- La variable PIB **macAckWaitDuration** a dû être augmentée par rapport au standard, car la couche physique prend plus de temps pour commuter entre les modes réception et transmission que le prévoit le standard. Le calcul du CRC, qui prend trop de temps, est également une partie de la cause.

4.5 Remarques personnelles

Le rapport qui résulte de ce travail de diplôme peut paraître fastidieux à parcourir, mais par le contexte du projet, qui consistait à étudier un standard, à l'adapter et à l'implémenter en C, il a été difficile de proposer un dossier simple d'accès et passionnant à lire. Ceci n'a heureusement pas entravé à l'enthousiasme du travail à réaliser. De plus le fait de pouvoir réaliser ce projet au CSEM, de travailler parmi plusieurs ingénieurs et d'avoir à disposition du très bon matériel de développement sont tous des avantages qui m'ont permis d'apprendre énormément de chose durant ce travail de diplôme.

Cela a également été une bonne expérience de devoir apprendre à connaître un standard dans les moindres détails. La difficulté étant que l'implémentation devait commencer rapidement, bien avant d'avoir eu le temps de lire complètement le standard. C'est d'ailleurs pour une raison de mal connaissance du standard que l'implémentation a pratiquement été reprise depuis zéro en cours de route. Mais c'est également en commençant à implémenter que la compréhension du standard s'accélérait. Finalement ce que j'en retiens est que la connaissance d'un standard peut prendre énormément de temps, car même après plusieurs lectures, des détails peuvent toujours apparaître...

Je pense que la partie centrale de ce projet de diplôme a été de trouver une bonne gestion de l'utilisation des trois Timers à dispositions. Le standard ne se préoccupant pas du nombre de Timer à utiliser. En effet, dans la description SDL du standard, on y trouve la déclaration de 2 Timers au niveau physique et de 10 Timers au niveau MAC. Ces Timers qui doivent être indépendants ne sont évidemment pas tous présents sur un microprocesseur basse consommation comme le MSP430. Et c'est ce qui peut sembler dommage pour un standard promouvant la basse consommation.

Les deux dernières semaines furent très pénibles, car l'adjonction de la synchronisation, l'utilisation des Timers et les mesures de temps ont fait accroître la difficulté à retrouver et à corriger d'éventuelles erreurs. Pendant plusieurs jours, je ne comprenais pas pourquoi l'association ne fonctionnait pas toujours dépendant des intervalles des Beacons. J'ai d'abord mis la faute sur les exceptions, mais il en n'était rien.

C'est l'avant dernier jour que je découvris la vraie raison, les comparaisons de variables temporelles étaient pour la plupart incorrectes. La raison est que les comparaisons se faisaient avec deux types de 32 bits. Par exemple, pour un test où l'on attendait tant qu'une première valeur de 32 bits était plus grande que la seconde valeur de 32 bits. Il se passait que lorsque la première valeur passait de 0xFFFF à 0x10000, la condition de test devenait positive, malgré le fait que cette première valeur était toujours plus grande que la deuxième. Il semble qu'il ne soit pas possible de réaliser des comparaisons sur 32 bits, car le test se déroule que sur le poids faible (16 bits) du type de 32 bits. Ce mystère n'a pu être élucidé.

Un grand regret concerne la structure générale des fichiers et du projet. C'est l'obligation de devoir déclarer toutes les variables globales dans le même fichier que le programme principal (*main*) qui n'a pas permis d'avoir une meilleure structure. De plus, le fait de posséder beaucoup de variables globales rend plus difficile la compréhension du standard. Comme une réorganisation de structure aurait pris trop de temps, cela n'a pas été fait. L'idéal aurait été de définir des variables statiques par couches et par fonctionnalités, la structure de l'implémentation aurait été mieux définie.

Toujours concernant les regrets, il est dommage que je n'ai pas eu le temps d'arriver à une application se basant sur un coordinateur et deux dispositifs associés. Les possibilités de câblage ayant même été abordées. Cette application aurait permis d'aller beaucoup plus loin dans les fonctionnalités du standard concernant les échanges d'informations.

Un dernier regret est de ne pas avoir eu l'occasion de tester une application avec l'interface radio UWB intégrée. Il aurait certainement été intéressant de vérifier si le fonctionnement d'une des applications réalisées fonctionnait également avec l'interface radio.

Afin de finir sur une note positive, je dirais que ce travail de diplôme a été une expérience formidable tant au niveau apprentissage qu'au niveau contact humain. J'ai eu beaucoup de plaisir à partager la vie du secteur 231 durant trois mois. L'accueil chaleureux et l'ambiance fabuleuse m'ont rapidement permis de bien m'intégrer (même un peu trop...). Je remercie tout particulièrement Alexandre et Andreas qui m'ont apporté une aide indispensable et qui ont réservé une partie de leur temps à mon égard.

Jérôme Vernez

Lexique :

ACL	Access Control List
ADC	Analog to Digital Converter
AES	Advanced Encryption Standard
BE	Backoff Exponentiel
BI	Beacon Interval
BO	Beacon Order
BPSK	Binary Phase Shift Keying
BSN	Beacon Sequence Number
CA	Collision Avoidance
CCA	Clear Channel Assessment
CAP	Contention Access Period
CFP	Contention Free Period
CMOS	Complementary Metal-Oxide Semiconductor
CPLD	Complex Programmable Logic Device
CPU	Central Processing Unit
CRC	Cyclic Redundancy Check
CSMA	Carrier Sense Multiple Access
CW	Contention Window
DCO	Digitally Controlled Oscillator
DSN	Data Sequence Number
ECG	Equine Chorionic Gonadotrophin
FCC	Federal Communications Commission
FFD	Full Function Device
GIE	General Interrupt Enable
GPS	Global Positioning System
GPRS	General Packet Radio Service
GTS	Guaranteed Time Slot
HDLC	High-level Data Link Control
IEEE	Institute of Electrical and Electronics Engineers
IFS	Inter Frame Space
ISM	Industrial Scientific Medical

LAN	Local Area Network
LIFS	Long Inter Frame Spacing
LLC	Logical Link Control
LR	Low Rate
MAC	Medium Access Control
MCPS	MAC Common Part Sublayer
MFR	MAC Footer
MHR	MAC Header
MLME	MAC Layer Management Entity
MPDU	MAC Protocol Data Unit
MSDU	MAC Service Data Unit
MU	Monitoring Unit
O-QPSK	Offset Quadrature Phase Shift Keying
OSI	Open System Interconnection
PAN	Personal Area Network
PBS	Portable Base Station
PC	Program Counter / Personal Computer
PD	PHY Data service
PDU	Protocol Data Unit
PIB	PAN Information Base
PLME	PHY Layer Management Entity
POS	Personal Operating Space
PPDU	PHY Protocol Data Unit
PSDU	PHY Service Data Unit
QoS	Quality of Service
RAM	Random Access Memory
RFD	Reduced Function Device
ROM	Read Only Memory
SAP	Service Access Point
SD	Superframe Duration
SDL	Specification and Description Language
SDU	Service Data Unit
SIFS	Short Inter Frame Spacing
SO	Superframe Order

SP	Stack Pointer
SPI	Serial Peripheral Interface
SR	Status Register
SSCS	Service Specific Convergence Sublayer
UART	Universal Asynchrone Receiver/Transmitter
URSAFE	Universal Remote Signal Acquisition for health
USART	Universal Synchrone/Asynchrone Receiver/Transmitter
UWB	Ultra Wide Bandwidth
WLAN	Wireless Local Area Network
WPAN	Wireless Personal Area Network
WU	Wrist Unit

Documents et liens Internet :

- Draft IEEE 802.15.4, version D18, février 2003
- Texas User's guide MSP430
- Le langage C – Norme ANSI (2^e édition), Brian W. Kernighan & Denis M. Ritchie, Masson, Paris, 1997
- <http://www.ieee.org/portal/index.jsp>
- <http://www.zigbee.org>
- <http://www.ti.com>
- <http://www.iar.com/>
- <http://rcswww.urz.tu-dresden.de/%7Esr21/crc.html>