Faculté
des Sciences
Département d'Informatique

UMONS
Université de Mons

Technologie Ultra Wide Band dans l'Internet des Objets

Projet de Master 1

Réalisé par **Maximilien CHARLIER**

Service: Réseaux et Télécommunications
Directeur: Bruno Quoitin

Année académique 2015–2016

Faculté des Sciences • Université de Mons • Place du Parc 20 • B-7000 Mons

Table des matières

Abréviations et acronymes

8DPSK - « 8 Differential Phase-Shift Keying » - Modulation par changement de phase à 8 symboles.

ACK - « Acquittement » - Désigne un message envoyé pour signaler la bonne réception d'un message.

CC2420 - Émetteur/récepteur 2,4 GHz utilisant le standard IEEE 802.15.4-2003.

CAN - « Convertisseur Analogique-Numérique ».

CCA - « Clear Channel Assessment » - Méthode de détection d'occupation d'un canal. Détaillée en section « Clear Channel Assessment » (page 56).

CSMA/CA - « Carrier Sense Multiple Access / Collision Avoidance » - Technique d'écoute d'un canal avant l'envoi d'un message pour prévenir d'une collision. Détaillé en section « Standard IEEE 802.15.4 » (page 16).

CRC - « Cyclic Redundancy Check » - Code de détection d'erreur permettant au récepteur de détecter si une trame est corrompue.

DC - « Duty Cycling » - Protocole régulant l'utilisation de l'antenne radio afin d'économiser de l'énergie. Détaillé en section « Pile réseau » (page 33).

FCS - « Frame Check Sequence » - Désigne un CRC ajouté à la fin d'une trame par un émetteur pour permettre au récepteur de vérifier l'intégrité de la trame.

GFSK - « Gaussian frequency-shift keying » - Modulation utilisée dans le mode Basic Rate et Low Energy du Bluetooth.

PHR - « `PHY header` » - Désigne une section de 19 bits présente dans une trame IEEE 802.15.4-2011 UWB juste après le SFD et avant la trame MAC qui définit certaines caractéristiques de cette trame MAC afin de permettre au récepteur une bonne réception [19] comme la longueur de la trame. Détaillé en section « Format physique des trames » (page 26).

PRF - « Pulse Repetition Frequency » - Définit dans le standard IEEE 802.15.4-2011, il s'agit de la fréquence de répétition des impulsions dans le préambule et dans la portion de donnée d'une trame IEEE 802.15.4-2011 en fonction de la configuration. Détaillé en section « Format physique des trames » (page 26).

RPL - « IPv6 Routing Protocol for Low power and Lossy Networks » - un protocole de routage décrit en section « Pile réseau » (section 1.4.1, page 33).

RX - « Réception » ou « Récepteur ».

SFD - « Start of Frame Delimiter » - Désigne une section de la trame physique servant de délimiteur de début de trame. Détaillé en section « Format physique des trames » (page 26).

SPI - « Serial Peripheral Interface » - Désigne une interface permettant de commander des circuits imprimés grâce à une communication série synchrone, introduite par Motorola [19]. Détaillé en section « Interaction avec le DW1000 » (section 2.1.1).

TOF - « Time Of Flight » - Désigne le temps de propagation d'un message entre un émetteur et un récepteur. Détaillé en section « Temps de propagation » (page 20).

TX - « Transmission » ou « Émetteur ».

UEID - « Unique Extended IDentifier » - Désigne un identifiant unique de 42 bits que possède chaque transceiver. Détaillé dans la section « Mode d'adressage » (page 26).

UWB - « Ultra Wide Band » - Désigne une technologie radio utilisant des canaux radio de bande de fréquence de largeurs plus grande ou égale à 500 MHz.

Introduction

Le projet consiste en la réalisation d'un driver pour un émetteur-récepteur radio. Cet émetteur-récepteur radio, que l'on abrégera en transceiver, utilise une technologie radio différente de celle habituellement répandue sur le marché. Cette technologie radio, appelée « Ultra Wide Band », permet une consommation moindre et une plus grande densité d'émetteurs au mètre carré. Ces deux critères sont deux contraintes fortes dans l'Internet des objets, un réseau conçu pour la communication entre machines et connecté à Internet grâce à l'emploi d'IPv6.

Le projet se déroule en plusieurs phases. La première est l'étude d'une partie des notions nécessaires à la compréhension du projet, ces notions sont détaillées en chapitre 1. Une fois les notions de base et une vue globale des technologies radio acquises, une phase d'étude du transceiver utilisant l'Ultra Wide Band est nécessaire. Cette étude se fait en chapitre 2. Après celle-ci et la maîtrise des fonctionnalités minimales du transceiver, une intégration dans l'Internet des objets est réalisée. Pour cela, un driver radio a été intégré à Contiki OS, un système d'exploitation dédié aux réseaux de capteurs et à l'Internet des objets. Cette intégration est décrite en chapitre 3. Une étude des performances du driver et de sa fiabilité est réalisée en chapitre 4. Enfin, une conclusion de ce projet est donnée dans le chapitre 5.

Chapitre 1

État de l'art

1.1 Réseau de capteurs sans fil

Un réseau de capteurs sans fil « Wireless Sensor Network » est un ensemble d'équipements (*nœuds*) reliés entre eux par une connexion sans fil via des transmissions optiques ou via des communications par radiofréquences.

Ils sont souvent utilisés pour relever des données de façon autonome. Par exemple, ils permettent de mesurer l'acidité de l'eau, mesurer la température, détecter la présence de personnes, enregistrer les vibrations dans un véhicule, etc.

Ils peuvent aussi être utilisés dans les objets du quotidien tels que des interrupteurs, des lampes, des thermomètres, des vannes thermostatiques intelligentes, etc.

Les nœuds de réseaux de capteurs sans fils sont des systèmes embarqués très petits, de l'ordre de quelques millimètres cubes embarquant les capteurs, l'émetteur/récepteur radio, de la mémoire, un processeur, des convertisseurs analogiques-numériques et une batterie. L'émetteur/récepteur radio sert à l'envoi des données recueillies par les capteurs et la propagation d'information. Ils sont exceptionnellement munis d'actionneurs tels que par exemple des moteurs. La Figure 1.1.1 illustre les diverses parties d'un nœud de réseaux de capteurs.

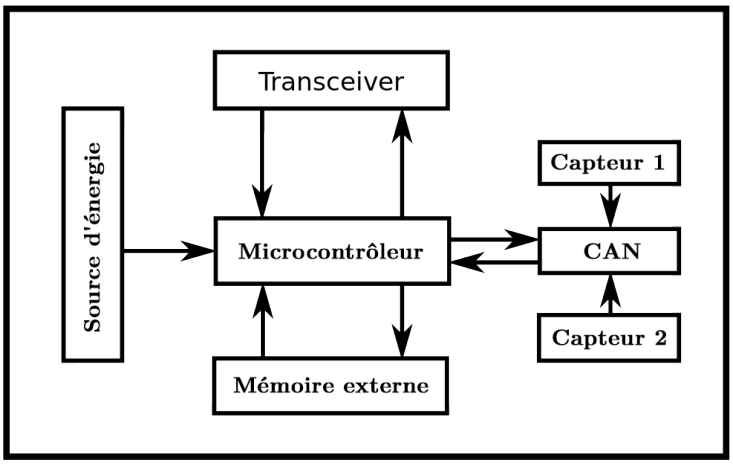


FIGURE 1.1.1 – Schéma d'un nœud [49].

La batterie est de faible capacité, souvent non remplaçable. On cherche à minimiser les transmissions radio qui représentent la majeure partie de la consommation des nœuds.

Lorsque les réseaux de capteurs sont reliés à Internet, on parle d'**Internet des objets** (Internet of Things - IoT). L'espace d'adresses de l'IPv4 étant épuisé dans certaines régions du monde et au vu du grand nombre d'objets qui pourraient voir le jour, ces réseaux de capteurs implémentent le support de l'IPv6.

La Figure 1.1.2 illustre, quant à elle, deux nœuds d'un réseau de capteurs utilisé en 2001 pour une démonstration qui comptait 800 nœuds [9]. Cette démonstration avait pour but de mettre en avant un réseau de capteurs qui s'organisait automatiquement, ce afin de démarrer l'Intel Developers Forum [48].

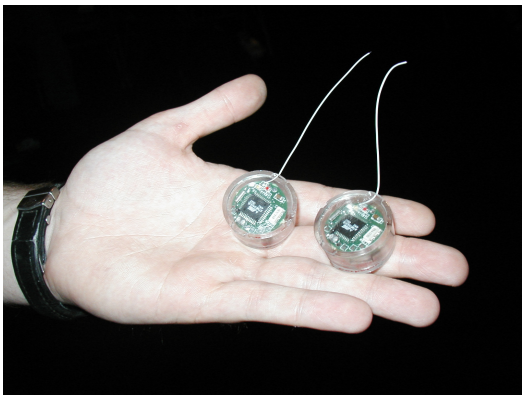


FIGURE 1.1.2 – Deux nœuds «dots» d'un réseau de capteurs expérimental [9].

Un exemple d'équipement de l'Internet des objets est le thermostat Nest [35] (Fig 1.1.3). Il s'agit d'un thermostat «intelligent» qui se place dans une pièce de vie. Celui-ci détecte la présente des habitants et apprend les habitudes de ceux-ci. En se basant sur ces habitudes, le thermostat Nest [35] régule la température de la maison, allume automatiquement le chauffe-eau. Cela permet de réduire la facture énergétique. Il peut aussi être contrôlé à distance. Cet équipement illustre la façon dont il est possible de réinventer un objet du quotidien afin de réduire les dépenses énergétiques.

FIGURE 1.1.3 – Thermostat Nest [35].

1.1.1 Type de noeuds

Dans un réseau de capteurs sans fil, on peut retrouver plusieurs types de nœuds :

1. **Nœud de collecte ou «nœud-puits»** (*sink* en anglais) qui centralise les mesures effectuées par le réseau de capteurs auquel il appartient. Il sera optionnellement relié au réseau Internet et permettra de renvoyer sur celui-ci les données collectées. Ce type de nœud consomme plus que les autres et est souvent pourvu d'une batterie de plus grande capacité ou est relié au réseau électrique. Il est aussi généralement pourvu d'une plus grande capacité processeur et mémoire.
2. **Nœud standard** qui relèvera des données à l'aide de ses capteurs et qui les transférera jusqu'aux nœuds de collecte. Ce type de nœud se chargera aussi de faire passer les messages des autres nœuds. En effet, un routage multi-saut consomme moins que d'émettre à plus forte puissance (la perte de signal diminuant de façon polynomiale avec la distance).
3. **Nœud fortement contraint** qui n'est par exemple pas pourvu de batterie, mais d'un condensateur qui se recharge via son environnement (la chaleur ou les vibrations par exemple). Ce type de capteur sera souvent en sommeil pour de longues périodes et ne propagera pas les messages des autres nœuds du réseau.

1.1.2 Topologies des réseaux de capteurs

Dans un réseau de capteurs sans fil, il existe différents types de topologie. La topologie modélise la façon dont les nœuds sont reliés entre eux et contraint le chemin par lequel les données vont transiter. Il existe typiquement 3 types de topologies [32] qui sont illustrées en Figure 1.1.4 :

1. La topologie en **étoile** qui se caractérise par un nœud principal qui va centraliser les communications. Ce nœud principal peut aussi se charger de l'ordonnancement des communications en attribuant les temps d'émissions de chaque noeud.
2. La topologie en **arbre** qui attribue un niveau à chaque nœud. Pour rejoindre la racine de l'arbre, les messages vont devoir transiter par les nœuds parents du nœud envoyant le message. La racine de l'arbre étant un nœud-puits.

3. Enfin, la topologie **maillée**, qui sera la plus couramment utilisée, car elle est plus dynamique. Elle est quant à elle composée de nœuds qui considèrent tous les liens possibles et choisissent le meilleur en fonction de leur besoin et des contraintes. L'utilisation d'un protocole de routage y est nécessaire.

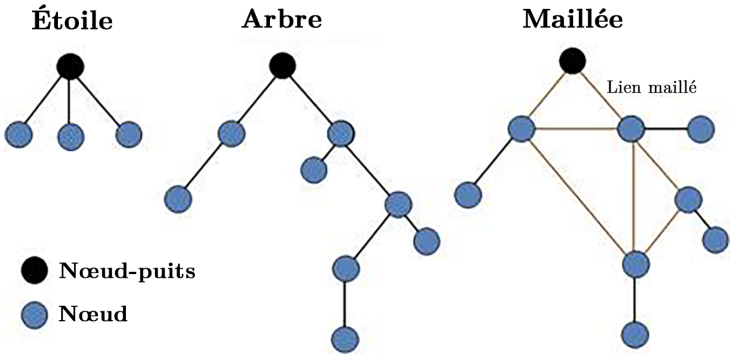


FIGURE 1.1.4 – Types de topologie dans un réseau de capteurs [32].

1.2 Technologies radio

Afin de partager les données récoltées ou simplement permettre d'être commandés à distance, les nœuds communiquent entre eux par radio fréquence. Contrairement aux terminaux utilisés couramment, tels qu'un smartphone ou un ordinateur portable, les nœuds sont beaucoup plus contraints. Leur **consommation** doit être la plus faible possible afin d'augmenter leur durée de vie. Afin de réduire fortement leur consommation, ceux-ci seront en veille pour des périodes relativement longues, de l'ordre de 99% du temps. De plus, les réseaux de capteurs impliquent une très forte concentration de noeuds au mètre carré. Il en résulte le besoin d'une technologie de communication qui préviendra les risques de collision afin de minimiser les retransmissions. C'est le cas du WiFi (IEEE 802.11). Ce dernier n'est pas utilisé dans les réseaux de capteurs, car il ne satisfait pas les prérequis. En effet, il entraîne une consommation électrique trop importante et il ne tolère pas une forte densité de nœuds.

Certains standards sont développés spécifiquement pour les réseaux de capteurs sans fil tels que l'IEEE 802.15.4 ou la version Low Energy du Bluetooth (BLE).

Nous allons à présent voir plus en profondeur différents types de technologies radio.

1.2.1 Bluetooth

Ce paragraphe présente le standard de communication Bluetooth. Il s'inspire des informations de la présentation de M. Vongvilay et al. [34], ainsi que du standard défini par l'IEEE [7] et la page Wikipédia [52].

Bluetooth est un standard de communication sans fil créé en 1994 et normalisé dans les années 2000. Il permet des échanges sur de courtes distances, de l'ordre de 10 à 20 mètres et peut aller jusqu'à 100 mètres en fonction des antennes.

Bluetooth repose sur un émetteur de très faible complexité ne pouvant moduler le signal qu'avec au maximum une modulation par changement de phase (8DPSK) à 8 symboles. Cela permet de produire des transceivers Bluetooth à très faible coup.

Bluetooth possède un nombre important de canaux (79) d'une largeur de bande de 1 MHz. Le tableau 1.2.1 présente les différents canaux utilisables en Bluetooth. Son principal défaut est le fait que son unique bande d'émission se trouve dans la bande ISM (Industrie, Science et Médical). Les communications dans cette bande de fréquence sont susceptibles d'être perturbées. En effet, les fours à micro-ondes et le WiFi, entre autres, émettent dans cette bande de fréquences.

Bande de fréquences (GHz)	**Nombre de canaux**	**Largeur de fréquence** (MHz)
2,400 à 2,4835	79	1,5 à 3

TABLE 1.2.1 – Tableau des bandes de fréquences en Bluetooth [7, p45-46].

Du côté du débit, Bluetooth a subi des améliorations au fil du temps. Le tableau 1.2.2 présente les différents modes ajoutés au Bluetooth de version en version. La première version permettait un débit de 0,7 Mb/s, la version 2 un débit de 3 Mb/s. En 2010, la version 4.0 est rendue publique. Celle-ci a pour but de réduire considérablement la consommation des émetteurs/récepteurs via l'ajout du mode Low Energy [45]. Le mode Low Energy utilise une modulation « Gaussian Frequency Shift Keying » (GFSK) comme le mode Basic Rate. La modulation GFSK est une modulation par changement de fréquence avec en plus l'ajout de l'utilisation d'un filtre gaussien [28].

Mode	Version	Débit théorique maximal (Mb/s)	Modulation [44]
Basic Rate (BR)	$\geqslant 1.x$	1	GFSK
Enhanced Data Rate (EDR)	$\geqslant 2.x$	3	8DPSK
High Speed (HS)	$\geqslant 3.x$	24 *	8DPSK
Low Energy (LE)	$\geqslant 4.x$	1	GFSK

TABLE 1.2.2 – Tableau de différents modes en Bluetooth [45].

La version 1 du protocole a été utilisée comme précurseur dans les réseaux de capteurs, car il était un des premiers à permettre le partage de données sur faible distance à faible coût (du fait de la simplicité de ses émetteurs). Malheureusement, il souffre de défauts, sa consommation est importante, ce qui est très handicapant dans les réseaux de capteurs sans fil. Les nœuds de réseaux de capteurs pouvant tomber en panne (et donc le nœud maître), la topologie en étoile n'est pas forcement appropriée. Le mode Low Energy, introduit dans la version 4, permet des communications à bas débit avec une très faible consommation, la priorité est donnée à la consommation électrique plutôt qu'au débit. Cela rend le Bluetooth utilisable dans les réseaux de capteurs sans fil [45]. Le support du Bluetooth a par exemple, été intégré à la plateforme Mulle [25], il s'agit d'un système embarqué adapté aux réseaux de capteurs connectés à l'Internet des objets [24].

1.2.2 Standard IEEE 802.15.4

Le standard 802.15.4 est défini par l'IEEE. Il est spécifiquement conçu pour les plateformes à très faible autonomie et très faible capacité de calcul et de mémoire. En d'autres mots, le standard vise des plateformes dont le coût de fabrication est faible comme pour les réseaux de capteurs. Il permet une communication à faible débit. Le débit exact est fonction de la bande de fréquence utilisée [1]. La première version de ce protocole est définie en 2003 et porte le nom de IEEE 802.15.4-2003.

∗. Repose sur l'utilisation du 802.11 (le Wifi) pour les transfères de données à haut débit.

Bandes de fréquences	**Nombre de canaux**	**Largeur de fréquence** (MHz)	**Débit théorique** (kb/s)
2,4 à 2,4835 GHz	16	5.2	250
902 à 928 MHz	10	2.6	40
868 à 868,6 MHz	1	0.6	20

TABLE 1.2.3 – Tableau des bandes de fréquences et des débits du standard IEEE 802.15.4-2003 [6].

Il est conçu pour fonctionner sur deux types de topologies de réseaux : étoile ou mesh (maillé). Il implémente «Carrier Sense Multiple Access / Collision Avoidance» (CSMA/CA) pour communiquer. CSMA/CA a pour principe d'écouter le canal d'émission avant d'émettre. Le temps d'écoute est fixé aléatoirement. L'émetteur émet si il n'y pas a eu d'autre émission pendant la durée d'écoute. Cela permet d'éviter de façon probabiliste d'émettre en même temps qu'une autre station, mais n'empêche pas forcément les collisions.

Comme le tableau 1.2.3 le montre, le standard IEEE 802.15.4 recouvre la bande de fréquence du 2,4 GHz, mais aussi d'autres parties du spectre à 800 et 900 MHz. Les spectres 800 et 900 MHz seront moins victimes de perturbations, car ils ne sont pas dans le range du 2,4 GHz et sont moins perturbés par les standards d'émission à haut débit (Wifi/Bluetooth).
On observe aussi que les débits des canaux hors 2,4 GHz sont beaucoup plus faibles.

1.2.3 Standard IEEE 802.15.4-2011

Le standard IEEE 802.15.4-2011 est une variante du standard IEEE 802.15.4-2003 qui couvre plus de fréquences. Certains canaux y ont aussi des bandes de fréquences plus larges (voir tableau 7.2.1). Ces canaux couvrent les bandes de fréquences spécifiques à plusieurs pays [8, p.1-2] (Chine et Japon) en plus des bandes fréquences ISM ce qui permet une utilisation des émetteurs/récepteurs de façon mondialisée.

Il permet également d'avoir des débits de 20 kb/s à 250 kb/s en fonction de la bande de fréquences utilisée. Le standard IEEE 802.15.4-2011 ajoute la standardisation de l'**Ultra Wide Band** qui est décrite plus loin dans ce rapport (en section 1.2.4).

Le tableau 7.2.1 (en annexe) liste les canaux disponibles pour le standard 802.15.4-2011 en excluant les canaux UWB. Certains canaux ne sont permis que dans certains pays. On remarque que le spectre 900 MHz a été étendu par rapport au précédent standard. Le nombre de canaux sur ces spectres étant passé de 11 à 34.

1.2.4 Technologie « Ultra Wide Band »

L'**Ultra Wide Band** [22] [30] désigne l'utilisation d'un spectre plus large pour l'émission de données en opposition aux émissions à canal étroit ou moyen tel qu'utilisé par le WiFi et le Bluetooth. L'UWB utilise une modulation par impulsion radio fréquence. Une impulsion radio fréquence (RF) de l'UWB a une durée inférieure à une nanoseconde (Figure 1.2.1a). Il existe différents types de modulation comme la bi-phase modulation (BPM) dont un exemple est donné en Figure 1.2.1b, où l'on remarque que le « 0 » est en opposition de phase avec le « 1 ».

(a) Une impulsion UWB **(b)** Modulation BPM en UWB

FIGURE 1.2.1 – Illustration d'une impulsion UWB [27].
Verticalement l'amplitude, horizontalement le temps.

La Figure 1.2.2 illustre la différence entre une émission en canal étroit et une émission UWB. En haut à gauche de la figure on retrouve une émission sur canal étroit utilisant une modulation en fréquence et en bas à gauche une suite d'impulsions UWB. Sur le canal étroit, on remarque que la transmission est continue du fait qu'on utilise une sinusoïde que l'on fait varier en fréquence pour porter le signal. En UWB, on remarque qu'entre chaque impulsion aucun signal n'est émis. Sur les graphiques de droite, la différence de largeur de spectre de l'UWB avec un canal étroit est illustrée. La valeur verticale représentant la puissance d'émission, on remarque qu'un signal UWB est émis à plus faible puissance.

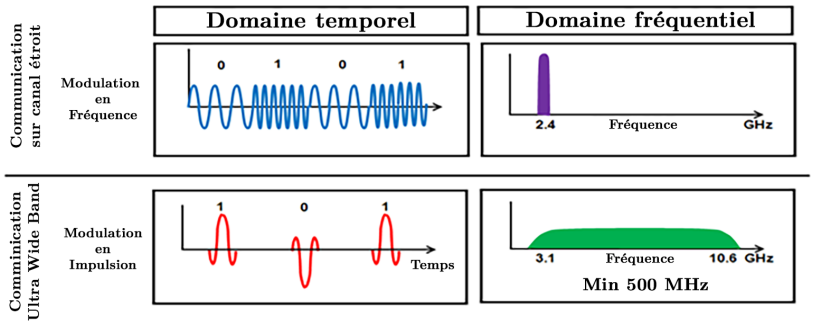


FIGURE 1.2.2 – Comparaison graphique entre une communication UWB et une communication sur canal étroit. [21].

1.2.4.1 Cohabitation avec les systèmes existants

Avant le déploiement de l'Ultra Wide Band dans la bande 3 GHz - 10 GHz, des études ont été menées. Cela afin d'établir des standards permettant la non-concurrence de l'UWB avec les systèmes à bande étroite/moyenne, les deux systèmes cohabitent via l'utilisation de filtres. Pour un système radio standard (à bande étroite/moyenne) un signal UWB sera considéré comme du bruit blanc. Le bruit blanc est du bruit dont la densité spectrale de puissance est constante sur toute la largeur de la bande de fréquence du système radio standard [41].
Cela est illustré dans la Figure 1.2.3, la puissance d'un signal UWB est beaucoup plus faible que les autres technologies.

⋆ PCS est un opérateur américain utilisant la fréquence 1,9 GHz.

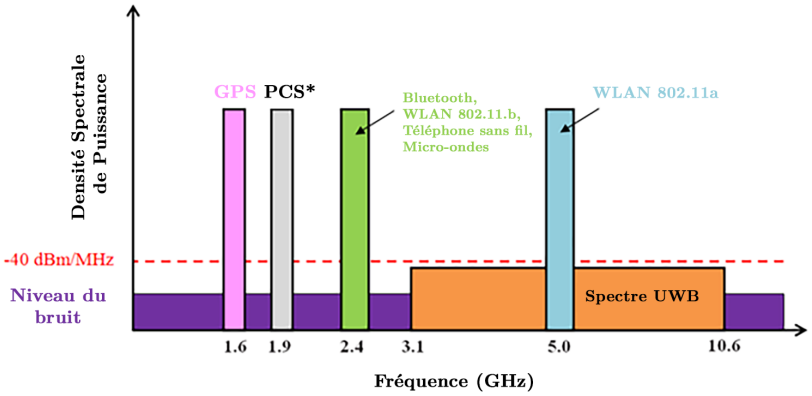


FIGURE 1.2.3 – Graphique représentant la répartition de la puissance d'un signal suivant les fréquences des technologies sans fil [31].

1.2.4.2 Temps de propagation

L'UWB permet de connaître le **temps de propagation** des trames de façon transparente dans la couche MAC. Cette fonction repose sur la mesure du **Time Of Flight** (*Tof*) et permet de déterminer la distance entre deux émetteurs. Le transceiver DW1000 (section 2.1), par exemple, permet de déterminer cette distance avec une erreur de dix centimètres maximum, sous la contrainte que les deux transceivers ne se déplaceraient pas à plus de 18 km/h l'un par rapport à l'autre.

Beaucoup d'applications de cette technique sont possibles telles que l'utilisation de l'UWB pour le suivi de robots dans des entrepôts, la surveillance et le suivi de détenus en prison afin de savoir s'ils ne se rendent pas dans des zones interdites ou leur participation éventuelle à des rixes. Le centre de recherche NASA Johnson Space Center (JSC) a mené une étude afin d'utiliser l'UWB pour le suivi des véhicules et d'astronautes sur Mars lors de mission d'exploration quand des satellites de géolocalisation ne sont pas disponibles [36]. Pour l'Internet des objets, cette fonctionnalité permettrait par exemple de cartographier les capteurs présents dans une habitation. Cela peut être utile, par exemple, pour une application de détection d'incendie.

La mesure du **Time Of Flight** est rendue possible grâce à la modulation par impulsion qui permet de détecter le chemin direct par rapport aux réflexions. Sur la Figure 1.2.4, on distingue que pour le récepteur le chemin direct se repère via la première variation dans le signal. Pour une communication en canal étroit cela n'est pas possible à détecter, car les signaux provenant de réflexions se confondent avec du bruit. La Figure 1.2.5 illustre les réflexions pour une émission sur canal étroit.

Le récepteur peut détecter le signal n'ayant pas subi de rebond (Figure 1.2.6) avec une erreur de moins de cent picosecondes [27]. Soit une erreur théorique d'environ trois centimètres (en prenant le rapport vitesse de la lumière avec le temps).

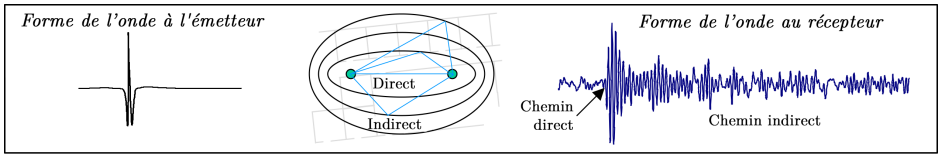


FIGURE 1.2.4 – Illustration de l'influence des réflexions sur la réception d'une onde UWB [27].

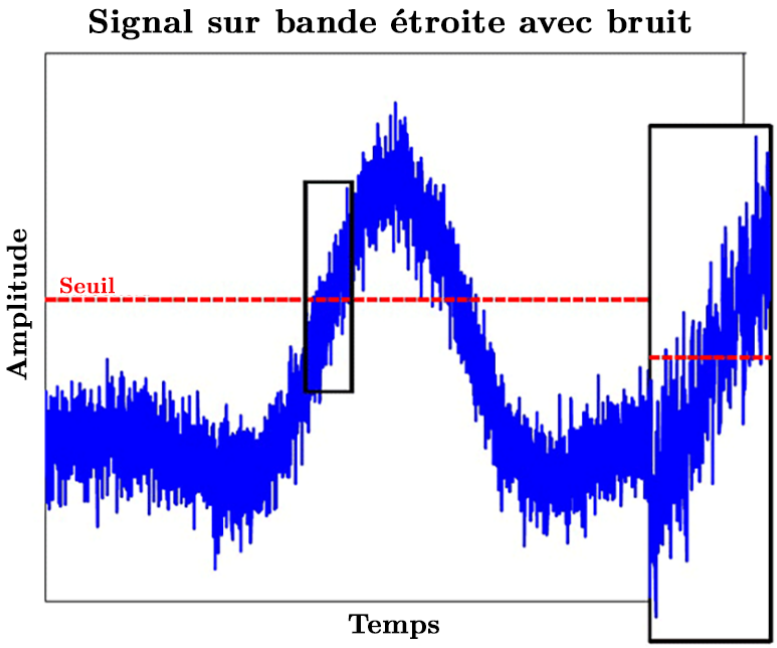


FIGURE 1.2.5 – Illustration du bruit sur un signal à bande étroite [13].

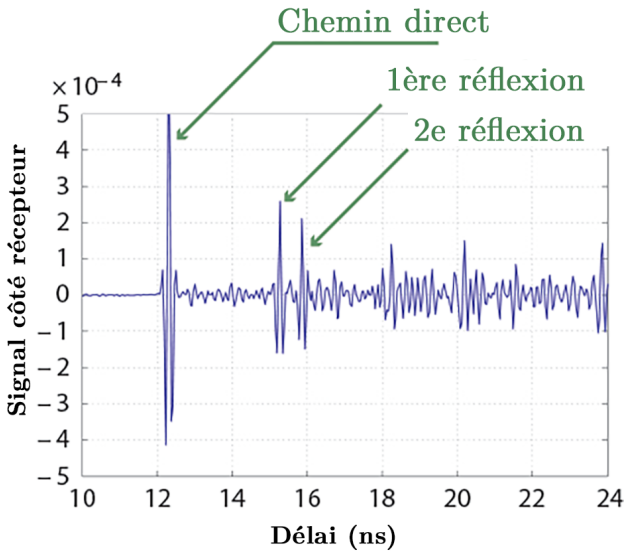


FIGURE 1.2.6 – Illustration théorique des réflexions à la réception d'un signal UWB [33].

Pour déterminer le **Time of Flight** entre deux transceivers A et B, la méthode **two-way time-of-arrival** (TW-TOA) est utilisée. La méthode TW-TOA consiste en la mesure du temps écoulé lors d'un envoi aller-retour de messages entre deux transceivers. Celle-ci est illustrée en figure 1.2.7.

Le Time of Flight (T_{ToF}) peut être calculé comme suit :

$$T_{ToF} = T_t = \frac{T_{tot} - T_{ta}}{2}$$

avec :
T_{tot} = le temps écoulé entre le début d'un message envoyé par A vers B et la réponse reçue par A de B.
T_{ta} = le temps de traitement du transceiver B (réception du message de A et début de l'émission de la réponse).

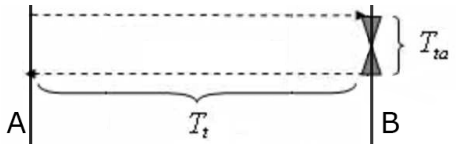


FIGURE 1.2.7 – Illustration du two-way time-of-arrival entre deux transceivers A et B.

L'utilisation du **Time of Flight** pour l'IEEE 802.15.4 est décrite dans le rapport en section 1.2.5.5.

1.2.4.3 Avantages de l'UWB

— **Une grande densité de capteurs.**
L'UWB offre la possibilité de placer beaucoup d'émetteurs dans un espace réduit, grâce à la faible durée des transmissions. Par exemple, un transceiver UWB Decawave 1000 peut compter jusqu'à 11 000 nœuds dans un rayon de 20 mètres [15].

— **Un faible coût de production**
En raison d'une faible complexité de l'émetteur et de l'antenne, un transceiver UWB est très concurrentiel par rapport aux autres technologies. Celui-ci n'a par exemple pas besoin d'une électronique lourde pour faire varier en fréquence le signal émis contrairement aux technologies concurrentes. La Figure 1.2.8 montre par exemple qu'un émetteur UWB n'aura pas besoin d'un oscillateur local et de mixers.

— **Une faible consommation énergétique**
La technologie Ultra Wide Band a été conçue afin de consommer le moins d'énergie possible. Il émet avec une puissance 10 à 1000 fois inférieure aux technologies à canal étroit [29]. Grâce à cela, les capteurs munis d'un transceiver UWB ont une durée de vie de batterie beaucoup plus grande que ceux munis de technologies telles que 802.14.4 ou Bluetooth LE.

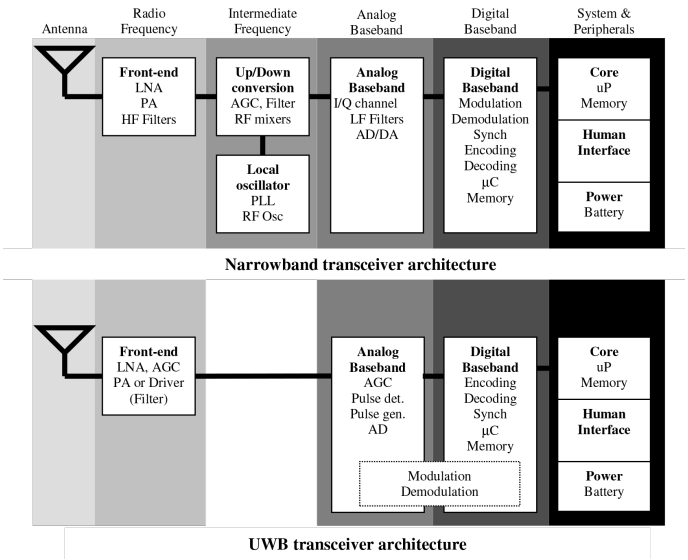


FIGURE 1.2.8 – Comparaison de l'architecture des transceivers UWB avec les transceivers sur canal étroit [14].

1.2.5 Standard IEEE 802.15.4-2011 UWB

En plus des améliorations présentées à la section 1.2.3, l'IEEE 802.15.4-2011 ajoute la prise en charge des canaux Ultra Wide Band. Les trames y gardent le même format, mais permettent en plus de retourner le Time Of Flight.

Les canaux sont répartis sur 3 bandes de fréquences distinctes [8, p.194] :

— Une bande sous-gigahertz, qui consiste en un seul canal et qui occupe le spectre de 249,6 MHz à 749,6 MHz.

— Une bande basse consistant en 4 canaux occupant le spectre de 3,1 à 4,8 GHz.

— Une bande haute consistant en 11 canaux occupant le spectre de 6,0 à 10,6 GHz.

Les canaux UWB sont détaillés dans le tableau 7.2.2 (en annexe). Comme on peut le constater, il y a 3 canaux présents sur tous les transceivers UWB les 12 autres sont optionnels.

1.2.5.1 Format des trames

Une trame MAC 802.15.4-2011 est un message formaté pour correspondre au standard du même nom. Sa taille est de maximum 127 octets. Comme illustré en Figure 1.2.9, on constate qu'une trame est constituée de 3 parties :

1. Le **MAC Header** qui contiendra les informations concernant le type de message : destinataire, type de message, taille, sécurité, etc.
2. Le **MAC payload** qui contiendra les données que l'on souhaite transmettre.
3. Le **MAC Footer** contiendra le **FCS** (Frame Check Sequence) sur deux octets qui permet de vérifier que le message reçu n'a pas été corrompu.

MAC Header (MHR)							MAC Payload	MAC Footer (MFR)
Frame Control	Sequence Number	Destination PAN Identifier	Destination Address	Source PAN Identifier	Source Address	Aux Security Header	Frame Payload	FCS
2 octets	1 octet	0 or 2 octets	0, 2 or 8 octets	0 or 2 octets	0, 2 or 8 octets	0, 5, 6 10 or 14 octets	Variable number of octets	2 octets

FIGURE 1.2.9 – Structure générale d'une trame 802.15.4-2011 [19][fig. 34].

1.2.5.2 MAC Header

La MAC Header est divisée en maximum 7 blocs.

- Le **Frame Control** sur deux octets illustré en figure 1.2.10 permet de décrire la suite de la trame.
  - **Frame Type** définit le type de trame : data, acquittement (ACK), annonce (beacon) ;
  - **Security Enabled** permet de définir si la trame suivra un protocole de sécurité ou pas ;
  - **Frame Pending** permet de spécifier au récepteur que d'autres trames seront envoyées après celle-ci afin qu'il reste en écoute. Cela peut permettre des économies d'énergie ;
  - **ACK Request** permet de spécifier une demande d'envoi d'acquittement par le récepteur ;
  - **Destination/Source Address Mode** et **PAN ID Compress** permet de définir quelles adresses sont présentes dans la trame. Leur fonctionnement est décrit dans la sous-section suivante (1.2.5.3).
- Le **Sequence Number** un entier de 0 à 255 qui permet de différencier les messages entre eux et de les ordonner ; Cela s'avère utile en cas de retransmission de message.

Les 4 champs ci-dessous sont détaillés dans la sous-section suivante.

- Le champ **Destination PAN Identifier** permet de définir un identifiant de groupe ;
- Le champ **Destination Address** permet de définir l'adresse de destination ;
- Le champ **Source PAN Identifier** permet de définir un identifiant de groupe ;
- Le champ **Source Address** permet de définir l'adresse de l'expéditeur ;
- Le champ **Aux Security Header** peut contenir des directives de sécurité pour permettre le cryptage du `MAC Payload` ; On précise la présence de ce champ via le bit `Security Enabled` dans le `Frame Control`.

Bits 0 to 2	Bit 3	Bit 4	Bit 5	Bit 6	Bits 7 to 9	Bits 10 & 11	Bits 12 & 13	Bits 14 & 15
Frame Type	Security Enabled	Frame Pending	ACK Request	PAN ID Compress	Reserved	Dest. Address Mode	Frame Version	Source Address Mode

FIGURE 1.2.10 – Structure des deux octets de `Frame Control` d'une trame 802.15.4 [19][fig. 35].

1.2.5.3 Mode d'adressage

En fonction du type de topologie, plusieurs façons d'adresser un message sont possibles. Les PAN ID permettent de spécifier un groupement (« cluster ») de destinateurs/émetteurs sur 16 bits. Afin que les messages envoyés ne soient pris en compte que par les membres du cluster correspondant. Cela permet par la même occasion de spécifier une adresse de source/destination plus petite pour les membres de Cluster utilisant 16 bits plutôt que 64 avec l'UEID. Le **Frame Control** (fig. 1.2.10) permet de spécifier quel type d'adressage sera utilisé via les bits `Source/Destination Address Mode` et `PAN ID Compress`. `PAN ID Compress` spécifie si les PANID source et destination sont présents ou si seule le `destination PANID` est présent, impliquant que les PAN ID source et destination sont identiques.
Quatre champs sont disponibles à cet effet :

- **Destination PAN Identifier** désignant le « cluster » (PAN) de destination ;
- **Destination Address** désignant une adresse de destination de 16 ou de 64 bits qui dépend de la présence ou non de « cluster » utilisant un même `PANID`. L'adresse sur 64 bits est définie par un standard : ces 24 premiers bits désignent le fabricant et les 40 bits de poids faible le périphérique. Ils sont attribués au périphérique par le fabricant et sont supposés uniques comme pour une adresse MAC. On l'appelle donc la **UEID** pour Unique Extended IDentifier ;
- **Source PAN Identifier** ;
- **Source Address**, l'adresse source. Cette adresse source peut, comme l'adresse de destination, avoir une taille de 16 ou 64 bits.

1.2.5.4 Format physique des trames

Une trame physique UWB est découpée en 4 sections illustrées en figure 1.2.11.

Voici le détail des sections formant une trame physique et leurs utilités.

- L'**en-tête de synchronisation** (SHR) qui permet au récepteur de détecter le message émis via la répétition d'un schéma (pattern). Cet en-tête permet en plus de la détection du message de « synchroniser » le récepteur avec l'émetteur. Plus l'en-tête sera long, plus la synchronisation sera précise. Cela est utile pour l'utilisation de `Rangin Frame` décrite en section suivante (section 1.2.5.5).

L'en-tête de synchronisation est divisée en deux :
— D'une part, une **séquence de préambule** de taille variable de 64 à 4096 symboles (en puissance de 2) permet la détection et la synchronisation ;
— D'autre part, un **délimiteur de début de trame** (SFD) qui peut être de deux types en fonction du PRF (Pulse Repetition Frequency) choisi [19][p.202]. Le PRF désigne la fréquence de répétition des impulsions dans le préambule et dans la portion de donnée d'une trame IEEE 802.15.4-2011.
  * PRF à 16 MHz, SFD de 8 symboles ;
  * PRF à 64 MHz, SFD de 64 symboles.

  Ce délimiteur de trame permet au récepteur d'identifier la fin du préambule et le début de la trame envoyée par l'émetteur ;

- Un **header physique** (PHR) de 21 bits est ensuite émis. Celui-ci contient des informations concernant la MAC frame, comme la taille de celle-ci ;
- La **trame MAC** que l'on fournit au transceiver.

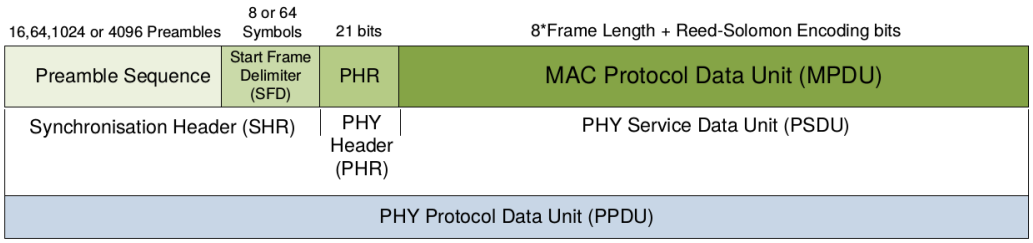


FIGURE 1.2.11 – Structure physique d'une trame IEEE802.15.4-2011 [18]

1.2.5.5 Ranging Frame

Le standard IEEE 802.15.4 UWB supporte le **Time Of Flight** via l'utilisation d'un protocole de **Ranging Frame** (RFRAME) [39].

Il existe deux façons différentes de mesurer le Time Of Flight, la « single-sided » et la « symmetric double-sided ».

La façon « **single-sided** » permet la mesure du Time Of Flight par le transceiver en faisant la demande. Ce protocole repose sur la mesure du temps de propagation aller-retour entre deux transceivers. Il s'agit du `TW-TOA` présenté en section 1.2.4.2. Son utilisation se fait via la couche MAC et la couche `PHY` (physique). L'utilisateur va spécifier au transceiver qu'il souhaite utiliser une `Ranging Frame` via la modification d'un registre. Le transceiver va ensuite modifier le paquet de sorte à utiliser le protocole. L'échange de messages se fera de façon transparente. Une fois l'aller-retour effectué par le message, l'utilisateur ira lire des registres spécifiques dans le transceiver pour récupérer le temps de propagation enregistré sous forme de `timestamp`. L'utilisation de `Ranging Frame` est illustrée en figure 1.2.12, on y voit que le demandeur place un paquet dans la couche MAC ce qui permet de spécifier le destinataire et l'émetteur du paquet. Ensuite, le paquet descend à la couche physique, celle-ci va modifier le `PHY Header` (illustré en figure 1.2.11). Le paquet va ensuite être envoyé et le transceiver correspondant va envoyer une réponse automatique dès la réception du paquet.

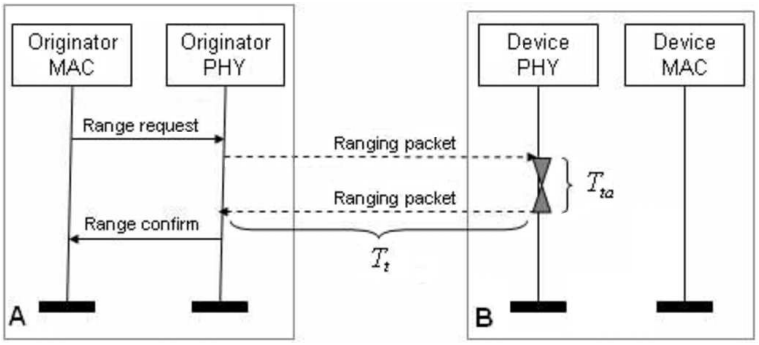


FIGURE 1.2.12 – Échange de messages basé sur le temps aller-retour [39].

La façon « **symmetric double-sided** » permet à deux transceivers de calculer leur Time Of Flight de façon symétrique. La figure 1.2.13, illustre l'utilisation du protocole "Symmetric Double Sided (SDS) TW-TOA". Ce protocole se déroule en 3 phases :

— Une phase de configuration, qui permet d'initialiser le protocole ;
— Le calcul du Time Of Flight entre l'Originator et le Recipient, puis entre le Recipient et l'Originator. Ce calcul permet d'obtenir deux mesures de Time Of Flight, ce qui est plus précis ;
— Le partage du Time Of Flight via le `Timestamp report`.

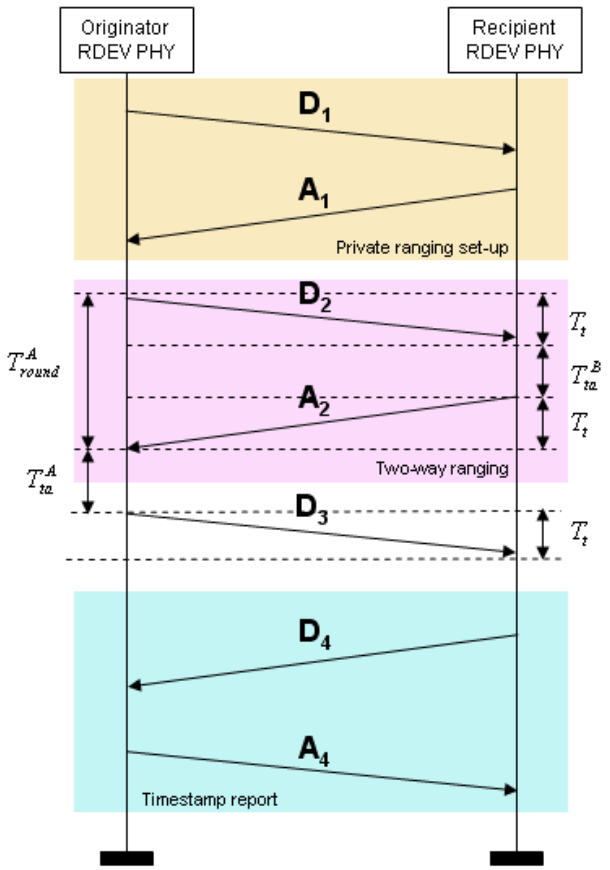


FIGURE 1.2.13 – Illustration du ranging protocol "Symmetric Double Sided (SDS) TW-TOA" supporté par le standard IEEE 802.15.4a [39]

1.3 Comparaison des technologies radio

Comparé à Bluetooth et 802.15.4-2003, l'UWB a la particularité de fournir le support du temps de propagation d'un message. Cela peut permettre de situer géographiquement et avec précision les nœuds dans une topologie. L'UWB est spécifiquement conçue pour les réseaux de capteurs grâce à sa faible consommation et ses différents canaux qui permettent de s'adapter aux besoins et aux contraintes des nœuds (ce que le 802.15.4-2003 permet lui aussi).

On constate que les standards 802.15.4-2003 et -2011 (donc l'UWB) implémentent CSMA/CA ce qui permet de prévenir les collisions et le cas échéant de réduire les pertes en cas de retransmission.
Le coût de fabrication des transceivers UWB est plus faible que celui des transceivers Bluetooth et 802.15.4-2003 du fait que l'UWB n'utilise pas de modulation en fréquence. Le tableau 1.3.1 compare quelques caractéristiques des technologies.

	Bluetooth 4.0	**802.15.4** *	**802.15.4-UWB**
Bande passante (kb/s)	250 à 3000	20 à 250	**20 à 8000**
Nombre de canaux	79	51	3 à 15
Largeur canaux(MHz)	1	0,6 à 5,2	500 à 1355
Taille max des messages (octets)	349	127	**127 ou 1048** †
Nœuds par réseaux	3 Maîtres/21 Esclaves	**65535**	**65535**
Nœuds par réseaux sécurisés [43] [40]	3 Maîtres/21 Esclaves	255	255
CSMA/CA	Non ‡	**Oui**	**Oui**
Time Of Flight	Non	Non	**Oui**
Autonomie (jours)	1 à 7	**100 à 1000+**	**100 à 1000+**
Puissance à l'émission (mW) [29]	< 100	> 1	$\leqslant 0.1$ §

TABLE 1.3.1 – Comparaison des technologies sans fil.

On peut conclure que la technologie 802.15.4-2011 UWB comprend les fonctionnalités des technologies concurrentes avec en prime de nouvelles fonctionnalités telles que le « Time of Flight » et un grand choix de canaux permettant divers débits ce qui permet une grande adaptation aux besoins.

∗. IEEE 802.15.4-2011 sans UWB

†. Les trames de 1048 octets sont dans un format non standard

‡. Du fait que Bleutooth utilise un fonctionnement maître/esclave.

§. La puissance dépend de la largeur du canal utilisé, avec un canal de 500MHz celle-ci est inférieure à 0.037 mW (0.074mW/GHz) [29]

1.4 Contiki OS

Contiki est un système d'exploitation libre crée par le *Swedish Institute of Computer Science* (SICS) de recherche appliquée en informatique [23]. Contiki est conçu pour être très léger et pouvoir être utilisé pour les réseaux de capteurs.
Ses caractéristiques sont :

1. Sa **faible empreinte mémoire**, ne nécessitant que 40 Ko de ROM et 2 Ko de RAM [26].
2. Sa faible **consommation énergétique** via l'utilisation de divers mécanismes ;
   — **Contiki MAC** qui va minimiser le temps où le transceiver est actif (ou en veille) afin qu'elle ne soit pas alimentée 99 % du temps ;
   — La Couche **6LoWPAN** qui va permettre une compression des paquets de données avant leur envoi afin de minimiser les transmissions ;
   — Un **support du routage**, pour minimiser le coût d'un envoi (en nombre de saut/de retransmission) grâce notamment au protocole RPL [2] décrit dans la section suivante ;
   — L'utilisation optionnelle d'une base de données des derniers relevés du capteur afin d'éviter de retransmettre des données si celle-ci n'a pas changé [50].
3. L'implémentation de **μIP** avec les protocoles IPv4, IPv6, TCP, UDP et ICMP ;
4. L'implémentation d'une couche de protocole Rime, qui fournit au niveau applicatif une série de primitives permettant de broadcaster des données dans un réseau de capteurs en Best-Effort à moindre coup. C'est une alternative à la pile IP, cette couche de protocole est conçue pour les réseaux de capteurs et est moins gourmande en ressource que la couche IP [26] ;
5. Sa **portabilité**, écrit en `c`, Contiki OS est compatible avec la plupart des systèmes embarqués utilisés pour les réseaux de capteurs. Les compilateurs `c` sont en général fournis avec les systèmes embarqués.

Contiki utilise un Ordonnanceur événementiel, qui ne gère pas les interruptions. Le fonctionnement du système d'exploitation est basé sur des protothreads, des tâches (suite d'instructions) qui s'exécutent jusqu'à leur terme. Un protothread peut cependant décider volontairement de rendre la main à l'ordonnanceur en invoquant des fonctions qui permettront l'attente d'un événement, l'expiration d'un timer par exemple. À la fin de l'exécution d'un protothread, l'ordonnanceur donnera la main à un autre protothread. Si une interruption se déclenche, le processus (protothreads) en cours d'exécution finira son exécution avant que l'ordonnanceur ne donne la main au protothread correspondant à l'interruption. C'est donc au développeur à faire en sorte que ses protothreads s'exécutent rapidement et à scinder ceux-ci en plusieurs processus s'ils sont trop longs.

1.4.1 Pile réseau

La pile réseau de Contiki est divisée en 7 couches. Chaque couche est dépendante des couches qui lui sont inférieures.

La figure 1.4.1 met en avant les différentes couches de la pile réseau de Contiki ainsi qu'un parallèle avec les différents fichiers qui y sont relatifs.

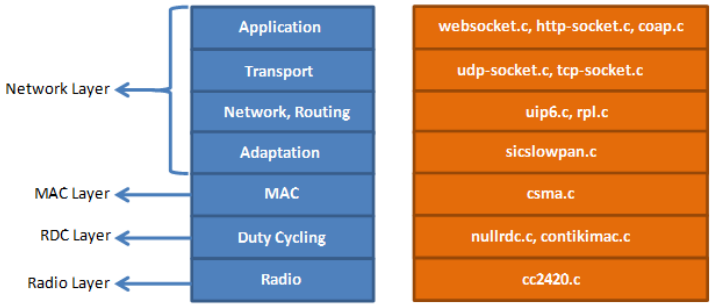


FIGURE 1.4.1 – La pile de réseau de Contiki OS [5].

Nous allons détailler la fonction de chaque couche de cette pile réseau.

7. **Application** il s'agit de la couche applicative, cette couche sert de point d'accès aux services réseaux. Dans Contiki, on y retrouve les sockets web et HTTP ainsi que le « Constrained Application Protocol » (CoAP). CoAP est un protocole permettant les interactions entre machines pour des nœuds contraints et des réseaux contraints dans l'Internet des objets [12]. Celui-ci repose sur UDP (défini au point suivant) et IP, il utilise un Header de petite taille, 4 octets, qui permet de limiter la fragmentation [12] ;

6. **Transport** La couche de transport regroupe les protocoles TCP (Transmission Control Protocol) et UDP (User Datagram Protocol) [37]. TCP est un protocole de transmission fiable qui permet de transmettre des données en détectant les pertes de paquets. UDP quant à lui, est un protocole « Best Effort » de transmission. « Best effort » signifiant « faire au mieux », c'est-à-dire que l'on ne garantit pas que les messages sont délivrés à leurs destinataires. Ce protocole est orienté vitesse de transfert et est adapté à des utilisations où la perte de message n'est pas critique ;
5. **Network**, **Routing** Cette couche implémente le protocole de routage RPL. RPL est l'acronyme de « IPv6 Routing Protocol for Low power and Lossy Networks » [37], il s'agit d'un protocole de routage à vecteur de distance conçu pour les réseaux de capteurs. Il prend en compte le fait que les nœuds peuvent tomber en panne d'où le « Lossy » et que ceux-ci ont des contraintes énergétiques fortes d'où le « Low Power » ;
4. **Adaptation** La couche d'adaptation, composée de 6LoWPAN, sert à compresser et découper les datagrammes (messages) générés par la couche de routage. La couche de routage génère des datagrammes IPv6. Les datagrammes IPv6 ont une taille maximum de 1280 octets [37], or une trame IEEE 802.15.4 utilisée par le CC2420 et le DW1000 ne peut faire que 127 octets maximum en incluant son propre header. Cette couche va donc compresser l'en-tête du datagramme IPv6 reçu et ensuite découper ce datagramme en plusieurs messages afin qu'elle puisse être transportée sur des trames IEEE 802.15.4 ;
3. **MAC** Il s'agit de la couche de contrôle d'accès au support, en anglais Media Access Control (MAC). Cette couche sert d'interface entre les couches logicielles et les couches physiques de la pile réseau. La couche réseau de Contiki comporte un **FRAMER** qui se charge de la construction de trame pour la couche MAC physique. Dans notre cas, le `FRAMER` est configuré pour générer des trames 802.15.4 ;

2. **Duty Cycling** Il s'agit d'une couche de protocole ayant pour but de gérer la consommation du transceiver en la mettant en veille. Contiki supporte plusieurs protocoles : Contiki MAC, X-MAC, NullRDC, etc. Contiki MAC est un protocole ayant pour but de minimiser le temps où le transceiver est actif. Celle-ci peut rester endormie jusqu'à 99% du temps. Sachant que l'usage du transceiver est ce qui est le plus gourmand en énergie, ce protocole permet d'allonger grandement l'autonomie des nœuds. Contiki MAC repose sur l'usage d'un mécanisme de CCA pour « Clear Channel Assessment ». Un CCA consiste à allumer le transceiver en réception et mesurer l'énergie reçue pendant un certain temps. Si cette énergie dépasse un seuil, cela indique qu'un message est peut être émis et Contiki MAC active donc la réception de message. X-MAC est un protocole qui contrairement à Contiki MAC, utilise de petits messages répétés pour avertir un récepteur qu'un message lui est adressé. L'utilisation de petit message permet d'éviter un temps d'attente quand le récepteur détecte qu'un message lui est adressé et permet aux récepteurs dont le message ne leur est pas adressé de se rendormir plus rapidement. NullRDC est quant à lui un protocole qui désactive le radio duty cycle, le transceiver reste donc active 100% du temps en réception ;
1. **Radio** Cette couche comprend le driver radio. Elle est la couche la plus basse dans la pile réseau. Celle-ci interagit avec le transceiver en exécutant les instructions que les couches supérieures lui envoient.

Chapitre 2

Expériences avec un transceiver UWB

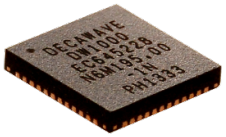
Ce projet consistant à intégrer la technologie UWB dans l'Internet des objets, nous allons nous atteler dans cette section à présenter l'émetteur / récepteur UWB que nous allons utiliser. À savoir le DW1000, la plateforme physique et le système d'exploitation qui interagiront avec le transceiver.

2.1 Transceiver UWB DW1000

Le DW1000 est un émetteur/récepteur(*transceiver*) sans fil fabriqué par DecaWave, utilisant la technologie 802.15.4-2011 UWB (section 1.2.4). Le dispositif se présente sous la forme d'un circuit intégré et d'une antenne planaire en céramique. La figure 2.1.1b présente le dispositif. Sa taille est de 20*10*3 mm.
Les points forts de ce transceiver sont les suivants [16] :

1. Le support du standard IEEE 802.15.4-2011 Ultra Wide Band ;
2. La localisation des objets en temps réel avec une précision de 10 cm si la vitesse de déplacement ne dépasse pas 18 km/h † et disponible avec une précision plus faible jusqu'à 1 750 km/h ‡ ;
3. Un débit nominal allant jusqu'à 6,8 Mb/s ;
4. Une faible durée d'émission qui permet une grande densité de nœuds, jusqu'à 11 000 dans un rayon de 20 mètres ;
5. Une faible consommation qui assure une longue période de fonctionnement sur batterie (en fonction du mode d'utilisation) ;

†. Avec un débit nominal de 110 kb/s [18][Table 5].
‡. Avec un débit nominal de 6,8 Mb/s [18][Table 5].



(a) Transceiver DW1000 [17].

(b) Module DWM1000 [15] (transceiver et son antenne planaire céramique).

FIGURE 2.1.1 – Photos du transceiver DW1000.

6. Le prix du module est faible (20 dollars à la pièce [42]).

Le transceiver supporte 3 vitesses de transfert 150, 850 et 6800 kb/s. Il supporte 6 des 16 canaux du standard IEEE 802.15.4-2011 Ultra Wideband. Ces canaux sont mis en gras dans le tableau Table 7.2.2 présent en annexe. Le DW1000 fournit des automatismes comme la vérification du checksum (CRC) ou l'envoi automatique d'ACK.

Le site du constructeur fournit quelques documents comme un mode d'emploi [19] et une fiche technique [18] très complète. Le mode d'emploi fournit des explications concernant les démarches générales pour utiliser le DW1000. Il fournit aussi des démarches très détaillées de chaque option du transceiver.

2.1.1 Interaction avec le DW1000

Le DW1000 utilise une interface série pour être commandé, plus précisément une liaison SPI (Serial Peripheral Interface). Ce type de liaison est basé sur le principe de maître/esclave. Le périphérique auquel est relié le DW1000 joue le rôle de maître et donne les commandes, il gère aussi le rythme de l'horloge qui régit la transmission de chaque bit sur le bus. Le transceiver joue le rôle d'esclave et exécute ces commandes.

Deux types de commandes sont supportées : des lectures ou des écritures. La lecture permet de vérifier l'état de registres et/ou de lire la dernière trame reçue par le transceiver. L'écriture permet de modifier l'état de registres. La modification de certains registres peut changer l'état du transceiver. Par exemple déclencher l'émission d'une trame.

Un bus SPI utilise quatre signaux logiques :

1. **SCLK** qui sera le signal d'horloge. Ce signal régule l'envoi des bits sur le bus. La vitesse d'envoi des messages dépend de ce signal ;
2. **MOSI** qui signifie Master Output, Slave Input. Il permet l'envoi de bits vers le périphérique esclave ;
3. **MISO** qui signifie Master Input, Slave Output. Il permet de lire les bits transmis par l'esclave ;
4. **SS** qui signifie **Slave Select**. Il permet d'avertir un esclave qu'un signal va lui être transmis (via un état bas). Plusieurs esclaves peuvent être connectés et ils auront chacun un signal SS différent.

Les signaux sont générés par le périphérique maître à l'exception du **MISO** qui lui est géré par l'esclave.
Physiquement, on retrouvera sur le transceiver 6 fils pour les communications SPI, quatre pour les signaux logiques (un par signal), un pour l'alimentation et le fil de masse (GND). La figure 2.1.2 illustre l'interconnexion SPI entre un périphérique maître et un DWM1000.

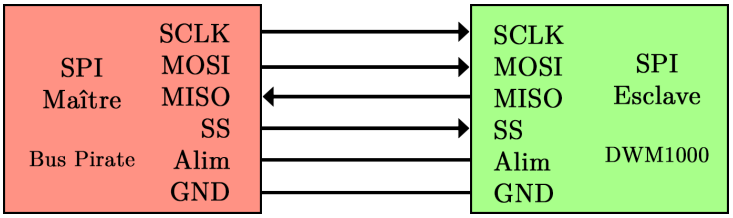


FIGURE 2.1.2 – Schéma résumant l'interconnexion SPI entre un périphérique maître et un DWM1000.
Inspiré d'une illustration Wikipédia [11].

Un exemple d'interaction avec le DW1000 est la lecture du registre « Device ID « (fig. 2.1.3). Celui-ci contient l'identifiant du transceiver et a la valeur constante de `0xDECA0130`. Le maître va envoyer un octet spécifiant le type de commande (lecture) et le registre que l'on souhaite lire, suivi de 4 octets « vides » permettant de garder le rythme de l'horloge et la réception de la réponse de l'esclave. La réponse de l'esclave est envoyée en « little-endian » (partie de poids faible d'abord).

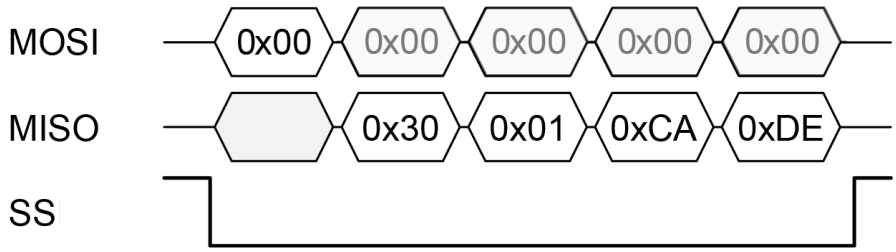


FIGURE 2.1.3 – Exemple de lecture du registre `0x00` `DEVICE_ID` contenant la valeur `0xDECA0130`.
Issue du mode d'emploi du DW1000 [19][p12].

2.1.2 Test au travers d'un Bus Pirate

Nous avons commencé par interagir avec le transceiver à partir d'un ordinateur classique plutôt qu'à partir d'un système embarqué. Les ordinateurs récents ne disposant pas d'interface série SPI accessible facilement (présente sur la carte mère) ; un adaptateur USB/SPI nommé Bus Pirate a été utilisé afin de communiquer avec le transceiver. Le Bus Pirate étant un adaptateur général, on peut y définir les différentes valeurs pour chaque signal logique (SCLK, MOSI, MISO, SS) ainsi que l'alimentation afin de correspondre au périphérique avec lequel on veut communiquer. Enfin, pour communiquer avec le Bus Pirate, la librairie `libbuspirate` [38] a été utilisée. Cette librairie C permet de commander le Bus Pirate et donc d'assurer la communication avec le DW1000 grâce à celle-ci.

La Figure 2.1.4, illustre les connexions entre le Bus Pirate et le transceiver DW1000. La carte électronique verte (sur la droite) étant un `test board` de Decawave fournit par l'université de technologie de Leauleå (Suède) et possédant en son centre un transceiver DW1000 avec son antenne. La carte comprend divers connecteurs et 4 LEDs. À cette carte, nous sommes venus connecter un Bus Pirate (en rouge à gauche sur la photo). On distingue 5 fils sur le bas du `test board` qui correspondent aux connecteurs SPI et à la masse. Les deux fils rouges en haut du `test board` sont des fils d'alimentation. Le Bus Pirate quant à lui se relie à l'ordinateur par un port USB.

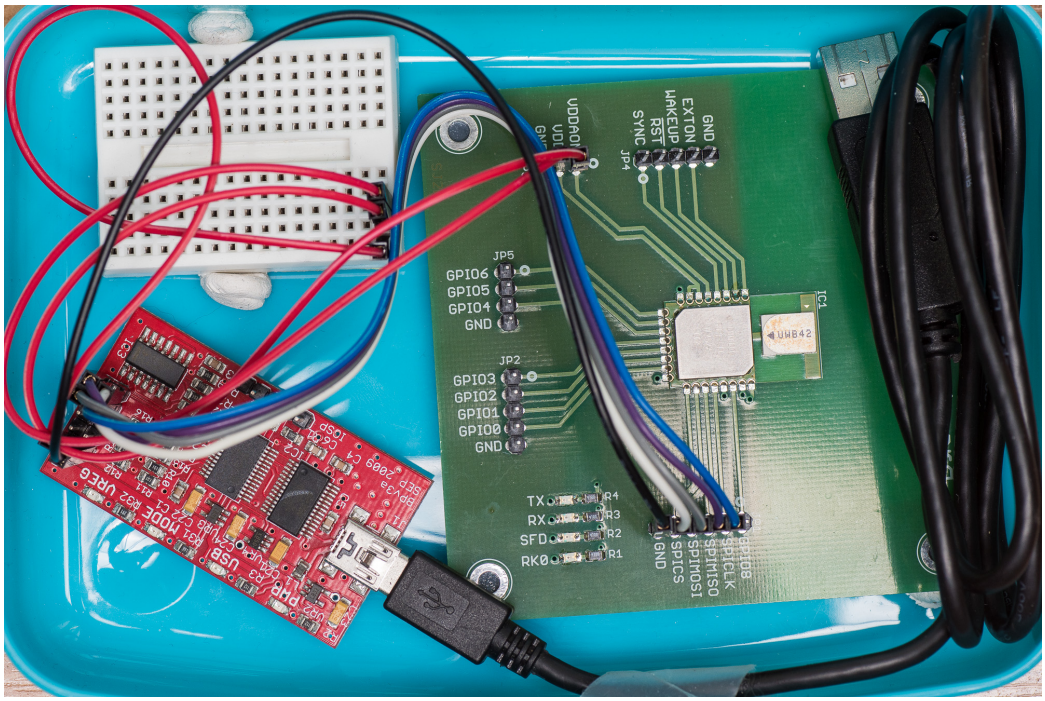


FIGURE 2.1.4 – Un Bus Pirate relié à un `test board` Decawave DWM1000.

Une fois la communication entre l'ordinateur et le DW1000 assurée par un Bus Pirate, nous avons pu commencer à interagir avec le DW1000. Pour cela, nous nous sommes basés sur un driver écrit par Hasan Derhamy et Kim Albertsson de la Lulea University of Technology en Suède. Ce driver conçu pour le DW1000 sur la plateforme Mulle (décrite en section 1.2.1) a dû être adapté pour le Bus Pirate.
Ce driver comprend notamment les fonctionnalités suivantes :

1. La déclaration d'une grande partie des registres et des masques de registre du DW1000 ;
2. La lecture et l'écriture de registres et de sous-registres sur le DW1000 ;
3. Une fonction d'initialisation du DW1000 ;
4. Une structure permettant la configuration du DW1000 en fonction du canal d'émission choisi ;
5. L'envoi de trame ;
6. La réception de trame ;
7. La lecture de divers états du transceiver : Device ID, température, tension, etc ;

8. Des fonctions de diagnostics : Niveau de bruit, amplitude du signal à la réception, la puissance de réception et d'émission ;
9. La génération d'un identifiant unique sur 64 bits ;
10. L'utilisation de ranging frame pour mesure la distance entre deux transceivers.

2.1.2.1 Adaptation du driver

Pour utiliser le DW1000, certaines fonctions ont dû être réécrites. C'est le cas des fonctions de lecture et d'écriture de registres. Afin de nous assurer du bon fonctionnement de ces fonctions, nous avons réalisé des tests à savoir :

— Une fonction qui lira le registre DEVICE ID, registre invariable identifiant le DW1000, et qui comparera la valeur lue avec la valeur attendue afin de vérifier que la lecture est correcte ;
— Une fonction qui ira écrire une chaîne de caractères dans un registre puis qui les lira afin de vérifier la lecture et l'écriture sur le DW1000 ;
— Enfin, un test via le contrôle de GPIO via des registres permet d'allumer des LEDs présentes sur le test board du DW1000.

La création de trames IEEE 802.15.4 a été implémentée afin d'effectuer des tests d'envoi et de réception de trames. Ces fonctions permettent la création de trames avec un adressage long (64 bits) ou court (16 bits). Enfin, une fonction qui permet de tester la construction d'une trame et d'en afficher le contenu a été créée.

2.2 Transmission entre DW1000

La réception de message se fait en plusieurs étapes. Tout d'abord la configuration et l'initialisation du transceiver. Ensuite, en fonction que l'on veuille recevoir ou envoyer un message, l'activation ou non de l'antenne en réception. Enfin, la réception et le traitement du message reçu de façon synchrone ou asynchrone ou l'envoi d'un message. Une fois un message envoyé ou reçu, le transceiver peut à nouveau recevoir ou envoyer un message.

2.2.1 Initialisation du transceiver

L'initialisation du transceiver permet de configurer le transceiver de façon à pouvoir recevoir des messages. Deux transceivers voulant communiquer entre eux, doivent avoir la même configuration (pour la partie obligatoire).

- La **sélection du canal** permet au transceiver d'utiliser un des canaux disponibles. On peut le choisir en fonction de l'utilisation de celui-ci. Théoriquement certain canaux consomme moins d'énergie que d'autres car ont une largeur de bande plus étroite [29], mais cela n'est pas confirmé par la fiche technique du transceiver. La choix du canal implique d'autres configuration plus bas niveau que nous ne détailleront pas (le choix du code de préambule par exemple) ;
- Le choix du **débit** nominal, celui-ci peut être de 110 kb/s, 850 kb/s ou 6,8 Mb/s ;
- Le choix de la longueur du **préambule** qui peut-être fixé à 64, 128, 256, 512, 1024, 2048 ou 4096. La longueur se choisit en fonction du débit, plus le débit est faible, plus le préambule doit être long ;
- La configuration de l'**adressage du transceiver** comprend l'adresse courte (16 bits) et l'adresse « Unique Extended IDentifier » (64 bits) et le PAN Identifier. Comme décrit dans la section 1.2.5.3.

2.2.2 Envoi

Une fois l'étape de configuration réalisée, le transceiver est en mesure d'envoyer des messages. Pour cela, il suffit, après avoir construit une trame au format IEEE 802.15.4, de la placer dans le buffer de transmission présent sur le transceiver et d'indiquer dans un autre buffer la taille de ce message. Il faut tenir compte du CRC présent en fin de trame IEEE 802.15.4. Deux options s'offrent à l'utilisateur : préciser une taille incrémentée de deux au transceiver et celui-ci calculera automatiquement le CRC ; ou ajouter le CRC à la trame et préciser au transceiver que la trame contient déjà le CRC. Ensuite, on initialise la transmission en passant le bit « Transmit Start » (`TXSTRT`) du registre « System Control » à 1.

2.2.3 Réception

Une fois l'étape de configuration réalisée, le transceiver est en mesure de recevoir les messages. Pour cela, il suffit d'activer le mode de réception en allumant l'antenne en réception.

2.2.3.1 Réception synchrone

Un réception synchrone désigne le fait que l'on laisse le périphérique maître vérifier constamment le status de réception de message. Cela implique que le périphérique maître reste dans une boucle et ne fait donc rien d'autre qu'attendre la réception d'un message. Pour cela, on va vérifier l'état du bit `RXDFR` « Receiver Data Frame Ready » du registre `SYS_STATUS` indiquant si un message est reçu tant que `RXDFR` vaut 0 [19]. Si `RXDFR` vaut 1, un message est reçu et on peut le traiter.

2.2.3.2 Réception asynchrone

Un réception asynchrone désigne le fait que l'on donne au transceiver la tâche de signaler la réception d'un message via une interruption (cela ce fait à l'initialisation). On laisse donc le périphérique maître exécuter un autre code pendant que le transceiver est en train d'attendre la réception d'un message. Une fois qu'un message est reçu, une routine d'interruption est déclenchée et celle-ci traite le message. Cette approche n'est pas possible avec un Bus Pirate qui ne supporte pas les interruptions.

2.2.3.3 Traitement du message reçu

Une fois qu'un message est reçu par le transceiver, celui-ci le place dans le buffer de réception. Le transceiver place aussi divers informations dans d'autres buffers comme la longueur du message, ou encore le niveau de bruit mesuré lors de la réception de celui-ci. Si à l'envoi, la demande du « Time Of Flight » était précisée, le transceiver enregistre un `timestamp` dans un registre spécifique. Pour lire un message, il suffit de lire la longueur de celui-ci dans le registre « RX Frame Information » (`RX_FINFO`) sous le masque « Receive Frame Length ». Une fois la longueur du message connue, on peut lire le registre « RX Frame Buffer » (`RX_BUFFER`) contenant le message en se basant sur sa taille précédemment lue.

2.3 Validation du driver de l'Ulea

Les transceivers DW1000 utilisent le standard 802.15.4. Pour les faire communiquer entre eux, il est nécessaire de construire des trames respectant ce standard.

Cette construction se fera en 3 étapes :

— La construction manuellement du **MAC Header**, comme illustré en section 1.2.5.1.
  Concernant la taille de la trame, celle-ci se calcule en additionnant la taille du Header avec celle du payload et les 2 octets du FCS présent dans le MAC Footer. Si la taille dépasse 127 octets, il faudra alors fragmenter la trame. C'est-à-dire scinder la trame en morceaux plus petits afin de ne pas excéder 127 octets. Afin de garder le message fragmenté compréhensible pour le récepteur, on incrémentera le numéro de séquence. Le récepteur sera alors en mesure de fusionner les payloads de chaque trame afin de retrouver le message original ;

— L'ajout du **MAC payload** ;

— Le **MAC footer**, le FCS sera calculé et ajouté automatiquement à la trame par le transceiver. Sa présence n'est donc prise en compte que lors du calcul de la taille de la trame.

Une fois la construction de la trame effectuée, nous avons pu faire communiquer de façon synchrone les transceivers entre eux en utilisant les paramètres décrits dans le tableau 2.3.1.

Paramètre	**Configuration**
Débit	850 kb/s
Canal	n°5 (499 MHz)
Préambule code	n°3
Préambule	256 symboles
Adressage	16 bits
Acquittement	Non
Broadcast	Non

TABLE 2.3.1 – Paramètre choisi pour le test du driver de l'Ulea.

À la suite de cela, nous avons considéré que le driver était fonctionnel et pouvait donc être porté sur une autre plateforme, la plateforme Zolertia Z1 décrite dans le chapitre suivant. Cette plateforme étant conçue pour faire fonctionner Conkiti OS.

Chapitre 3

Intégration à Contiki d'un transceiver UWB

3.1 Structure d'un driver radio dans Contiki

3.1.1 Place du driver dans la pile réseau

Le driver radio se situe dans la couche la plus basse de la pile réseau détaillée en section 1.4.1. Ce driver sert d'interface entre le système d'exploitation et le transceiver et permet d'interagir avec celui-ci. La figure 3.1.1, ré illustre la pile réseau de Contiki et fait un parallèle entre les couches réseaux à gauche et l'implémentation en `c` à droite. On peut voir que la couche radio comporte le fichier « `cc2420.c` ». Ce fichier est le driver radio du transceiver CC2420, ce même transceiver est présent sur le Zolertia Z1. Ce fichier va nous servir de modèle pour l'implémentation du driver radio.

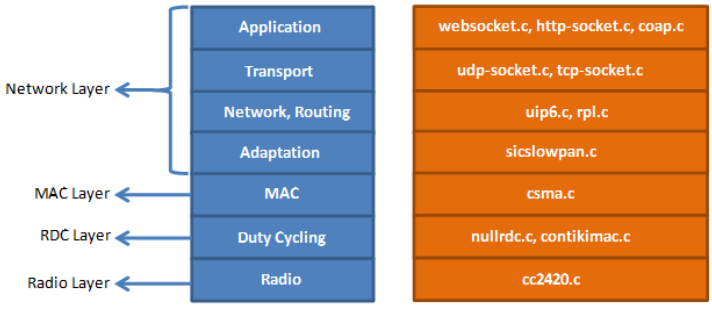


FIGURE 3.1.1 – La pile de réseau de Contiki OS [5].

Le driver radio est défini à l'aide d'une interface modélisée sous forme d'une structure `c`, la structure « `radio_driver` ». Cette structure est détaillée dans la section ci-dessous.

3.1.2 Structure « radio_driver »

La structure « `radio_driver` », illustrée en figure 3.1.2, permet d'avoir une vue d'ensemble des différentes fonctionnalités du driver radio.

```
struct radio_driver {
  int (* init)(void);
  int (* prepare)(const void *payload,
                  unsigned short payload_len);
  int (* transmit)(unsigned short transmit_len);
  int (* send)(const void *payload,
                unsigned short payload_len);
  int (* read)(void *buf,
                unsigned short buf_len);
  int (* channel_clear)(void);
  int (* receiving_packet)(void);
  int (* pending_packet)(void);
  int (* on)(void);
  int (* off)(void);
};
```

FIGURE 3.1.2 – Structure « `radio_driver` » définie dans « `/core/dev/radio.h` » de Contiki 2.7.

Cette structure est composée de 10 pointeurs vers des fonctions dont une brève description est donnée dans le tableau 3.1.1. Comme on peut le voir, ces 10 fonctions permettent l'envoi et la réception de message sur le transceiver ainsi que son initialisation.

Concernant la réception, celle-ci est signalée via une interruption. Une interruption désigne un signal électrique émis par un périphérique pour signaler un événement. Dans notre cas, le transceiver émet un signal électrique pour indiquer la réception d'un message. Ce signal électrique provoque une interruption dans le système d'exploitation. Cette interruption se présente sous forme d'une fonction qui est appelée quand le système détecte le signal électrique. On appelle cela une interruption du fait que le protothread en train d'être exécuté au moment de la réception du signal et est interrompu le temps de lancer la fonction d'interruption. Dans Contiki, on peut assigner un mécanisme d'interruption à un port grâce à la macro « `ISR(port, nom_interruption){...}` » où « `port` » désigne le port à surveiller et « `nom_interruption` » désigne un nom que l'on souhaite donner à l'interruption et qui peut-être utilisé pour le débogage. Entre les accolades, on placera la routine à exécuter lors du déclenchement de l'interruption.

Primitive	Description
init()	Se charge de l'**initialisation** de la configuration SPI et du transceiver.
prepare(...)	**Prépare la transmission** en envoyant la trame pointée par le pointeur « `payload` » dans le buffer du transceiver.
transmit(...)	**Initialise la transmission** et renvoi si la transmission s'est bien déroulée.
send(...)	Effectue un **envoi complet** en appelant la fonction « `prepare` » puis la fonction « `transmit` » et retourne le résultat de la transmission.
read(...)	**Lit un message** contenu dans le buffer du transceiver et le copie dans le tableau pointé par « `*buf` » et la taille de ce message est retourné par « `buf_len` ».
channel_clear()	Active le mécanisme de **Clear Channel Assessment** (CCA) décrit dans la couche Duty Cycling de la pile réseaux en section 3.
receiving_packet()	Lire l'état de réception du message sur le transceiver pour savoir si celui-ci est en train de recevoir un message. Cette méthode permet d'éviter d'interrompre la réception d'un message lorsque l'on souhaite éteindre l'antenne ou envoyer un message.
pending_packet()	Permet de savoir si un message a été reçu et que celui-ci est en attente de traitement.
on()	Allume l'antenne du transceiver en réception.
off()	Éteint l'antenne du transceiver.

TABLE 3.1.1 – Brève description des primitives de la structure « `radio_driver` » [51] [3].

3.1.3 Exemple du CC2420

Pour l'intégration du driver radio, nous nous sommes basés sur les fichiers du `driver_radio` du `CC2420` comme mentionné en section 3.1.1. Ce driver est divisé en plusieurs fichiers. Des fichiers présents dans le dossier `/core/dev/` :

- « **cc2420.c/h** » qui contient la structure « `radio_driver` » et l'implémentation des fonctions de celle-ci ;
- « **cc2420_const.h** » qui contient les déclarations des constantes nécessaires à l'utilisation du CC2420, le nom des registres et leurs adresses mémoires entre autres ;
- « **cc2420-arch.c** » présent dans le dossier `/platform/z1/dev/` qui contient la configuration SPI dépendante de la plateforme Zolertia Z1. Ce fichier nous informe que la routine d'interruption utilise le même port que l'accéléromètre adxl345 présent sur le Zolertia Z1. De ce fait, la routine d'interruption du CC2420 est intégrée à celle de l'accéléromètre adxl345. Cette routine est présente dans le driver de l'adxl345 situé dans le fichier « `/platform/z1/dev/adxl345.c` ».

3.2 Plateforme utilisée

Une plateforme hôte autonome est nécessaire pour y incorporer Contiki. Dans le cadre du projet, la plateforme **Zolertia Z1** a été choisie. La plateforme Zolertia Z1 (fig 3.2.1) est une des cartes électroniques les plus utilisées comme plateforme de développement pour les réseaux de capteurs sans fil [53]. Celle-ci possède un processeur MSP430, de la RAM, une mémoire flash ainsi que divers capteurs. La plateforme Zolertia Z1 est supportée par plusieurs systèmes d'exploitation comme RIOT, OpenWSN et Contiki [53]. Celle-ci comporte un transceiver, le CC2420 qui supporte le standard IEEE802.15.4-2003 (sans UWB). Le CC2420 est un transceiver qui utilise des bandes de fréquences étroites contrairement au transceiver Decawave DW1000.

La première partie de l'intégration sous Contiki s'est donc déroulée comme avec le Bus Pirate. Il a s'agit de faire communiquer le Zolertia Z1 avec le DW1000. Pour cela, il a fallu configurer les ports SPI du Zolertia et réécrire les fonctions de lecture et d'écriture SPI vers le DW1000.

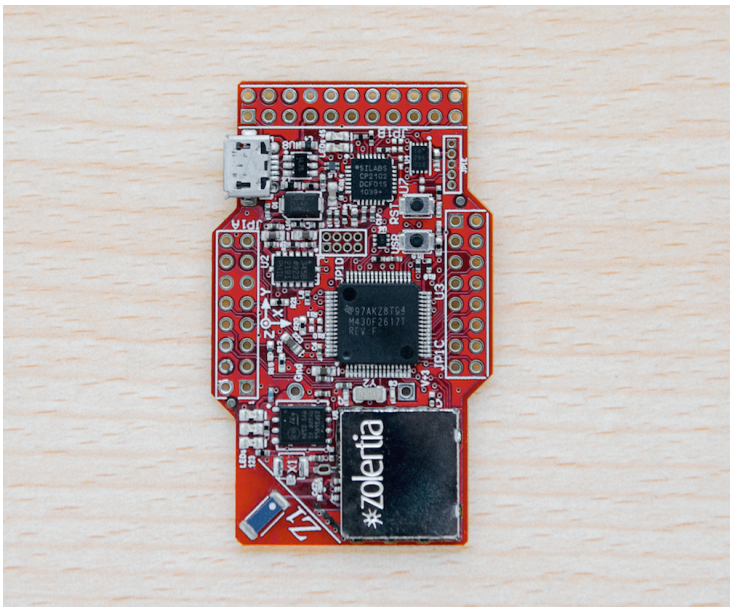


FIGURE 3.2.1 – Photographie d'une carte Zolertia Z1 [53].

La base du driver du DW1000 est relativement indépendante des plateformes. En effet, seules les méthodes d'accès aux registres de base, les lectures et écritures de registre générique sont dépendantes de la plateforme qui exécute le code. Nous avons pris en compte cette observation. La structure du code a donc été réfléchie afin d'être compatible avec le Zolertia Z1 et le DW1000 en même temps. La structuration est décrite en section 3.4.1.

Contrairement au Bus Pirate, le code développé pour le Zolertia Z1 est exécuté par le processeur MSP430 du Zolertia Z1. À chaque mise à jour du code, celui-ci doit être recompilé puis écrit sur le Zolertia Z1. Ensuite, le Zolertia Z1 redémarre et exécute le code fraîchement écrit dans sa mémoire. Cela entraine un délai non négligeable au développement, il faut donc s'assurer que le code est correctement écrit avant de le tester sur le Zolertia Z1.

Lors des premières phases du développement, nous avons utilisé deux transceivers, l'un connecté à un Zolertia Z1 et le second connecté à un Bus Pirate. Cette approche permet de monitorer les communications du transceiver connecté au Zolertia Z1. Elle permet aussi d'apporter des modifications rapides au code et de le tester sans devoir réécrire la mémoire flash du Zolertia Z1 à chaque fois.

3.2.1 Zolertia Z1 et DW1000

Il est important de noter que le Zolertia Z1 ne possède pas une alimentation suffisante pour alimenter lui-même le DWM1000. Pour alimenter le transceiver, nous avons donc utilisé un Bus Pirate configuré pour fournir une alimentation 3.3 volts en sortie. La figure 3.2.2 illustre le schéma de connexion SPI en incluant l'alimentation (le Bus Pirate) entre un Zolertia Z1 et un DWM1000.

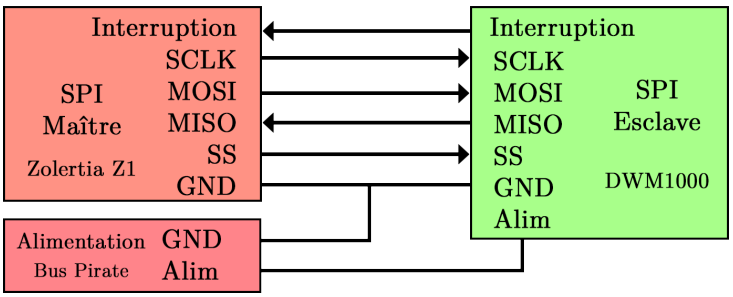


FIGURE 3.2.2 – Schéma résumant l'interconnexion SPI entre un Zolertia Z1 et un DWM1000.
Inspiré d'une illustration Wikipédia [11].

Le montage électrique est illustré sur la photographie présente en figure 3.2.3. Le DWM1000 se situe en haut à droite, le Zolertia Z1 en haut à gauche et le Bus Pirate en bas à gauche. Deux câbles relient le DW1000 au Bus Pirate pour l'alimentation (masse et alimentation), 6 autres câbles sont utilisés pour connecter le DW1000 au Zolertia Z1. La configuration du câblage est décrite en annexe dans la Table 7.3.1. Dans ce tableau, on remarque l'ajout d'un câble par rapport au câblage du Bus Pirate, celui-ci permet au DW1000 d'envoyer des interruptions vers le Zolertia Z1.

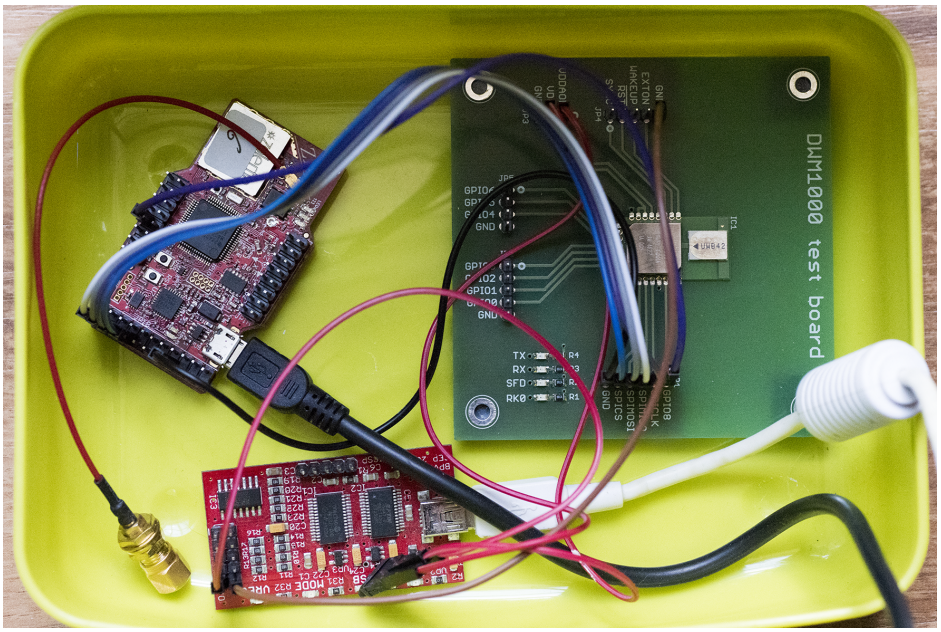


FIGURE 3.2.3 – Photographie des connexions entre un Zolertia Z1, un Bus Pirate et un DWM1000.

3.3 Mode opératoire du transceiver DW1000

Nous allons détailler le fonctionnement des mécanismes utilisés dans le driver ainsi que le mécanisme d'interruption.

3.3.1 Initialisation du transceiver

L'initialisation du transceiver est une étape fort semblable en réception synchrone et asynchrone. Celle-ci permet de configurer le transceiver de façon à pouvoir recevoir des messages Nous allons présenter cette étape sous forme de liste, les éléments marqués par une « boulette » (•) sont des étapes de configuration obligatoire que l'on retrouve déjà dans la section 2.2.

- La **sélection du canal** ;
- Le choix du **débit** nominal ;
- Le choix de la longueur du **préambule** ;
- La configuration de l'**adressage du transceiver**.

— **Activation du filtrage des trames**, il s'agit d'un mécanisme qui va analyser les messages reçus et rejeter ceux mal formés automatiquement. Cela permet aussi au système de détecter le type de message reçu ;

— **Activation de l'envoi automatique d'acquittement**. Les filtres précédemment activés permettent au transceiver de reconnaître le type de message reçu. Si un message reçu a un CRC correct et qu'il contient une demande d'acquittement, le transceiver envoie automatiquement un acquittement. Une demande d'acquittement étant détectée à la lecture du bit `ACK Request` dans le `Frame Header` décrit en section 1.2.5.1 ;

— **Activation des interruptions**. Sur le transceiver, cela se fait via le passage à 1 d'un bit dans le registre `SYS_STATUS` pour System Event Status Register. Le bit en question est le bit 13 `RXDFR` pour Receiver Data Frame Ready [19]. Dans notre cas, on active l'interruption lors de la réception d'un message bien formé. Pour cela on passe dans le registre System Event Mask (`SYS_MASK`), le bit Mask receiver FCS good event (`MRXFCG`) à 1 [19] ;

— **Activation du ré allumage automatique de l'antenne en réception** via le bit `RXAUTR` (Receiver Auto-Re-enable) dans le registre de configuration système `SYS_CFG` [19]. Cela permet à l'antenne de se réactiver automatiquement en cas d'erreur à la réception d'un message. Par exemple, la réception d'un préambule correct suivi d'un header de la couche physique mal formé cause une erreur. En mode normal, l'antenne s'éteint, ici elle se rallumera automatiquement ;

— **Désactivation de la mise en veille de l'antenne**. On configure le Receive Wait Timeout Enable à faux [19]. Pour cela, on passe le bit `RXWTOE` (Receive Wait Timeout Enable) à 0 dans le registre de configuration système `SYS_CFG` [19].

3.3.2 Envoi

Pour envoyer un message, il suffit après avoir construit une trame au format IEEE 802.15.4. Cela est réalisé par les couches supérieures à celle du driver dans la pile réseau, de la placer dans le buffer de transmission présent sur le transceiver et d'indiquer dans un autre buffer la taille de ce message. Il faut tenir compte du CRC présent en fin de trame IEEE 802.15.4. Pour cela, on précise une taille incrémentée de deux au transceiver et celui-ci calculera automatiquement le CRC. Ensuite on initialise la transmission.

3.3.2.1 Acquittement automatique

Une fois la transmission finit, et si le message envoyé contenait une demande d'acquittement, on active l'antenne en réception. Une fois l'antenne activée on va effectuer une réception synchrone en attendant l'acquittement, i.e une boucle. Cette réception va être majorée par un temps maximal. En effet, l'acquittement peut ne jamais arriver si celui-ci est perdu ou corrompu. Un acquittement se caractérise par un message d'une longueur de 5 octets ; 2 octets de `Frame Control`, 1 octet avec le numéro de séquence de la trame acquittée et 2 octets de CRC. Si l'on ne reçoit pas d'acquittement dans le temps impartit, on le signal via une valeur de retour spécifique.

3.3.2.2 Valeurs de retour

Une transmission peut mal se dérouler, la structure radio driver possède donc plusieurs valeurs de retour qui sont définies dans une énumération illustrée en figure 3.3.1. Dans cette énumération, on retrouve 4 valeurs possibles :

- « **RADIO_TX_OK** » qui signifie que la transmission s'est bien déroulée ;
- « **RADIO_TX_ERROR** » qui signifie qu'il y a eu une erreur à la transmission ;
- « **RADIO_TX_COLLISION** » qui signifie qu'une collision a été détectée pendant la transmission ;
- « **RADIO_TX_NOACK** » qui signifie que le transceiver n'a pas reçu d'acquittement après l'envoi de la trame.

```
enum {
  RADIO_TX_OK,
  RADIO_TX_ERR,
  RADIO_TX_COLLISION,
  RADIO_TX_NOACK,
};
```

FIGURE 3.3.1 – Valeurs de retour des fonctions « `send` » et « `transmit` » définies dans « `/core/dev/radio.h` ».

3.3.3 Réception

Une fois l'étape de configuration réalisée, le transceiver est en mesure de recevoir les messages. Pour cela, il suffit d'activer le mode de réception en allumant l'antenne en réception. Dans le cadre de ce driver, une réception asynchrone est implémentée. On peut donc laisser le Zolertia Z1 exécuter un autre code pendant que le transceiver est en train d'attendre la réception d'un message. Une fois qu'un message est reçu, une routine d'interruption est envoyée. Cette routine se décline en une fonction qui consiste a aller lire si le message reçu par le DW1000 est bien formé et qu'il n'y a pas d'erreur. Si c'est le cas, celui-ci prévient Contiki qu'un processus doit être exécuté le plus rapidement possible afin de lire le message.

3.3.4 Clear Channel Assessment

Les technologies radios « narrow band » permettent de détecter l'émission de paquet grâce à l'augmentation de la puissance reçue sur le canal utilisé. Il suffit de laisser l'antenne en réception un certain laps de temps et de vérifier après celui-ci si une hausse de puissance a été détectée pour déduire si le canal est occupé ou non. En « Ultra Wide Band » cette approche n'est pas possible, un message est émis à une très faible puissance. Sous le seuil de bruit des technologies UWB comme illustré en figure 1.2.3. Afin de détecter un message, un transceiver UWB commence son message par un préambule comme illustré en section 1.2.5.4. Ce préambule permet la détection du message. Des symboles ayant un schéma (pattern) spécifique permettent au transceiver de détecter le message et ensuite de se synchroniser avec l'autre émetteur. La suite du message ne contient plus de préambule et peut être interprétée grâce à la synchronisation effectuée précédemment.

Dans le cas de l'UWB, on peut donc détecter un message via ce préambule. Nous allons regarder la proportion qu'occupe le préambule dans une trame. Plus cette proportion est grande, plus ont aura de chance de détecter un paquet entrain d'être émis. La proportion du préambule dans la trame physique varie en fonction de la vitesse de transmission, et ce, sur deux facteurs. Premièrement, sa longueur, plus le débit est faible, plus le préambule va contenir de symboles. Le préambule est plus long du fait que chaque symbole d'un préambule est toujours émis à la même vitesse à savoir 1 μs par symboles. Deuxièmement, avec un plus faible débit, la partie MAC de la trame physique est émise en plus de temps. Le tableau 3.3.1 met en avant le temps passé pour émettre le préambule et le `SFD` en fonction du débit du DW1000. Une formule est donnée en figure 4.2.2 pour calculer le temps passé en transmission pour un paquet. En nous basant sur cette formule et sur le tableau 3.3.1, nous allons générer un graphique afin de mettre en avant la proportion du temps de transmission prise par le préambule. Comme on peut le voir sur la figure 3.3.2, la proportion du temps passé pour émettre le préambule est fort variable en fonction de la configuration du transceiver. Elle varie en fonction du débit nominal choisi, des paramètres qui peuvent varier selon le type d'utilisation et en fonction de la longueur de la trame MAC. Plus un préambule représentera une grande proportion de la trame émise, moins la durée d'un « Clear Channel Assessment » sera grande. Un débit de 6.8 Mb/s est donc à favoriser par rapport à un débit de 100 kb/s.

Débit (kb/s)	**Temps d'émission du préambule** $t_{preamble}$ (μs)	**Temps d'émission du délimiteur de trame** t_{SFD}(μs)
6800	64 à 256	8
850	128 à 1024	8
110	1024 à 4028	64

TABLE 3.3.1 – Paramètres recommandés en fonction du débit [19][p196] et/ou utilisés dans la fiche technique [18][p29].

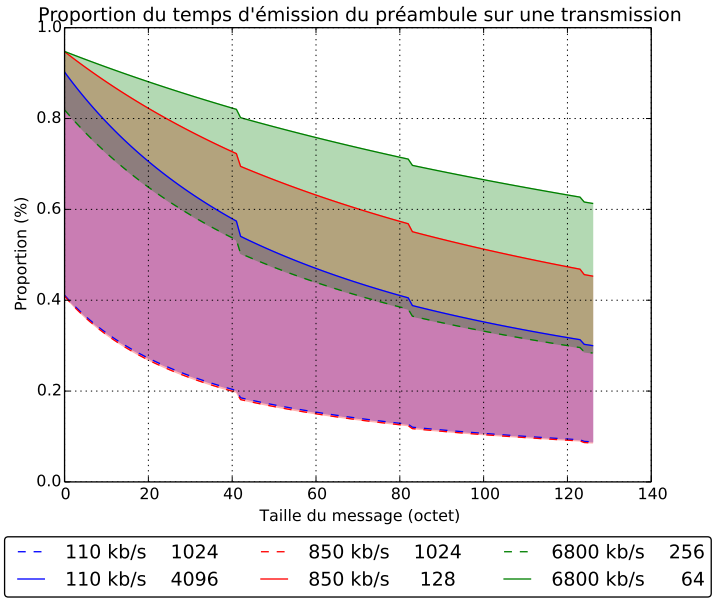


FIGURE 3.3.2 – Proportion théorique du temps d'émission du préambule sur une transmission de taille variable.

3.4 Implémentation du driver du DW1000

Afin d'intégrer au mieux le driver dans Contiki, nous avons créé une nouvelle plateforme appelée « z1-dw1000 » en copie de l'original afin de garder la plateforme avec le driver du CC2420 utilisable. En pratique, deux dossiers ont été ajouté :

— un dossier `/platform/z1-dw1000/` contenant la configuration de la plateforme Zolertia Z1 ;

— un dossier `/example/z1-dw1000/dw1000/` contenant les fichiers du driver ; Pour l'utilisation du driver, il suffit de donner à la variable « `TARGET` » la valeur « `z1-dw1000` » dans le Makefile utilisé pour compiler votre programme.

3.4.1 Structuration du driver

Le langage `c` repose sur deux types de fichiers : les fichiers « **.c** » qui contiennent les fonctions exécutées par le langage et les fichiers « **.h** » qui contiennent des déclarations de fonction, structures et constantes. Les fichiers « `.h` » sont inclus en début de fichier « `.c` ». Nous avons décidé de découper le code en plusieurs fichiers :

- « **dw1000**.c/.h » le fichier de base du driver, celui-ci regroupe toutes les fonctions permettant les interactions avec le transceiver ;
- « **dw1000-driver**.c/.h » le driver radio pour le DW1000 sur Contiki décrit en section 3.1.2 ; Il fait appel aux fonctions présentes dans le fichier `driver.c`.
- « **driver-const**.h » contient les déclarations des constantes nécessaires à la manipulation des registres du DW1000. À savoir, les adresses des registres, les positions de bits spécifiques et leurs masques ;
- « **dw1000-arch-**x.c/.h » contient les fonctions permettant de lire et décrire dans un registre du DW1000 sur la plateforme x et d'initialiser la configuration SPI sur celle-ci. Dans notre cas, nous avons deux fichiers de ce type : `dw1000-arch-bp` pour le Bus Pirate et `dw1000-arch-z1` pour le Zolertia Z1 ;
- « **dw1000-util**.c/.h » est un fichier qui regroupe des fonctions utilitaires pour le DW1000 :
  — Construire une trame 802.15.4 ;
  — Afficher une trame 802.15.4. en la décomposant suivant sa structure ;

— Afficher de façon détaillée l'état du registre `SYS_STATUS` du DW-1000. Ce registre permet de diagnostiquer divers statuts du DW-1000. Par exemple, le fait qu'un message a été reçu ou les erreurs déclenchées lors de la réception d'un message : Erreur dans le « MAC Header » par exemple.

Ces fichiers sont placés dans le dossier `/examples/z1-dw1000/dw1000/` afin de pouvoir conserver leurs utilisations avec le Bus Pirate. Les fichiers `dw1000` et `driver-const` pourrait être regroupé en un seul fichier, mais cela deviendrait trop peu lisible.

3.4.2 Substitution du driver radio

Le Zolertia Z1 utilise de base le transceiver CC2420 pour communiquer. Dans le cadre de ce projet, il a fallu remplacer la couche driver radio utilisant le CC2420 par le driver radio que nous avons créé. Concrètement, Contiki utilise des variables pour définir les différentes couches réseaux à utiliser. On les retrouve dans le fichier « `contiki-conf.h` » et « `platform-conf.h` » présent dans le dossier `/platform/z1-dw1000/`. La couche radio driver est définie grâce à la constante « `NETSTACK_CONF_RADIO` », nous avons remplacé sa valeur originale « `cc2420_driver` » par « `dw1000_driver` », qui est la structure radio illustrée en figure 3.4.3. La pile réseau est illustrée en figure 3.4.1, on constate que l'on a défini le protocole « NullRDC » comme protocole de Duty Cycle, l'antenne restera donc constamment allumée.

Une autre constante a été définie, « `DW1000_CONF_AUTOACK` » qui permet de spécifié si le driver radio utilise ou non la fonctionnalité d'envoi automatique d'acquittement (si la trame contient une demande d'acquittement).

```
1 /* Network setup for IPv6 */
2 #define NETSTACK_CONF_NETWORK sicslowpan_driver
3 #define NETSTACK_CONF_MAC     csma_driver
4 #define NETSTACK_CONF_RDC     nullrdc_driver
5 #define NETSTACK_CONF_RADIO   dw1000_driver
6 #define NETSTACK_CONF_FRAMER  framer_802154
```

FIGURE 3.4.1 – Configuration de la pile réseau du Zolertia Z1 présente dans le fichier « `platform/z1-dw1000/contiki-conf.h` ».

Pour la compilation de Contiki, il est nécessaire de rajouter les fichiers utilisés pour le driver du DW1000 dans le `Makefile` de la plateforme. Un `Makefile` est un fichier permettant l'exécution de commande, dans notre cas les commandes de compilation et d'upload vers le Zolertia Z1. Le Makefille principal du Zolertia Z1 dans Contiki se situe dans le dossier `/platform/z1-dw1000/` et porte le nom de `Makefile.common`. Dans ce fichier, une variable `ARCH` contient les noms des fichiers utilisés pour compiler Contiki pour la plateforme Zolertia Z1. Cette variable est illustrée en figure 3.4.2, on constate en ligne 3 l'ajout des 3 fichiers nécessaires au radio driver. Ces 3 fichiers sont présents dans le dossier `examples/z1-dw100/dw1000/`.

```
1 ARCH=msp430.c leds.c watchdog.c xmem.c \
2 spi.c cc2420.c cc2420-aes.c cc2420-arch.c cc2420-arch-sfd.c \
3 dw1000-arch-z1.c dw1000.c dw1000-util.c dw1000-driver.c \
4 node-id.c sensors.c button-sensor.c cfs-coffee.c \
5 radio-sensor.c uart0.c uart0-putchar.c uip-ipchksum.c \
6 checkpoint-arch.c slip.c slip_uart0.c \
7 z1-phidgets.c sht11.c sht11-sensor.c light-sensor.c \
8 battery-sensor.c sky-sensors.c tmp102.c temperature-sensor.c
     light-ziglet.c \
9 relay-phidget.c tlc59116.c
```

FIGURE 3.4.2 – Variable « `ARCH` » contenue dans le fichier « `platform/z1-dw1000/Makefile.common` ».

L'utilisation du driver radio dans Contiki pour la plateforme Zolertia Z1 se fait ensuite de façon complètement transparente.

3.4.3 Organisation du driver

Nous avons placé la structure « `radio_driver` » et les définitions des fonctions de celle-ci dans le fichier « `dw1000-driver.c` » décrit en section 3.4.1. La structure « `radio_driver` » du DW1000 est illustrée en figure 3.4.3.

```
const struct radio_driver dw1000_driver =
{
dw1000_driver_init,
dw1000_driver_prepare,
dw1000_driver_transmit,
dw1000_driver_send,
dw1000_driver_read,
dw1000_driver_cca,
dw1000_driver_receiving_packet,
dw1000_driver_pending_packet,
dw1000_driver_on,
dw1000_driver_off,
};
```

FIGURE 3.4.3 – Structure « `radio_driver` » du DW1000 présent dans le fichier « `examples/z1-dw100/dw1000/dw1000-driver.c` ».

Nous allons détailler les fonctions qui ont nécessité une réflexion particulière pour leur implémentation.

3.4.4 Routine d'initialisation

La fonction « `dw1000_driver_init` » permet l'initialisation du transceiver. Celle-ci se déroule en plusieurs étapes :

- **Initialisation du SPI**, on se charge d'abord de configurer le SPI du Zolertia Z1 pour pouvoir communiquer avec le DW1000 ;
- **Réinitialisation du DW1000**, comme on peut utiliser le DW1000 en l'ayant utilisé précédemment et sans avoir coupé son alimentation, on réinitialise complètement celui-ci ;
- **Initialisation du transceiver**, on configure le DW1000 pour la réception et l'envoi de message en suivant les étapes décrites en section 3.3.1 ;
- **Envois d'acquittement**, on active l'envoi automatique d'acquittement. Le DW1000 permet l'envoi automatique d'acquittement au niveau hardware, celui-ci enverra automatiquement un acquittement si une trame reçue contient une demande d'acquittement ;
- « **Double buffering** » que l'on active, celui-ci est décrit en section 3.4.9 ;
- **LED**, optionnellement, on active le clignotement des LEDs présentent sur le DW1000. Celui-ci possède 4 LEDs, dont voici les descriptions :
  - **TX**, la LED TX clignote lorsque d'un message est correctement envoyé ;

— **RX**, cette LED clignote répétitivement lorsque l'antenne est activée ;
— **SFD**, cette LED s'allume lorsque le transceiver détecte un `SDF`. `SFD` signifie « start of frame delimiter » et désigne le fait que le transceiver a détecté un préambule physique suivi du début d'une trame IEEE 802.15.4 ;
— **RX0**, la LED RXOK s'allume quand le transceiver a fini de recevoir une trame et que celle-ci a un bon CRC.

3.4.5 Choix du débit et du canal

Le choix de débit et du canal a été implémenté via une fonction dans le driver. Ils sont paramétrables via des directives de compilation, mais peuvent être redéfinis après l'initialisation du driver. Les débits disponibles sont 6,8 Mb/s, 850 kb/s, 110 kb/s. Le changement de débit implique de changer la longueur du préambule et du SFD des trames physiques. Le format des trames physiques est décrit en section 1.2.5.4. Les paramètres utilisés en fonction du débit sont illustrés dans le tableau 3.4.1. La figure 3.4.4 illustre la fonction `dw1000_driver_config(...)` qui permet le choix du débit et du canal.

Débit nominal (kb/s)	**Longueur du préambule**	**Taille du SFD**
110	1024	64
850	256	8
8600	128	8

TABLE 3.4.1 – Paramètres utilisés en fonction du débit.

```
/**
* Set the channel:
*  Available channel: 1, 2, 3, 4, 5, 7.
*       Channel 4 and 7 have a bandwidth of 999MHz
*       Other channels have bandwidth of 500MHz
*  Pulse Repetition Frequency set to 16MHz
*  Preambule length set to the minimum for data transfert:
*       DW_DATA_RATE_110_KBPS  > 1024
*       DW_DATA_RATE_850_KBPS  > 256
*       DW_DATA_RATE_6800_KBPS > 128
*
*/
void dw1000_driver_config(dw1000_channel_t channel,
    dw1000_data_rate_t data_rate);
```

FIGURE 3.4.4 – Documentation de la fonction de configuration de débit et de canal.

3.4.6 Routine de préparation

La fonction « `dw1000_driver_prepare` » fonctionne en suivant la description donnée en section 3.3. Nous avons ajouté un mécanisme d'analyse du premier octet de la trame. Ceci afin de détecter la présence d'une demande d'acquittement au niveau du `MAC Header` d'une trame IEEE 802.15.4. Le `MAC Header` est décrit en section 1.2.5.1. Une variable sera instanciée en fonction de la présence ou non d'une demande d'acquittement au niveau physique afin d'en tenir compte lors de la transmission. De plus, en cas de demande d'acquittement, le numéro de séquence présent dans le 3e octet du `MAC Header` sera stocké dans une variable afin d'être comparé à celui présent dans l'acquittement reçu après la transmission.

La figure 3.4.5 illustre le diagramme d'activité de la fonction `dw1000_driver_prepare` ».

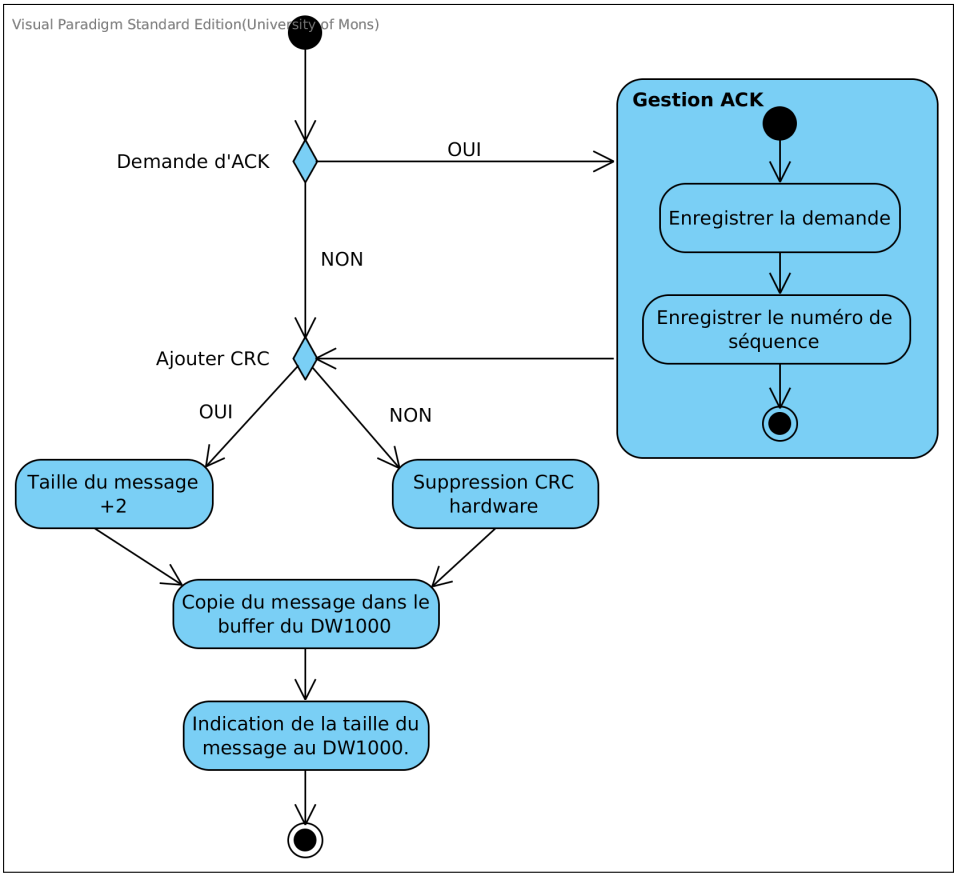


FIGURE 3.4.5 – Diagramme d'activité de la fonction « `dw1000_driver_prepare` ».

3.4.7 Routine d'émission

La fonction « `dw1000_driver_transmit` » est appelée après la fonction de préparation. « `dw1000_driver_transmit` » se charge de l'envoi du message et de l'attente d'un acquittement si cela lui est spécifié. La fonction est illustrée en figure 3.4.6 sous forme d'un diagramme d'activité. De plus, elle est divisée en deux blocs :

- **Envoi** l'envoi se fait via l'activation de l'antenne en émission. Pour cela, il suffit de passer le bit « Transmit Start » (`TXSTRT`) du registre con-
  trôle système (`SYS_CTRL`) à 1. La transmission est ainsi initialisée, la fin de celle-ci est indiquée par le drapeau « Transmit Frame Sent » `TXFRS` du registre statuts système (`SYS_STATUS`) du transceiver. Nous allons donc lire la valeur du drapeau jusqu'à ce celle-ci passe à 1 ou qu'un temps limite soit dépassé. Si le temps limite est dépassé, la fonction renvoie une erreur via « `RADIO_TX_ERROR` » ;
- **Acquittement** si le message contient une demande d'acquittement, le transceiver passe automatiquement en réception après l'envoi du message. Étant donné que la fonction doit retourner si l'acquittement n'a pas été reçu, nous allons attendre la réception de celui-ci. Comme pour l'émission, il faudra lire continuellement un drapeau. Celui-ci est le « Receiver FCS Good » (`RXFCG`) du registre statut système (`SYS_STATUS`). Il passe à 1 après la réception d'un message avec un CRC valide. Si ce bit ne passe pas à 1 dans le temps imparti, la fonction retournera « `RADIO_TX_NOACK` ». Si l'acquittement est reçu, on vérifie la taille du message reçu, elle doit être de 5 octets : 2 octets de header, 1 octet de numéro de séquence et 2 octets de CRC. On vérifié aussi que le numéro de séquence correspond au numéro de séquence du message envoyé que l'on a enregistré dans une variable pendant la préparation du message. Si toutes les conditions sont remplies, la fonction retournera « `RADIO_TX_OK` ».

En cas de demande d'acquittement, les interruptions sont désactivées momentanément au début de la fonction « `transmit` » et rétablies à la fin de celle-ci, la réception de l'acquittement se faisant de façon synchrone. On changera aussi le pointeur du tampon de réception dû au double buffering.

En cas de réception asynchrone, la fonction terminerait avant d'avoir reçu ou non l'acquittement. Cela trouve son origine dans le fait que celle-ci ne se trouve pas dans un protothread (à sa déclaration) et que l'on ne peut donc pas attendre un événement. Le driver du CC2420 quant à lui, n'implémente pas le support des acquittements physique. La fonction « `dw1000_driver_transmit` » peut retourner « `RADIO_TX_COLLISION` » d'après la documentation. Le DW1000 n'intègre pas de mécanisme de mesure de la puissance du signal reçu comme le font les transceivers radio à faible largeur de bande. Dans le cas de ce driver, nous ne pouvons donc pas détecter s'il y a eu une collision, cette valeur de retour n'est donc pas utilisée.

Les temps alloués à la transmission et à l'acquittement sont définis dans la partie « Durée d'attente maximale » de la section « Test et validation » (section 4.7, page 93).

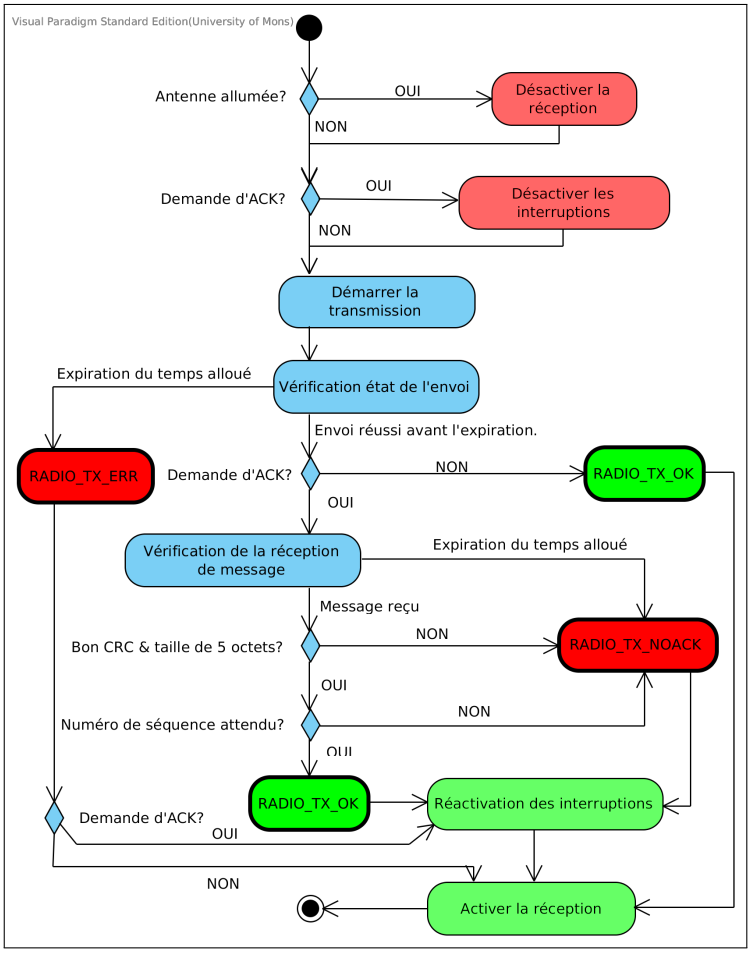


FIGURE 3.4.6 – Diagramme d'activité de la fonction « `dw1000_driver_transmit` ».

3.4.8 Routine de lecture

La fonction « `dw1000_driver_read` » se charge de la lecture du paquet reçu. Cette fonction vérifiera la taille du paquet et sera appelée par la routine d'interruption. Si le paquet reçu est trop court, moins de deux octets soit la taille du CRC celui-ci est jeté et la fonction incrémente un compteur de message trop court via la commande « `RIMESTATS_ADD(tooshort);` ». Si le paquet est trop long, la commande « `RIMESTATS_ADD(toolong);` » est exécutée. En cas d'erreur, un paquet trop long ou trop court, la fonction retourne une taille de 0. Sinon, elle copie la trame dans le buffer qui lui est passé en argument en excluant le CRC. Elle retourne la taille du message copié dans le buffer, soit la taille de la trame moins les deux octets de CRC.

3.4.9 Routine d'interruption

La routine d'interruption est déclenchée par le DW1000 à la réception d'un message et se divise en deux parties.

— Une fonction « **ISR(port, nom_interruption){...}** » qui gère l'interruption générée par le DW1000. Cette fonction vérifie l'état de registre du DW1000 afin d'identifier les statuts de la réception. Ses statuts peuvent indiquer que la réception s'est mal déroulée et qu'aucun message n'est donc à lire. Si les statuts indiquent que le message reçu est correctement formé, à savoir qu'il n'y a pas d'erreur détectée dans le registre de statuts, une commande est effectuée afin d'indiquer qu'un processus doit gérer le reste de l'interruption ;

— On utilise un **protothread** pour réduire la durée de l'interruption et pour rendre la main au processus qui était en cours d'exécution. Une fois que celui-ci a fini son travail, le processus de l'interruption peut avoir la main. Ce protothread est le protothread « `dw1000_driver_process` », il est chargé de lire le message sur le transceiver et d'en fournir le contenu aux couches supérieures de la pile réseau.

Sur les premières versions du driver, à la réception d'un message, l'antenne s'éteignait automatiquement à la réception d'un message. À la fin de la gestion de l'interruption (c'est à dire du processus « `dw1000_driver_process` ») l'antenne était donc réactivée après avoir effacé les statuts d'interruption sur le transceiver.

Afin que l'antenne reste active pendant la gestion de l'interruption, un mécanisme nommé « double buffering » a été implémenté. Le « **double buffering** » consiste en l'utilisation d'un registre tampon pour la réception des messages. Concrètement, le transceiver comprend deux tampons de réception. Ces deux tampons permettent d'en utiliser un pour lire le dernier message reçu et d'avoir le second pour recevoir un message dans le même temps. Cette fonctionnalité est fort utile dans le cas d'une utilisation à haut débit du transceiver. Cella permet donc de recevoir un message pendant que l'on traite le précédent et donc éviter de perdre des paquets en ayant l'antenne éteinte le temps de traiter le message reçu. Dans le cas de l'échange de trame IPv6, environ 10 messages IEEE 802.15.4 doivent être envoyés d'affiliés, cette option permet donc leurs envois successifs sans perte.

L'utilisation du « double buffering » peut entraîner des erreurs de tampons appelées « Overrun ». Celles-ci se produisent lorsque le transceiver reçoit un message alors que les deux tampons sont déjà remplis. Pour éviter ce type d'erreur, il faut lire assez rapidement les tampons après la réception de l'interruption et indiquer au transceiver que le message a été lu. Néanmoins, ce type d'erreur peut tout de même arriver, quand elle se produit, les deux messages présents sur le transceiver peuvent être corrompus. Dans ce cas, il faut jeter tous les messages présents dans les buffers du transceiver. Cela se fait en éteignant l'antenne, en effectuant une réinitialisation logicielle de celle-ci (via la modification de registre spécifique). Et enfin en ré allumant l'antenne, car celle-ci reste toujours active (on est en mode « nullRDC »). Cette erreur peut être détecté via un drapeau « `Receiver Overrun` » (`RXOVRR`) que l'on peut lire dans le registre statut système. Le même que l'on parcourt afin de connaître l'état de la réception d'un message. Il faut donc veiller à vérifier lorsque que l'on lit un message que cette erreur n'a pas eu lieu et ceux à plusieurs moments. Cette procédure est décrite via deux diagrammes d'activités. La figure 3.4.7 décrit la procédure à adopter en cas d'interruption. Le choix de gérer l'« Overrun » dans la fonction d'interruption vient du fait que celui-ci provoque la perte de message. Plus vite cette erreur est traitée, plus vite le transceiver est de nouveau opérationnel.

En cas d'interruption indiquant la réception d'un message et en absence d'« overrun », on ajoute le protothread « `dw1000_driver_process` » à la file d'exécution pour qu'il soit exécuté dès que l'ordonnanceur le décidera. Ce processus est décrit en figure 3.4.8. La première chose qu'on y fait est de vérifier la présence ou non du drapeau d'overrun afin de ne pas lire un message inutilement. Ce processus est lancé uniquement si le drapeau de réception de message est levé. Celui-ci n'indique pas si le message à un bon ou un mauvais CRC, celui-ci est donc vérifié. Il est possible de placer comme interruption le fait que le message doit avoir un bon CRC. Dans le cas du double buffering cette approche n'est pas envisage du fait que le message même avec un mauvais CRC est placé dans le buffer. On doit alors absolument changer le pointeur de tampon afin d'éviter un « overrun ». Si le message a un bon CRC, il est lu. Une fois la lecture finie, on vérifie qu'il n'est pas corrompu, c'est-à-dire qu'il n'y a pas eu d' « overrun ». Le cas échéant, on finalise la réception en l'indiquant à la couche réseau supérieure. Enfin, on réinitialise les interruptions et on change le pointer de tampon pour préparer la réception suivante.

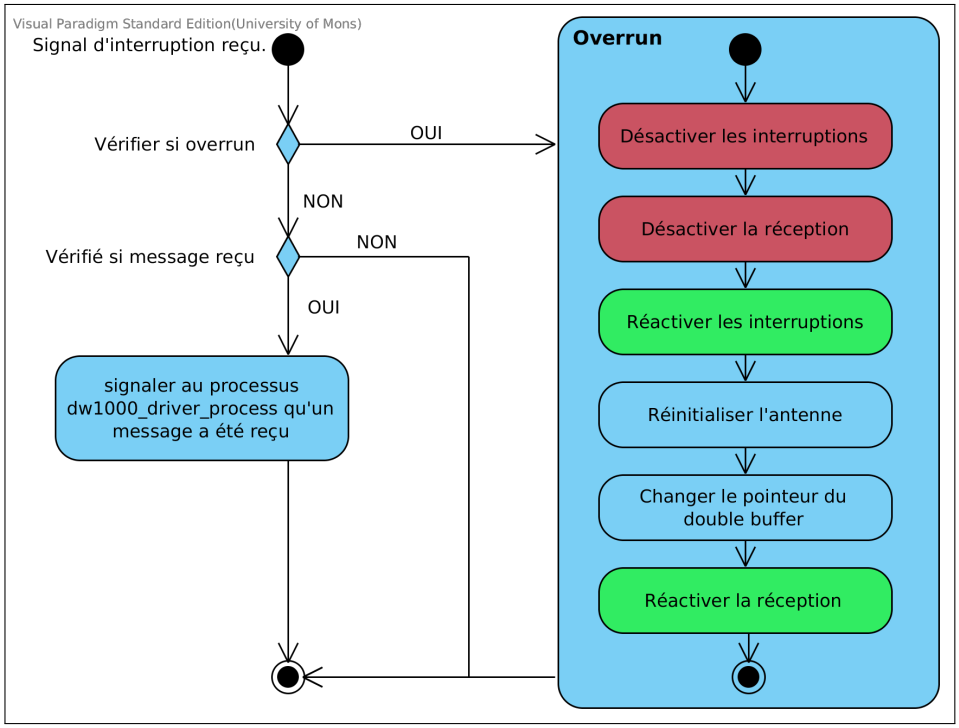


FIGURE 3.4.7 – Diagramme d'activité de la fonction lancé en cas d'interruption.

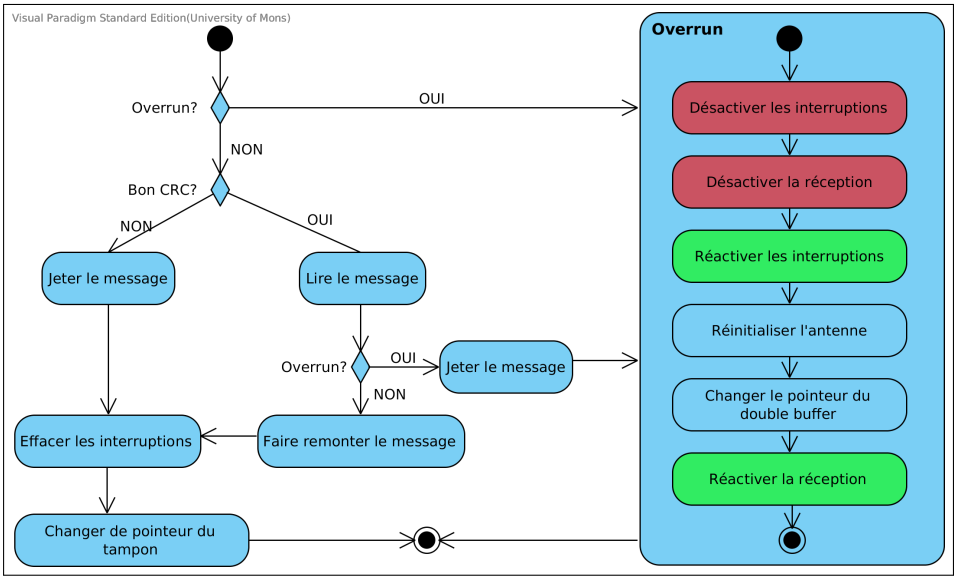


FIGURE 3.4.8 – Diagramme d'activité du processus `dw1000_ driver_process` lancé par la fonction d'interruption.

3.4.10 Clear Channel Assessment

L'implémentation de la fonction de « Clear Channel Assessment » demande une étude plus approfondie afin de déterminer plusieurs paramètres comme la durée d'écoute, ainsi que les intervalles à utiliser entre chaque appel d'un « Clear Channel Assessment ». C'est-à-dire que cela nécessite de modifier les protocoles de duty cycle.

La fonction « `dw1000_driver_cca` » n'est donc pas implémentée. Elle retourne donc constamment que le canal est libre. Néanmoins, on peut retrouver une ébauche d'une implémentation en commentaire dans le driver. La figure 3.4.9 illustre le fonctionnement de la fonction si celle-ci avait été implémentée. On peut voir qu'elle consiste en une réception synchrone avec une vérification continuel de la réception ou non d'un préambule. La durée autorisée avant l'expiration nécessite une étude approfondie des protocoles.

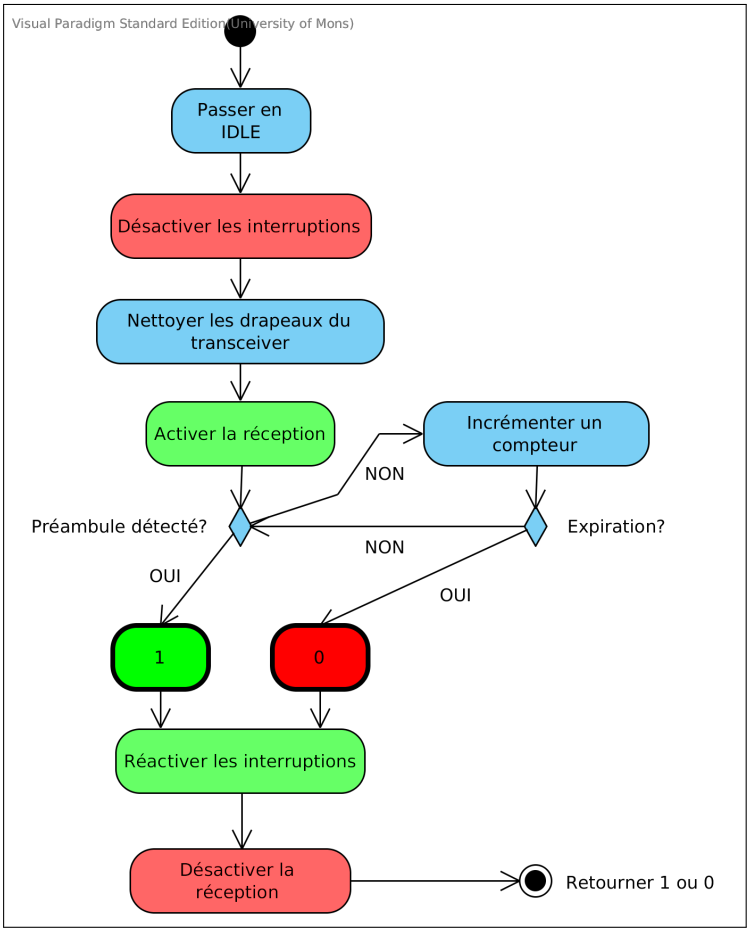


FIGURE 3.4.9 – Fonctionnement théorique de la fonction de « Clear Channel Assessment » en UWB.

3.4.11 Paramètres

Le driver possède plusieurs directives de compilation (macro C) permettant de modifier son comportement. Nous allons les lister et les détailler.

- **DW1000_CONF_CHECKSUM** peut valoir **1** (vrai) valeur par défaut ou **0** (faux) :
  - Si cette constante est définie à 1 le driver calcule et ajoute le `CRC` à la trame qu'on lui fournit ; En réalité, on spécifie simplement une taille de trame avec deux octets de plus, soit les 2 octets pris par le CRC. Le DW1000 calcule ensuite automatiquement le CRC lorsque l'envoi de la trame est demandé ;

  - Si la constante est à 0, on spécifie au transceiver de ne pas calculer le CRC et l'on envoie le message sans rajouter les 2 octets de CRC. On considère qu'ils ont été calculés avant l'appel de la fonction.
- **DW1000_CONF_AUTOACK** peut valoir **1** (vrai) valeur par défaut ou **0** (faux). Cette constante définit l'activation ou non des acquittements automatiques. Si cette constante est à vrai, le DW1000 enverra un acquittement à la réception d'un message automatiquement, comme décrit en section 3.4.7. À l'envoi du message contenant une attente d'acquittement, on attendra la réception de l'acquittement et on retournera une erreur correspondante si le transceiver ne le reçoit pas ;
- **DW1000_DATA_RATE** permet de définir le débit de transmission nominal du transceiver. Le transceiver supporte 3 débits nominaux :
  - **DW_DATA_RATE_6800_KBPS** qui définit un débit de 6,8 Mb/s qui est le débit maximal du transceiver. Cette valeur est la valeur par défaut ;
  - **DW_DATA_RATE_850_KBPS** qui définit un débit de 850 kb/s ;
  - **DW_DATA_RATE_110_KBPS** qui définit un débit de 110 kb/s.
- **DW1000_IEEE802154_EXTENDED** qui permet de définir l' utilisation (1) ou non (0) du format étendu de trame. Cette valeur se spécifie au niveau du fichier `/platform/z1/platform-conf.h`. Comme décrit en section 3.5.3, l'utilisation de ce format de trame ne peut se faire qu'entre DW1000 ;
- **RF_CHANNEL** est une constante inclue par défaut dans Contiki qui permet la sélection du canal. Les valeurs de canaux disponibles sont : 1, 2, 3, 4, 5 et 7. Où les canaux 1, 2, 3 et 5 ont une largeur de bande de 500 MHz et où les canaux 4 et 7 ont une largeur de 999 MHz. Le canal par défaut est le canal 7 ;
- **DEBUG** qui permet d'afficher les appels aux fonctions sur la ligne série. Défini à faux (0) par défaut ;
- **DEBUG_VERBOSE** qui permet d'afficher les appels aux fonctions et plus de détails dans celle-ci sur la ligne série. Défini à faux (0) par défaut ;
- **DEBUG_LED** qui permet d'activer ou non les LED présentent sur le DWM1000. Définit à vrai (1) par défaut.

En cas d'utilisation à 110 kb/s, les acquittements physiques ne sont pas reçus par le transceiver. Nous désactivons dès lors l'option d'acquittement automatique si l'utilisateur choisit ce débit.

3.5 Divers

Nous allons décrire la démarche à suivre pour utiliser le driver ainsi que donner un examen d'utilisation. Nous allons aussi discuter de l'usage de trame au format étendu optionnellement utilisée dans le driver.

3.5.1 Utilisation du driver

Pour compiler un programme utilisant le driver, il suffit d'utiliser la plateforme « `z1-dw1000` » comme expliqué en section 3.4. La compilation nécessite l'installation de certains outils [46] [4] :

- `msp430-gcc-4.7.x` qui est un compilateur C pour le processeur MSP-430 qui équipe le Zolertia Z1. Les versions moins récentes peuvent entrainer des problèmes avec la communication séries ;
- `GNU Binutils` qui est une collection de divers outils [10] ;
- `Python-serial support (BSL)` pour la communication série.

Les commandes d'installation se trouvent aux adresses suivantes :

- Site de l'ANRG : http://anrg.usc.edu/contiki/ ;
- Wiki de Zolertia : http://zolertia.sourceforge.net/wiki/.

Ensuite, pour le faire fonctionner, il faut alimenter les transceivers. Pour ce faire, il faut exécuter le programme présent dans le dossier `/examples/z1-dw1000/power/` avec les commandes suivantes :

- « `make power` » pour le compiler le programme ce qui donne un exécutable `power` ;
- « `./power /dev/ttyUSB1` » pour demander au Bus Pirate branché sur le port « `/dev/ttyUSB1` » d'alimenter en 3,3 volts sa sortie d'alimentation.

3.5.2 Exemple d'utilisation

Un exemple d'utilisation peut de se faire grâce aux fichiers exemples présents dans Contiki. Prenons l'exemple d'envois de message en Broadcast depuis Rime. Le fichier `/examples/rime/example-broadcast.c` permet cela. Il suffit de se placer dans le dossier et après avoir allumé l'alimentation sur les transceivers comme décrit en section 3.5.1, de compiler et d'uploader le fichier sur les Zolertias Z1. Cela via la commande donnée en ligne 1 de la figure 3.5.1 où « `ttyUSBX` » est le port USB connecté au Zolertia. Ensuite, de lire les retours sur la ligne série via la commande donnée en ligne 2 de la figure 3.5.1. Vous devriez avoir des messages de la forme « broadcast message received from 06.00 : Hello » sur la ligne série où « 00.06 » est le numéro du nœud.

```
make example-broadcast.upload TARGET=z1-dw1000 MOTES=/dev/
    ttyUSBX
make z1-reset MOTES=/dev/ttyUSBX && make login MOTES=/dev/
    ttyUSBX
```

FIGURE 3.5.1 – Compilation et exécution de l'exemple « `example-broadcast.c` ».

3.5.3 Format étendu

Le Decawave DW1000 permet l'utilisation de trame IEEE 802.15.4 de taille maximale de 1023 octets contre 127 octets pour le format défini par le standard. Ce type de trame n'est pas standard et n'est reconnu que par les transceivers DW1000. Pour ce faire le transceiver modifie le Header physique (PHR) de la trame physique décrit en section 1.2.5.4, qui contient la taille en octets de la trame MAC transportée par le message. Le standard IEEE 802.15.4-2011 définit le format du PHR. Il lui définit une taille de 21 bits, dont 3 bits dédiés aux évolutions futures du standard. Le DW1000 utilise ces 3 bits pour définir une plus grande taille de trame MAC. Dans le standard la taille est définie sur 7 bits soit une taille maximale de $(2^7 - 1)$ soit 127 octets. Avec les 3 bits supplémentaires on obtient une taille maximale de $(2^{10} - 1)$ soit 1023 octets.

Dans le cadre du driver, nous avons ajouté le support des trames non standard. Pour ce faire, il a fallu modifier une constante au niveau du tampon de paquet (`packetbuff`) de Contiki. De base celui-ci ne peut contenir que des messages de plus de 127 octets, cette taille peut être modifiée via la constante « `PACKETBUF_CONF_SIZE` ». Nous avons constaté que l'on devait se limiter à une valeur d'environ 500 pour cette constante. Au-delà une erreur du type : `msp430-elf/bin/ld: region 'RAM' overflowed by 2000 bytes` apparait. Cette erreur signale que l'on « consomme » trop d'espace mémoire. Nous avons ensuite remarqué un dysfonctionnement au niveau de `Rime`, les messages étant correctement envoyé et reçu, mais ignorés une fois renvoyés à la couche supérieure du driver. En diminuant la longueur des trames, nous avons constaté que leur taille maximale pour être supporté par les couches supérieures est de 265 octets.

Notre driver supporte donc des trames de format non standard jusqu'à une taille de 265 octets.

3.6 Discussion et perspectives futures

Le driver actuellement conçu permet son utilisation dans façon quasi transparente. Sa structure pourrait être améliorée pour ressembler à celle du driver du CC2420 (fichier dans le dossier `/core/dev/` et `/platform/`) mais cela enlèverait le support du driver pour le Bus Pirate.

L'ajout du « Clear Channel Assessment » permettrait une utilisation réelle pour des nœuds fortement contraints en diminuant fortement la consommation grâce à l'utilisation de protocole de Duty Cycle. Il faudrait pour cela, ajouter une alimentation physique qui ne nécessite pas d'ordinateur pour l'activer (ce qui est le cas avec le Bus Pirate) afin que les nœuds soient totalement autonomes.

Un autre ajout pourrait être la modification de la pile réseau afin de prendre en compte les « Rangings Frames » et de permettre de mesurer la distance entre deux transceivers de façon transparente. Actuellement, le driver n'est pas conçu pour supporter les « Ranging Frame », mais cela pourrait être fait « facilement », car cela ne nécessite pas de grandes modifications. Il suffirait d'ajouter une méthode d'envoi de « Ranging Frame » dans le driver (les primitives sont déjà présentes dans le driver de l'Ulea) et de modifier les paramètres de configurations pour utiliser des préambules plus long afin d'avoir une meilleure précision.

Actuellement, la fonction d'envoi « `dw1000_driver_transmit` » possède deux constantes, `DW1000_TX_TIMEOUT` et `DW1000_ACK_TIMEOUT` définissant respectivement les durées maximales d'attente de fin de transmission et de réception d'acquittement et qui sont détaillées en section 4.7. Celles-ci pourraient être remplacées par une fonction se basant sur la durée de transmission théorique afin de ne pas devoir stocker plusieurs versions des constantes en fonction du débit nominal. Cela permettrait aussi de pouvoir modifier le débit choisi de façon dynamique lors de l'utilisation du driver qui actuellement est fixé à la compilation.

On pourrait, enfin, effectuer une étude de consommation du Zolertia Z1 en établissant un modèle pour comparer la consommation UWB (avec le DWM1000) et celle du transceiver CC2420 à bande étroite présente sur le Zolertia Z1. Cette étude pourrait aussi se focaliser sur la consommation des canaux UWB pour voir si elle correspond avec la théorie qui indique que la consommation d'un canal deux fois moins large doit être deux fois plus faible [29] ce qui n'est pas indiqué dans la fiche technique du DW1000.

Chapitre 4

Test et validation

Le driver précédemment réalisé est testé via les couches de la pile réseau supérieure. Dans notre cas, nous allons utiliser la couche de protocole `Rime` faisant appel à la couche MAC.

Le test de validation se déroule en procédant à l'envoi de messages entre deux transceivers. Un transceiver étant utilisé comme émetteur et un transceiver utilisé comme récepteur. Depuis un protothread, après avoir configuré `Rime` pour envoyer les paquets vers le transceiver récepteur, nous allons procéder à des envois successifs de paquets via une primitive de `Rime`. Pour chaque envoi, nous allons effectuer plusieurs mesures de temps :

- Le **temps de préparation**, le temps passé dans la fonction « `dw1000_driver_prepare(...)` » décrite en section 3.4.6 ;
- Le **temps de transmission**, le temps écoulé entre le début de la transmission et l'indication par le transceiver que la trame à bien était envoyée via le drapeau « Transmit Frame Sent » (`TXFRS`) du registre « System Event Status » ;
- Le **temps d'attente de l'acquittement**, une fois le message envoyé et en cas de demande d'acquittement, nous allons mesurer le temps nécessaire à sa réception et son traitement ;
- Le **temps total de l'envoi**, le temps écoulé entre le début de l'envoi et la fin de celui-ci, incluant les modifications de registre nécessaire : extinction de l'antenne, en cas d'acquittement : désactivation des interruptions, rétablissement de celles-ci.

Nous avons effectué les mesures en fonction de la taille du message. Le message peut avoir une taille de 15 à 127 octets en format standard ou de 17 à 265 octets en format étendu. La taille de 15 octets étant imposée par l'overhead dû au MAC header (9 octets), au CRC (2 octets), et aussi à 4 octets utilisés par Rime. Chaque message de taille x est envoyé 1000 fois et les mesures considérées sont la moyenne des 1000 itérations.

Nous utilisons la configuration du transceiver décrite dans le tableau 4.0.1.

Paramètre	**Configuration**
Débit	6800 kb/s
Canal	n°5 (499 MHz) pour les trames de format standard.
Canal	n°7 (999 MHz) pour les trames de format non standard
Préambule	128 symboles

TABLE 4.0.1 – Paramètre choisi pour les tests en 6,8 Mb/s.

Les mesures sont effectuées grâce à la macro `RTIMER_NOW()` qui permet de récupérer le temps actuel du processeur. Ce temps est exprimé en signal d'horloge. La constante `RTIMER_SECOND` représente une seconde dans cette unité soit 32784 incrémentations. La précision en microseconde sera donc donnée par $\frac{1000000}{32784} \approx 30.5$ μs. L'emplacement des enregistrements de temps et leurs affichages sont illustrés en annexe 7.5. Les `printf(...)` ont volontairement été placés en fin de fonction afin de ne pas fausser les mesures.

Les sections suivantes représentent les résultats obtenus pour les différentes métriques considérées.

4.1 Préparation

Le temps de préparation est le temps passé dans la fonction « `dw1000_driver_prepare(...)` » décrite en section 3.4.6. La figure 4.1.1 illustre le temps passé pour la préparation du message. On constate que le temps de préparation pour le format standard et non standard est identique, ce qui était attendu.

Le temps de préparation à une vitesse qui varie entre 787 kb/s pour les paquets de 18 octets ($\frac{18*8}{183} \approx 0,787$) et 1,92 Mb/s pour les paquets de 265 octets ($\frac{265*8}{1104} \approx 1,92$). La vitesse relative (qui ne prend pas en compte le 18 premier octets) est de $\frac{(265-18)*8}{1104-183}$ soit environ $2,14$ Mb/s. Celle-ci est plus grande en raison de la perte de l'overhead induit par des commandes effectuées avant le début de la copie du message dans le buffer :

— L'analyse de la présence d'une **demande d'acquittement** qui se fait « instantanément » (dans notre échelle de mesure) étant donné que cela se fait entièrement sur le Zolertia Z1 ;

— La copie de la taille de la trame dans un registre spécifique du transceiver.

Le remplissage illustre les écarts entre les valeurs de temps maximal et minimal observé. La variation est d'environs 60 μs ce qui représente environ 30 % du temps pour les messages de 17 octets et environ 6 % du temps pour les messages de 265 octets.

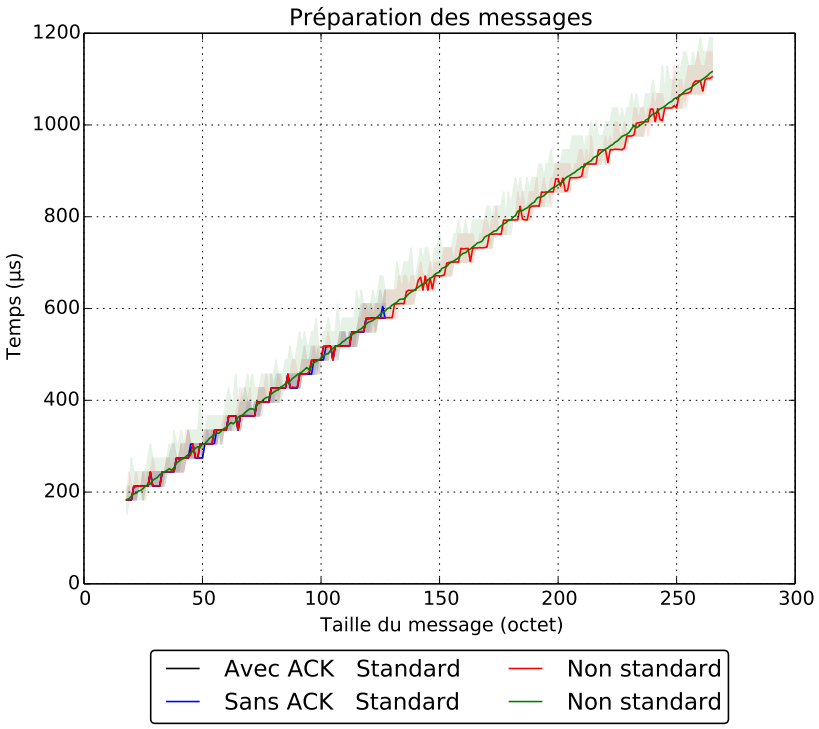


FIGURE 4.1.1 – Temps de préparation d'un message avant son envoi, moyenne sur 1000 envois.

Le début d'une communication SPI comprend des « ordres » comme on l'a vu dans la section 2.1.1. La fonction permettant d'écrire des données dans un registre du DW1000 est illustrée en annexe en section 7.6. Dans le cas de la copie sur le buffer, un overhead de 2 octets est constaté, un octet désignant le type d'ordre (écriture) et un autre l'adresse du registre. Les octets suivants sont les octets contenant les données à copier. L'overhead induit par la communication SPI est donc très faible voir quasi nul. Néanmoins, la vitesse mesurée est presque inférieure de moitié aux valeurs attendues. On a mesuré une vitesse de $2,14$ Mb/s pour la copie alors que le Zolertia Z1 est configuré pour avoir une vitesse de 4 Mb/s (4 Mhz) et que le DW1000 supporte une vitesse de 20 Mb/s [18]. Cette différence peut s'expliquer par la façon dont l'on interagit en SPI. La figure 4.1.2 illustre la communication en SPI pour écrire dans un registre. On remarque deux boucles, la première permet l'écriture des deux octets d'instruction (ligne 5 à 7), la seconde permet la copie de données dans le registre du transceiver (ligne 9 à 11). Le rythme d'horloge du Zolertia Z1 est de 4 MHz, soit un débit de 4 Mb/s, mais on ne copie pas les données en un bloc, on le divise par octets. Cette division par octet peut entraîner des pertes de synchronisation avec l'horloge du SPI et donc induire une perte de vitesse. Cela peut expliquer la forte perte de vitesse observée.

```
1  // SPI communications
2  // Asserting CS
3  DW1000_SPI_ENABLE();
4  //write instruction
5  for (i = 0; i < reg_inst; i++){
6    SPI_WRITE( (char) instruction[i]);
7  }
8  //write data
9  for (i = reg_inst; i < n_len; i++){
10   SPI_WRITE( (char) *(p_data++));
11 }
12 // De-asserting CS
13 DW1000_SPI_DISABLE();
```

FIGURE 4.1.2 – Illustration du code permettant l'écriture de données dans un registre en SPI issue du code présent en figure 7.6.1.

4.2 Transmission & acquittement

Cette section va détailler le temps de transmission et d'attente des acquittements.

4.2.1 Modèle théorique du temps de transmission

Nous allons définir la vitesse théorique que nous devrions obtenir en nous basant sur les informations envoyées dans une trame physique décrite en section 1.2.5.4. Pour rappel, une trame physique UWB est découpée en 4 sections illustrées en figure 4.2.1. Ces 4 sections ne sont pas émises au même débit, leur vitesse d'émission dépendant des paramètres fixés à l'initialisation du transceiver. Voici la description des vitesses de transmissions de chaque section.

- La séquence de préambule (SHR) est composée de deux sections :
  - Une **séquence de préambule** de taille variable (de 64 à 4096 symboles en puissance de 2) dont l'envoi de chacun des symboles durs environ 1 μs ;
  - Un **délimiteur de début de trame** (SFD) qui peut être de deux types en fonction du `PRF` « Pulse Repetition Frequency » [19][p202].
    - ⋆ PRF à 16 MHz, SFD de 8 symboles émis en $\frac{496}{499,2}$ μs soit 8 μs ;
    - ⋆ PRF à 64 MHz, SFD de 64 symboles émis en $\frac{508}{499,2}$ μs soit 64 μs.
- Un **header physique** (PHR) de 21 bits dont la durée des symboles varie en fonction de la vitesse d'émission et du PRF. En annexe, à la table 7.4.1 se trouve un tableau donnant la correspondance en temps pour les différents réglages ;
- La **trame MAC** que l'on fournit au transceiver. Cette trame est modifiée par un algorithme « reed solomon » [18]. Un algorithme « reed solomon » est un algorithme d'encodage de message qui retourne un code correcteur d'erreurs. On lui fournit un message et celui-ci le code de façon à ce que le contenu soit consolidé. Dans le cas de l'IEEE 802.15.4-2011, l'algorithme prend comme entrée des blocs de 330 bits, ou moins, auquel il rajoute I bits afin d'avoir un bloc de 330 bits si besoin et retourne 330-I bits modifiés ainsi qu'une table de codage de 48 bits. Les bits donnés en sortie (330-I + 48 bits) sont utilisés pour l'envoi [8]. Le débit réel d'envoi est de $\frac{1}{0,87}$ fois [19] la vitesse nominale donnée par le fabricant avec $\frac{300}{330+48} \approx 0,87$. Celui-ci tenant compte de l'overhead du « reed solomon » grâce au facteur $\frac{1}{0,87}$. Une vitesse d'envoi configuré à 6.8 Mb/s est en réalité de 7,81 Mb/s.

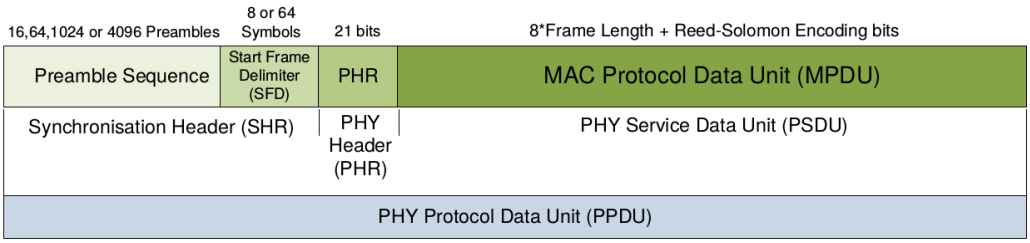


FIGURE 4.2.1 – Structure physique d'une trame IEEE802.15.4-2011 [18]

Le tableau 4.2.1 donne le temps passé pour émettre chaque section d'une trame physique. Son contenu se base sur le tableau 7.4.1 donné en annexe et les paramètres choisis à l'initialisation du transceiver. La vitesse de transmission est quant à elle donnée par la formule décrite en figure 4.2.2.

$$
\begin{aligned}
f_{trans}(x) &= t_{preamble} + t_{SFD} + t_{PHR} + t_{MAC}(x), \\
&= \left(128 + 8\right) * \frac{993,59}{1000} + 21 * 1,025 + 0,128 \left(8x + 48 \left\lceil \frac{8x}{330} \right\rceil\right), \\
&\approx 135,13 + 21,5 + 1,024x + 6,14 \left\lceil \frac{8x}{330} \right\rceil, \\
&\approx 156,63 + 1,024x + 6,14 \left\lceil \frac{8x}{330} \right\rceil.
\end{aligned}
$$

FIGURE 4.2.2 – Temps d'émission en fonction du débit dans le cas du 6,8 Mb/s.

La fonction $f_{trans}(x)$ sera utilisée dans les graphiques afin de comparer la valeur mesurée avec la valeur théorique. Son implémentation est illustrée en annexe en figure 7.4.1 (page 112). Il est important de noté que les messages envoyés successivement pour les mesures le sont en ayant un certain délai entre chaque envoi, induit par les envois des mesures sur la ligne série, la préparation du message par les couches supérieures, et la copie sur le buffer d'envoi. Ce délai est supérieur au temps « Inter-Frame Spacing » définit dans le registre « Transmit Frame Control » qui est de 6 μs (6 symboles) [19], la copie sur le buffer d'envoi prenant déjà au moins 200 μs à elle seule.

Section	**Paramétrage**	**Temps d'émission** (μs)
Séquence de préambule	128 symboles	$t_{preamble} = 128 * \frac{993,59}{1000} \approx 127,18$
Délimiteur de début de trame (SFD)	8 symboles	$t_{SFD} = 8 * \frac{993,59}{1000} \approx 7,94$
Header physique (PHR)	21 bits	$t_{PHR} = 1,025 * 21 \approx 21,5$
Trame MAC de **x** octets après passage par le « reed solomon »	6,8 Mb/s	$t_{MAC} = 0,128 * (8 * x + \lceil \frac{8*x}{330} \rceil * 48)$

TABLE 4.2.1 – Temps d'émission pour chaque section d'une trame physique UWB.

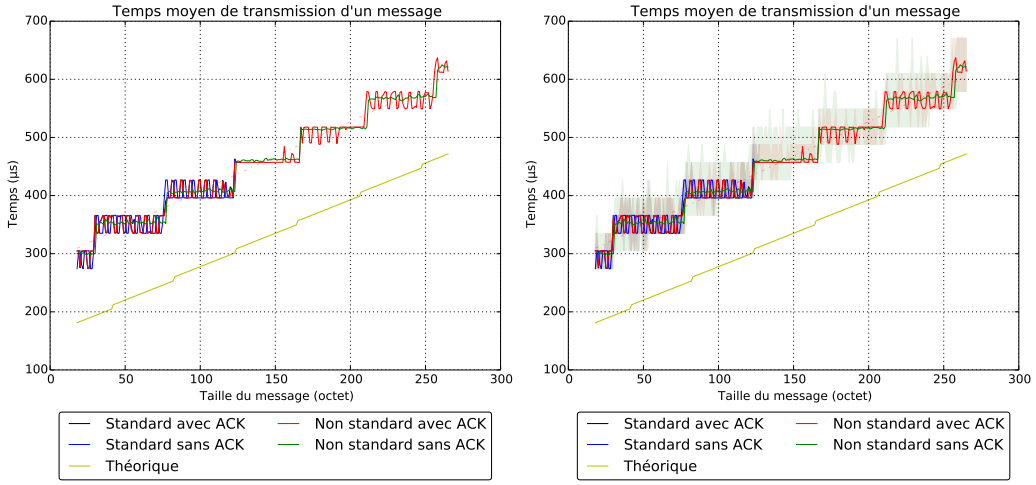
4.3 Mesure du temps de transmission

La figure 4.3.1 illustre le temps passé par le transceiver en état de transmission (sans attendre d'acquittement). On remarque que le type de trame, standard (max 127 octets), ou non-standard (max 1023 octets) n'influence pas la vitesse d'envois, car les deux courbes se superposent. On remarque aussi que l'ajout d'une demande d'acquittement n'influence pas le temps passé en émission. Cela confirme la théorie, car une demande d'acquittement n'est rien d'autre qu'un bit à 1 plutôt qu'à 0. Et une trame non standard n'a pas de bit en plus à envoyer pour une trame MAC de même taille.

Nous allons considérer deux vitesses d'envoi, celle avec l'overhead de départ (vitesse absolue) et celle sans overhead, qui sera donc plus grande (vitesse relative). La vitesse est donnée par la formule suivante : $\frac{nombre\ de\ bit}{temps\ en\ \mu s}$. Soit $\frac{(265-16)*8}{614-304}$ = **6,42** Mb/s pour la vitesse relative et $\frac{265*8}{614}$ = **6,97** Mb/s pour la vitesse absolue. Ses vitesses sont proches de la vitesse nominale de 6,8 Mb/s avec laquelle on a configuré le transceiver. On remarque même que la vitesse absolue dépasse ce seuil, ce qui est acceptable théoriquement du au « facteur » $\frac{1}{0,87}$ appliqué sur la vitesse nominale pour tenir compte de l'overhead du « reed-solomon ».

La figure 4.3.1b, où les axes démarrent à zéro permet de voir le temps perdu dû à l'overhead. Le remplissage met en avant le temps maximum et minimum passé pour l'envoi de chacun des 1000 messages par taille de message. On constate que cette fenêtre est assez étroite et que les valeurs ne dépassent pas les 700 μs.

L'évolution du temps passé en envoi n'est pas linéaire en fonction de la taille des paquets. Elle subit des sauts, ceux-ci coïncident avec les sauts dus au « reed solomon », mais ceux-ci sont plus grands que dans la théorie. On remarque que l'on a un écart d'environ 150 μs entre la valeur théorique du temps de transmission et la valeur mesurée. Il ne devrait pas avoir de délai, car la transmission est censée commencer instantanément après que l'on ait spécifié le drapeau de début de transmission du fait que l'on est en mode IDLE avant de commencer à émettre [18].



(a) Moyenne uniquement

(b) Incluant les durées minimum et maximum observées

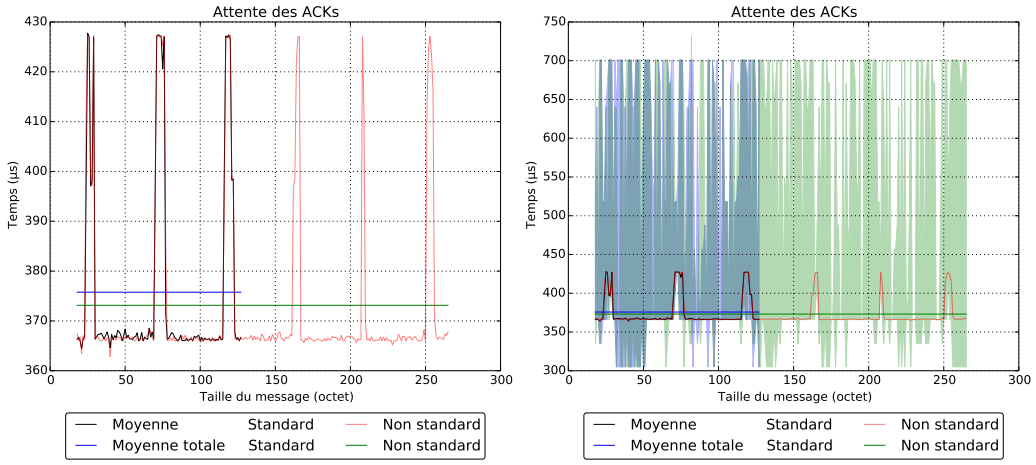
FIGURE 4.3.1 – Temps moyen de transmission d'un message. (L'attente de l'ACK n'est pas considérée).

4.3.1 Temps d'attente des acquittements

Théoriquement, l'attente de l'acquittement pour un message ne dépend pas de la taille de celui-ci, étant donné que celle-ci commence une fois l'envoi terminé. Sur la figure 4.3.2a, on illustre la durée moyenne d'attente des acquittements. On remarque étrangement que la taille des paquets influence la durée d'attente moyenne des acquittements. Les pics observés coïncident avec « les marches d'escalier » présent dans la durée de transmission. Effectuer un calcul de « reed solomon » induit peut-être un délai supplémentaire pour le récepteur, si celui-ci avait pris en entrée un bloc de 330 bits où presque tous les bits étaient à 0. L'attente des acquittements est en moyenne d'environ 365 μs avec des écarts dans la moyenne de 50 μs le temps d'attente théorique est de 182 μs (169 μs pour la transmission de l'acquittement et 13 μs d'overhead comme expliqué en section 4.7.2).

La moyenne d'attente des acquittements sur les 110 000 envois standard est d'environs 375 μs. Quant aux 248 000 envois avec acquittement du format non standard la moyenne est d'environ 373 μs soit une valeur presque identique.

La figure 4.3.2b illustre quant à elle l'attente des acquittements en prenant compte des attentes minimum et maximum observé. On remarque que les temps d'attente varient entre 300 et 700 μs. Ces valeurs sont dans le même intervalle que les valeurs mesurées pour le temps d'envoi. Les pics à 700 μs indiquent que l'acquittement n'a pas été reçu et que l'on a déclenché une erreur de type « `RADIO_TX_NOACK` ». On remarque que l'on a le déclenchement d'une expiration à l'attente de l'acquittement presque pour chaque taille d'envoi, soit pour 1000 envois. La figure 4.3.3 met justement en avant la proportion d'acquittements non reçus. Cette proportion est d'en moyenne d'environ 0,1 % soit un message perdu tout les 1 000 envois.



(a) Moyenne uniquement

(b) Incluant les durées minimum et maximum observées

FIGURE 4.3.2 – Temps moyen d'attente pour la réception de l'acquittement.

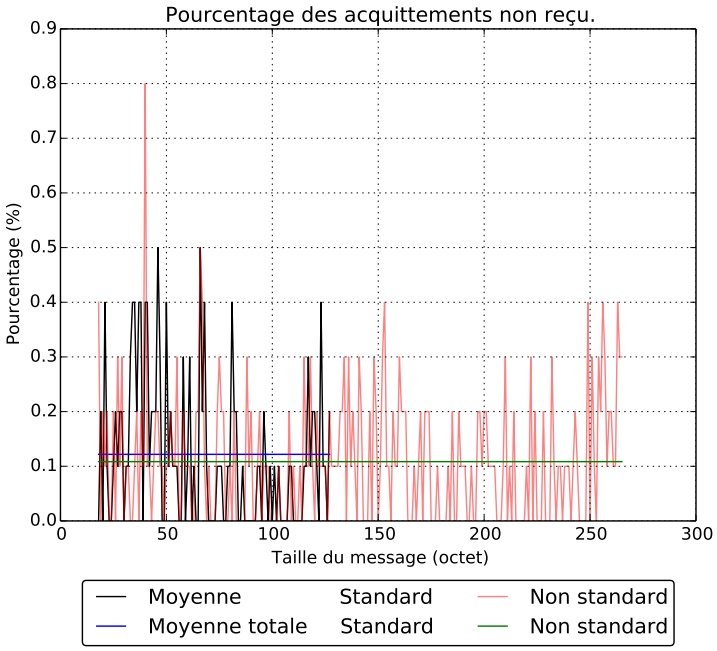
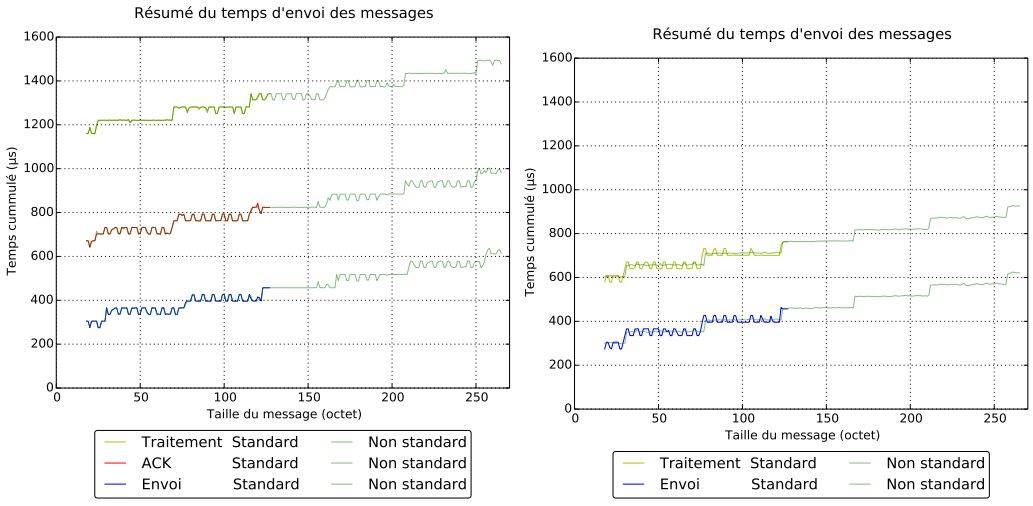


FIGURE 4.3.3 – Pourcentage de perte des acquittements.

4.3.2 Envois et acquittements cumulés

Nous allons mettre en évidence le temps total passé dans la fonction « `dw1000_driver_transmit(...)` ». Cela inclut l'ajout de temps induit par l'attente des acquittements, cette attente provient de deux facteurs : l'attente de l'acquittement proprement dit, mais aussi la désactivation et l'activation des interruptions.

La figure 4.3.4 met en relation les deux types d'envois. On constate que le temps passé à l'attente de l'acquittement est presque de la même durée que le temps passé à l'envoi. La courbe « traitement » prend en compte le temps passé pour désactiver/réactiver les interruptions ainsi que le temps nécessaire à la désactivation de la réception avant l'envoi. Cet overhead est de 300 μs pour l'envoi sans acquittement et 450 μs pour celui avec acquittement. L'overhead de 300 μs représente presque autant de temps que l'envoi d'un message. Il pourrait être intéressant d'améliorer la communication SPI afin de diminuer ces temps d'attentes. En augmentant le débit de transmission entre le DW1000 et le Zolertia Z1 qui est actuellement de 4 Mb/s (limité par le Zolertia) alors que le DW1000 accepte un débit allant jusqu'à 20 Mb/s (fréquence maximale de 20 MHz [18]). Le débit peut théoriquement monter à 8 MHz (8 Mb/s) sur le Zolertia Z1, mais en pratique des perturbations sur les câbles provoquent des dysfonctionnements. En raccourcissant les fils, il serait peut-être possible de monter à ce débit-là.



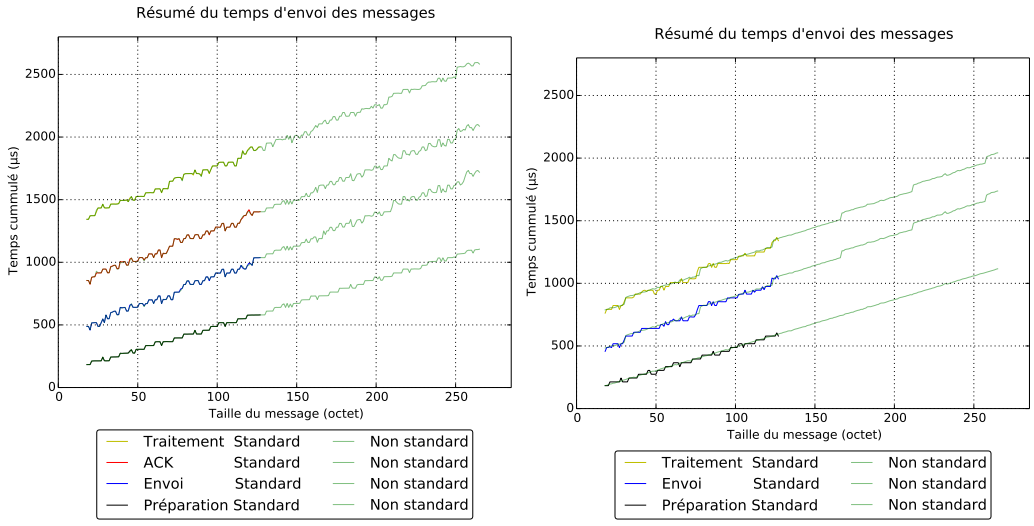
(a) Avec ACK **(b)** Sans ACK

FIGURE 4.3.4 – Temps moyen passé pour l'envoi d'un message, l'attente de l'acquittement et le traitement total. (Ne tiens pas compte de la préparation du message).

4.4 Temps total

La figure 4.4.1 permet de mettre en relation le temps passé à la préparation du message et celui passé à l'envoi de celui-ci. Ce graphique permet de donner une vitesse effective d'envoi des données sans tenir compte du temps passé à la création du message par les couches supérieures.

Avec acquittement, le temps passé à l'envoi d'un message varie en moyenne entre 1400 μs pour un message de 16 octets et 2500 μs pour un message de 265 octets(figure 4.4.1a). Sans acquittement, le temps passé à l'envoi d'un message varie en moyenne entre 800 μs pour un message de 16 octets et 200 μs pour un message de 265 octets(figure 4.4.1a).



(a) Avec ACK **(b)** Sans ACK

FIGURE 4.4.1 – Temps moyen passé pour l'envoi d'un message.

Dans le tableau 4.4.1, on considère les vitesses effectives avec et sans acquittement. On constate que celles-ci sont plus faibles qu'attendu à cause du fait que l'envoi d'un message ne se limite justement pas qu'à son envoi. Les divers overheads successifs font, qu'au final, on obtient une vitesse d'envoi d'environ 1/6 ème de la vitesse nominale configurée. Il faut aussi tenir compte que cette vitesse est plus grande que la vitesse de communication SPI avec le transceiver (4 Mb/s) ce qui limite déjà très fortement la vitesse d'envoi final.

Taille	**Avec ACK**	**Sans ACK**
16	$\frac{16*8}{1400} = \mathbf{90\ kb/s}$	$\frac{16*8}{800} = \mathbf{160\ kb/s}$
127	$\frac{127*8}{1900} = \mathbf{534\ kb/s}$	$\frac{127*8}{1400} = \mathbf{725\ kb/s}$
265	$\frac{265*8}{2500} = \mathbf{848\ kb/s}$	$\frac{265*8}{2000} = \mathbf{1060\ kb/s}$

TABLE 4.4.1 – Résumé des vitesses effectives mesurées pour l'envoi à une vitesse nominal de 6800 kb/s.

La figure 4.4.2 met en avant les écarts de durée maximale pour chaque taille de paquet sur 1 000 envois par taille de paquet. On remarque que l'écart est plus grand en cas d'attente d'acquittement. Les valeurs maximales sont beaucoup plus grandes que les valeurs attendues à cause de l'attente de l'expiration du temps imparti à la réception d'acquittement si celui-ci n'est pas reçu.

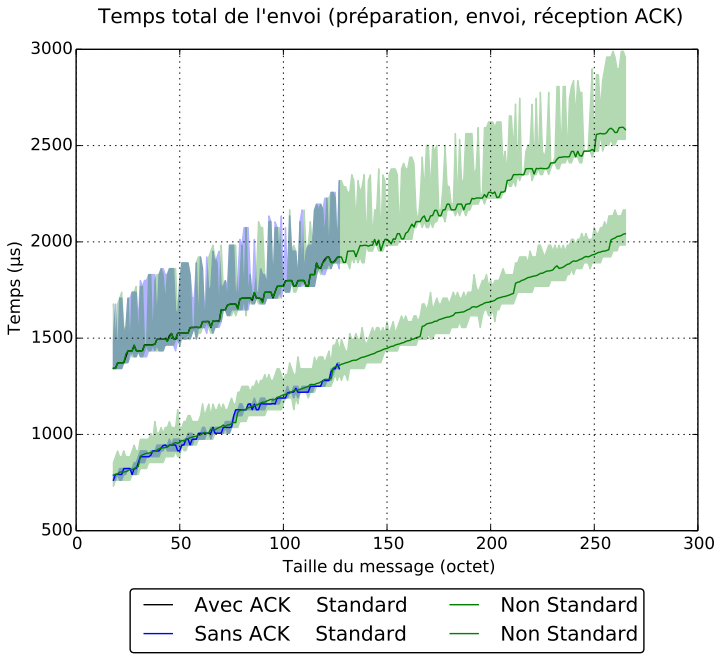


FIGURE 4.4.2 – Temps total d'envoi d'un message.

4.5 Débit nominal de 850 kb/s

Nous avons analysé le temps d'envoi pour le comparer avec le temps théorique. Cela est illustré en figure 4.5.1. Contrairement au débit de 6800 kb/s, le débit mesuré est presque identique au débit théorique.

Le débit effectif est illustré en figure 4.5.2, à partir de ce graphique nous avons calculé les vitesses effectives pour plusieurs tailles de message. Ces vitesses sont illustrées dans le tableau 4.5.1. On constate que pour de longs messages, on atteint une vitesse relative de 490 kb/s soit la moitié du débit. Ce qui est bien mieux qu'en 6,8 Mb/s où l'on atteint seulement 1/6 ème du débit nominal.

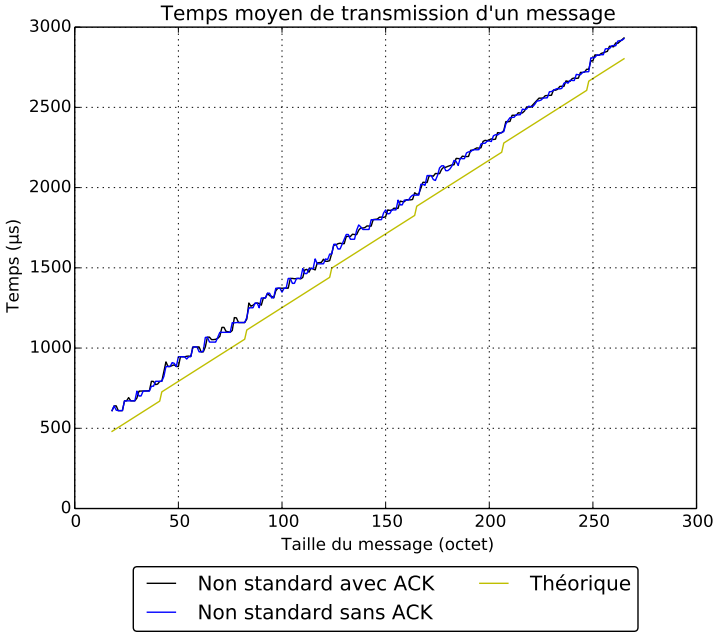
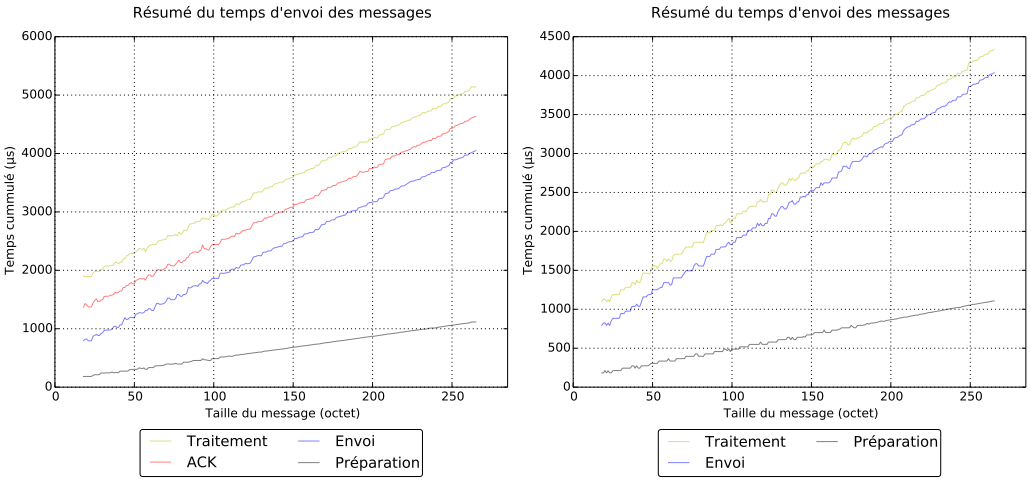


FIGURE 4.5.1 – Temps moyen de transmission d'un message à un débit nominal de 850 kb/s.



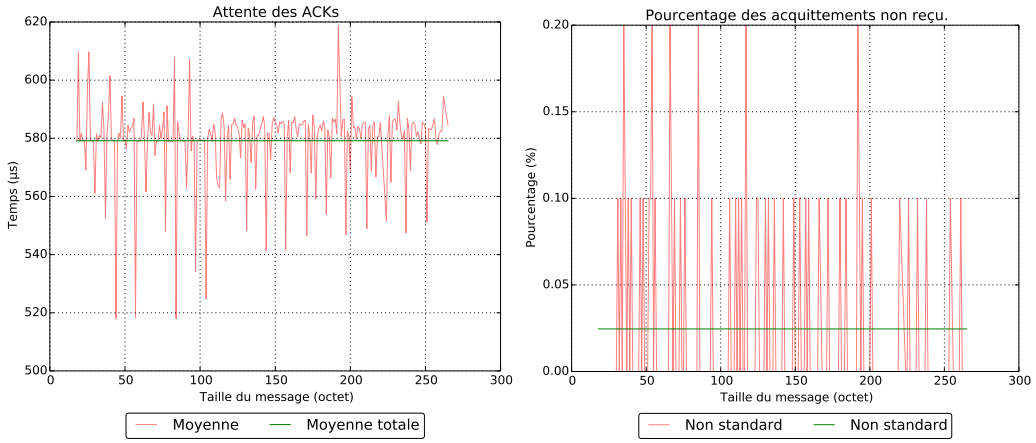
(a) Avec ACK **(b)** Sans ACK

FIGURE 4.5.2 – Temps moyen total passé pour l'envoi d'un message à un débit nominal de 850 kb/s.

Taille	**Avec ACK**	**Sans ACK**
16	$\frac{16*8}{1900}$ = **56 kb/s**	$\frac{16*8}{1150}$ = **111 kb/s**
127	$\frac{127*8}{3250}$ = **313 kb/s**	$\frac{127*8}{2500}$ = **406 kb/s**
265	$\frac{265*8}{5100}$ = **415 kb/s**	$\frac{265*8}{4300}$ = **490 kb/s**

TABLE 4.5.1 – Résumé des vitesses effectives mesurées pour l'envoi à une vitesse nominale de 850 kb/s.

La figure 4.5.3a donne le temps d'attente nécessaire avant la réception de l'acquittement en moyenne. Leurs réceptions sont enregistrées après environs 580 μs soit 210 μs plus tard qu'avec un débit nominal de 6,8 Mb/s. La figure 4.5.3b, permet de déterminer la proportion de messages non acquittés qui est de 0,02 % soit un message non acquitté en moyenne tous les 5 000 messages. Ce ratio est 5 fois plus faible qu'avec un débit de 6,8 Mb/s. Deux facteurs peuvent expliquer ce comportement. Premièrement, on utilise un préambule plus long ce qui permet une meilleure synchronisation à la lecture du message. Ensuite, avec un débit plus faible, un bit étant codé sur un plus « long » signal, la probabilité d'erreur en est atténuée.



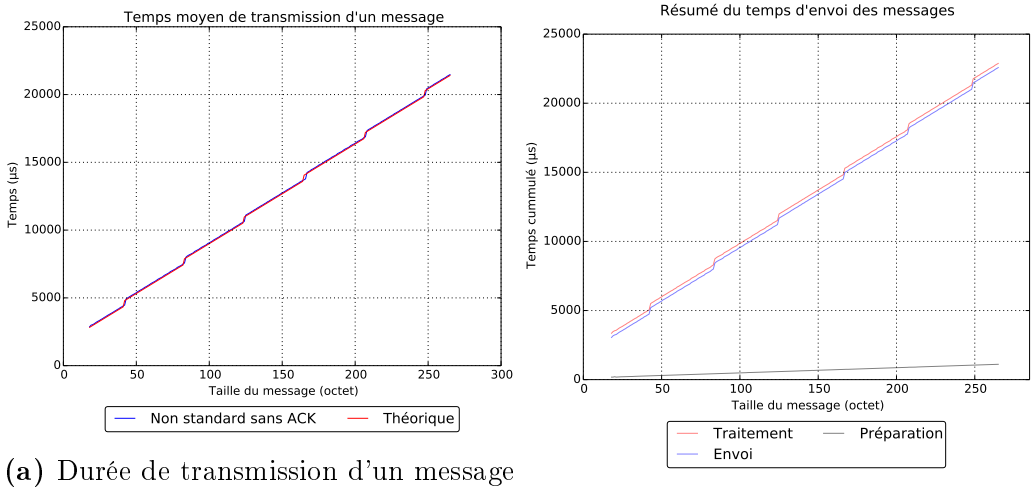
(a) Moyenne uniquement

(b) Pourcentage de perte des acquittements.

FIGURE 4.5.3 – Temps moyen d'attente pour la réception de l'acquittement à un débit nominal de 850 kb/s.

4.6 Débit nominal de 110 kb/s

Pour le débit nominal de 110 kb/s, nous constatons en figure 4.6.1a que le débit théorique est très proche du débit mesuré. En figure 4.6.1b, on illustre le temps total passé pour émettre un paquet, on remarque que le temps de la mise en tampon sur le transceiver est dérisoire comparé à celui de l'envoi.



(a) Durée de transmission d'un message

(b) Temps total d'envois d'un message.

FIGURE 4.6.1 – Mesure pour un débit nominal de 110 kb/s.

4.7 Durée d'attente maximale

La fonction d'envoi « `dw1000_driver_transmit` » possède deux constantes, `DW1000_TX_TIMEOUT` et `DW1000_ACK_TIMEOUT` définissant respectivement les durées maximales d'attente de fin de transmission et de réception d'acquittement. La valeur de ses constantes varie en fonction du débit nominal utilisé pour envoyer les données. Pour définir chaque constante, nous allons comparer les valeurs théoriques et les valeurs mesurées en pratique. À partir de celles-ci, nous allons définir les valeurs en tenant compte des écarts rencontrés.

4.7.1 Durée d'attente maximale de fin de transmission

La durée théorique peut être calculée à partir des paramètres choisis lors de la configuration du transceiver. Cette durée est donnée par la fonction $f_{trans}(x)$ définie en figure 4.2.2. Pour rappel, voici son équation :

$$f_{trans}(x) = t_{preamble} + t_{SFD} + t_{PHR} + t_{MAC}(x)$$

Le choix du débit influence les variables utilisées dans la fonction $f_{trans}(x)$, les valeurs de ses variables peuvent être retrouvées grâce au tableau 3.4.1.

À partir de cette fonction et pour chaque débit du transceiver (6,8 Mb/s, 850 kb/s et 110 kb/s), nous allons établir 2 valeurs, à savoir la valeur du temps de transmission d'un paquet de 127 octets (format standard) et d'un paquet de 265 octets (format étendu). Ces deux valeurs vont être comparées aux valeurs mesurées sur la moyenne de 1 000 envois pour un débit de 8,6 Mb/s et 100 envois pour les autres. Le tableau 4.7.1 donne les valeurs mesurées et les valeurs théoriques ainsi que l'écart entre celle-ci. Premièrement, on constate que cet écart varie de 34 μs à presque 150 μs en fonction de la configuration. On observe que l'écart reste dans le même intervalle pour chaque débit. Pour une trame de 265 octets à un débit de 110 kb/s, on remarque que la durée théorique est supérieure à la durée mesurée, cet écart peut s'expliquer par un débit d'envois plus rapide que celui donné dans la fiche technique du transceiver.

Débit	**Format standard** (127 octets)			**Format étendu** (265 octets)		
(kb/s)	**Théorique**	**Mesuré**	**Écart**	**Théorique**	**Mesuré**	**Écart**
110	11 196	11 230	34 (0,3 %)	21 572	21 469	-103 (0,4 %)
850	1 526	1 650	124 (8 %)	2 823	2 932	109 (4 %)
8 600	312	457	145 (46 %)	474	614	140 (30 %)

TABLE 4.7.1 – Temps de transmission (μs) de paquet de format standard et de format étendu.

Le tableau 4.7.2 donne les durées minimum et maximum mesurées pour les transmissions. On constate que ses durées sont assez stables et varie au maximum de moins de 100 μs. Les variations sont plus faibles pour de petits paquets à haut débit, mais plus grandes pour de longs paquets à haut débit. Cela est peut être dû à une latence ajoutée par l'algorithme « reed-solomon » qui est fixe, car le temps de calcul est identique pour tous les débits et qui n'est pas correctement « absorbé » par le facteur $\frac{1}{0,87}$ de convertion entre le débit nominal et le débit réellement utilisé par le transceiver.

Étant donné l'écart de temps entre un envoi de 127 octets et un envoi de 265 octets, qui est presque d'un facteur deux, nous allons fixer la constante `DW1000_TX_TIMEOUT` en fonction du débit et du format utilisé. Le tableau 4.7.3 donne les valeurs fixées pour `DW1000_TX_TIMEOUT`, on constate que l'on a ajouté entre 100 et 200 μs d'écart avec les valeurs maximale mesurée, et ce afin d'éviter des expirations si l'on a dans des conditions différentes de celles de test.

Débit	**Format standard** (127 octets)			**Format étendu** (265 octets)		
(kb/s)	**Min**	**Max**	**Écart**	**Min**	**Max**	**Écart**
110	11 199	11 260	70 (0,6 %)	21 453	21 514	61 (0,2 %)
850	1 617	1 678	61 (3,7 %)	2 899	2 990	91 (3,1 %)
8 600	457	488	31 (6,7 %)	579	671	92 (15,8 %)

TABLE 4.7.2 – Temps de transmission mesuré (μs) de paquet de format standard et de format étendu.

Débit	**Format standard** (127 octets)		**Format étendu** (265 octets)	
(kb/s)	**Max mesuré**	**Max fixé**	**Max mesuré**	**Max fixé**
110	11 260	**13 000**	21 514	**23 000**
850	1 678	**1 900**	2 990	**3 100**
8 600	488	**625**	671	**800**

TABLE 4.7.3 – Temps de transmission (μs) de paquet de format standard et de format étendu.

4.7.2 Durée d'attente maximale de réception d'acquittement

Pour l'attente des acquittements, il faut prendre deux paramètres en considération. Le premier est le temps de transmission d'une trame de 5 octets, i.e la trame contenant l'acquittement. Le second est la distance entre les deux transceivers qui rajoutera un délai de propagation ainsi que le temps nécessaire pour passer de réception à émission. La portée de transmission des DW1000 est de 290 mètres [16]. La vitesse de propagation est de $\frac{2c}{3}$ m/s [37] où c est la vitesse de la lumière (c.-à-d. 299 792 458 m/s). La vitesse de propagation est de $290 * 2 \div \frac{2c}{3} \approx 2902$ ηs soit 2,9 μs. À cela, il faut rajouter le temps nécessaire pour passer de réception à émission qui est de 10 μs [18]. On obtient donc une valeur de 13 μs à ajouter au temps de transmissions d'un paquet de 5 octets. Pour ce temps de transmission, nous allons comparer la valeur théorique avec les valeurs mesurées. Le tableau 4.7.4 met en parallèle les valeurs mesurées avec les valeurs théoriques. On remarque que les valeurs théoriques sont plus petites que les valeurs mesurées. Le mode d'emploi indique que le préambule d'un acquittement peut être deux fois plus long que celui du message acquitté [19][p52]. La valeur de `DW1000_ACK_TIMEOUT` est fixée à 460 μs en 6,8 Mb/s et 640 μs en 850 kb/s, ce qui laisse une certaine marge au cas où, en dehors des mesures de tests, les valeurs mesurées puissent être plus grandes (les mesures précédentes ont montré que le modèle théorique n'était pas strictement correcte).

Débit	**Valeur**	**Valeurs mesurées**		**Valeur**
(kb/s)	**Théorique**	**Moyenne**	**Minimum**	**Fixée**
110	2 023 + 13	-	-	-
850	380 + 13	580	500	**640**
8 600	169 + 13	375	300	**460**

TABLE 4.7.4 – Valeurs mesurées et définies (en μs) pour l'attente d'un acquittement

4.8 Conclusion

Avec ces tests, on remarque que le taux de perte de paquet est assez faible. On ne reçoit pas les acquittements environ une fois sur 1 000 pour un débit de 6.8 Mb/s et une fois sur 5 000 pour un débit de 850 kb/s. Si l'on ne reçoit pas un acquittement tout les 1 000 messages, cela signifie que l'on reçoit l'acquittement pour 999 messages (aller-retour) soit environ 2 000 messages. Cela indique donc que l'on a environ 1 message sur 2 000 qui est perdu ou corrompu à 6,8 Mb/s et 1 message sur 10 000 en 850 kb/s. À un débit de 6,8 Mb/s, les overheads sont assez conséquents comparé au temps de transmission étant donné que le débit de transmission est plus élevé que le débit de communication entre le Zolertia et le transceiver. Avec des débits plus faibles, 850 kb/s et 110 kb/s, on obtient de « meilleures » performances dans le sens où l'on a proportionnellement moins de pertes dues à une communication SPI trop lente.

Enfin, les tests à 6,8 Mb/s ont nécessité environ 250 000 envois par « mode » sans que cela ne pose de problème au niveau du Zolertia (redémarrage ou blocage). Le driver est donc correctement intégré à la plateforme et ne produit pas de dysfonctionnement sur celle-ci.

Chapitre 5

Conclusions

Dans le cadre de ce projet, nous avons pu réaliser l'implémentation d'un driver radio pour un transceiver Ultra Wide Band. Celle-ci a permis d'acquérir des connaissances sur les technologies radio et le développement sur système embarqué. Cette implémentation a aussi permis de mettre en évidence les défis liés au développement sur de tels systèmes en raison des limites en ressources de ceux-ci. Un exemple de défis est l'optimisation via des directives de compilation afin éviter des conditions où les valeurs sont toujours identiques à l'exécution. Le driver développé possède les fonctionnalités du driver du CC2420 présent sur le Zolertia Z1, hormis le support du « Clear Channel Assessment », ce qui implique une modification des protocoles du duty cycle discuté en section 3.3.4. Il possède aussi certaines fonctionnalités non présentes dans le driver du CC2420, comme le choix d'un débit nominal de 6800, 850 ou 110 kb/s contre uniquement 250 kb/s pour le CC2420. Il implémente aussi le support physique des acquittements IEEE 802.15.4 automatique ce qui n'est pas le cas du driver du CC2420, du moins pas au niveau de la transmission (pas d'attente d'acquittement après l'envoi). Le driver permet l'utilisation de trames de format étendu ; leur utilisation peut se faire uniquement entre transceivers DW1000, mais permet de meilleures performances en limitant les overheads grâce à des trames plus longues, 265 octets maximum contre 127 octets pour le format standard. Ce driver est conçu de façon modulable et peut facilement être modifié facilement afin de l'utiliser sur d'autres plateformes compatibles avec Contiki OS. Enfin, la section Test & Validation à permis de mettre en avant la fiabilité de celui-ci et ses bonnes performances.

Chapitre 6

Références

[1] Le groupe de travail 802.15.4. http://www.ieee802.org/15/pub/TG4.html, 2015. [En Ligne ; Accès : 2015-12-16].

[2] Site web de contiki os. http://www.contiki-os.org/, 2016. [En Ligne ; Accès : 2016-05-16].

[3] Sourcefoge Contiki 2.6. radio_driver Struct Reference. http://contiki.sourceforge.net/docs/2.6/a00071.html. [En Ligne ; Accès : 2016-05-16].

[4] ANRG. Contiki Installation on Ubuntu. http://anrg.usc.edu/contiki/index.php/Installation. [En Ligne ; Accès : 2016-05-16].

[5] ANRG. The network stack of Contiki OS. http://anrg.usc.edu/contiki/index.php/File:Contikinetstack.png. [En Ligne ; Accès : 2016-04-22].

[6] IEEE STANDARDS ASSOCIATION. 802.15.4-2003. https://standards.ieee.org/getieee802/download/802.15.4-2003.pdf, 2003. [En Ligne ; Accès : 2015-12-17].

[7] IEEE STANDARDS ASSOCIATION. 802.15.1-2005. https://standards.ieee.org/getieee802/download/802.15.1-2005.pdf, 2005. [En Ligne ; Accès : 2015-12-17].

[8] IEEE STANDARDS ASSOCIATION. 802.15.4-2011. https://standards.ieee.org/getieee802/download/802.15.4-2011.pdf, 2011. [En Ligne ; Accès : 2015-12-17].

[9] Berkeley and the Intel Berkeley Research Lab. Large-Scale Demonstration of Self-Organizing Wireless Sensor Networks. http://webs.cs.berkeley.edu/800demo/, 2001. [En Ligne ; Accès : 2016-02-21].

[10] GNU Binutils. GNU Binutils. https://www.gnu.org/software/binutils/. [En Ligne ; Accès : 2016-05-16].

[11] Cburnett. Liaison SPI : un maître et un esclave. https://fr.wikipedia.org/wiki/Serial_Peripheral_Interface#/media/File:SPI_single_slave.svg. [En Ligne ; Accès : 2016-05-16].

[12] CoAP. CoAP, RFC 7252 Constrained Application Protocol. http://coap.technology/. [En Ligne ; Accès : 2016-04-22].

[13] Ciaran Connell. What's The Difference Between Measuring Location by UWB, Wi-Fi, and Bluetooth ? http://electronicdesign.com/communications/, 2015. [En Ligne ; Accès : 2015-12-17].

[14] Heinz Jäckel David Barras, Frank Ellinger. A Comparison between Ultra-Wideband and Narrowband Transceivers. http://www2.ife.ee.ethz.ch/case/ife/publications/comparisonOfTransceivers.pdf. [En Ligne ; Accès : 2016-02-20].

[15] Decawave. Decawave - DW1000. http://www.decawave.com/products/dwm1000-module, 2015. [En Ligne ; Accès : 2015-12-27].

[16] Decawave. Decawave - DW1000. http://www.decawave.com/products/dw1000, 2015. [En Ligne ; Accès : 2015-12-29].

[17] Decawave. Decawave - DW1000. http://www.decawave.com/products/overview, 2015. [En Ligne ; Accès : 2015-12-27].

[18] Decawave. Decawave - DW1000 Datasheet. http://www.decawave.com/support, 2015. [En Ligne ; Accès : 2016-02-09].

[19] Decawave. Decawave - DW1000 USER MANUAL. http://www.decawave.com/support, 2015. [En Ligne ; Accès : 2016-02-10].

[20] Daniel Drake. UDEV : Comment ça marche ? https://doc.ubuntu-fr.org/udev. [En Ligne ; Accès : 2016-04-16].

[21] Traitement du signal. Traitement du signal : Ultra wideband. http://www.traitement-signal.com/ultra_wideband.php, 2016. [En Ligne ; Accès : 2016-02-14].

[22] Samuel Dubouloz. *Développement d'architectures avancées pour communications ultra large bande (UWB) dans des applications bas débit.* Theses, Télécom ParisTech, June 2008.

[23] A. Dunkels, B. Gronvall, and T. Voigt. Contiki - a lightweight and flexible operating system for tiny networked sensors. In *Local Computer Networks, 2004. 29th Annual IEEE International Conference on*, pages 455–462, Nov 2004.

[24] Eistec. Mulle wireless sensor platform. http://www.eistec.se/mulle/, 2016. [En Ligne ; Accès : 2016-04-16].

[25] Jens Eliasson, Per Lindgren, and Jerker Delsing. A bluetooth-based sensor node for low-power ad hoc networks. *Journal of Computers*, 3(5) :1–10, 2008.

[26] Muhammad Omer Farooq and Thomas Kunz. Operating systems for wireless sensor networks : A survey. *Sensors*, 11(6) :5900–5930, 2011.

[27] D. Kell G. Shreve. A Precision Location Network Using Ultra Wideband WLAN Radios. http://www.wlan01.wpi.edu/proceedings/wlan62d.pdf. [En Ligne ; Accès : 2016-02-20].

[28] Sabih H. Gerez. Implementation of Digital Signal Processing : Some Background on GFSK Modulation.

[29] Greg Hackmann. 802.15 Personal Area Networks. http://www.cse.wustl.edu/~jain/cse574-06/ftp/wpans/index.html, 2016. [En Ligne ; Accès : 2016-03-16].

[30] J. Hamon. *Asynchronous oscillators and architectures for UWB impulse radio signal processing*. Theses, Institut National Polytechnique de Grenoble - INPG, October 2009. ISBN : 978-2-84813-138-2.

[31] Osama Haraz. Why do we need Ultra-wideband ? http://www.vlsiegypt.com/home/?p=518, 2012. [En Ligne ; Accès : 2016-02-17].

[32] National Instriments. What Is a Wireless Sensor Network ? http://www.ni.com/white-paper/7142/en/, 2012. [En Ligne ; Accès : 2016-02-21].

[33] Pozyx Labs. How does ultra-wideband work. https://www.pozyx.io/Documentation/doc_howDoesUwbWork, 2016. [En Ligne ; Accès : 2016-03-16].

[34] Grégory Wolowiec Michel Vongvilay, Gabriel Nguyen Ngoc. Nguyen Vongvilay Wolowiec présentation Bluetooth. http://www-igm.univ-mlv.fr/~duris/NTREZO/20042005/Nguyen-Vongvilay-Wolowiec-Bluetooth.pdf, 2005. [En Ligne ; Accès : 2015-12-17].

[35] Nest. Nest Learning Thermostat. https://store.nest.com/product/thermostat/, 2016. [En Ligne ; Accès : 2016-02-21].

[36] Jianjun Ni, D. Arndt, Phong Ngo, Chau Phan, K. Dekome, and J. Dusl. Ultra-wideband time-difference-of-arrival high resolution 3d proximity tracking system. In *Position Location and Navigation Symposium (PLANS), 2010 IEEE/ION*, pages 37–43, May 2010.

[37] Bruno Quoitin. Cours de Réseaux I/II. [Cours].

[38] Bruno Quoitin. Sourceforge - libbuspirate. http://sourceforge.net/projects/libbuspirate/, 2015. [En Ligne ; Accès : 2015-12-29].

[39] Z. Sahinoglu and S. Gezici. Ranging in the ieee 802.15.4a standard. In *IEEE Wireless and Microwave Technology Conference (WAMICON)*, December 2006.

[40] Naveen Sastry and David Wagner. Security considerations for ieee 802.15. 4 networks. In *Proceedings of the 3rd ACM workshop on Wireless security*, pages 32–42. ACM, 2004.

[41] Futurama Science. Bruit blanc. http://www.futura-sciences.com/, 2016. [En Ligne ; Accès : 2016-03-16].

[42] SemiconductorStore. SemiconductorStore - DWM1000. http://www.semiconductorstore.com/Pages/asp/search.asp?PL=0207&Query=Search%20DecaWave%20at%20SemiconductorStore.com, 2016. [En Ligne ; Accès : 2016-02-09].

[43] Bluetooth SIG. Security, Bluetooth Smart (Low Energy). https://developer.bluetooth.org/TechnologyOverview/Pages/LE-Security.aspx. [En Ligne ; Accès : 2016-05-16].

[44] Bluetooth SIG. Core Version 4.2. https://www.bluetooth.com/specifications/adopted-specifications, 2014. [En Ligne ; Accès : 2016-05-16].

[45] Bluetooth SIG. What is Bluetooth Technology ? https://www.bluetooth.com/what-is-bluetooth-technology, 2016. [En Ligne ; Accès : 2016-03-16].

[46] Sourceforge. Contiki Installation on Ubuntu. http://zolertia.sourceforge.net/wiki/index.php/Mainpage:Contiki_installation_Ubuntu. [En Ligne ; Accès : 2016-05-16].

[47] Sutekh. Create your own udev rules to control removable devices. http://ubuntuforums.org/showthread.php?t=168221. [En Ligne ; Accès : 2016-04-16].

[48] Chávez Edgar (Eds.) Syrotiuk, Violet R. 4th international conference, adhoc-now 2005, cancun, mexico, october 6-8, 2005, proceedings. In *Ad-Hoc, Mobile, and Wireless Networks*, page 100, 2005.

[49] Tsaitgaist. File :Sensornode.svg. https://en.wikipedia.org/wiki/File:Sensornode.svg, 2009. [En Ligne ; Accès : 2016-14-05].

[50] Nicolas Tsiftes and Adam Dunkels. A database in every sensor. In *Proceedings of the ACM Conference on Networked Embedded Sensor Systems, ACM SenSys 2011*, Seattle, WA, USA, November 2011.

[51] Glacsweb Wiki. Contiki Radio Driver. http://wiki.glacsweb.info/index.php/Contiki_Radio_Driver, 2014. [En Ligne ; Accès : 2016-05-16].

[52] Wikipédia. Wikipédia - Bluetooth. https://fr.wikipedia.org/wiki/Bluetooth, 2015. [En Ligne ; Accès : 2015-12-17].

[53] Zolertia. Site constructeur du Zolertia Z1. http://zolertia.io/. [En Ligne ; Accès : 2016-04-10].

Chapitre 7

Annexes

7.1 Nommage des périphériques USB

L'utilisation de plusieurs périphériques USB n'étant pas aisée dû au fait que leurs noms sur la machine dépendent de l'ordre dans lequel ils y sont connectés, nous avons choisi de leur donner des noms fixes. Pour cela, nous avons utilisé l'outil « `udev` » disponible sur Ubuntu [20]. En suivant un tutoriel disponible sur un forum [47], nous avons obtenu le script présent en figure 7.1.1. Ce script se base sur le nom des produits « `ATTRS{product}` » et sur leur numéro de série « `ATTRS{serial}` » pour renommer les deux Zolertia Z1 utilisés pour le développement dont les numéros de nœuds sont 6 et 10 en respectivement « `z1_6` » et « `z1_10` » et les Bus Pirates associés en « `bp_6` » et « `bp_10` ».

```
1 SUBSYSTEMS=="usb", ATTRS{product}=="BusPirate V3", ATTRS{
    idProduct}=="6001", ATTRS{serial}=="BP4", NAME="bp_6",
    SYMLINK="bp_6"
2 SUBSYSTEMS=="usb", ATTRS{product}=="Zolertia Z1", ATTRS{
    serial}=="Z1RC3583", NAME="z1_6", SYMLINK="z1_6"
3 SUBSYSTEMS=="usb", ATTRS{product}=="BusPirate V3", ATTRS{
    idProduct}=="6001", ATTRS{serial}=="BP2", NAME="bp_10",
    SYMLINK="bp_10"
4 SUBSYSTEMS=="usb", ATTRS{product}=="Zolertia Z1", ATTRS{
    serial}=="Z1RC3582", NAME="z1_10", SYMLINK="z1_10"
```

FIGURE 7.1.1 – Commandes utilisant « `udev` » pour renommer les périphériques `USB` (`Zolertia Z1` et `Bus Pirate`).

7.2 IEEE 802.15.4-2011

Le tableau 7.2.1 illustre les différents canaux de fréquences disponibles dans la bande industrielle, scientifique et médicale pour le standard IEEE 802.15.4-2011. On remarque que les attributions ne sont pas les mêmes partout dans le monde.

Région	**Bandes de fréquences** (MHz)	**Nombre de canaux**	**Largeur de bande** (MHz)	**Débit théorique** (kb/s)
Chine :		8	-	-
	430 – 434	2	2	250
	314 – 316	2	2	250
	779 - 787	4	2	250
Europe	868 - 868,6	1	0,6	20/250
Amérique	902 – 928	10	2	40/250
Japon :		22	-	-
	951,2 - 955,4	8	0,6	100
	954,4 - 954,8	2	0,2	20
	951,1 - 955,5	11	0,4	20
Monde	2400 - 2483,5	16	5	250

TABLE 7.2.1 – Tableau des bandes de fréquences et des débits (sans UWB) de l'IEEE 802.15.4-2011 [8, p.147-148].

Le tableau 7.2.2 illustre les canaux Ultra Wide Band du standard IEEE 802.15.4-2011. Les numéraux de canaux mis en gras sont ceux disponibles sur le transceiver DecaWave DW1000 utilisé dans le cadre de ce projet.

Numéro du canal	**Centre du spectre** (MHz)	**Largeur de bande** (MHz)
Bande sous-gigahertz :		
0	499,2	499,2
Bandes basses :		
1	3494,4	499,2
2	3993,6	499,2
3	4492,8	499,2
4	3993,6	1331,2
Bandes hautes :		
5	6489,6	499,2
6	6988,8	499,2
7	6489,6	1081,6
8	7488,0	499,2
9	7987,2	499,2
10	8486,4	499,2
11	7987,2	1331,2
12	8985,6	499,2
13	9484,8	499,2
14	9984,0	499,2
15	9484,8	1354,97

TABLE 7.2.2 – Description des canaux UWB IEEE 802.15.4-2011 [8, p.211].

7.3 Connexions entre Zolertia Z1 et DWM1000

Le tableau 7.3.1, illustre les connexions filaires nécessaires entre le Zolertia Z1 et le DWM1000 afin de permettre l'utilisation de communication SPI entre eux.

Fonction	**Port sur le Zolertia Z1**	**Port sur leDWM1000**
Interruption	JP1C pin 46, Port2.3	GPIO8
SCLK	JPI1B pin 34, SPI.CLK	SPICLK
MOSI	JPI1B pin 36, SPI.SIMO	SPIMOSI
MISO	JPI1B pin 38, SPI.SOMI	SPIMISO
SS	JPI1B pin 32, Port4.0	SPICS
GND	JPI1B pin 24, DGND	GND

TABLE 7.3.1 – Description des connexions entre le Zolertia Z1 et le DWM1000.

7.4 Durée d'une transmission UWB

Le tableau 7.4.1 permet de connaître la durée d'émission des symboles dans une trame physique UWB du transceiver DecaWave DW1000 en fonction du paramétrage de celui-ci. Il est utilisé pour permettre de calculer le temps théorique d'une transmission.

PRF (MHz)	**Data Rate** (Mb/s)	**SHR** (ns)	**PHR** (ns)	**Data** (ns)
16	0, 11	993, 59	8205, 13	8205, 13
16	0, 85	993, 59	1025, 64	1025, 64
16	6, 81	993, 59	1025, 64	128, 21
64	0, 11	1017, 63	8205, 13	8205, 13
64	0, 85	1017, 63	1025, 64	1025, 64
64	6, 81	1017, 63	1025, 64	128, 21

TABLE 7.4.1 – Durée des symboles émis par le DW1000 [18].

La figure 7.4.1 illustre le code Python utilisé pour calculer la durée théorique d'une transmission à partir de la longueur du préambule, du débit nominal et de la taille x de la trame MAC. Elle se base sur le tableau 7.4.1 et la formule donnée en figure 4.2.2.

```python
def theorical_speed(l_preamble, speed, x, PRF=16):
    # duree d un symbole dans le SHR
  if(PRF == 16):
    s_shr = 993.59 / 1000
  else:
    s_shr = 1017.63 / 1000

  # Longueur SFD et duree d un symbole PHR
  if(speed == 110):
    l_sfd = 64
    s_phr = 8205.13 / 1000
  else:
    l_sfd = 8
    s_phr = 1025.64 / 1000

  l_phr = 21 # longueur du PHR

  # Duree d un symbole dans la trame MAC
  if(speed == 110):
    s_mac = 8205.13 / 1000
  elif(speed == 850):
    s_mac = 1025.64 /1000
  else: #speed == 6.8 Mb /s
    s_mac = 128.21 / 1000

  t_shr = (l_preamble + l_sfd) * s_shr
  t_phr = l_phr * s_phr

  def reedSolomonParityBit(x):
    return ((int(x*8)/330)+1)*48
  t_mac = s_mac * ((8*x) + reedSolomonParityBit(x))
  return t_shr + t_phr + t_mac
```

FIGURE 7.4.1 – Fonction python permettant le calcul du temps théorique d'une transmission.

7.5 Temps de préparation et de transmission

La figure 7.5.1 illustre la fonction « `dw1000_driver_prepare(...)` » et l'emplacement des mesures de temps dans celle-ci.

```
static int dw1000_driver_prepare(const void *payload,
                                 unsigned short payload_len){
  PRINTF("dw1000_driver_prepare\r\n");

  rtimer_clock_t tinit = RTIMER_NOW();

  /**
  * Copy data to the TX buffer.
  */

  rtimer_clock_t tend = RTIMER_NOW();

  printf("%u, ", data_len); //len
  printf("%u, ", (((tend - tinit)*1000000)/RTIMER_SECOND));

  return 0;
}
```

FIGURE 7.5.1 – Illustration de l'emplacement des mesures pour la fonction « `dw1000_driver_prepare(...)` ».

La figure 7.5.2 illustre la fonction « `dw1000_driver_transmit(...)` » et l'emplacement des mesures de temps dans celle-ci.

```
static int dw1000_driver_transmit(unsigned short payload_len)
    {
  PRINTF("dw1000_driver_transmit\r\n");

  rtimer_clock_t tinit = RTIMER_NOW();
  // dw1000 to idle
  // disable intteruption if ACK
  rtimer_clock_t t1 = RTIMER_NOW();
  // init TX and wait until transmit down
  rtimer_clock_t t2 = RTIMER_NOW();
  rtimer_clock_t tack1 = RTIMER_NOW();
  // wait ACK and proccess ACK
  rtimer_clock_t tack2 = RTIMER_NOW();
  // if ACK > enable interrupt
  //          dw1000 to idle (fix no reception of ACK)
  // reenable the rx
  rtimer_clock_t tend = RTIMER_NOW();

  printf("%u, ", (((t2 - t1)*1000000)/RTIMER_SECOND)); //
      send time
  printf("%u, ", (((tack2 - tack1)*1000000)/RTIMER_SECOND));
      // ACK time
  printf("%u, ", (((tend - tinit)*1000000)/RTIMER_SECOND));
      //total
  if(tx_return == RADIO_TX_NOACK)
    printf("1, ");
  else
    printf("0, ");
  if(tx_return == RADIO_TX_ERR)
    printf("1");
  else
    printf("0");
  printf("\r\n"); //end
  return tx_return;
}
```

FIGURE 7.5.2 – Illustration de l'emplacement des mesures pour la fonction « `dw1000_driver_transmit(...)` ».

7.6 Écriture d'un registre via SPI

La figure 7.6.1 mets en avant une fonction permettant d'écrire des données dans un registre du DW1000 via une communication SPI. Cette fonction est issue du fichier `dw1000-arch-z1.c` présent dans le dossier `/examples/z1-dw1000/dw1000/`.

```
void dw_write_subreg(uint32_t reg_addr, uint32_t subreg_addr,
    uint32_t subreg_len, uint8_t *p_data)
{
  int reg_inst = 3;
  int i;
  int n_len = subreg_len + reg_inst;
  // Check if 3-octet header is required or if 2 will do
  uint8_t isThreeOctet = (uint8_t) (subreg_addr > 0x7F ? (1
      << 7) : 0);
  // Prepare instruction
  uint8_t instruction[3] = {0x00, 0x00, 0x00};
  instruction[0] = (uint8_t) (0xC0 | (reg_addr & 0x3FUL)); /*
      write bit = 1, subreg present bit = 1 */
  instruction[1] = (uint8_t) (isThreeOctet | (subreg_addr & 0
      x7F)); /* isThreeOctet is the extended address bit */
  // Write instruction
  if (isThreeOctet != 0) {
    instruction[2] = (uint8_t) ((subreg_addr & 0x7F80UL) >>
        7);
    reg_inst = 3;
  } else {
    reg_inst = 2;
    n_len--;
  }
  // SPI communications
  // Asserting CS
  DW1000_SPI_ENABLE();
  //write instruction
  for (i = 0; i < reg_inst; i++){
    SPI_WRITE( (char) instruction[i]);
  }
  //write data
  for (i = reg_inst; i < n_len; i++){
    SPI_WRITE( (char) *(p_data++));
  }
  // De-asserting CS
  DW1000_SPI_DISABLE();
}
```

FIGURE 7.6.1 – Illustration de la fonction « `dw_write_subreg(...)` » permettant l'écriture de données dans un registre en SPI.