
DeSem
DeSenet laboratory report
Rémy Macherel
January 15, 2022

Authors :
Macherel Rémy
Sterren Thomas

Teacher :
Rieder Medard

Table des matières

1	Introduction	5
2	Modifications du code	6
3	Classes implémentées	7
3.1	MPDU	7
3.1.1	Méthodes implémentées	7
3.1.1.1	MPDU()	7
3.1.1.2	getEPDUCount()	7
3.1.1.3	commitSv(SvGroup,uint8_t)	7
3.1.1.4	commitEv(EvId,uint8_t)	8
3.1.1.5	insertBuffer()	8
3.1.1.6	reset()	8
3.1.1.7	setEPDUCount()	8
3.1.1.8	getBuffer()	8
3.1.1.9	getRemainingSpace()	8
3.1.1.10	evPDUWrite(SharedByteBuffer)	8
3.1.2	Séquence générique d'écriture d'un MPDU	9
3.1.3	svPDU	9
3.1.4	evPDU	9
3.1.5	Transmission du MPDU	10
3.2	JoystickApplication	10
3.2.1	Propriétés	10
3.2.2	Méthodes implémentées	10
3.2.2.1	JoystickApplication()	10
3.2.2.2	start()	10
3.2.2.3	processEvent()	11
3.2.2.4	joystick()	11
3.2.2.5	onPositionChanged(IJoystick::Position)	11
3.2.2.6	_readJoystickValue()	11
3.2.3	Machine d'état	11
4	Diagramme de séquence de l'application	12
4.1	Diagrammes de séquence détaillés	13
4.1.1	Joystick	13
4.1.1.1	Création et écriture MPDU	14
5	Tests	15
5.1	Remarques sur les tests	16
5.1.1	Test appui simple	16
5.1.2	Appui continu	16
5.1.3	Appuis multiples	16
5.1.3.1	Déplacement de la fenêtre sensor	17
5.1.3.2	Affichage d'erreurs dans le test bench	17
5.1.3.3	Déplacements rapides du sensor	18
5.1.3.4	Comparaison des frames entre mon programme et le sensor de démo	18

6 Conclusion

19

List of Figures

2.1	Aperçu des modifications à apporter	6
3.1	Diagramme de la classe MPDU	7
3.2	Séquence d'écriture du MPDU	9
3.3	JoystickApplication UML Diagram	10
3.4	JoystickApplication machine d'état	11
4.1	Diagramme de séquence générique de l'application	12
4.2	Diagramme de séquence détaillé du joystick	13
4.3	Diagramme de séquence détaillé du MPDU	14
5.1	Position initiale du sensor et valeurs d'accéléromètre	17
5.2	Position modifiée du sensor et valeurs d'accéléromètre	17
5.3	Frames générées par le programme développé	18
5.4	Frames générées par le sensor de démo	18

List of Tables

1. Introduction

Dans le cadre du cours MA-DeSem, il nous a été demandé de réaliser le protocole DeseNET sur un système de type STM32 Nucleo. Les spécifications du protocole de communication ont été fournies pour ce travail.

La structure de base du projet fût fournie et nous avons du apporter les modifications nécessaires au bon fonctionnement du protocole.

2. Modifications du code

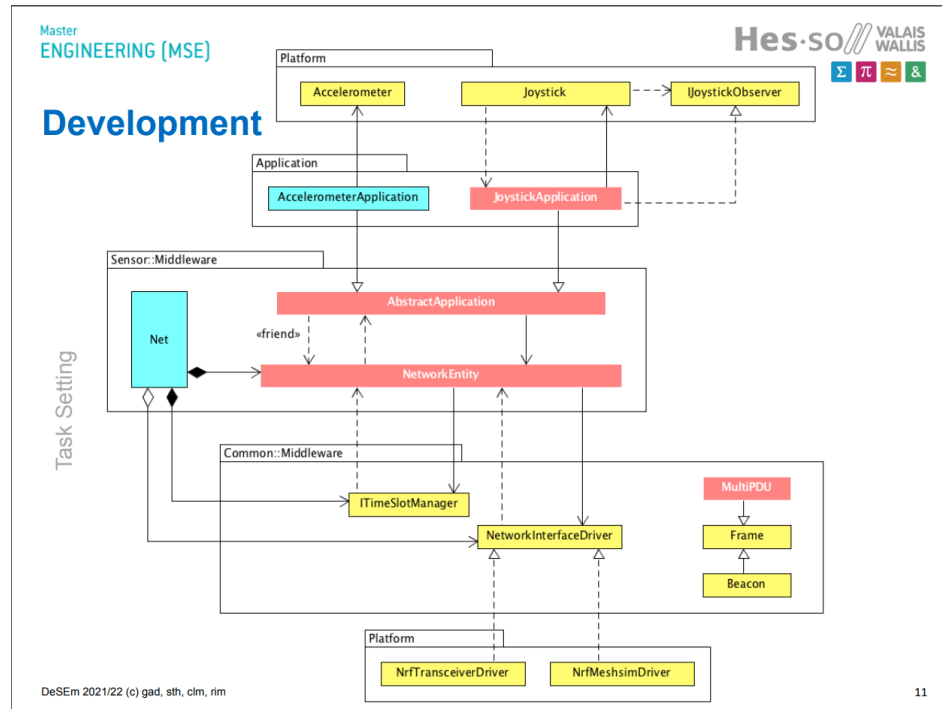


Figure 2.1: Aperçu des modifications à apporter

La figure suivante nous a été présentée afin d'illustrer les classes que nous devons compléter/créer. Les classes à créer sont les suivantes :

1. JoystickApplication (.h et .cpp)
2. MPDU (.h et .cpp)

Alors que les classes à compléter sont :

1. AbstractApplication (.h et .cpp)
2. NetworkEntity (.h et .cpp)

D'autres fichiers ont cependant également été modifiés comme par exemple Factory.cpp afin d'y ajouter l'initialisation du Joystick ainsi que le addObserver().

3. Classes implémentées

3.1 MPDU

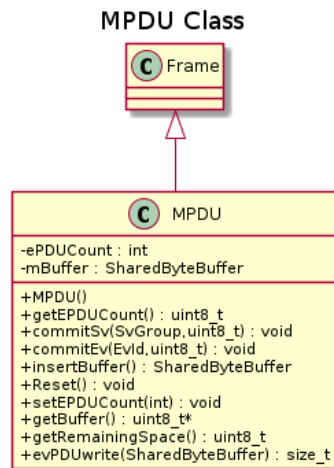


Figure 3.1: Diagramme de la classe MPDU

Comme on peut le voir sur cette figure, la classe MPDU hérite de la classe *Frame*. Ceci va permettre de réutiliser de nombreuses fonctions déjà implémentées dans celle-ci.

3.1.1 Méthodes implémentées

3.1.1.1 MPDU()

Cette méthode est le constructeur, elle permet d'initialiser également la classe parent *Frame* en lui passant en paramètre la variable *Mtu* (égale à 37) qui représente la taille maximale d'une Frame.

3.1.1.2 getEPDUCount()

Cette méthode permet d'obtenir le nombre de EPDU qui sont actuellement dans la Frame. La valeur de ce compte est également écrite dans la Frame en question.

3.1.1.3 commitSv(SvGroup,uint8_t)

Cette méthode permet de finaliser l'écriture d'une *SampleValue* (*Sv*) en ajoutant dans la Frame le header de celle-ci. Le header est composé de trois champs :

- count : la taille en bytes de la valeur
- svGroupOrEvId : Le numéro du groupe auquel l'application s'est inscrite ou l'Id de l'évènement (pour les valeurs event, voir plus tard)
- type : 0 pour sampleValue et 1 pour event

Les paramètres de cette fonction sont l'identifiant de groupe (type SvGroup) et le byteCount (type uint8_t).

3.1.1.4 commitEv(EvId,uint8_t)

Comme la précédente méthode, celle-ci se charge de finaliser l'écriture d'un event dans la trame. Son fonctionnement est identique hormis qu'elle inscrira dans le champ type du header la valeur 1. Les paramètres sont l'identifiant de l'évènement ainsi que le byteCount.

3.1.1.5 insertBuffer()

Retourne un proxy du buffer principal (issu de Frame) qui permet d'inscrire des valeurs dans le buffer du MPDU. Le début du buffer renvoyé par cette fonction varie en fonction du contenu du buffer du MPDU.

3.1.1.6 reset()

Cette méthode sert à réinitialiser le MPDU afin qu'il soit à nouveau prêt à être réécrit pour de nouvelles données.

3.1.1.7 setEPDUCount()

Cette méthode sert à écrire au bon endroit dans le buffer le compte actuel de EPDU contenus dans le buffer.

3.1.1.8 getBuffer()

Cette méthode fut utilisée dans networkentity au moment de transmettre le MPDU. Elle ne fait qu'appeler la méthode buffer() de la classe Frame mais celle-ci étant déclarée protected, elle ne peut être appelée depuis l'extérieur de la classe MPDU. La méthode getBuffer() sert donc à contourner ceci.

3.1.1.9 getRemainingSpace()

Cette méthode est utilisée lors de l'écriture des EvPDU dans le networkentity afin de connaître l'espace disponible restant dans le buffer du MPDU.

3.1.1.10 evPDUWrite(SharedByteBuffer)

Utilisée afin d'écrire les données d'un evPDU (ici une position du Joystick) dans le buffer du MPDU. Les evPDU étant basés sur un évènement et donc pas inscrits dans le publishArray, j'ai créé cette méthode afin d'inscrire la valeur de l'event dans le buffer.

3.1.2 Séquence générique d'écriture d'un MPDU

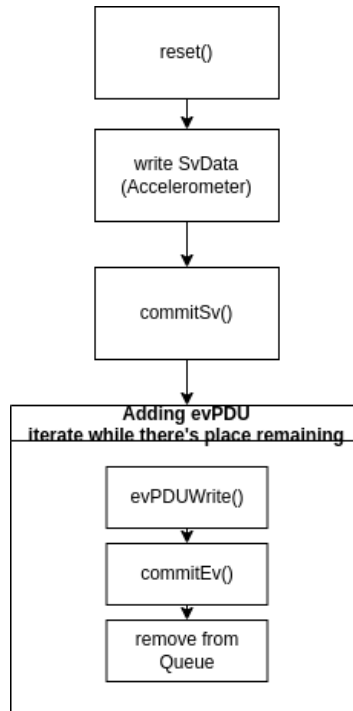


Figure 3.2: Séquence d'écriture du MPDU

Ci dessus sont les étapes suivies lors de l'écriture d'un MPDU, ces étapes utilisent les méthodes précédemment décrites de la classe MPDU et sont réalisées dans la méthode *onReceive()* de la classe *NetworkEntity*.

3.1.3 svPDU

Un svPDU représente une *Sample Value*, c'est-à-dire une valeur mesurée par un capteur (dans notre cas l'accéléromètre). Afin d'écrire la valeur dans le MPDU (étape 2 de la séquence de la figure 3.2), on vérifie que le beacon réceptionné demande bien les valeurs du groupe dans lequel notre mesure est effectuée. Si le beacon demande cette mesure, on va appeler la méthode *svPublishIndication* de l'application contenue dans le groupe souhaité (ici l'accéléromètre) en lui passant en paramètres un proxy du buffer du MPDU dans lequel cette méthode se chargera d'inscrire ses valeurs (exemple pour notre cas : la méthode *svPublishIndication* dans le fichier *AccelerometerApplication.cpp*). La méthode *commitSv()* permettra ensuite de finaliser l'écriture de cette sample value.

3.1.4 evPDU

Le deuxième type de valeurs reçues dans ce projet sont les valeurs reçues par évènement. Ces valeurs sont dans notre cas des positions de click sur le Joystick. Pour l'écriture de ces valeurs, le protocole prévoit d'ajouter autant de evPDU qu'il reste de place dans le buffer du MPDU. Lorsque soit la queue d'évènements est vide ou que le MPDU n'a plus de place, le MPDU est prêt à être transmit. Note : S'il reste des évènements dans la queue, celle-ci sera vidée lorsque le MPDU est plein, ceci peut engendrer

la perte de certains évènements mais permet également de ne pas "polluer" les prochains timeSlots et ainsi retarder la réception de certains évènements.

3.1.5 Transmission du MPDU

Le projet est défini sur un système de Time Slots, c'est-à-dire que chaque station a sa période durant laquelle elle peut transmettre des données au Gateway. Afin de transmettre le MPDU, on utilisera la méthode *onTimeSlotSignal()* de *NetworkEntity*. Dans cette méthode on vérifie que le signal du timeslot correspond bien à un start de notre timeSlot (OWN_SLOT_START) puis on appelle *transceiver().transmit(responseMPDU.getBuffer(),responseMPDU.length())* qui va transmettre au Gateway le MPDU construit selon les étapes ci-dessus.

3.2 JoystickApplication

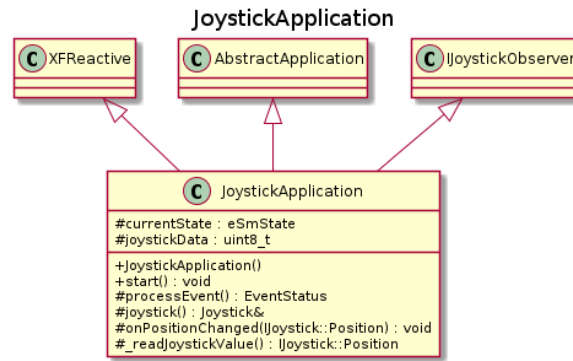


Figure 3.3: JoystickApplication UML Diagram

Cette classe fut construite en se basant sur la classe *AccelerometerApplication*, et hérite de *AbstractApplication* (classe abstraite de base pour les applications), de *IJoystickObserver* (afin de pouvoir utiliser la méthode *onPositionChanged()*), et de *XFRactive* (utilisé pour la machine d'état).

3.2.1 Propriétés

Les propriétés de cette classe sont *currentState* qui contient l'étape actuelle de la machine d'état de l'application et *joystickData* qui contient la valeur de la position du joystick.

3.2.2 Méthodes implémentées

3.2.2.1 JoystickApplication()

Le constructeur de cette classe permet d'initialiser ses propriétés. Il permet également de définir l'état initial de la machine.

3.2.2.2 start()

Cette méthode permet d'appeler la méthode *startBehavior()* héritée de *XFRactive* qui permet de démarrer la machine d'état via *XFRactive*.

3.2.2.3 processEvent()

Cette méthode est utilisée afin d'implémenter la machine d'état (qui sera décrite plus bas) et de traiter les évènements.

3.2.2.4 joystick()

Cette méthode sert à récupérer le singleton correspondant au Joystick() depuis la Factory.

3.2.2.5 onPositionChanged(IJoystick::Position)

Cette méthode héritée IJoystick permet de générer un évènement XFReactive de type *POSITION_CHANGED_EVENT* (voir explications de la machine d'état).

3.2.2.6 __readJoystickValue()

Cette méthode utilise l'instance de Joystick afin de récupérer sa position actuelle.

3.2.3 Machine d'état

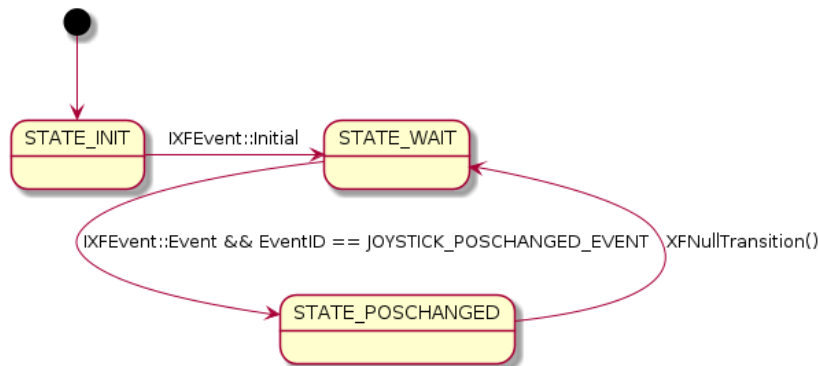


Figure 3.4: JoystickApplication machine d'état

La machine d'état implémentée pour la classe *JoystickApplication* est celle présentée dans la figure 3.4, lors de la réception d'un évènement de type *IXFEvent::Initial*, on passe dans l'état wait, puis, à la réception d'un évènement ayant comme Id *JOYSTICK_POSCHANGED_EVENT*, on passe dans l'état *STATE_POSCHANGED*. La machine d'état a été implémentée selon le principe du double switch vu en cours.

4. Diagramme de séquence de l'application

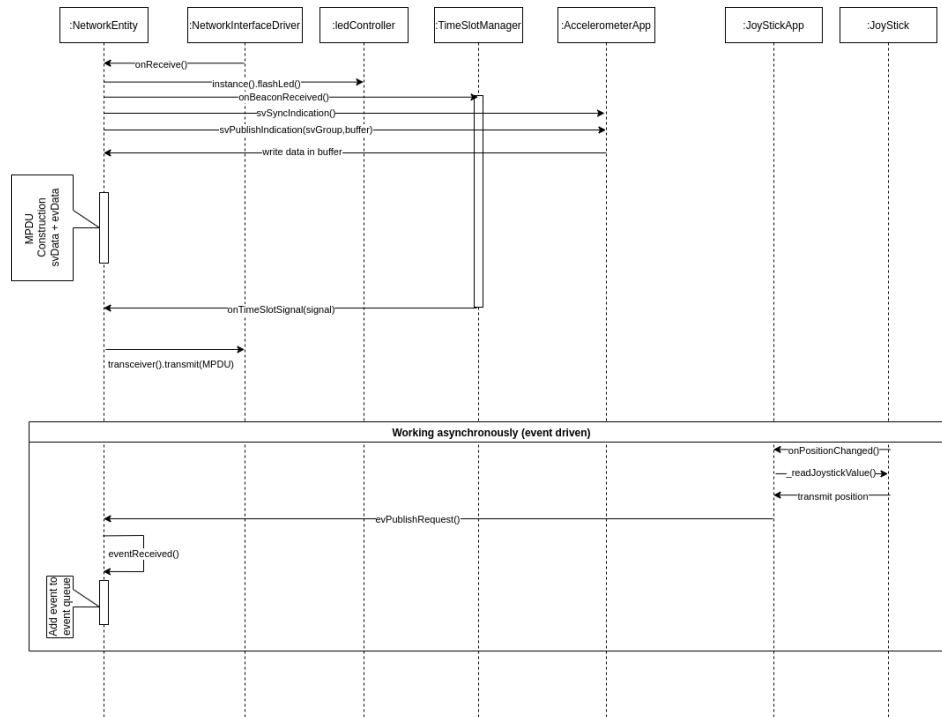


Figure 4.1: Diagramme de séquence générique de l'application

La figure 4.1 illustre la séquence appliquée lors de la réception d'un beacon. On peut voir qu'une partie est exécutée de manière asynchrone et n'est pas soumise à un enregistrement pour diverses indications. Il en est ainsi car les valeurs du Joystick sont gérées de manière événementielle et comme on peut le voir sur le diagramme, lorsque la position du joystick change, la joystickApplication lit la nouvelle valeur puis transmet directement au NetworkEntity qui va stocker l'information dans une queue. C'est seulement au moment où le NetworkEntity aura terminée de recevoir la sample value qu'il va placer dans le MPDU autant de données d'événements qu'il y a de place restante dans celui-ci. La partie supérieure du diagramme illustre la synchronisation et la collecte de sample value à chaque réception d'un beacon.

4.1 Diagrammes de séquence détaillés

4.1.1 Joystick

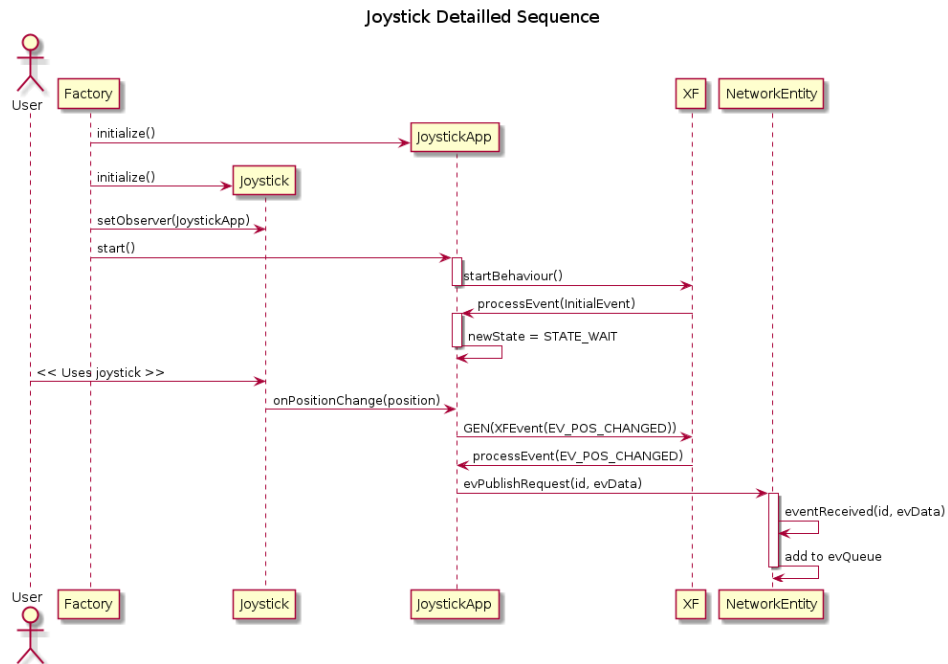


Figure 4.2: Diagramme de séquence détaillé du joystick

4.1.1.1 Création et écriture MPDU

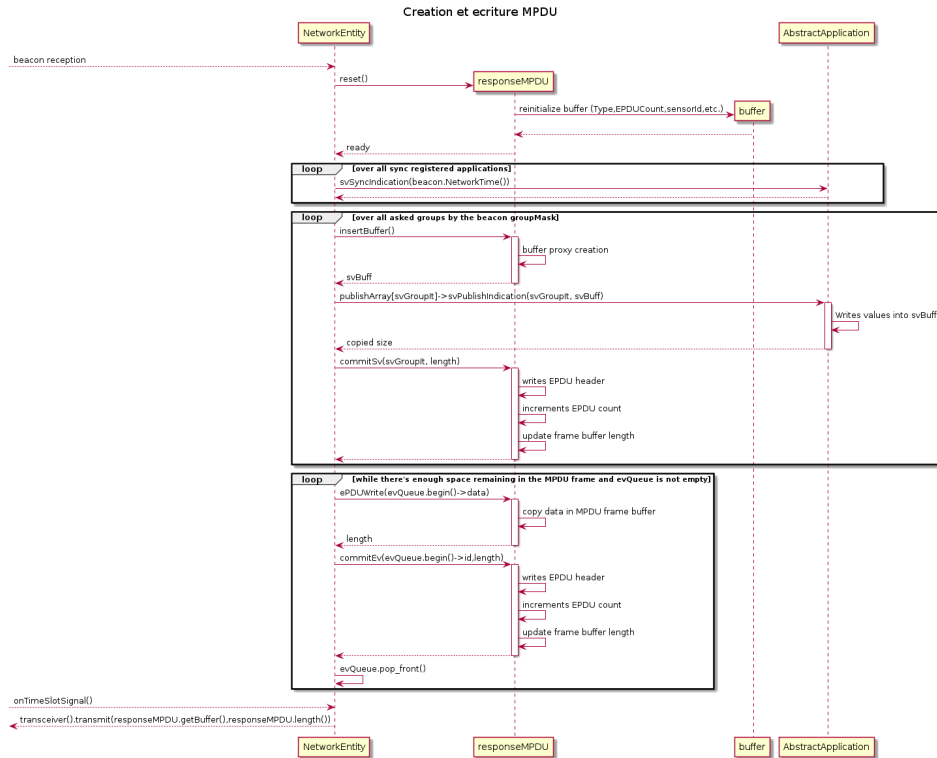



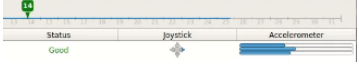
Figure 4.3: Diagramme de séquence détaillé du MPDU

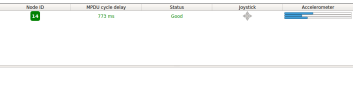
5. Tests

Afin d'effectuer des tests sur le programme nous avons à disposition deux possibilités :

1. Une simulation
2. Une carte de test avec un Gateway

Les tests effectués sont donc les suivants :

Description	Objectif	Résultat	Preuve	Remarque
Flash de la led à la réception du beacon	Contrôler le bon fonctionnement de la réception	OK		-
Affichage d'un message dans la console à la réception du beacon	idem	OK	<pre>-- Starting Desenet SENSOR -- -- Compiled: Jan 15 2022 10:36:21 -- Beacon Received ! Beacon Received ! Beacon Received !</pre>	-
Test appui simple (puis relâchement) sur un bouton du joystick pendant le cycle	Vérifier que la queue se remplit bien de deux événements	OK	<pre>-- Starting Desenet SENSOR -- -- Compiled: Jan 15 2022 10:36:21 -- Event data : 4 Event data : 1</pre>	5.1.1
Appui continu sans relâchement sur un bouton du joystick	Vérifier l'affichage de l'appui sur le bouton dans le test-Bench	OK		5.1.2
Appuis multiples sur le joystick	Observer comment la trame est remplie	OK	<pre>-- Starting Desenet SENSOR -- -- Compiled: Jan 15 2022 10:36:21 -- Event data : 4 Event data : 1 Event data : 4 Event data : 1 Event data : 4 Event data : 1 Event data : 4 Event data : 1 Event data : 4 Event data : 1 Event data : 4 Event data : 1</pre>	5.1.3

Description	Objectif	Résultat	Preuve	Remarque
Déplacement de la fenêtre de simulation et modifications des données de l'accéléromètre	Observer le bon fonctionnement de la transmission des données de l'accéléromètre	OK	images dans les remarques	5.1.3.1
Observation des erreurs dans le test Bench	Vérifier qu'il n'y a pas d'erreurs dans le test bench afin de vérifier que la trame MPDU est bien construite selon les spécifications du protocole	OK		5.1.3.2
Déplacement rapide du sensor	Observer si la lecture des valeurs du sensor est au bon moment	OK	-	5.1.3.3
Comparaison de frames entre le sensor de démo et mon programme	Vérifier la justesse des frame transmises par mon programme	OK	voir remarques	5.1.3.4

Les mêmes tests ont été effectués sur la board distribuée en cours avec l'aide du Gateway également distribués et les résultats se sont avérés identiques. La seule différence fût qu'avec les tests physiques il arrivait parfois que le timeSlot ne soit pas parfaitement respecté. J'en ai tiré pour conclusion que cela pouvait être dû à d'éventuelles perturbations sur le wifi ou des problèmes de temps de calcul sur les diverses stations.

5.1 Remarques sur les tests

5.1.1 Test appui simple

A l'affichage sur le testBench, on ne voit pas la pression sur le bouton car le testBench affiche tous les évènements rapidement et le dernier étant un relachement, nous n'avons pas le temps de voir le clignotement du bouton. Cependant on voit dans la preuve qu'il y a bien eu deux eventData qui correspondent à un appui à droite puis un état sans appui du joystick.

5.1.2 Appui continu

On observe ici que le testBench affiche bien un appui sur le bouton droit du joystick car le dernier (et seul) évènement reçu correspond à ceci. On observe aussi que l'affichage éteint le bouton au beacon suivant le relâchement du click.

5.1.3 Appuis multiples

Ce test permet d'observer qu'il y a bien plusieurs évènements qui peuvent être inscrits dans le MPDU. Cependant, à cause du choix de vider la Queue à la fin de l'écriture du MPDU (voir [3.1.4](#)), s'il y avait trop d'évènements ceux-ci seront perdus.

5.1.3.1 Déplacement de la fenêtre sensor

Pour des questions de visibilité dans le tableau, la position initiale ainsi que les valeurs de l'accéléromètre sont disponibles ici :

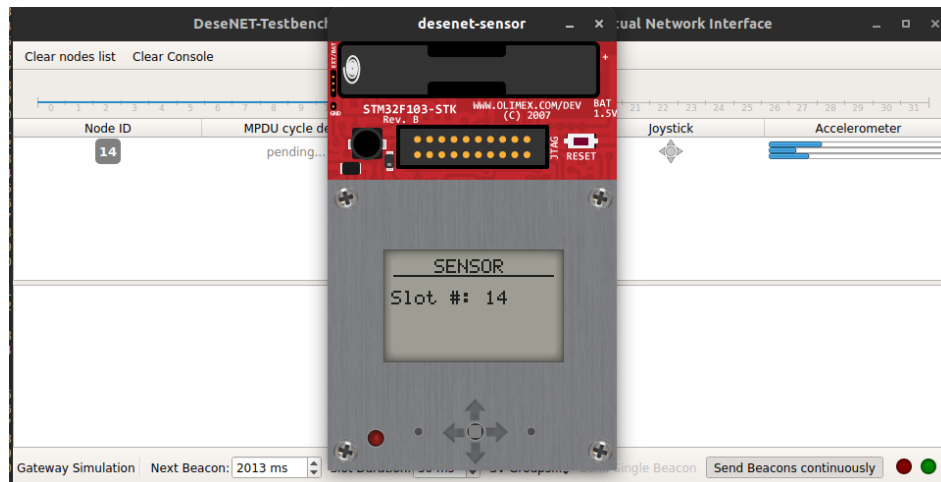


Figure 5.1: Position initiale du sensor et valeurs d'accéléromètre

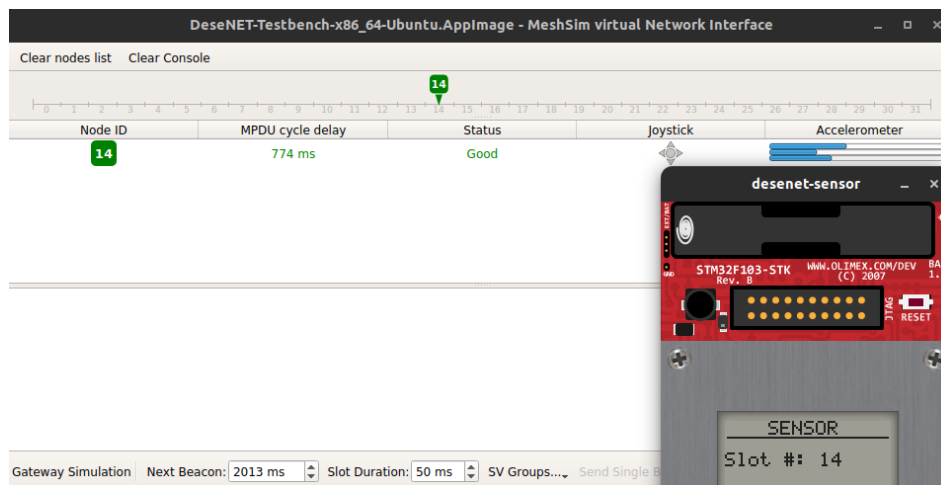


Figure 5.2: Position modifiée du sensor et valeurs d'accéléromètre

On voit donc bien dans ces images qu'après déplacement de la fenêtre sensor, les valeurs ont changé. Ceci confirme donc le bon fonctionnement du programme.

5.1.3.2 Affichage d'erreurs dans le test bench

On voit dans cette figure que le test bench ne présente aucune erreur, cela veut dire, en accordance avec les précédents tests, que la trame MPDU est bien construite (autant pour les svPDU que pour les evPDU) et bien lisible. Les headers sont donc correctement formés et les data bien transmises.

5.1.3.3 Déplacements rapides du sensor

J'ai pu observer que si après le timeslot on déplace le sensor puis toujours avant le prochain beacon on le remet en place la valeur transmise ne change pas. C'est pour moi un comportement correct car les valeurs sont lues et transmises à l'arrivée du beacon.

5.1.3.4 Comparaison des frames entre mon programme et le sensor de démo





Frame Capture							
Delta Time			Framesize	Time on Network	Destination address	Source address	Frame data
0			20	0.000 s	C7C7C7C7	C7C7C7C7	AAC7C7C7C70C00BFF40D03DD07003200FFFFBC4A
0.754 s			17	0.002 s	E2E2E2E2	E2E2E2E2	AAE2E2E2E2098E01167DF9E7F9B2F9711E

Figure 5.3: Frames générées par le programme développé





Frame Capture							
Delta Time			Framesize	Time on Network	Destination address	Source address	Frame data
0			20	0.000 s	C7C7C7C7	C7C7C7C7	AAC7C7C7C70C00A0D90E03DD07003200FFFF0036
0.754 s			17	0.002 s	E2E2E2E2	E2E2E2E2	AAE2E2E2E2098E01167DF9E7F9B2F9711E

Figure 5.4: Frames générées par le sensor de démo

On peut vérifier à l'aide de ces figures le bon remplissage des frame du MPDU car les tailles sont identiques entre les valeurs transmises par le sensor de démo et celles transmises par mon programme. La seule différence ici est au niveau des valeurs transmises car en réalisant ce test avec la simulation, les deux fenêtre de sensor ne se trouvaient pas au même endroit et donc les valeurs d'accéléromètre étaient différentes.

6. Conclusion

En conclusion, le programme est fonctionnel et permet bien de remplir les MPDU, ainsi que de les transmettre à la station principale en respectant les spécifications du protocole DeseNET. Les principales difficultés ont été pour ma part de bien rentrer dans le protocole et d'en comprendre tous les fonctionnements. La classe MPDU a donc été la plus complexe à réaliser mais une fois ceci compris, la suite s'est bien déroulé et lors de l'implémentation sur la carte STM, je n'ai rencontré aucun problème et le système semblait toujours fonctionner. Comme cité ci-dessus, les seuls défauts de fonctionnement rencontrés furent parfois le timeSlot qui n'était pas parfaitement respecté ou des petits problèmes matériels comme par exemple des rebonds sur les boutons du joystick qui génèrent donc parfois une légère perte d'informations.

Le choix de vider la queue d'évènement est un choix personnel qui fut pris en discutant avec certains de mes camarades. En effet, nous avons jugé plus important de transmettre uniquement les données générées durant le time slot plutôt que de garder indéfiniment et provoquer des potentiels retards dans la transmission d'informations. Par exemple, si la queue est maintenue, des informations générées durant un timeslot pourraient n'être transmises uniquement 2 timeslot plus tard.