MSE | MASTER OF SCIENCE IN ENGINEERING

Hes·so

Projet DeseNET

Département : EIE
Unité d'enseignement : MA_DeSEm

Auteur Sébastien Deriaz
Professeurs Medard Rieder
Assistants Thomas Sterren

Date 11 janvier 2022

Table des matières

1 Introduction

L'objectif de ce travail est de mettre en œuvre le protocol DeseNET sur une carte STM32 Nucleo. Le protocole a été rédigé par M. Clausen et nous a été fourni pour ce travail.
Un projet de base nous a été fournit et il conviendra de le remplir pour implémenter les différentes fonctions manquantes.
Le projet sera séparé en deux parties

1. Simulation : Test du programme dans un environnement simulé
2. Implémentation sur la carte : Test du programme sur la carte Nucleo équipée d'un module de communication sans-fil

2 Fichiers crées

1. `joystickapplication.h` et `joystickapplication.cpp` : Machine d'état du joystick et polling de l'état
2. `mpdu.h` et `mpdu.cpp` : Classe héritée de `frame` qui implémente les méthodes pour créer une frame MPDU (stockage des événements / sample values et headers spécifiques).

3 Modifications apportées aux fichiers

1. `platform_config.h` (`ide-stm32cubeide\platform\nucleo-stm321476rg\platform-config.h`)
   (a) Changement du `DESENET_SLOT_NUMBER`
2. `factory.h` (`src\app`)
   (a) Ajout du joystick (plus include pour `joystickapplication.h`)
3. `factory.cpp` (`src\app`)
   (a) Include de `joystick.h`
   (b) Initialisation du joystick (+ `setObserver`)
   (c) Start Joystick / joystickApplication
   (d) méthodes `joystickApplication()` et `joystick()`
4. `abstractapplication.cpp` (`src\common\mdw\desenet\sensor`)
   (a) Ajout de `subscribeToSvGroup()` dans `svPublishRequest`
   (b) Ajout de `eventReceived()` dans `evPublishRequest`
5. `abstractapplication.h` (`src\common\mdw\desenet\sensor`)
   (a) Ajout `const` a l'event id
6. `net.cpp` (`src\common\mdw\desenet\sensor`)
   (a) Ajout de `slotNumber` à l'initialisation de `networkentity`
7. `joystick.cpp` (`src\common\platform\nucleo-stm321476rg\board`)
   (a) Ajout du namespace `board`
8. `joystick.h` (`src\common\platform\nucleo-stm321476rg\board`)
   (a) Ajout du namespace `board`
9. `joystick.cpp` (`src\common\platform\qt-meshsim\board`)
   (a) Ajout du namespace `board`
10. `joystick.h` (`src\common\platform\qt-meshsim\board`)
    (a) Ajout du namespace `board`
11. `networkentity.h` (`src\common\mdw\desenet\sensor`)
    (a) Ajout de l'include pour `mpdu.h`
    (b) Héritage de `ITimeSlotManager` + Ajout de la méthode virtuelle `onTimeSlotSignal`
    (c) Ajout de paramètre `slotNumber` à initalize
    (d) Ajout des méthodes `subscribeToSvGroup` et `unsubscribe`
    (e) Ajout de la méthode `eventReceived`
    (f) Liste des applications et struct `AppBind`
12. `networkentity.cpp` (`src\common\mdw\desenet\sensor`)
    (a) Ajout de `slotNumber` dans l'initialisation
    (b) Initialisation des relations de `pTimeSlotManager`
    (c) Complétion de la méthode `onReceive`
    (d) Ajout de la méthode virtuelle `onTimeSlotSignal`
    (e) Ajout des méthodes `subscribeToSvGroup` et `unsubscribe`
    (f) Ajout de la méthode `eventReceived`

4 Fonctionnement

La station de base (que l'on programme) doit être capable, lorsqu'elle reçoit une trame spéciale (beacon), de renvoyer les données qu'elle a collecté. Ces données sont :

1. Événements (par exemple un mouvement sur le joystick)
2. État mesuré (par exemple l'accéléromètre embarqué)

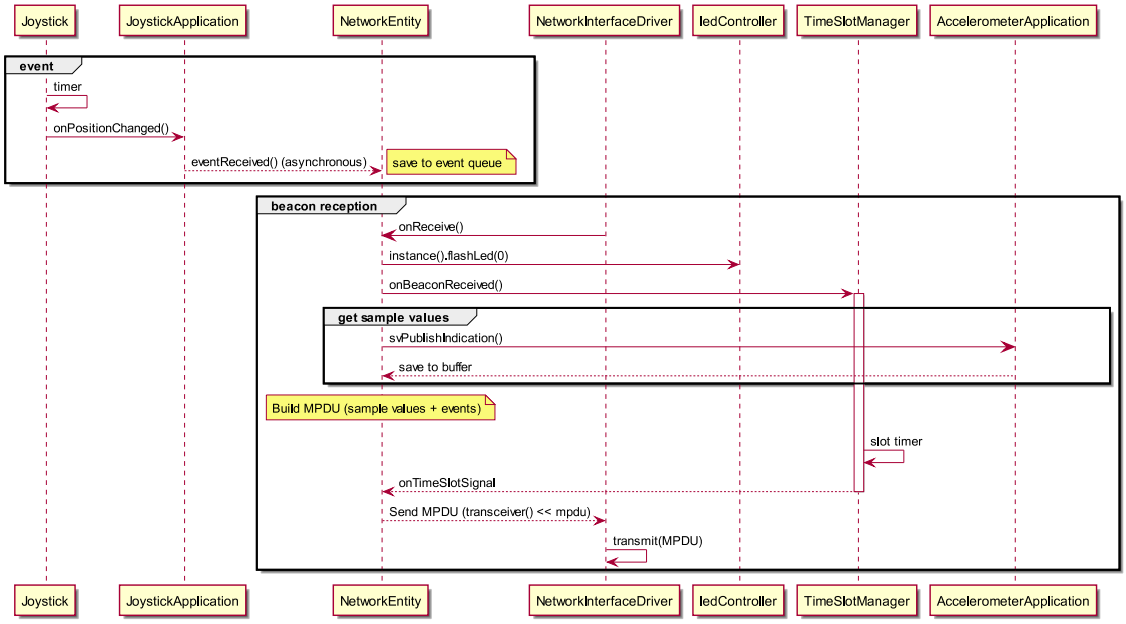


FIGURE 1 – Diagramme de séquence

4.1 Classe `MPDU`

Le classe MPDU permet de construire une frame contenant des données d'événements et/ou des "sampled values" (valeurs mesurées demandées par la station de base).

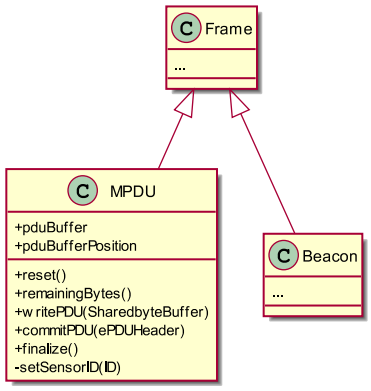
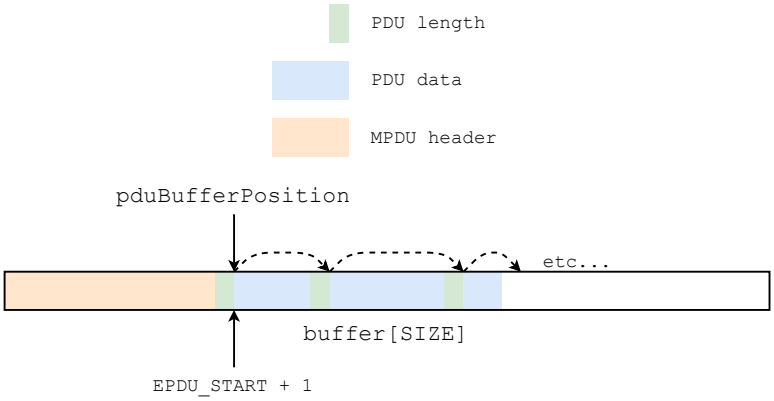


FIGURE 2 – Classe `MPDU`

Les informations des classes `Beacon` et `Frame` n'ont pas été notées car elles n'ont pas été modifiées.

4.1.1 Description des méthodes

1. `pduBuffer` : `SharedByteBuffer` qui permet l'écriture sur le buffer du MPDU. Le début de ce buffer avance continuellement à mesure que de nouvelles données sont écrites dans le MPDU.
2. `pduBufferPosition` : Position actuelle du `pduBuffer` à l'intérieur du "vrai" buffer. Ceci permet à une application d'écrire dans la frame au bon endroit



3. `reset()` permet de réinitialiser le MPDU (position dans le buffer, données, etc...)
4. `remainingBytes()` : donne une information sur la place restante pour écrire les données des applications
5. `writePDU()` : Écrire des données dans le buffer (à la position donnée par `pduBufferPosiiton`
6. `finalize()` : Conditionne le MPDU pour l'envoi final (headers)
7. `setSensorID()` : Enregistre le numéro de slot dans le MPDU (pour l'écrire du header MPDU)

La création d'un MPDU (pour l'envoi à la station de base) se fait de la manière suivante :

1. Reset
2. Ajout des sample values (accéléromètre)
3. Ajout des événements (jusqu'à ce qu'il n'y en ait plus ou que le buffer soit plein). Effectuer un "commit" après chaque événement
4. Finalisation

Une fois qu'un MPDU est prêt, il est envoyé au moment où le temps du slot est atteint (slot timer).

4.1.2 svPDU

Pour stocker les données mesurées, on commence par vérifier si le groupe de mesures (`svGroup`) est demandé dans le beacon.
Si c'est le cas, on appelle la méthode `svPublishIndication` de chaque application en passant un buffer. L'application va écrire les données dans ce buffer puis retourner le nombre de bytes écris. On appelle ensuite la méthode `commit` du MPDU qui va créer le header pour ces données. On recommence cette séquence pour chaque svGroup.

4.1.3 evPDU

Une fois que tous les svPDU sont stockés, on rempli le MPDU avec des evPDU (tant qu'il y en a ou jusqu'à ce que le MPDU soit plein). Pour cela on va lire une liste des événements (`eventsQueue`) et ajouter chaque élément dans l'ordre.
Si le MPDU est plein, on supprime les événements restants (il est possible de perdre des changements d'état sur le joystick).

4.1.4 Envoi

Lorsque le timeslot est atteint (Temps du slot × numéro de slot en millisecondes), on envoie le MPDU à la station de base.
On utilise la fonction `onTimeSlotSignal` en vérifiant que le signal émis est `OWN_SLOT_START`. Si cette vérification n'est pas faite, le MPDU sera envoyé plusieurs fois (lors d'autres signaux qui ne correspondent pas).

4.2 Classe `JoystickApplication`

La classe `JoystickApplication` a été copiée à partir de la classe `AccelerometerApplication`

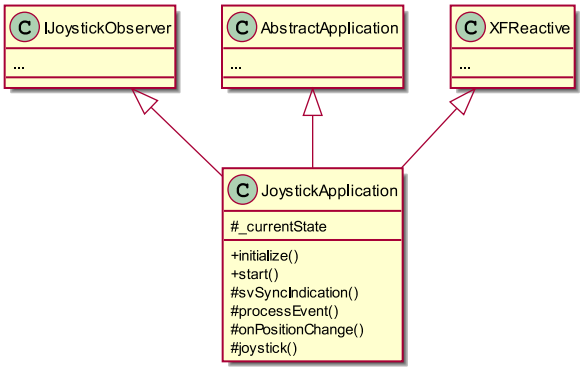


FIGURE 3 – Classe `JoystickApplication`

1. `_currentState` : état actuel de la machine d'état
2. `initialize()` : Initialisation de l'application (inscription vers le `networkentity` avec le bon numéro de groupe)
3. `start()` : démarrage de la machine d'état
4. `svSyncIndication()` : méthode appelée lorsque le beacon est reçu (pour la synchronisation des applications)
5. `processEvent()` : traitement des événements de la machine d'état
6. `onPositionChange()` : méthode appelée lorsqu'un mouvement est détecté sur le joystick (vient de `IJoystick`)
7. `joystick()` méthode permettant d'obtenir le singleton `joystick` (depuis la `Factory`)

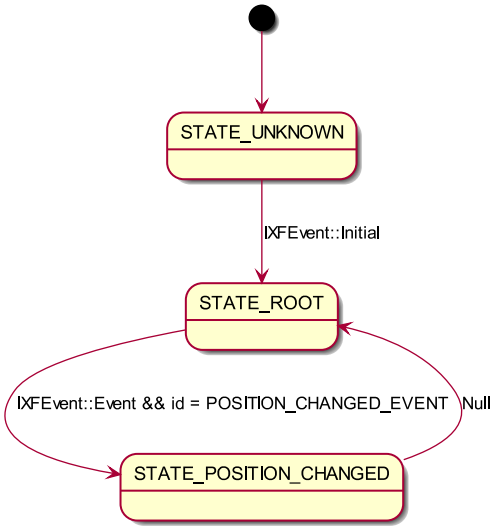


FIGURE 4 – Machine d'état de `JoystickApplication`

5 Tests

Les tests seront réalisés en deux parties

1. Test avec le simulateur
2. Test sur la carte

Le simulateur permet de trouver rapidement les premières erreurs et de mettre en place le fonctionnement du récepteur ainsi que l'envoi des trames.
Les tests sur la carte permettent de finaliser le système et de détecter les erreurs liées au système embarqué (au lieu du système simulé).
Les tableaux suivants montrent l'état du système dans sa version finale.

5.1 Simulation

Description	État	Remarque
Beacon		
Réception du beacon (envoi d'un message dans le port série lorsque le beacon est reçu)	✓	
Led allumée (clignotement de la led lorsque le beacon est reçu. Ce test permet de valider le précédent sans utiliser le port série)	✓	
Joystick		
Test de flanc (mettre le testbench en mode "continuous" et appuyer sur le bouton sans le relâcher). On devrait observer le bouton correspondant s'allumer puis s'éteindre au beacon suivant le relâchement)	✓	
Test de flanc sur tous les boutons (un seul à la fois)	✓	
Test d'appui puis relâchement dans un seul cycle (le bouton ne devrait pas s'allumer dans le testbench)	✓	
Valeurs correctes lors du remplissage de la queue d'événements (appuis à répétition sur le joystick lors d'un cycle)	✗	Comme la queue d'événements est effacée à chaque envoi des données, il est normal que des données soient perdues si trop d'événements ont étés enregistrés.
Accéléromètre		
Test en déplaçant la fenêtre en haut à droite et en haut à gauche (les valeurs de l'accéléromètre devraient changer et leurs valeurs doivent être répétables en fonction de la position de la fenêtre)	✓	
svPDU		
Formation correcte du header (vérifier si il n'y a pas d'erreurs dans le testbench)	✓	
Données correctes (affichage des données dans le testbench)	✓	
evPDU		
Formation correcte du header (vérifier si il n'y a pas d'erreurs dans le testbench)	✓	
Données correctes (affichage des données dans le testbench)	✓	
Transmission		
Aucune erreur lors de l'envoi d'un svPDU	✓	
Aucune erreur lors de l'envoi de svPDU et d'un ou plusieurs evPDU	✓	
Envoi des données au bon moment (timeslot)	✗	Erreurs occasionnelles due à la transmission entre les applications et au système d'exploitation

5.2 Carte

Description	État	Remarque
Beacon		
Réception du beacon (envoi de données dans le port série lorsque le beacon est reçu) [1]	✓	
Joystick		
Test de flanc (mettre le testbench en mode "continuous" et appuyer sur le bouton sans le relâcher). On devrait observer le bouton correspondant s'allumer puis s'éteindre au beacon suivant le relâchement)	✓	
Test de flanc sur tous les boutons (un seul à la fois)	✓	
Test d'appui puis relâchement dans un seul cycle (le bouton ne devrait pas s'allumer dans le testbench)	✓	
Valeurs correctes lors du remplissage de la queue d'événements (appuis à répétition sur le joystick lors d'un cycle)	✗	voir commentaire dans 5.1
Accéléromètre		
Positionnement de la carte sur son côté / à l'envers (les trois mesures doivent s'échanger les valeurs en fonction de l'orientation de la carte à condition qu'elle soit posée "droite")	✓	
Vérifier que les valeurs de l'accéléromètre sont cohérentes avec les autres cartes des collègues (toutes les mesures doivent être quasi-identiques à condition que les cartes soient orientées de la même manière)	✓	
svPDU		
Formation correcte du header	✓	Voir 5.1
Données correctes	✓	
evPDU		
Formation correcte du header	✓	
Données correctes	✓	
Transmission		
Aucune erreur lors de l'envoi d'un svPDU	✓	
Aucune erreur lors de l'envoi de svPDU et d'un ou plusieurs evPDU	✓	
Envoi des données au bon moment (timeslot)	✗	Erreurs occasionnelles dues à des pertes (perturbations du WiFi) de paquets et/ou des erreurs de timing dans la station de base et/ou les stations subordonnées.

1. Vérifié avec un envoi sur le port série

6 Conclusion

Le système final est fonctionnel, autant en simulation avec le testbench que sur la carte (test à l'école avec la station de base). Les seules erreurs rencontrées sont des problèmes de timing du slot (délais du système d'exploitation) ainsi que des pertes de données lors des tests sur les cartes. Les pertes sont probablement dues aux perturbations du WiFi qui fonctionne sur des fréquences similaires.
Le code réalisé est compilable sans warnings avec STM32CubeIDE et l'implémentation sur la carte n'as montré aucun défaut.