Montre connectée

Projet pédagogique Bluetooth Low Energy, STM32, application smartphone Xamarin

Culture Sciences de l'Ingénieur

Thomas MONGAILLARD

Édité le 17/06/2022

école normale supérieure paris-saclay

Ce sujet de mise en œuvre d'une connexion Bluetooth Low Energy entre une carte micro-contrôleur et un smartphone est à destination des enseignants en BUT GEII Option Électronique et Systèmes Embarqués dans le cadre d'un projet de Situation d'Apprentissage et d'Évaluation en 3ème année. Il est écrit par Thomas Mongaillard, candidat à l'agrégation externe SII Option Informatique Industrielle, en lien avec son dossier industriel. Ce texte a été relu par Anthony Juton, enseignant au Département d'Enseignement et de Recherche Nikola Tesla de l'ENS Paris-Saclay.

Cette ressource présente les différentes étapes permettant la mise en œuvre d'une connexion Bluetooth Low Energy entre un micro-contrôleur et un smartphone. Elle s'appuie sur un projet de montre connectée permettant la transmission des données de pulsation cardiaque vers un smartphone. Une première partie expose les concepts principaux de la norme Bluetooth Low Energy. Ensuite, une deuxième partie détaille les étapes pour la programmation du micro-contrôleur. Enfin, une troisième partie présente le développement d'une application Android avec l'outil Xamarin pour la réception des données sur smartphone.

1 – Présentation du projet

Le but de ce projet est de réaliser une montre connectée communiquant avec un smartphone via une liaison Bluetooth Low Energy (BLE). La montre connectée se compose des éléments suivants :

- Un écran LCD ;
- Un capteur de pulsation cardiaque ;
- Un micro-contrôleur ;
- Une antenne ;
- Une batterie.

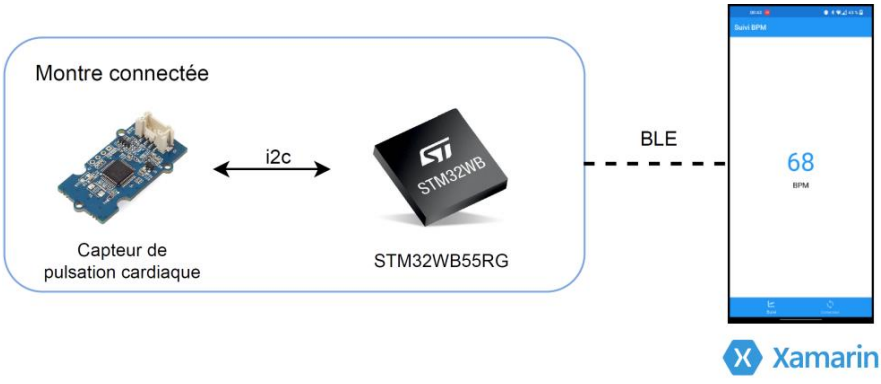

Figure 1 : Synoptique

Le système global doit pouvoir permettre d'effectuer les actions suivantes :

- Consulter l'heure sur la montre ;
- Consulter la pulsation cardiaque instantanée sur la montre ;
- *Consulter l'évolution sur une heure de la pulsation cardiaque sur le téléphone* ;
- Être alerté en cas de pulsation cardiaque anormale ;
- Consulter l'évolution entière de la pulsation cardiaque en ligne.

Dans ce sujet, on s'intéresse particulièrement à la liaison entre le smartphone et la montre et à la visualisation de l'évolution de la pulsation cardiaque sur le smartphone de l'utilisateur (action en italique). Deux parties sont donc à traiter : le programme BLE sur le micro-contrôleur et l'application mobile.

1.1 – Micro-contrôleur

On utilise la carte de développement *STM32 NUCLEO-WB55RG*[1] qui embarque le microcontrôleur STM32WB55RG[2] faisant partie de la gamme des micro-contrôleurs spécialisés pour les communications sans fils.

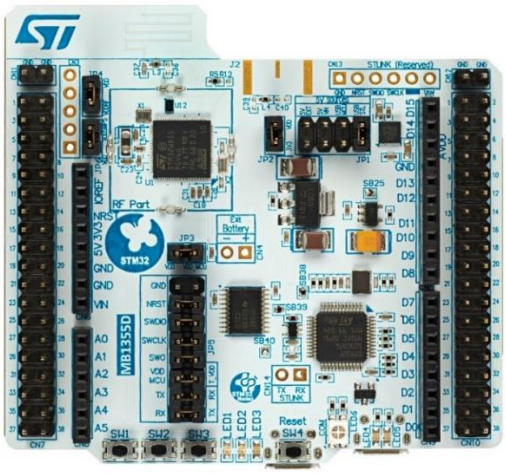

Figure 2 : Carte de développement Nucleo-WB55RG

Il intègre les normes Bluetooth 5.2 et ZigBee au sein d'un coprocesseur Arm® Cortex® M0+ couplé à un processeur Arm® Cortex® M4 sur lequel le programme hôte a lieu. Les fonctionnalités principales du STM32WB55RG sont :

- Émetteur/Récepteur Radio
- Modes d'ultra basse consommation
- Timers
- Mémoire Flash de 1 MB
- ADC 12 bits
- Protocoles USB, I2C, USART, SPI
- Algorithmes de chiffrement AES, RSA, Diffie-Helman.

La programmation de cette carte se fera par l'intermédiaire des outils de développement logiciels[3] [3] développés par ST (*cliquer sur les logos pour obtenir les pages de téléchargement*) :

- L'environnement de développement : STM32 Cube IDE
- Le générateur de code : STM32 Cube MX
- Le gestionnaire de programmation et monitoring : STM32 Cube Programmer

IDE MX Prg

[1] https://www.st.com/en/evaluation-tools/nucleo-wb55rg.html
[2] https://www.st.com/content/st_com/en/products/microcontrollers-microprocessors/stm32-32-bit-arm-cortex-mcus/stm32-wireless-mcus/stm32wb-series/stm32wbx5/stm32wb55rg.html
[3] https://www.st.com/content/st_com/en/products/development-tools/software-development-tools/stm32-software-development-tools.html

1.2 – Capteur de pulsation cardiaque

Le capteur choisi ici est le Heart Rate Sensor de chez Seeed basé sur le capteur optique PAH8001EI-2G du fabricant PXI. Il possède une interface I2C pour une communication simple avec le micro-contrôleur.

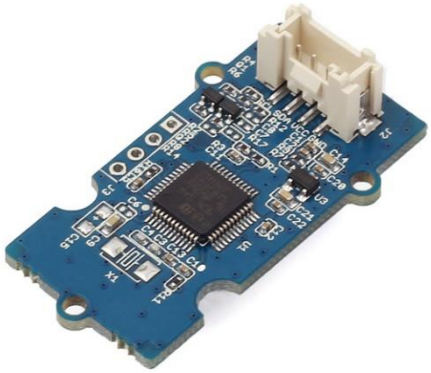

Figure 3 : Capteur de pulsation cardiaque

1.3 – Application Android et iOS

Pour pouvoir déployer une application mobile conjointement sur Android et iOS, on utilise l'outil Xamarin Forms développé par Microsoft et disponible sur l'environnement de développement Visual Studio. Les projets sont écrits en langage C#, un langage dérivé du C++ et très proche du Java. Xamarin propose un kit de développement écrit en C# (regroupant des éléments graphiques, des structures de navigation, etc) et s'occupe de transformer le code associé en deux projets Android et iOS à la compilation. Il est possible de gérer indépendamment les deux projets au niveau des icônes et des couleurs par exemple.

L'application mobile doit permettre les actions suivantes :

- Établir la connexion entre le téléphone et la montre ;
- Recevoir les données de pulsation cardiaque ;
- Afficher en temps réel l'évolution de la pulsation cardiaque de l'utilisateur ;
- Envoyer les données de pulsation cardiaque sur un serveur de base de données.

La mise en œuvre de la dernière action n'est pas détaillée dans cette ressource.

2 – Bluetooth Low Energy

Certaines des figures de cette sous-section proviennent du document *STM32WB BLE programming guidelines*[4], ainsi que du site de Bluetooth[5].

2.1 – Modélisation et séparation des fonctions

L'architecture BLE et son implémentation se décompose principalement en deux parties fonctionnelles : un contrôleur et un hôte. On représente la pile sous une forme de modèle en couche type OSI en figure 4.

[4] https://www.st.com/resource/en/programming_manual/pm0271-stm32wb-ble-stack-programming-guidelines-stmicroelectronics.pdf
[5] https://www.bluetooth.com/

Le contrôleur est composé de la liaison physique ainsi que du Link Layer. L'hôte gère la mise en forme des données et la gestion des connexions. On peut le décomposer par ses différentes fonctions : l'adaptation protocole (L2CAP), le security manager (SM), l'attribute protocol (ATT), le Generic Attribute profile (GATT) et le Generic Access Profile (GAP). Le contrôleur et l'hôte communiquent entre eux via l'interface HCI (Host Controller Interface). Ces différentes couches sont détaillées dans la suite.

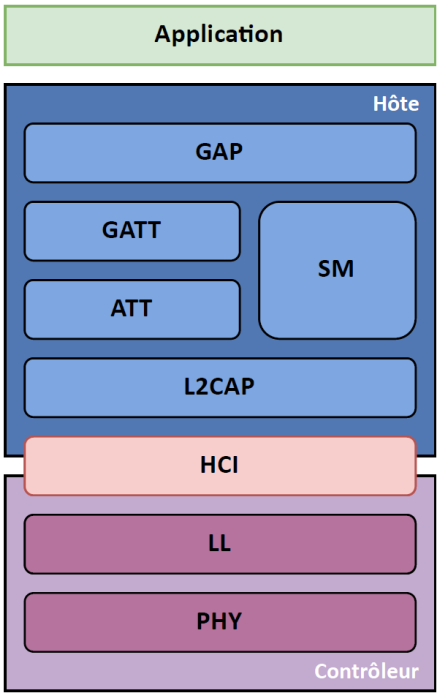

Figure 4 : Pile BLE

2.2 – Modes de communication

La norme Bluetooth Low Energy supporte plusieurs topologies de communication possibles présentées en figure 5.

Figure 5 : Topologies de communication

Le mode point à point est le mode le plus utilisé dans l'utilisation quotidienne. Ce mode est utilisé pour connecter deux appareils (une souris sans fil à un ordinateur par exemple). Le mode broadcast est utilisé dans des architectures plus grandes où certains appareils diffusent des données à d'autres appareils qui écoutent aux alentours. Il n'y a donc pas de connexion nécessaire entre les appareils et ce mode trouve ses applications dans les systèmes de mesures (un capteur diffuse sa mesure) ou dans la géolocalisation en intérieur (des balises indiquent leurs positions dans un bâtiment). Enfin, le réseau mesh est moins utilisé et plus récent que les autres. Dans cette topologie, chaque appareil est à la fois client et serveur de telle sorte que tous les appareils peuvent communiquer entre eux.

2.3 – Connexion et appairage dans le mode point à point

La connexion entre deux appareils BLE dans le mode point à point se réalise en plusieurs temps. On résume les interactions nécessaires à la connexion dans le diagramme de séquence en figure 6.

Ce mode met en jeu deux rôles :

- L'appareil *central* pouvant scanner autour de lui pour détecter des dispositifs BLE et initier une connexion ;
- L'appareil *peripherial* pouvant indiquer sa présence, accepter des connexions, mais pas les initier.

Ces rôles sont configurés dans les appareils à travers le *Generic Access Profile* (GAP). Ce profil permet aussi la configuration des options de connexion.

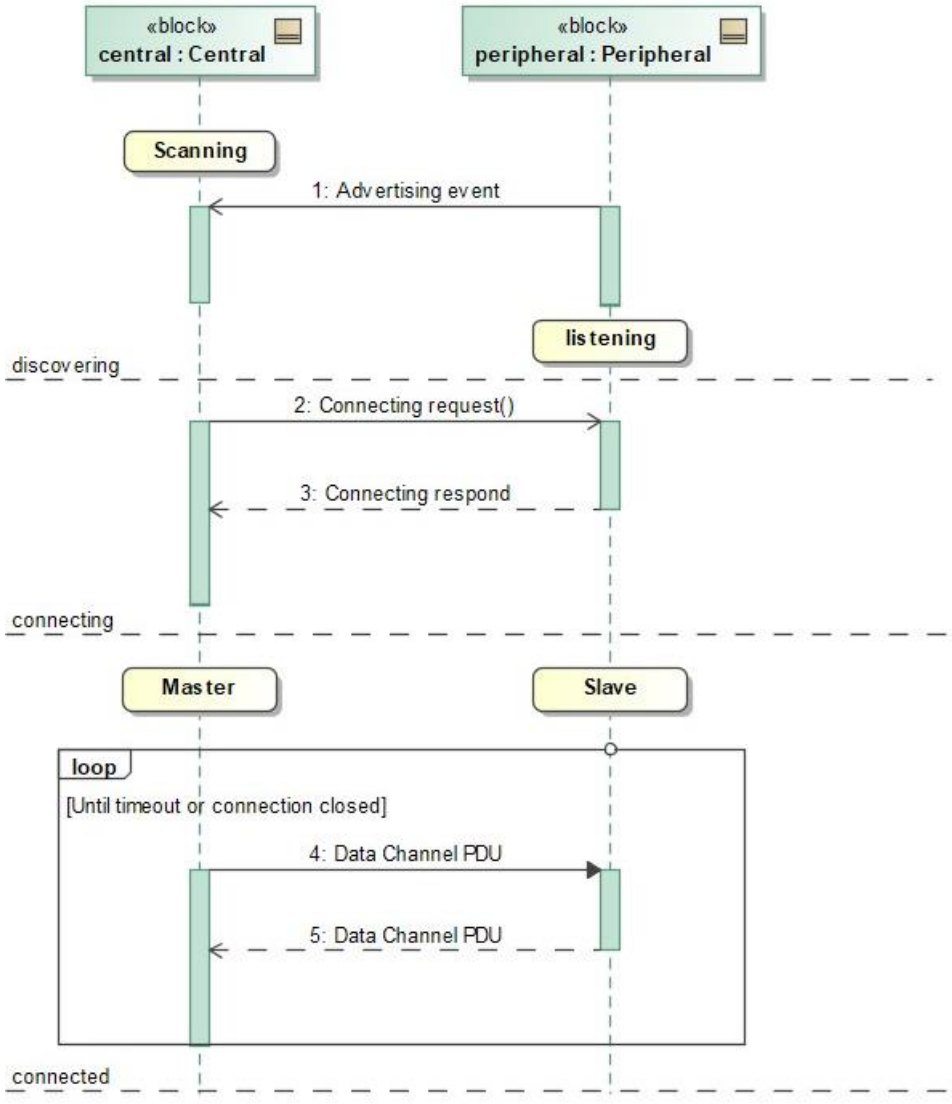

Figure 6 : Diagramme de séquence de connexion

Dans un premier temps, les deux appareils ne se connaissent pas. Un appareil *peripherial* qui cherche à communiquer va envoyer périodiquement des trames d'annonce (advertisment) qui contiennent des informations sur l'émetteur, sa fonction et sur la connexion. Après chaque émission, le périphérique va attendre une réponse avant d'émettre à nouveau. Après détection du message d'annonce et s'il y a volonté de se connecter, l'appareil *central* va envoyer une demande de connexion qui va être acquittée par le *peripherial*. On peut alors communiquer les informations

ou demandes d'informations souhaitées, et ce, jusqu'à ce que la connexion soit terminée. On parle alors de maître et d'esclave comme dans une liaison I2C.

2.4 – Statut d'un périphérique (Link Layer)

On peut définir le statut d'un périphérique par différents états qui relatent de la connexion. On présente le graphe d'état d'un appareil en figure 7.

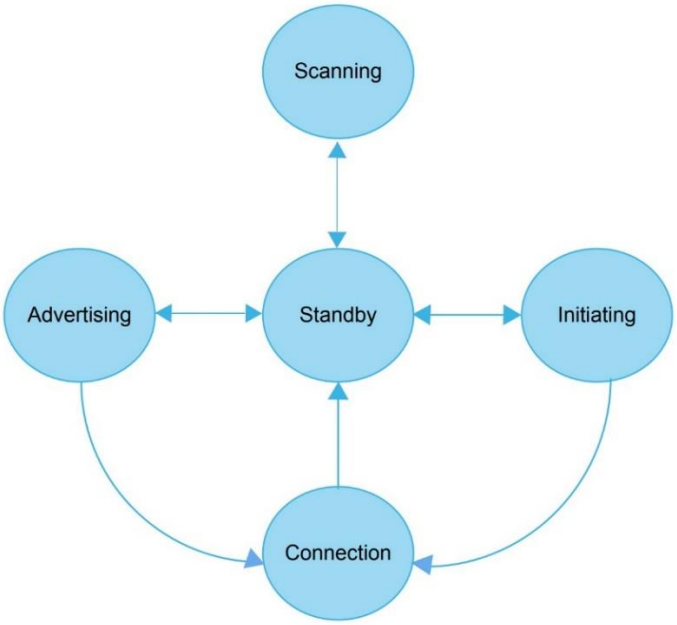

Figure 7 : Graphe d'état d'un périphérique

On explicite les différents états :

- Standby : traduit l'absence de flux de données entrant ou sortant ;
- Advertising : traduit la diffusion d'un message d'annonce ;
- Initiating : traduit l'initiation d'une connexion ;
- Connection : traduit l'échange de données en tant que maître/esclave ;
- Scanning : traduit l'attente et l'écoute de messages d'annonce.

2.5 - Paquets BLE

Les trames envoyées et reçues ont toujours la même structure, à savoir celle représentée sur la figure 8.

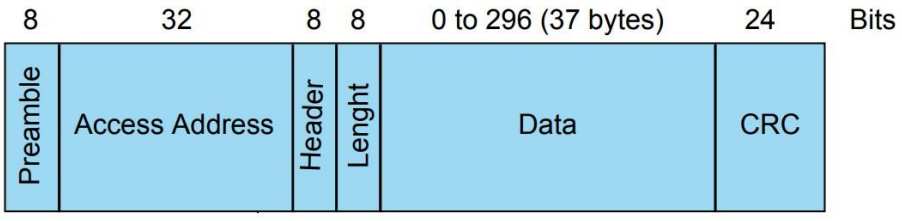

Figure 8 : Structure trame

On détaille les différents champs :

- Le champ *Preamble* est une séquence permettant la synchronisation radiofréquence ;
- Le champ *Access Address* est une adresse propre au contenu de la trame émise/demandée ;
- Le champ *Header* permet de définir le type de la trame (publicité, connexion, réponse à un scan, etc.) ;
- Le champ *Lenght* permet d'indiquer la quantité d'octets à lire ;

- Le champ *Data* contient les données transmises sous la forme d'un tableau d'octets ;
- Le champ *CRC* permet une protection des données contre les erreurs de bits lors de la transmission.

2.6 – Attribute Protocole

Lorsque l'on transmet des données en BLE, on utilise l'*Attribute protocole* (ATT). Ce protocole définit une structure pour la communication des données. Ces dernières prennent la forme d'attributs et sont définis par :

- Sa référence ;
- Son type défini par un Universally Unique IDentifier (UUID) ;
- Sa valeur ;
- Les permissions d'écriture et de lecture associées.

On appelle :

- Serveur l'appareil possédant les attributs à communiquer ;
- Client l'appareil demandant l'accès à des attributs.

Par exemple, si on donne l'accès à une température en lecture uniquement et sans sécurité, on aurait :

Attribute handle	Attribute type	Attribute value	Attribute permissions
0x0008	"Temperature UUID"	"Temperature Value"	"Read only, no authorization, no authentication"

Figure 9 : Exemple d'attribut

Protocol data unit (PDU message)	Sent by	Description
Request	Client	Client asks server (it always causes a response)
Response	Server	Server sends response to a request from a client
Command	Client	Client commands something to server (no response)
Notification	Server	Server notifies client of new value (no confirmation)
Indication	Server	Server indicates to client new value (it always causes a confirmation)
Confirmation	Client	Confirmation to an indication

Figure 10 : Description des PDU

Il est possible d'interagir de plusieurs manières avec un serveur. On définit alors plusieurs protocoles de communication sur les attributs nommés *Protocol Data Unit* (PDU) en figure 10.

On peut voir cette liaison client/serveur de la même manière qu'une base de données avec un client qui vient interagir avec des accès plus ou moins restreints à des tables de valeurs. Dans le message d'annonce transmis lors de la phase *advertising*, il est possible d'introduire un attribut représentant la fonction de l'appareil, permettant alors certains systèmes d'exploitation d'indiquer à l'utilisateur quel type d'objet a été scanné. Par exemple, ils affichent un logo de casque lorsqu'un casque Bluetooth est scanné.

2.7 – Generic Attribut Profile (GATT)

Afin d'utiliser plus simplement les attributs, on passe par une structure utilisant le protocole ATT. Cela permet de simplifier les échanges d'attributs. En effet, ATT ne concerne qu'une unité de

donnée alors que le GATT permet l'encapsulation de plusieurs attributs. Les GATT suivent également le format de la trame.

Il existe 3 types d'attributs :

- *Characteristic* : représente une donnée,
- *Descriptor* : décrit plus en détail une *characteristic*,
- *Service* : regroupe plusieurs *characteristics* pour un service particulier.

2.8 - Bluetooth Special Interest Group (SIG)

Il existe des spécifications/normes écrites par le Bluetooth SIG à suivre pour différentes applications qui sont des profils standards de Bluetooth. Les profils sont détaillés dans ces normes et comportent entre autres les rôles du GAP, les services présents, les paramètres de connexion. Les services eux-mêmes sont détaillés dans un document à part. Tous ces documents sont disponibles sur ce lien. Il existe un profil pour les dispositifs de mesure de pulsation cardiaque appelé *Heart Rate Profile* qui correspond parfaitement à notre application.

3 – Projet STM32

3.1 - Création d'un projet d'exemple pour WB55RG

L'outil CubeMX permet de générer un projet à partir des exemples fournis par ST. Pour notre projet, on va utiliser l'exemple BLE_HeartRate qui répond à la spécification du Bluetooth SIG mentionné précédemment. CubeMX peut générer l'exemple pour différents IDE, on utilisera ici CubeIDE. Les captures d'écran suivantes montrent les étapes à suivre :

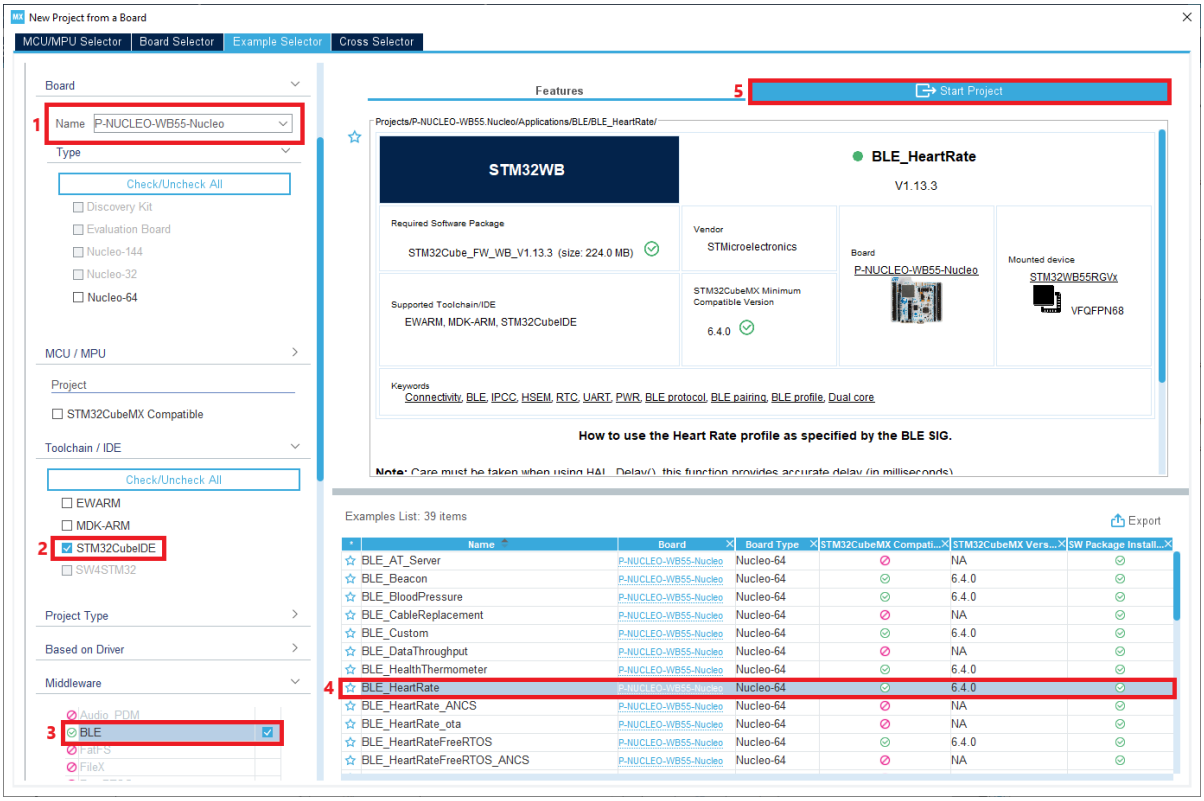

Figure 11 : Sélection de l'exemple

Le code a été généré, on lance alors CubeIDE. Pour importer le projet créé, aller dans **File** → **Import** → **General** → **Existing Projects into Workspace** puis sélectionner le dossier dans le lequel

a été créé le projet. Il est alors possible de sélectionner le projet BLE_HeartRate puis cliquer sur **Finish**.

Le fichier principal main.c se trouve dans **Application** → **User** → **Core**. L'en-tête de ce fichier précise qu'il est nécessaire que le Wireless Coprocessor soit flashé avec un certain fichier binaire grâce à CubeProgrammer.

3.2 – Flash du Wireless Coprocessor

Les fichiers binaires nécessaires pour mettre à jour le coprocesseur se trouvent sur cette page. Télécharger le paquet Patch-CubeWB puis le dézipper. Lancer à présent CubeProgrammer en ayant branché la carte de développement et suivre les étapes suivantes :

1. Déterminer quelle version du Firmware Upgrade Service (FUS) est installée (figure 14)
2. Mettre à jour le FUS :
    a. Cliquer sur Browse
    b. Depuis le dossier dézippé, aller dans Projects/STM32WB_Copro_Wireless_Binairies/STM32WB5x/ et ouvrir
        - stm32wb5x_FUS_fw_for_fus_0_5_3 si la version du FUS est 0.5.3
        - stm32wb5x_FUS_fw sinon
    c. Vérifier que le champ Start address vaut 0x080EC000
    d. Cliquer sur Firmware Upgrade
    En cas d'erreur, vérifier que le fichier ouvert est bien conforme à la version du FUS et que l'adresse de départ est correcte. En cas de version micro-contrôleur différente, consulter le fichier Release_Notes.html présent dans le même dossier que précédemment et déterminer la bonne adresse de départ.
3. Mettre à jour le stack BLE :
    a. Cliquer sur Browse
    b. Ouvrir le fichier stm32wb5x_BLE_Stack_full_fw dans le même dossier que précédemment
    c. Vérifier que le champ Start address vaut 0x080C7000
    d. Cliquer sur Firmware Upgrade

Le coprocesseur a été mis à jour, on va pouvoir exécuter l'exemple sur la carte.

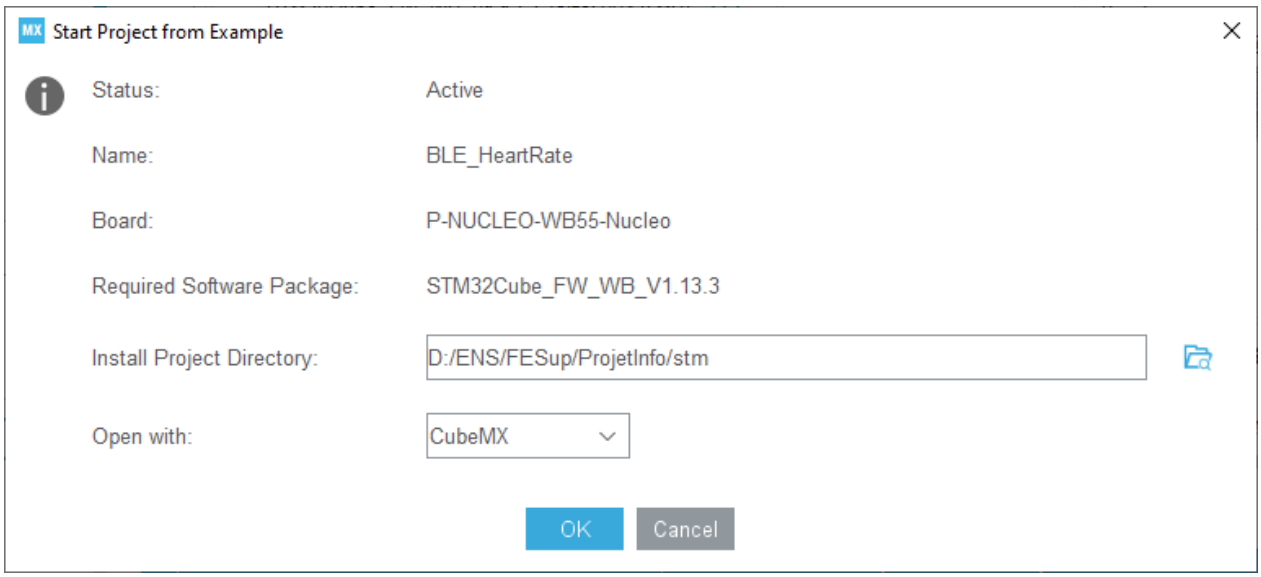

Figure 12 : Création du projet

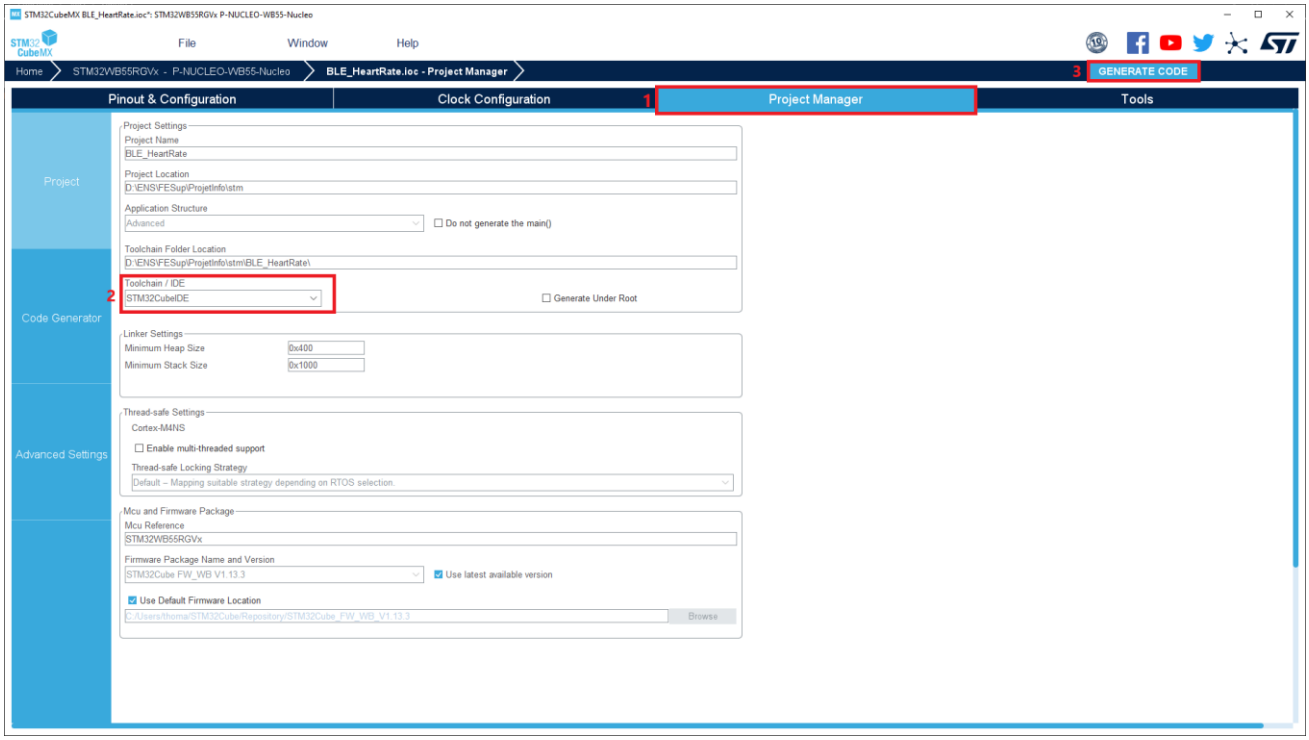

Figure 13 : Génération du code

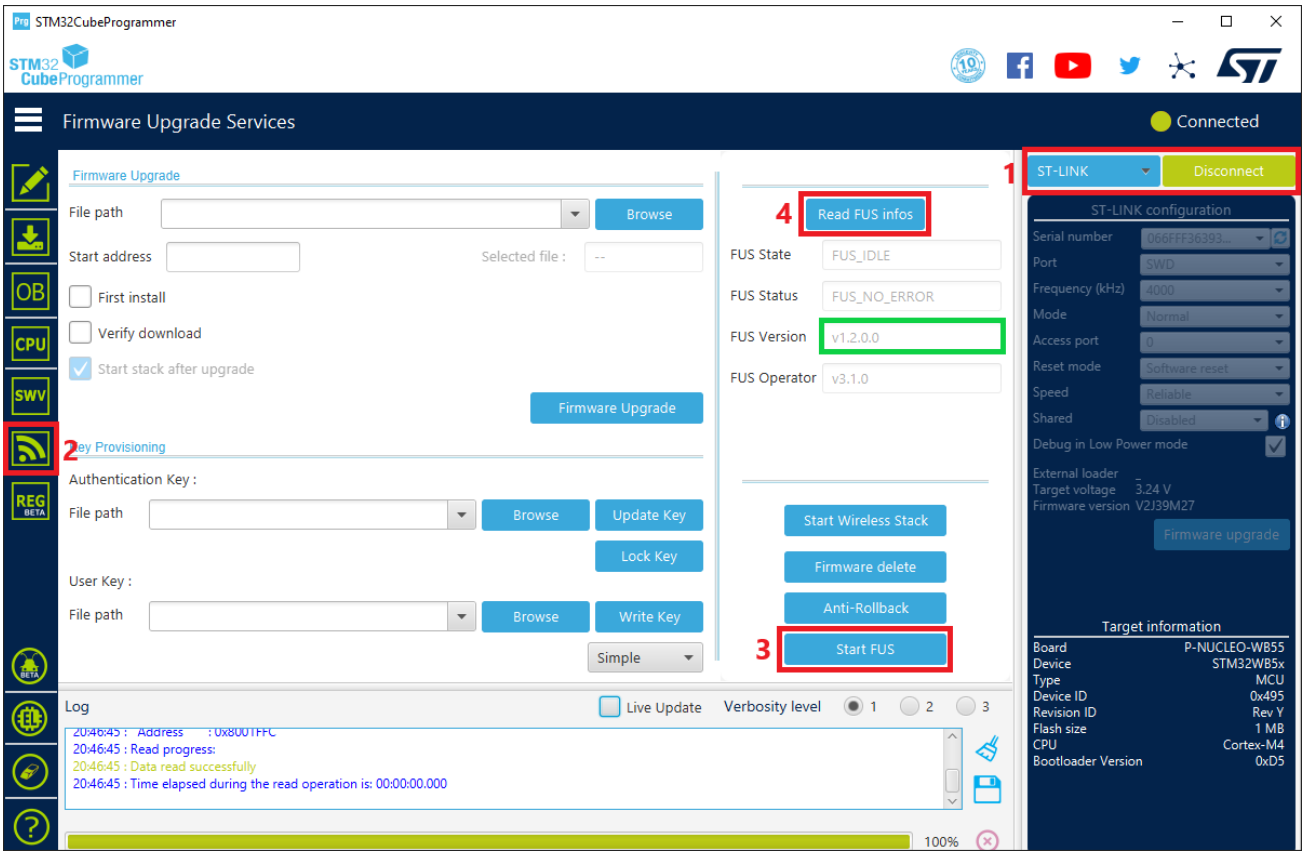

Figure 14 : Version du FUS

3.3 - Analyse de l'exemple

3.3.1 - Exécution du code

Par défaut, le débogage de la partie BLE n'est pas activé. Pour l'activer, double-cliquer sur le fichier BLE_HeartRate.ioc dans l'explorateur de projet à gauche puis suivre les étapes de la capture de la figure 15. Une fois terminé, enregistrer le fichier .ioc (Ctrl+S) pour regénérer le code.

On peut maintenant lancer le build (Ctrl+B) puis le débogage (F11). Le code étant téléversé sur la carte, on peut vérifier son bon fonctionnement avec l'application mobile *ST BLE Profile* (figure 17).

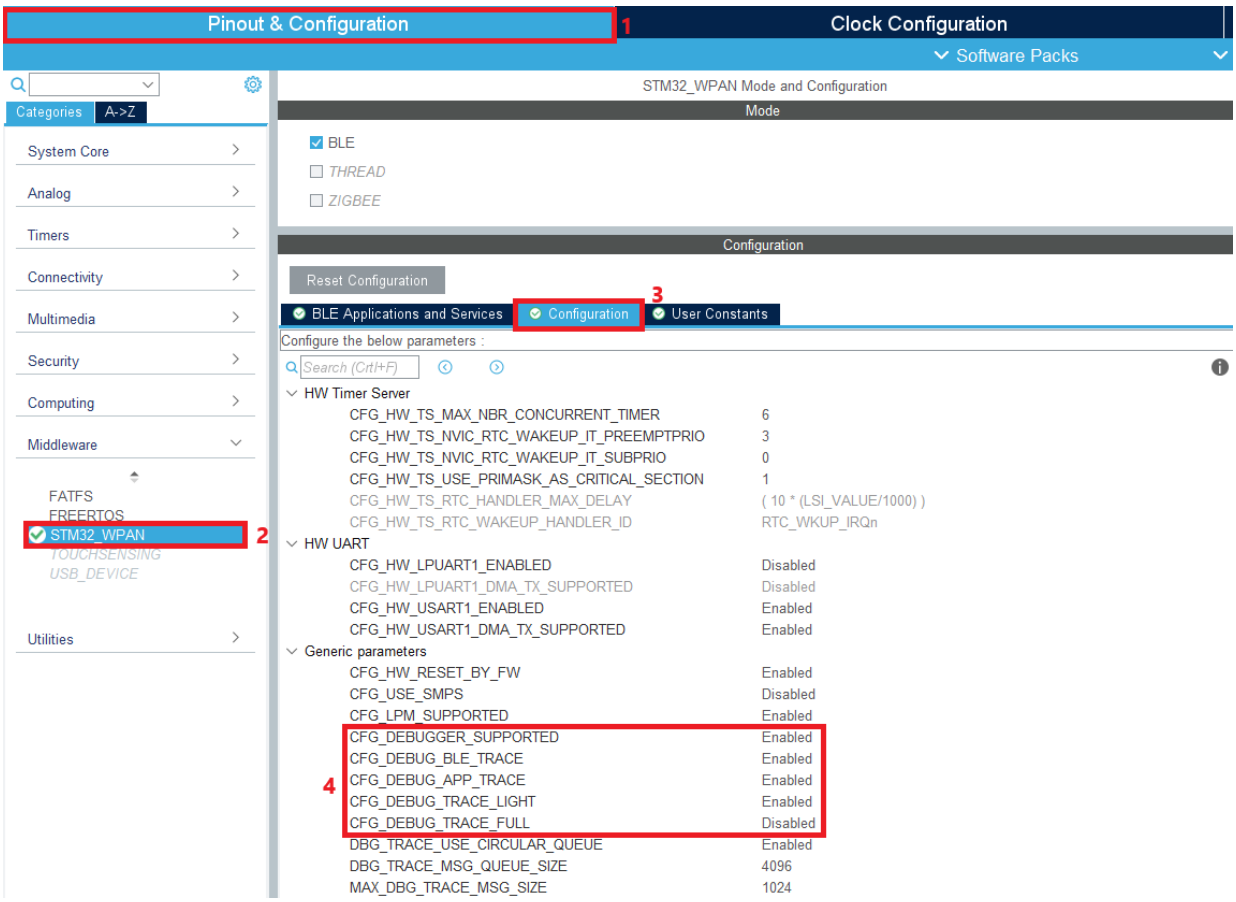

Figure 15 : Activation du débogage

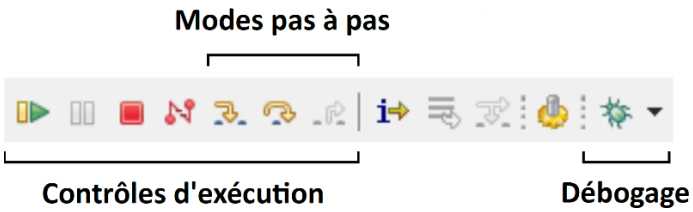

Figure 16 : Barre d'outils CubeIDE

On peut voir qu'une fois la connexion effectuée, on reçoit à intervalle régulier court (< 1s) une valeur de pulsation cardiaque en battement par minute. Il n'y a pas besoin de demander depuis l'application une nouvelle mesure puisque c'est le micro-contrôleur qui notifie le téléphone à chaque nouvelle mesure. On remarque sur la figure 17 que l'on obtient une courbe de pulsation cardiaque périodique alors qu'aucun capteur n'a encore été branché. Il s'agit donc de mesures arbitraires générées par l'exemple pour montrer le fonctionnement. Il s'agira par la suite de modifier le code associé pour le remplacer par la mesure réelle provenant du capteur de pulsation cardiaque.

3.3.2 - Configuration BLE

Le fichier qui permet de configurer la communication est Application/User/STM32_WPAN/app_ble.c.

La fonction principale est APP_BLE_Init dans laquelle on s'intéresse aux fonctions suivantes :

- Ble_Hci_Gap_Gatt_Init : initialise les couches HCI, GAP et GATT ;
- SVCCTL_Init : initialise les profils Bluetooth ;
- HRSAPP_Init : initialise les actions associées au profil Heart Rate.

Pour rentrer dans le code d'une fonction, utiliser le raccourci F3.

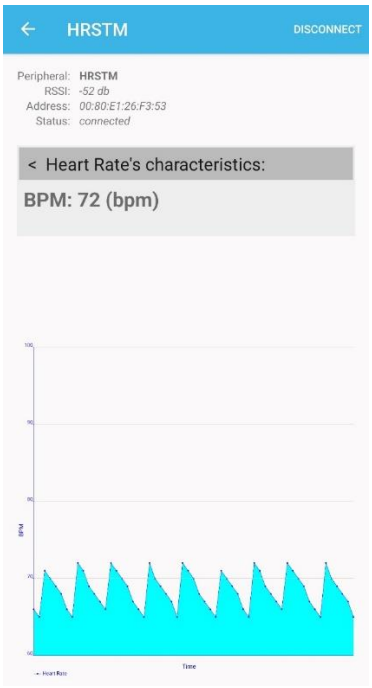

Figure 17 : Application ST BLE Profile

Initialisation GAP Dans Ble_Hci_Gap_Gatt_Init, on définit le rôle du GAP du côté du capteur, à savoir le rôle Peripherial puisqu'il est l'appareil avec le moins de ressources (calcul et autonomie).

On définit en plus le nom du capteur HRSTM qui est lu lors du scan par le téléphone ainsi que l'apparence (figure 18).

```
  /**
   * Initialize GAP interface
   */
  role = 0;

#if (BLE_CFG_PERIPHERAL == 1)
  role |= GAP_PERIPHERAL_ROLE;
#endif

#if (BLE_CFG_CENTRAL == 1)
  role |= GAP_CENTRAL_ROLE;
#endif

  if (role > 0)
  {
    const char *name = "HRSTM";
    aci_gap_init(role,
#if ((CFG_BLE_ADDRESS_TYPE == RESOLVABLE_PRIVATE_ADDR) || (CFG_BLE_ADDRESS_TYPE == NON_RESOLVABLE_PRIVATE_ADDR))
                 2,
#else
                 0,
#endif
                 APPBLE_GAP_DEVICE_NAME_LENGTH,
                 &gap_service_handle,
                 &gap_dev_name_char_handle,
                 &gap_appearance_char_handle);

    if (aci_gatt_update_char_value(gap_service_handle, gap_dev_name_char_handle, 0, strlen(name), (uint8_t *) name))
    {
      BLE_DBG_SVCCTL_MSG("Device Name aci_gatt_update_char_value failed.\n");
    }
  }

  if(aci_gatt_update_char_value(gap_service_handle,
                                gap_appearance_char_handle,
                                0,
                                2,
                                (uint8_t *)&appearance))
  {
    BLE_DBG_SVCCTL_MSG("Appearance aci_gatt_update_char_value failed.\n");
  }
```

Figure 18 : Initialisation du GAP

GATT Services Les deux services Bluetooth associés au profil Heart Rate sont le Heart Rate Service et le Device Information Service. Celui qui nous intéresse ici est le Heart Rate Service dont la déclaration des attributs est faite dans le fichier Middlewares/STM32_WPAN/hrs.c. La fonction HRS_Init permet d'ajouter le service avec les caractéristiques associés et définies dans la norme du Bluetooth SIG. La seule caractéristique qui va nous intéresser ici est le Heart Rate Measurement, les autres étant inutiles pour notre application. Pour les désactiver, lancer le fichier .ioc et suivre les étapes de la capture de la figure 19. On remarque que la caractéristique Heart Rate Measurement a la propriété NOTIFY.

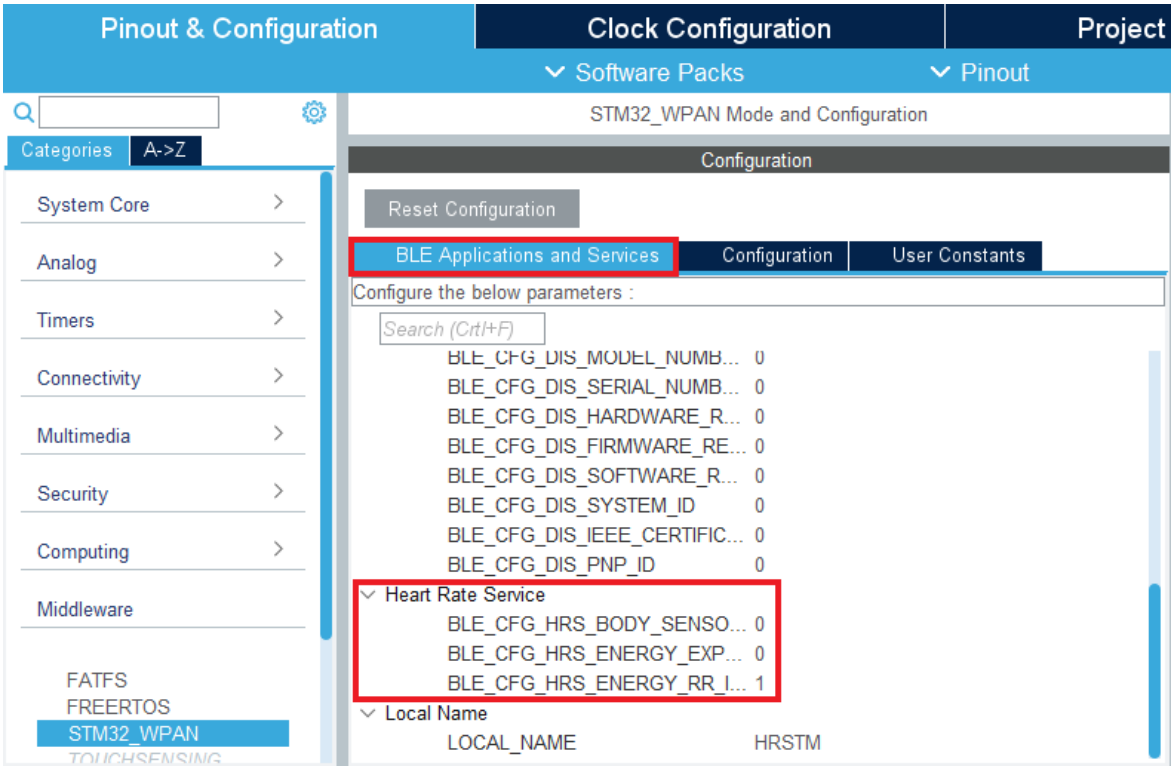

Figure 19 : Désactivation des caractéristiques inutiles

Notification La gestion de la notification est faite dans le fichier qui contient la fonction HRSAPP_Init. On remarque qu'un timer est créé à l'initialisation permettant de lancer la fonction HrMeas. Ce timer est lancé lorsque l'utilisateur demande la notification depuis le téléphone après s'être connecté, comme on peut le voir dans la fonction HRS_Notification. Lors du lancement du timer, on indique la période du timer HRSAPP_MEASUREMENT_INTERVAL. La fonction HrMeas permet de lancer la tâche qui a été enregistrée à la première ligne de la fonction d'initialisation, permettant elle-même d'exécuter la fonction HRSAPP_Measurement. C'est dans cette fonction que l'on effectue la mesure arbitraire avant de mettre à jour la caractéristique pour permettre la notification vers le téléphone.

Les valeurs de pulsation cardiaque sont générées par la ligne :

```
measurement = ((HRSAPP_Read_RTC_SSR_SS()) & 0x07) + 65;
```

La fréquence des notifications n'est pas explicite dans le code puisqu'elle est définie à partir d'une horloge provenant d'une division de celle de l'oscillateur. On peut tout de même la modifier en augmentant ou diminuant le facteur dans la définition de HRSAPP_MEASUREMENT_INTERVAL.

3.4 - Ajout du capteur de pulsation cardiaque

On souhaite maintenant ajouter le capteur pour envoyer des vraies mesures au téléphone.

3.4.1 - Configuration I2C

Lancer le fichier .ioc et suivre les étapes de la figure 20.

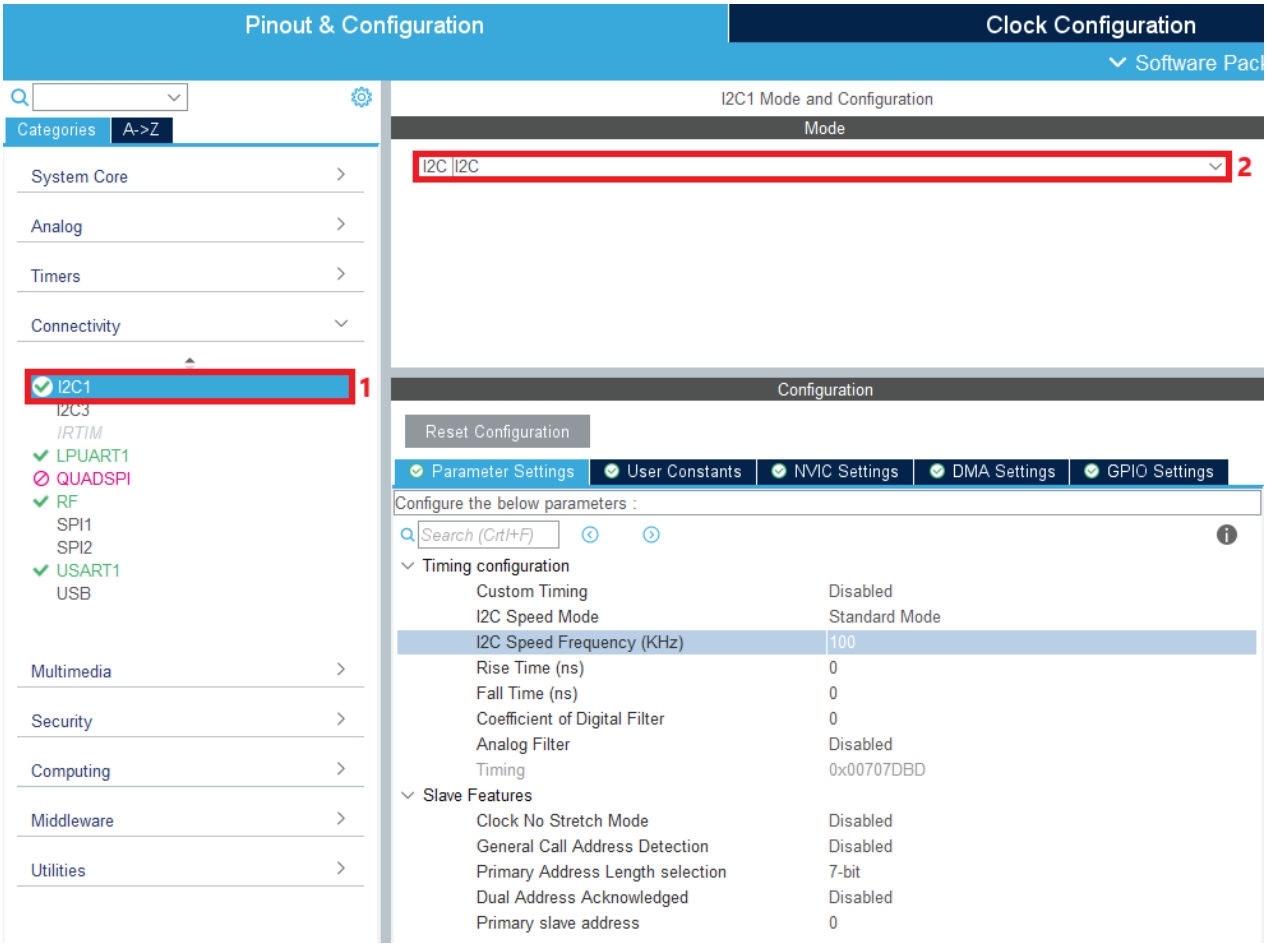

Figure 20 : Activation de la communication I2C

Dans l'onglet GPIO Settings, les pins associés aux signaux SCL et SDA sont PB8 et PB9. La figure 21 indique que les pins correspondantes sont nommées D15 et D14 sur la carte. Enregistrer le fichier pour regénérer le code. Dans le fichier main.c, une variable hi2c1 a été créée pour permettre la gestion de la communication I2C grâce à la bibliothèque HAL_I2C. La fonction d'initialisation a été générée automatiquement, il suffit maintenant d'écrire la fonction permettant de lire la valeur du capteur.

On peut alors brancher le capteur à la carte avec d'un côté l'alimentation en 5V et de l'autre les 2 signaux I2C. On s'assure que le capteur est bien alimenté en observant la DEL du capteur optique s'allumer.

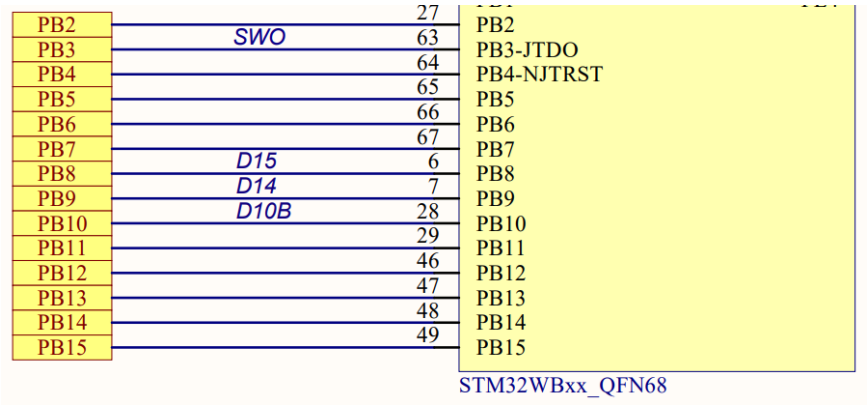

Figure 21 : Noms des pins Nucleo-WB55RG

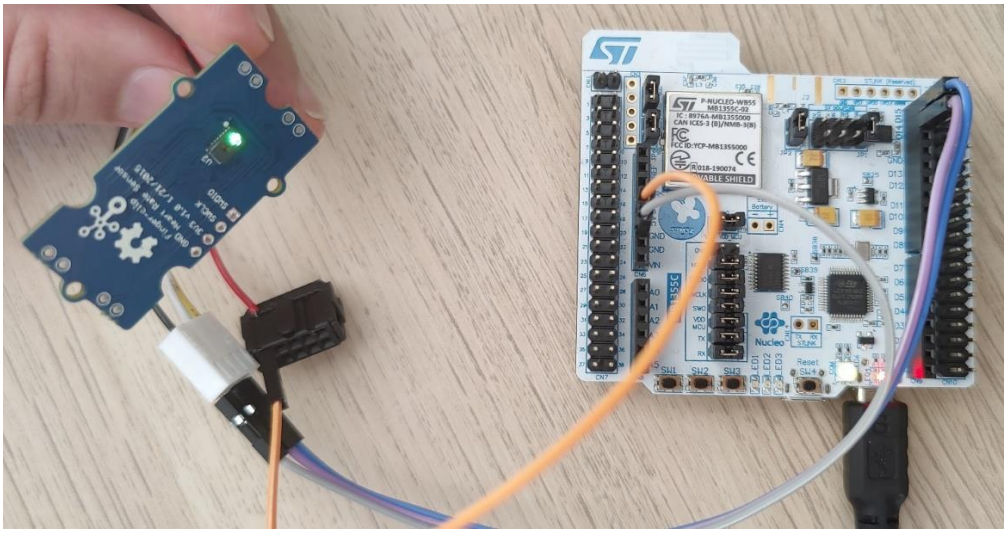

Figure 22 : Montage

3.4.2 – Récupérer la donnée capteur

La variable hi2c1 étant privée, on doit écrire cette fonction à l'intérieur du fichier main.c. En bas de celui-ci, une section USER CODE BEGIN 4 permet d'écrire des fonctions utilisateur sans qu'elles ne soient effacées lors d'une regénération de code. Le capteur renvoyant un unique octet représentant la valeur de la pulsation cardiaque, le prototype de la fonction sera uint8_t GetHeartRateValue(void).
Au préalable, on définit deux variables dans la section USER CODE BEGIN PV :

- uint8_t HeartRateValue : permet de stocker la valeur renvoyée par le capteur
- uint16_t SensorAddress = 0xA0 : adresse I2C du capteur

On écrit ensuite la fonction :

```
// Lecture en I2C
uint8_t GetHeartRateValue()
{
HAL_I2C_Master_Receive(&hi2c1,SensorAddress,&HeartRateValue,1,500);
return HeartRateValue;
}
```

La fonction HAL_I2C_Master_Receive prend en paramètre :

- Le *handle* de la liaison I2C ;
- L'adresse de l'appareil subordonné ;
- Le pointeur de la variable de stockage ;
- La taille du mot à lire en octet ;
- Le timeout.

Ne pas oublier de rajouter le prototype de la fonction dans le fichier main.h à la section USER CODE BEGIN EFP.

On va peut remplacer la ligne de génération de valeur dans la fonction HRSAPP_Measurement du fichier hrs_app.c par :

```
measurement = GetHeartRateValue();
```

en incluant main.h en en-tête de fichier.

4 – Application mobile

Les capsules vidéos mentionnées dans cette partie sont disponibles sur ce lien.

4.1 – Création d'un projet Xamarin Forms

Capsule vidéo :

- 01 : Création d'un projet Xamarin Forms pour le développement d'applications Android et iOS sous le format d'application à onglets.

4.2 – Interface utilisateur

Capsules vidéo :

- 02 : Utilisation du rechargement à chaud, permettant de modifier des caractéristiques graphiques dans le code et de visualiser en même temps
- 03 : Modification des icônes des onglets
- 04 : Nettoyage des fichiers non utiles provenant de l'exemple de base
- 05 : Ajout d'un bouton de scan sur la page de connexion dans le fichier .xaml avec sa fonction de callback
- 14 : Mise en page de la vue de suivi pour afficher la pulsation cardiaque

4.3 – Connexion avec le micro-contrôleur

Capsules vidéo :

- 06 : Pour utiliser les fonctionnalités Bluetooth LE, on utilise le package Plugin.BLE[6]. On va pouvoir utiliser facilement des fonctions pour les différentes procédures LE
- 07 : Ajout des autorisations android dans le fichier manifest.xml
- 08 : Fonction de callback pour le scan
- 09 : Fonction qui se déclenche lors d'une détection d'appareil : on ne garde que les appareils LE possédant un nom et le service Heart Rate
- 10 : Correction du code pour l'affichage
- 11 : Requête de connexion après avoir trouvé un appareil compatible
- 12 : Découverte des services et de la caractéristique de mesure de la pulsation cardiaque
- 13 : Affichage des informations de connexion

4.4 – Réception et affichage des données

Capsules vidéo :

- 15 : Liaison des données avec l'activation des notifications et la fonction qui s'exécute lors d'une réception. Une modification graphique doit passer nécessairement par le thread principal
- 16 : Correction du code, il faut initialiser le tableau
- 17 : Ajout des autorisations qui doivent être acceptées par l'utilisateur

[6] https://github.com/xabre/xamarin-bluetooth-le

4.5 – Export et test

Capsules vidéo :

- 18 : Export de l'application android
- 19 : Test sur téléphone

Le projet sous iOS doit subir les mêmes étapes, il faut se référer à la documentation de Plugin.BLE pour ce qui concerne les autorisations.

Références :

[1]: Site Web Bluetooth, https://www.bluetooth.com/

[2]: PM0271, STM32WB BLE stack programming guidelines, Programming manual, ST, https://www.st.com/resource/en/programming_manual/pm0271-stm32wb-ble-stack-programming-guidelines-stmicroelectronics.pdf

[3]: Documentation Xamarin, https://docs.microsoft.com/fr-fr/xamarin/get-started/what-is-xamarin

[4]: Github Plugin.BLE, https://github.com/xabre/xamarin-bluetooth-le