Bluetooth LE sur le STM32

Un moyen de lire les mesures à distance

Tam Hanna (Hongrie)

Pour de nombreux appareils électroniques, la connectivité avec les smartphones est indispensable, l'idéal pour la communication étant la technologie Bluetooth LE à faible puissance. En revanche, la mise en œuvre d'une application BLE sur un microcontrôleur n'est pas vraiment triviale. Les environnements de développement avancés avec génération de code sont d'une aide précieuse. Nous le démontrons ici avec une carte STM32WBA2 Nucleo et le STM32CubeIDE.

Figure 1. Le chipset radio proprement dit est placé sur une carte porteuse…

Figure 2. …tandis que le connecteur Arduino et le débogueur sont fournis par un circuit imprimé séparé.

Développer un produit doté de la communication Bluetooth LE n'est pas toujours simple. Vous devez tout d'abord sélectionner le meilleur SoC sans fil pour votre application, et tenir compte de la consommation d'énergie ultra-faible, mais aussi de l'ensemble de l'écosystème disponible pour le matériel.

STMicroelectronics a beaucoup œuvré pour offrir un écosystème convivial et libre pour tous les MCU STM32 d'usage général et maintient la même stratégie pour ses MCU sans fil. Le STM32WBA52 prend entièrement en charge la connectivité Bluetooth LE et est complètement intégré dans ses outils STM32Cube. Les contrôleurs de génération précédente, comme le produit BlueNRG de ST, migrent également vers l'écosystème STM32Cube. Dans cet article, nous allons montrer comment construire votre application Bluetooth LE en tirant parti de l'écosystème STM32Cube.

L'environnement de travail

L'un des principaux arguments en faveur des MCU de ST est le STM32CubeIDE. L'environnement de développement intégré est accompagné d'un générateur de code capable de produire des résultats d'une ingéniosité impressionnante.

Pour des exigences moyennes, il suffit souvent de configurer les périphériques à l'aide de la fonctionnalité graphique du STM32CubeIDE et de ne s'occuper que du code utilisateur. On ne perd pas de temps à étudier les fiches techniques et à « disséquer » les exemples de code. Cela permet d'économiser un nombre considérable d'heures de travail.

Dans les étapes suivantes, une carte NUCLEO-WBA52CG nous a servi d'environnement de travail, comme le montrent les **figures 1** et **2**.

Autres contrôleurs et cartes de développement

Le candidat alternatif numéro 1 est le STM32WL55 : il s'agit d'un MCU multicœur qui combine un cœur Arm Cortex-M4 et un cœur Cortex-M0+. L'émetteur-récepteur radio, optimisé pour divers protocoles sub-gigahertz, est intégré dans le MCU. Dans son annonce, ST promet la prise en charge des modulations « LoRa, (G)FSK, (G)MSK et BPSK. S'agissant d'un système sur puce sans fil entièrement ouvert, il est compatible avec les protocoles normalisés - et propriétaires - tels que LoRaWAN, Sigfox, wM-Bus et bien d'autres encore ».

La carte Nucleo, qui est principalement destinée à l'évaluation des nœuds LoraWAN, est disponible en deux versions - la WL55JC1 couvre la gamme de fréquences 865-928 MHz, tandis que la WL55JC2 couvre la gamme de fréquences 433-510 MHz.

Le candidat numéro 2 est le STM32WB-5MM-DK, une carte dont le module certifié est également basé sur le STM32WB55 à double cœur. Sur cette carte, le contrôleur diffère de son frère mentionné ci-dessus en ce qu'il prend en charge des protocoles supplémentaires - l'émetteur-récepteur sans fil installé ici est également capable de communiquer via Bluetooth LE et comprend également un MAC 802.15.4 et la prise en charge de Open Thread, Zigbee et Matter.

La carte d'évaluation contient également divers périphériques, notamment une mémoire Flash NOR QSPI externe et même un petit écran, ce qui permet de mettre en place et faire fonctionner des systèmes de capteurs sans avoir à connecter du matériel externe.

Gagnez 5 000 euros !

Dans le cadre du concours STM32 Wireless Innovation Design Contest, vous pouvez développer vos propres applications sans fil avec des cartes puissantes – prises en charge par le riche écosystème de STMicroelectronics. Vous pouvez vous attaquer à tout ce qui vous semble intéressant - de la manière que vous voulez ! L'IdO, la robotique, les jeux, la domotique, les tests et mesures ou l'IA ne sont que quelques-uns des domaines d'application possibles. Laissez libre cours à votre créativité, amusez-vous et gagnez ! Des prix d'une valeur totale de 5 000 euros sont à gagner ! Les détails concernant la participation au STM32 Wireless Innovation Design Contest, tels que le calendrier et les conditions exactes de participation, sont disponibles sur la page web du concours à l'adresse *elektormagazine.com/st-contest*.

Le processeur principal est un STM32WBA52CG - une puce à cœur Arm® Cortex®-M33 avec un haut niveau de sécurité (certifié PSA niveau 3), une puissance de sortie élevée (jusqu'à +10 dBm) et optimisée pour les capacités de messagerie à faible consommation afin de prolonger la durée de vie de la batterie. Toutes les fonctionnalités de la norme sans fil Bluetooth LE 5.3 sont prises en charge. Il convient de noter que ST fournit également plusieurs autres cartes d'évaluation, que nous avons brièvement décrites dans l'encadré **Autres contrôleurs et cartes de développement**.

Si vous regardez attentivement le dessous de la carte NUCLEO-WBA52CG, vous remarquerez qu'il y a un module bleu monté en surface sur le circuit imprimé. Il s'agit d'un débogueur ST Link complet appelé STLINKV3-MOD, vendu également en tant que produit autonome.

Pour commencer les expérimentations, nous utiliserons STM32CubeIDE. On peut le télécharger gratuitement à l'adresse [1] – dans les étapes suivantes, l'auteur travaille avec la version 1.13.2 de l'EDI. Une station de travail AMD à huit cœurs fonctionnant sous Windows 10 sert d'environnement d'exécution. STM32CubeIDE n'est pas trop exigeant en termes de performances système ; cependant, comme beaucoup d'autres systèmes basés sur Eclipse, l'interface utilisateur peut bénéficier de performances plus rapides sur un seul cœur.

Après avoir démarré l'EDI, nous cliquons sur l'option *File New STM32 Project* pour lancer le générateur de projet STM32. Ne soyez pas surpris si cela prend un peu de temps - STM32CubeIDE télécharge la dernière liste des définitions de cartes depuis un serveur ST à chaque démarrage pour s'assurer que le développeur dispose toujours d'un catalogue de modèles de projets à jour.

Dans l'étape suivante, nous passons à l'onglet *Board Selector,* où nous entrons NUCLEO-WBA52CG dans le champ *Commercial Part Number.* Une image de notre carte apparaît alors dans la fenêtre *Board List*, que nous sélectionnons en cliquant dessus. Nous cliquons ensuite sur *Next* pour lancer l'assistant de création de projet STM32. Vous pouvez attribuer le nom de projet que vous souhaitez tout en veillant à ne pas activer l'option *Enable TrustZone* et à laisser les autres paramètres tels que suggérés par STM32CubeIDE.

Après avoir cliqué sur *Finish*, le générateur commence à créer le squelette du projet – répondez par *Yes* à la question sur le paramétrage par défaut de tous les périphériques, ce qui signifie que nous nous appuierons sur le BSP - *Board Support Package* de la carte Nucleo. Dans certains cas, Cube affiche un avertissement (**figure 3**) qui indique l'absence d'un compte ST en ligne.

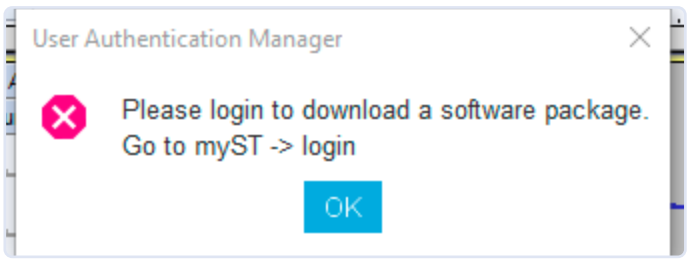

Figure 3. Pour télécharger des paquets, vous devez d'abord vous connecter.

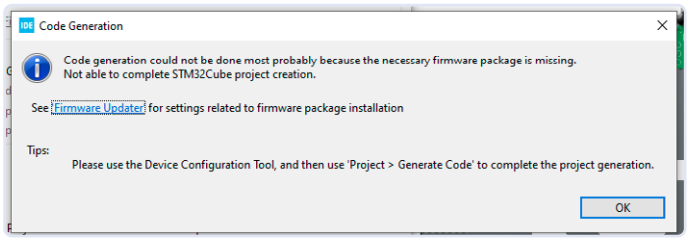

Figure 4. Cette boîte de dialogue est synonyme d'absence de paquet !

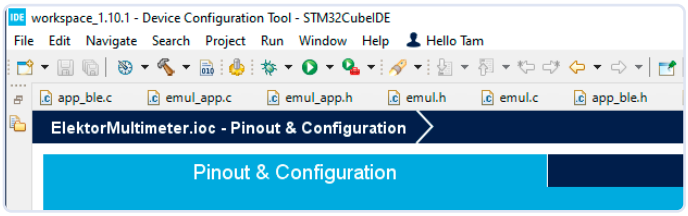

Figure 5. STM32CubeIDE est très convivial.

Si vous recevez ce message d'erreur, vous verrez – normalement - également la boîte de dialogue de la **figure 4**. Attention, car notamment sur les postes de travail multi-écrans, elle peut parfois être cachée derrière une fenêtre.
Si vous voyez ce message d'erreur, vous n'avez pas de chance – la solution la plus pratique est de supprimer l'ensemble du projet dans l'explorateur de projets. Attention : Dans la fenêtre *Delete Resource*, la case *Delete Project Content on Disk* doit être cochée, sinon STM32CubeIDE ne supprimera pas le projet sur le disque.
L'étape suivante consiste à cliquer sur *myST Login*. Dans la boîte de dialogue qui s'affiche, connectez-vous avec les détails de votre compte sur le site web de ST. Si vous n'avez pas encore de compte ST, nous vous conseillons d'en créer un. Ce compte vous permettra d'accéder à l'assistance technique proposée en ligne. Vous recevrez également des invitations régulières à des événements qui vous permettront d'en savoir plus sur le monde de ST : outre les événements en ligne, il y a également des événements réguliers en présentiel.
Après avoir réussi à vous connecter, vous êtes accueilli par votre nom dans la barre de menu, comme le montre la **figure 5**. Il s'agit d'une indication suffisante pour relancer la création du squelette du projet. Lors de l'extraction de l'archive contenant le pack logiciel STM32CubeWBA, vous devrez entre autres choses accepter un accord de licence. Une fois ce travail terminé, vous vous retrouvez dans la configuration standard d'un microcontrôleur STM32, qui vous permet de configurer les différents périphériques.

Comment accélérer l'extraction

L'extraction des archives du micrologiciel prend un certain temps sous Windows : Le principal responsable est l'antivirus intégré à Windows. En théorie, vous pouvez le désactiver temporairement dans le panneau de configuration pour accélérer le déploiement, mais nous ne le recommandons pas.

Mise en œuvre de la pile Bluetooth LE

Le développement manuel des diverses piles réseau est un de ces processus interminables qui tendent à se transformer en corvée. Heureusement, avec ST, la pile Bluetooth - également appelée « *middleware* » - est partiellement intégrée dans le générateur de code graphique.
Pour que le matériel Bluetooth soit opérationnel, il n'est même pas nécessaire de télécharger manuellement des paquets contenant des logiciels supplémentaires. En effet, les modules requis font généralement déjà partie du logiciel fourni ci-dessus.
Ici, le principal défi consiste à configurer les différents périphériques. Afin de réduire la consommation d'énergie, ST a toujours équipé ses contrôleurs 32 bits d'un système étendu de contrôle de puissance, qui permet de désactiver les unités fonctionnelles inutilisées. Il n'est pas surprenant que cette flexibilité accrue s'accompagne également d'une certaine responsabilité du développeur et éventuellement d'un effort supplémentaire lors de la phase de démarrage.
La pile Bluetooth LE nécessite toute une série de périphériques dont l'utilité n'est pas toujours évidente. Heureusement, ST propose une liste prête à l'emploi [2], que nous pouvons simplement parcourir. Il est important et nécessaire d'activer également des périphériques plus « exotiques » tels que le contrôle de la température du CA/N – il fournit des informations d'état dont la pile Bluetooth a absolument besoin pour démarrer.
Nous noterons également que les erreurs de configuration matérielle peuvent être délicates. Si, pour une raison quelconque, la pile Bluetooth LE ne fonctionnait pas au moment de l'exécution, vous devriez normalement commencer le dépannage de ce côté.
Dans ce qui suit, nous supposons que le lecteur a une connaissance préalable de l'utilisation de Bluetooth LE. Une brève introduction est disponible à l'adresse [3].
Pour la configuration proprement dite de la pile Bluetooth, nous passons à la catégorie *STM32_WPAN,* où nous sélectionnons *Mode BLE* et l'option *Select and configure your Server application*.
ST interprète la nomenclature du Bluetooth SIG de manière conviviale et épargne à l'utilisateur des puces les désignations particulières, *Center* et *Peripheral*.
L'étape suivante consiste à activer l'onglet *Configuration*, dans lequel on doit spécifier les paramètres *High Level*, y compris ceux du *GAP*. Comme nous supposons dans ce qui suit qu'il n'y a pas de renifleur de paquets Bluetooth LE dédié dans votre labo, nous recommandons une approche pas à pas pour configurer la pile Bluetooth LE. À noter cependant que ST propose aussi un exemple d'implémentation de renifleur Bluetooth LE basé sur le STM32WB. [5]
La première étape consiste à activer *l'Advertising* et à vérifier la réactivité à l'aide d'un appareil Android ou IOS doté d'une application de scanner Bluetooth LE ou, mieux encore, de l'une des applications mobiles de ST telles que ST BLE Toolbox.
Pour activer la fonction *Advertising*, nous passons à la section *Application parameters* et saisissons le nom d'utilisateur souhaité dans le champ *CFG_GAP_DEVICE_NAME* – nous utilisons la chaîne *TAMSMM* dans les étapes suivantes. Le logiciel met automatiquement à jour le champ *length*. Sachez que la norme Bluetooth LE impose des restrictions (assez strictes) sur la longueur maximale autorisée des paquets *advertising* et donc sur la longueur du nom.

Dans la section *Advertising Elements*, le contenu à inclure dans les paquets *advertising* proprement dits peut alors être ajusté – nous recommandons d'activer l'option *Include AD_TYPE_COMPLETE_LOCAL_NAME* dans la section *Advertising Elements* en sélectionnant *Yes.*

L'étape suivante consiste à définir les services exposés. Le STM32CubeIDE expose un assistant de configuration graphique convivial pour faciliter la vie des développeurs.

Pour cela, nous passons d'abord à l'onglet *BLE Applications and Services,* où nous spécifions le nombre de services BLE à créer dans la section*Server Mode Number of Services.* Par commodité, nous avons choisi l'option un, ce qui entraîne l'apparition d'un nouvel onglet portant le nom *SERVICE1*. L'ajout d'autres services entraînera l'apparition des onglets *SERVICE2*, *SERVICE3*, etc.

L'étape suivante consiste à ouvrir l'onglet *SERVICE1*, qui permet de configurer le service BLE nouvellement créé. Le champ *Number of Characteristics* est particulièrement important, car il détermine le nombre de caractéristiques du service concerné qui sont effectivement responsables de la fourniture des données.

Il est également nécessaire de spécifier l'*UUID*, ce qui est fait dans la section de dialogue illustrée à la **figure 6**.

Si vous sélectionnez la valeur *reduced* dans le champ *UUID 128 input type*, il suffit de saisir quatre nombres hexadécimaux dans le champ *UUID* - Cube compilera le reste à partir d'autres informations.

Nous avons décidé de créer deux caractéristiques dans les étapes suivantes, ce qui entraîne l'apparition des catégories *Characteristic 1* et *Characteristic 2.*

Outre la réattribution des UUID, il est important de remplir correctement les attributs *char_prop_write* et *char_prop_read.* Ils déterminent les droits accordés à un client établissant une connexion.

Dans la zone *Longueur de la valeur*, l'auteur a saisi 4 comme valeur pour la première caractéristique, tandis que nous avons donné la valeur 1 à la deuxième. Ces deux paramètres déterminent la quantité d'informations stockées dans la zone concernée.

Bien que vous puissiez définir des paramètres avancés de la pile Bluetooth LE dans la section *Platform Settings*, vous pouvez en général utiliser les valeurs par défaut ; ST propose une discussion détaillée des différentes options disponibles à l'URL [2] mentionnée ci-dessus. La fonction permettant à la pile Bluetooth LE d'envoyer des informations d'état à l'UART est particulièrement intéressante - une fonction dont nous ne discuterons pas davantage à ce stade, mais qui est vraiment très utile pour tout développeur pendant la phase de débogage.

Générer le code et effectuer un test de fumée

Les conditions logiques de mise en service de la pile Bluetooth LE sont à présent remplies. Passons au *Project Manager Code Generator* pour ajuster les options de conversion du fichier *.ioc* en fichiers compilables.

Dans la section *STM32Cube MCU Packages and Embedded Software Packs*, nous cochons *Add necessary library files as reference in the toolchain project configuration file* pour que le squelette de projet inclue les pilotes matériels requis et autres éléments.

L'étape suivante consiste à passer à la section *Advanced Settings*, où vous devriez – idéalement – adopter les paramètres fournis par ST

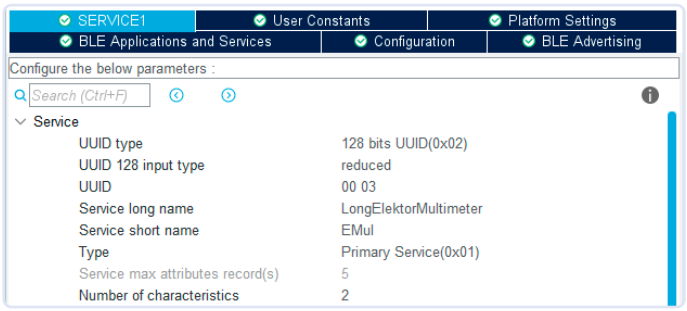

Figure 6. La configuration de l'UUID se fait ici.

(voir **figure 7**). Cependant, « déplacer » la priorité n'a pas fonctionné sur notre poste de travail, nous avons donc « seulement » ajusté les options *Do not generate function* et *Visibility.*

Il faut ensuite enregistrer le fichier *.ioc.* STM32CubeIDE vous invite alors à régénérer le squelette du projet comme d'habitude, une requête que vous pouvez et devez accepter.

Attention : Les dossiers peuvent contenir des fichiers d'utilisateur

Remarque importante : les dossiers à supprimer pour provoquer la recompilation sont également utilisés pour stocker le code utilisateur lors de la configuration de l'application Bluetooth LE. La solution consiste à ajouter un système de contrôle de version - toute ressemblance avec les batailles d'éditeurs dans la programmation Symbian est purement fortuite.

Préparation de l'Advertiser

Si votre projet est « compilable » à ce stade, vous pouvez le rendre visible pour le scanner Bluetooth LE mentionné ci-dessus. Dans un premier temps, nous vous recommandons de supprimer le champ *TX_POWER_LEVEL* dans le paquet et d'insérer le champ *AD_TYPE_APPEARANCE REIN :* Cela permet de s'assurer que le scanner affichera un symbole visible et facilement reconnaissable.

Il convient également de fixer à 20 l'intervalle de publication, défini via les champs *ADV_INTERVAL_MIN* et *ADV_INTERVAL_MAX* et de sélectionner *Yes* dans le champ *AD_TYPE_MANUFACTURING* pour le rendre visible depuis l'application ST Toolbox. Bien que le module radio consomme plus d'énergie dans cet état spécial, il transmet ses messages à une fréquence plus élevée et est donc beaucoup plus facile à reconnaître pour l'application scanner.

À ce stade, vous devez également passer à la section *Configuration RT GPIO Debug RT_DEBUG_GPIO_MODULE* et vous assurer que *NO* est sélectionné dans le champ *RT_DEBUG_GPIO_MODULE* – sans cela, la pile Bluetooth plantera avec une erreur grave pendant le démarrage de l'émetteur, à cause d'une liste de broches GPIO « vide ».

Pour terminer, vous devez ensuite ouvrir le fichier *app_ble.c,* où vous adaptez la section `USER CODE BEGIN APP_BLE_Init_3` de la fonction `APP_BLE_Init()` selon le schéma suivant. Veuillez tenir compte du caractère caché lorsque vous copiez et collez ce code :

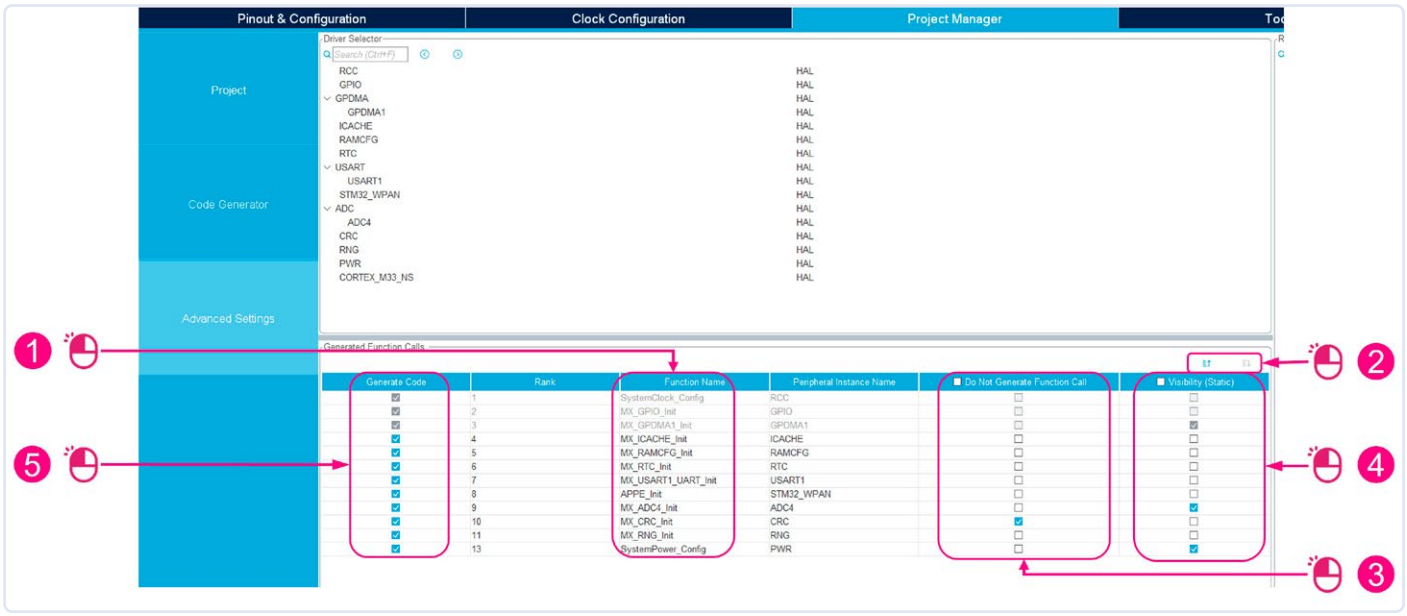

Figure 7. Ces réglages permettent d'atteindre l'objectif. (Source : STMicroelectronics [4])

```
/* USER CODE BEGIN APP_BLE_Init_3 */
tBleStatus ret =
  aci_hal_set_radio_activity_mask(0x0006);
if (ret != BLE_STATUS_SUCCESS)
{
  APP_DBG_MSG(«  Fail   :
   aci_hal_set_radio_activity_mask command,
   result: 0x%2X\n», ret);
}
else
{
  APP_DBG_MSG(«  Success:
   aci_hal_set_radio_activity_mask command\n\r»);
}

/* Start to Advertise to accept a connection */
APP_BLE_Procedure_Gap_Peripheral
 (PROC_GAP_PERIPH_ADVERTISE_START_FAST);

/* USER CODE END APP_BLE_Init_3 */
```

Pour éviter que l'*Advertising* ne se désactive lors d'une déconnexion, vous devez ajouter le code suivant dans le fichier *app_ble.c* :

```
SVCCTL_App_Notification - HCI_DISCONNECTION_COMPLETE_
EVT_CODE
/* USER CODE BEGIN EVT_DISCONN_COMPLETE */

APP_BLE_Procedure_Gap_Peripheral(PROC_GAP_PERIPH_ADVER-
TISE_START_FAST);

/* USER CODE END EVT_DISCONN_COMPLETE */
```

Le plus important ici est d'appeler `APP_BLE_Procedure_Gap_Peripheral(PROC_GAP_PERIPH_ADVERTISE_START_FAST)`, qui demande à la pile Bluetooth d'activer la logique *Advertising*. Veuillez noter que ce code n'est pas généré automatiquement par le configurateur STM32CubeIDE car il est considéré de la responsabilité du développeur. Le type de données `tBleStatus` est également intéressant : il s'agit d'un *typedef* destiné à renvoyer des informations sur l'état de Bluetooth LE. Notre application est à présent prête à flashée sur la carte. Nos efforts sont récompensés par l'apparition des fenêtres illustrées dans les **figures 8** et **9**ation.

Analyse de la structure de l'application

Plus haut, dans la section *SERVICE1*, nous avons attribué la valeur *EMUL* comme *Service Short Name*. Cela conduit maintenant à l'apparition d'un groupe de fichiers supplémentaires dans le sous-dossier app, comme le montre la **figure 10**.
Le fichier *emul.c est le* pivot du « stockage » des valeurs ; la fonction `EMUL_UpdateValue()` est particulièrement importante. Les premières lignes sont les suivantes :

```
tBleStatus EMUL_UpdateValue
 (EMUL_CharOpcode_t CharOpcode,
  EMUL_Data_t *pData)
{
 tBleStatus ret = BLE_STATUS_INVALID_PARAMS;

 /* USER CODE BEGIN Service1_App_Update_Char_1 */

 /* USER CODE END Service1_App_Update_Char_1 */

  switch(CharOpcode)
  {
    case EMUL_SONA1:
      ret = aci_gatt_update_char_value
    (EMUL_Context.EmulSvcHdle,
         EMUL_Context.Sona1CharHdle,
         0, /* charValOffset */
         pData->Length, /* charValueLen */
         (uint8_t *)pData->p_Payload);
```

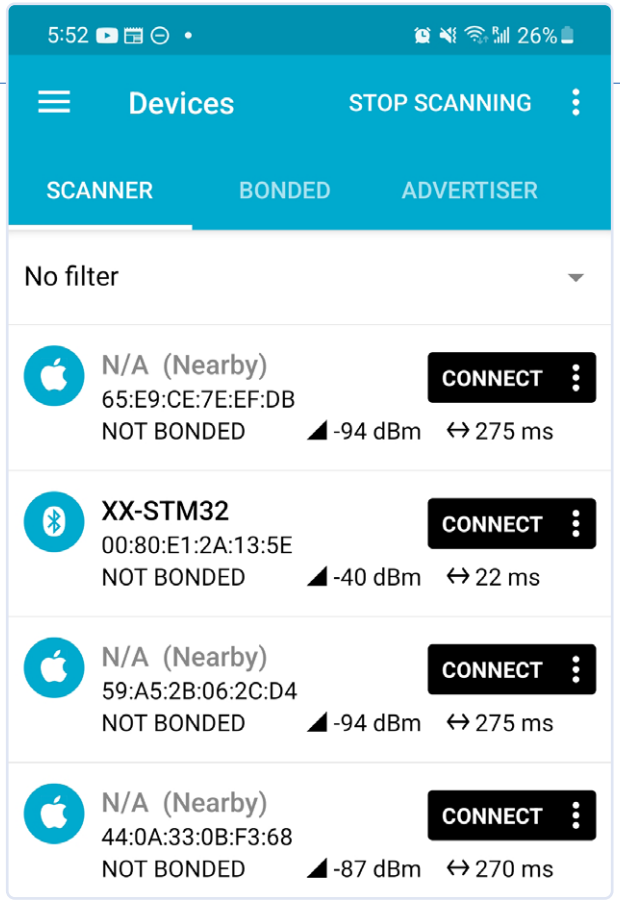

Figure 8. L'appareil est visible par le téléphone Samsung…

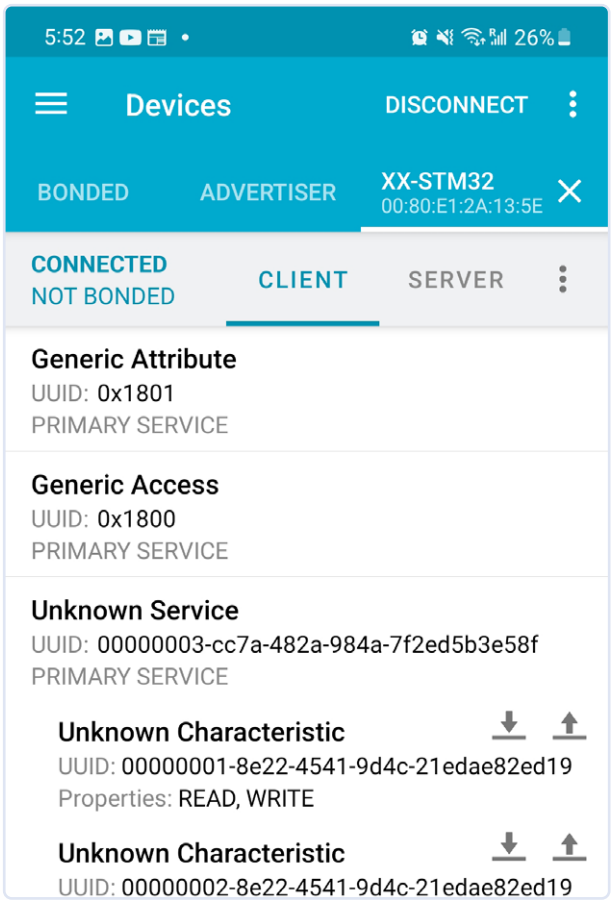

Figure 9. …et expose les deux caractéristiques dans le service !

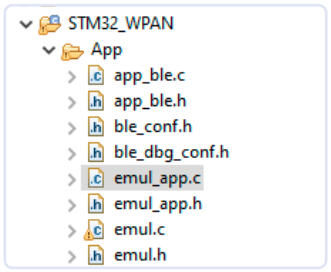

Figure 10. Les quatre fichiers qui « configurent » conjointement le service.

La fonction `aci_gatt_update_char_value()` est fournie par la pile Bluetooth qui écrit les informations fournies dans le cache. La pile prend le relais lorsque des demandes de lecture sont reçues d'appareils connectés à la puce.
Le traçage de la structure `EMUL_CharOpcode_t` conduit alors à la structure suivante – les entrées individuelles correspondent aux caractéristiques créées dans Cube ci-dessus (SONA1 et SONA2 sont les noms des caractéristiques) :

```
typedef enum
{
  EMUL_SONA1,
  EMUL_SONA2,
  /* USER CODE BEGIN Service1_CharOpcode_t */

  /* USER CODE END Service1_CharOpcode_t */
  EMUL_CHAROPCODE_LAST
} EMUL_CharOpcode_t;
```

La structure `EMUL_Data_t` sert à l'écriture ou à la transmission des valeurs et se présente ainsi :

```
typedef struct
{
  uint8_t *p_Payload;
  uint8_t Length;

  /* USER CODE BEGIN Service1_Data_t */

  /* USER CODE END Service1_Data_t */
} EMUL_Data_t;
```

Si vous approfondissez la fonction, vous trouverez également des codes structurés selon le schéma suivant. Il est chargé de recevoir les informations destinées à la deuxième caractéristique.

```
    case EMUL_SONA2:
      ret = aci_gatt_update_char_value
      (EMUL_Context.EmulSvcHdle,
             EMUL_Context.Sona2CharHdle,
             0, /* charValOffset */
             pData->Length, /* charValueLen */
             (uint8_t *)pData->p_Payload);
```

Forts de ces connaissances, nous pouvons revenir au fichier *main.c*, dans les inclusions duquel nous autorisons l'accès aux fichiers app de la première étape :

```
/* USER CODE BEGIN Includes */

#include «../../STM32_WPAN/App/emul.h»

/* USER CODE END Includes */
```

L'inclusion du fichier *common_blesvc.h* est également nécessaire car c'est lui qui permet d'accéder aux différentes primitives de la pile Bluetooth utilisées dans le fichier *Emul.h.*

```
/* USER CODE BEGIN Includes */
#include «common_blesvc.h»
#include «../../STM32_WPAN/App/emul.h»

/* USER CODE END Includes */
```

Dans l'étape suivante, nous allons exploiter une partie de la puissance de calcul de la boucle principale pour incrémenter un compteur et écrire régulièrement les mises à jour à la pile Bluetooth. C'est le moyen le plus simple, même s'il faut noter qu'un résultat plus élégant pourrait être obtenu en utilisant l'utilitaire Timer Server. Note : si le code ne se trouve pas dans la section *USER CODE*, il sera supprimé lors de la régénération :

```
  /* Infinite loop */
  /* USER CODE BEGIN WHILE */
  int aCounter = 0;
  int32_t aValue = 0;
  while (1)
  {
    /* USER CODE END WHILE */
    MX_APPE_Process();

    /* USER CODE BEGIN 3 */
    aCounter++;
    if(aCounter>10000)
    {
       aCounter=0;
       aValue++;
       EMUL_Data_t pData;
       pData.Length=4;
       pData.p_Payload = malloc(4);
       memcpy(pData.p_Payload, &aValue, 4);
       EMUL_UpdateValue(EMUL_SONA1,  &pData);
       free(pData.p_Payload);

    }
  }
  /* USER CODE END 3 */
```

Une remarque à propos de `malloc()` : Dans un système embarqué réel, il serait plus logique d'utiliser un tampon créé dans la pile, mais pour nos besoins de démonstration, cette procédure devrait être autorisée. Notez également que les appels à `EMUL_UpdateValue()` peuvent être très gourmands en énergie, en fonction de la configuration de la pile Bluetooth. En effet, selon la norme Bluetooth LE, les caractéristiques sont également capables d'envoyer des notifications. Dans ce cas, tous les clients enregistrés reçoivent un message, ce qui entraîne une consommation d'énergie considérable.
À ce niveau, un test plus approfondi du programme serait pertinent – cependant, les tentatives de lecture dans le scanner échoueraient avec un dépassement de délai. La raison de ce comportement, qui semble indésirable à première vue, est que les commandes individuelles de lecture et/ou d'écriture sont activées dans la pile ST Bluetooth – si nous avions été plus attentifs lors de la compilation, des avertissements du type *#warning user shall call aci_gatt_allow_read() function* **if** *allowed* nous auraient alertés de ce problème.
Pour le résoudre, activez comme suit la commande de lecture, dans le fichier *emul.c* dans l'`EMUL_EventHandler` :

```
if (p_read_req->Attribute_Handle ==
  (EMUL_Context.Sona1CharHdle +
   CHARACTERISTIC_VALUE_ATTRIBUTE_OFFSET))
{
  return_value = SVCCTL_EvtAckFlowEnable;
  /*USER CODE BEGIN Service1_Char_1_ACI_
   GATT_READ_PERMIT_REQ_VSEVT_CODE_1 */
  /*USER CODE END Service1_Char_1_ACI_
   GATT_READ_PERMIT_REQ_VSEVT_CODE_1*/
  /*USER CODE BEGIN Service1_Char_1_ACI_
   GATT_READ_PERMIT_REQ_VSEVT_CODE_2 */

 #warning user shall call aci_gatt_allow_read()
 function if allowed
   aci_gatt_allow_read
  (p_read_req->Connection_Handle);
   /*USER CODE END Service1_Char_1_ACI_
  GATT_READ_PERMIT_REQ_VSEVT_CODE_2*/
}
/* if(p_read_req->Attribute_Handle ==
 (EMUL_Context.Sona1CharHdle +
 CHARACTERISTIC_VALUE_ATTRIBUTE_OFFSET))*/
```

Le paramètre à passer à la fonction `aci_gatt_allow_read()` est important – il s'agit du *Connection Handle*, et il peut être obtenu via `p_read_req->Connection_Handle`.
Cette version du programme est enfin prête à être exécutée. Les **figures 11** et **12** montrent que la valeur mesurée enregistrée dans la caractéristique augmente avec le temps.

On suppose une connaissance préalable du BLE

Dans les étapes suivantes, l'auteur suppose que le lecteur a une connaissance préalable de l'utilisation de Bluetooth LE. Une brève introduction est disponible à l'adresse [3].

Conclusion et perspectives

Les premiers pas avec la pile Bluetooth sont quelque peu laborieux, mais l'EDI STM32Cube contribue grandement à accélérer le processus de développement. En plus d'une consommation d'énergie extrêmement faible, la pile vous permet de contrôler finement ce qui se passe dans l'interface sans fil et dans le microcontrôleur.

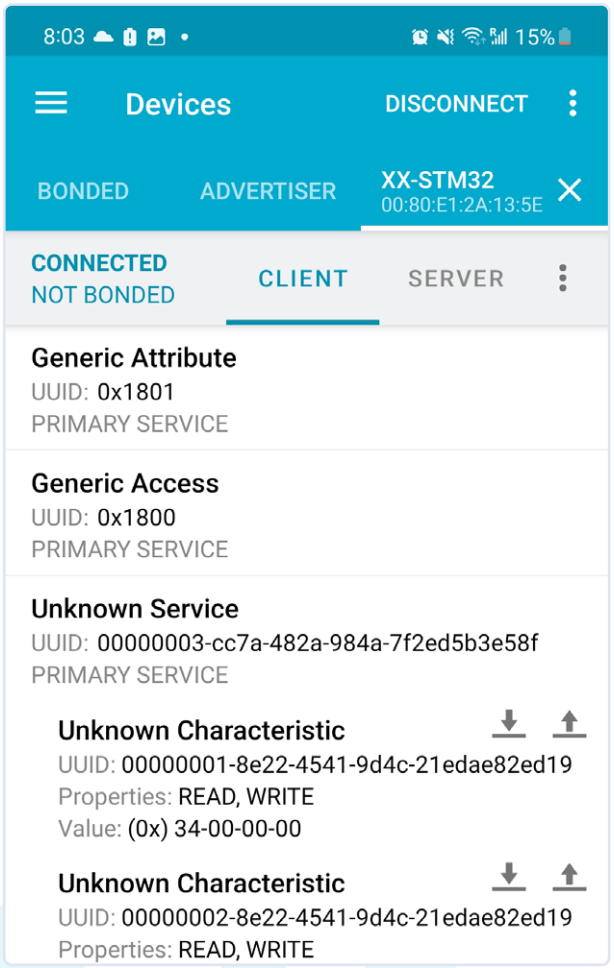

Figure 11. Comme du sable dans un sablier…

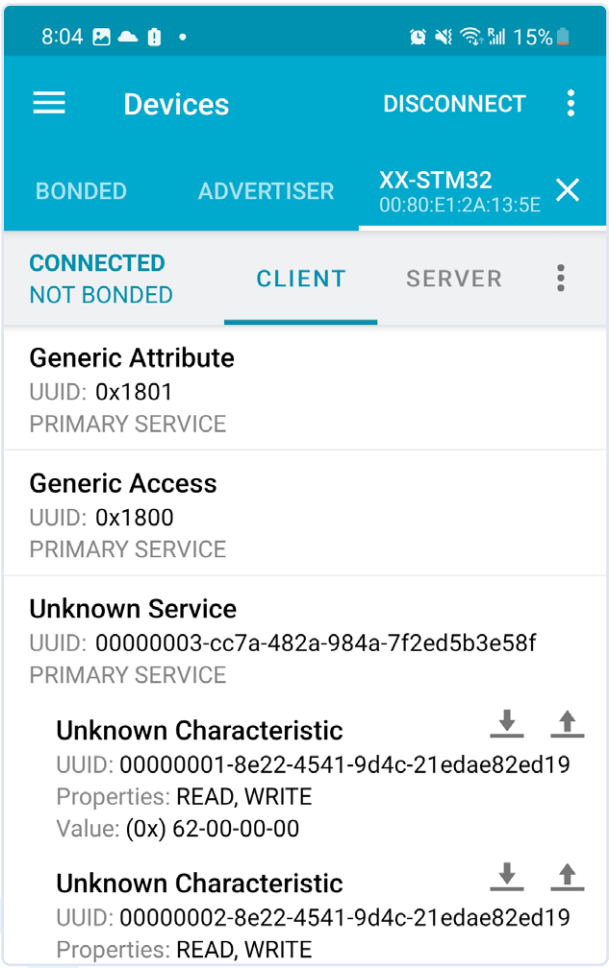

Figure 12. …la valeur transmise est incrémentée.

Bien entendu, la réalisation d'un véritable appareil de mesure qui envoie ses valeurs mesurées à un smartphone via BLE nécessiterait encore plus de travail : en plus d'un frontal analogique, il faudrait dans l'absolu une application d'accompagnement sur le téléphone, qui serait (idéalement) créée à l'aide de MAUI ou d'un cadre d'applications similaire indépendant de la plate-forme. Plus d'informations à ce sujet dans l'un des prochains numéros.

Nous espérons que ces premières étapes constitueront un bon guide pour travailler avec un système de développement Bluetooth fascinant !

VF : Denis Lafourcade — 230698-04

Questions ou commentaires ?

Vous avez des questions ou des commentaires sur cet article ? Envoyez un courriel à l'auteur à l'adresse tamhan@tamoggemon.com, ou contactez Elektor à l'adresse editor@elektor.com.

À propos de l'auteur

En tant qu'ingénieur, Tam Hanna travaille avec l'électronique, les ordinateurs et les logiciels depuis plus de 20 ans. Il est concepteur indépendant, auteur de livres et journaliste (@tam.hanna sur Instagram). Pendant son temps libre, Tam conçoit et produit des solutions imprimées en 3D et, entre autres, se passionne pour le commerce et la dégustation de cigares haut de gamme.

LIENS

[1] STM32 Cube IDE : https://www.st.com/en/development-tools/stm32cubeide.html
[2] BLE sur le STM32WBA, premiers pas : https://wiki.st.com/stm32mcu/wiki/Connectivity:STM32WBA_BLE_STM32CubeMX
[3] Brève introduction à Bluetooth Low Energy (BLE) : https://developer.android.com/develop/connectivity/bluetooth/ble/ble-overview
[4] Les paramètres dans STM32CubeIDE [capture d'écran] : https://tinyurl.com/stm32cubeidesettingsjpg
[5] Renifleur Bluetooth® Low Energy basé sur le STM32 : https://wiki.st.com/stm32mcu/wiki/Connectivity:STM32_Sniffer_for_BLE