HEIG-VD HAUTE ÉCOLE D'INGÉNIERIE ET DE GESTION DU CANTON DE VAUD

Prof. Yann Thoma

Laboratoire VSE

semestre d'automne 2025 - 2026

Laboratoire de vérification SystemVerilog
Bluetooth Low Energy Analyzer

Introduction

Dans le cadre du développement d'un analyseur Bluetooth Low Energy, nous avons développé un design sur FPGA pour l'analyse de paquets. Le système complet comprend toute la partie modulation-démodulation, suivie de la partie traitement des données. Le système embarqué est connecté à un PC via un port USB par lequel tous les paquets reçus sont transmis. Finalement une application graphique offre une vision générale des paquets observés.
Dans ce laboratoire vous allez mettre en place la vérification de la partie traitement des données. Ce composant reçoit, en entrée, les données fournies pour la partie démodulation, et en sortie envoie des données à l'interface USB.
Les deux interfaces disposent d'un protocole très simple, en *push*.

Spécifications

Bluetooth Low Energy exploite les mêmes 79 fréquences que le Bluetooth ($2402 + n$ MHz, $n \in [0, 78]$). Ces fréquences seront numérotées de 0 à 78. Toutefois, alors que Bluetooth exploite bien 79 fréquences, Bluetooth Low Energy ne se sert que des 40 fréquences paires ($2402 + 2n, n \in [0, 39]$).
Une communication standard se déroule de la manière suivante :

1. Un device envoie un paquet d'*advertising* qui contient une adresse sur 32 bits. Cet envoi se fait sur un des 3 canaux prévus (voir plus loin ci-dessous) ;
2. Ensuite, des paquets de données peuvent être envoyés, sur n'importe quel autre des 37 canaux, en utilisant l'adresse précédemment envoyée sur l'advertising.

Il y a donc, pour une adresse, un paquet d'advertising suivi d'un nombre quelconque de paquets de données.
Le format général d'un paquet est le suivant :

8	32	16	dataLength$\times$ 8
Préambule	Addresse	Entête	Données

Le préambule correspond à la valeur binaire 01010101.
L'adresse permet de savoir à quelle communication correspond le paquet.
L'entête contient diverses informations concernant le paquet.
Les données correspondent simplement aux données du paquet.
Il existe en outre deux types de paquets : Advertising et Data.
Les paquets d'advertising ont les caractéristiques suivantes :

1. L'adresse est toujours 0x12345678 ;
2. Les canaux d'advertising sont 2402, 2426, et 2480 MHz, soit les canaux 0, 24 et 78 selon notre numérotation qui suit le standard Bluetooth ;

3. Le nombre d'octets de données est représenté par les 4 bits de poids faible de l'entête (entete[3 :0]) ;
4. La taille maximale des données est donc de 15 octets ;
5. Les 4 premiers octets des données (poids forts des données) contiennent une adresse (deviceAddr) qui sert ensuite à détecter des paquets de données. Ceci implique que la taille minimale des données est de 4 octets.

Exemple d'un paquet d'advertising :

8	32	16	4× 8
Préambule	0x12345678	Entête	deviceAddr

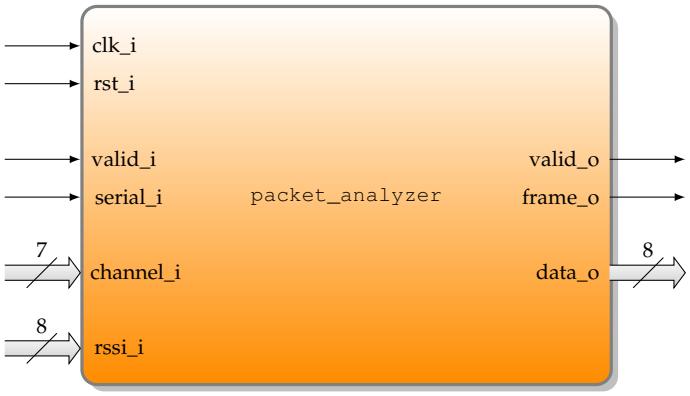
Les paquets de données ont les caractéristiques suivantes :

1. L'adresse est quelconque, mais doit correspondre à une adresse reçue via un paquet d'advertising ;
2. Les canaux sont les 37 canaux qui ne sont pas dévolus à l'advertising ;
3. Un paquet de données n'est valide que si un paquet d'advertising a déjà été envoyé avec l'adresse dans les données correspondant à l'adresse du paquet.
4. Le nombre d'octets de données est représenté par les 6 bits de poids faible de l'entête (entete[5 :0]) ;
5. La taille maximale des données est donc de 63 octets.

Il est à noter que l'analyseur n'est capable de stocker que 16 adresses. Donc si un 17ème paquet d'advertising est reçu, la nouvelle adresse remplace la plus ancienne. Il y a donc un FIFO pour les adresses considérées ensuite comme valides.
Au niveau système il est possible d'imaginer qu'il y a une ligne série par canal, les bits arrivant les uns après les autres. (Nous verrons que l'implémentation fait que nous multiplexons cette ligne dans le temps, mais pour l'instant restons sur la vue système).
Par défaut la ligne d'entrée doit être à 1. Ensuite, les paquets sont envoyés bit de poids fort en premier (en partant de la gauche dans le schéma du paquet présenté précédemment).
L'analyseur de paquet comporte une entrée sérielle par laquelle les paquets sont reçus. Une entrée supplémentaire sur 8 bits indique la qualité de réception du paquet. L'analyseur est chargé d'analyser les informations arrivant sur l'entrée et de fournir en sortie différentes informations sur un port parallèle 8 bits. Outre la sortie présentant une donnée 8 bits, une deuxième sortie permet d'indiquer si la donnée est valide ou non. Le composant possède évidemment également une horloge et un reset en entrée.
L'analyseur de paquets renvoie des paquets qui pourront ensuite être envoyés par USB, via un bus de 8 bits. Le format d'un paquet de sortie est le suivant :

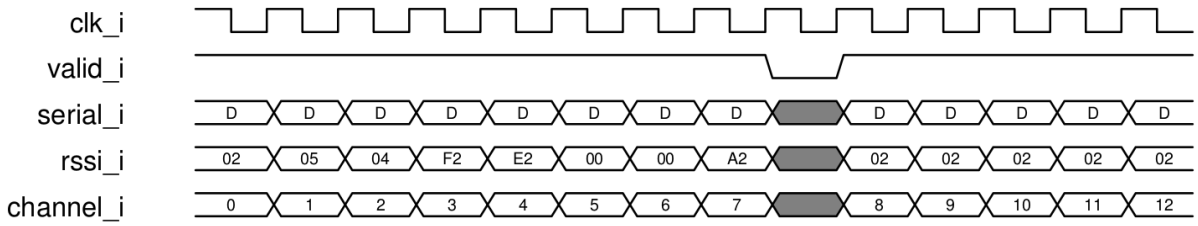
octet	Donnée							
	7	6	5	4	3	2	1	0
0	Size							
1	Rssi							
2	Channel							Adv
3	Reserved							
4	Adresse(7 :0)							
5	Adresse(15 :8)							
6	Adresse(23 :16)							
7	Adresse(31 :24)							
8	Entête(7 :0)							
9	Entête(15 :8)							
10	Données							
...	Données							
Size-1	Données							

Le champ `Size` correspond à la taille en octets du paquet USB. Le Rssi correspond au Rssi moyen observé durant la réception du paquet. Le champ `Adv` indique s'il s'agit d'un paquet d'advertising (1) ou de données (0).
Le composant possède les ports suivants :



Entrées

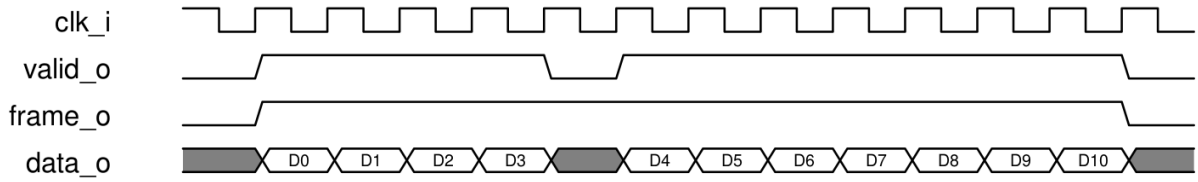
Le chronogramme suivant illustre le protocole d'entrée :



Il illustre le fait qu'à chaque cycle d'horloge, un bit d'un canal particulier est présent. Le numéro de canal est fourni en entrée (`channel_i`), de même que le bit (`serial_i`), et la valeur de RSSI (`rssi_i`). Les numéros de canaux seront toujours fournis dans l'ordre, de 0 à 80, en repassant ensuite à 0. La valeur de RSSI correspond à l'intensité du signal reçu. Nous serons intéressés à sortir la moyenne de cette valeur sur l'ensemble des bits reçus pour un paquet particulier.
Le démodulateur peut à tout moment désactiver le signal `valid_i` si nécessaire, et reprendre l'envoi dès sa réactivation.

Sorties

Le chronogramme suivant illustre le protocole de sortie :



Lors de l'envoi d'un paquet pour le circuit USB, `frame_o` doit être activé. Ce signal doit rester activé de manière ininterrompue pendant un envoi, et repasser à 0 dès que l'envoi d'un paquet est terminé. Il y a donc au moins un cycle à 0 sur `frame_o` entre deux paquets.
Le signal `valid_o` permet d'indiquer qu'une donnée est bien présente. Il peut être désactivé à tout moment par le composant, comme illustré sur le chronogramme.
Finalement, `data_o` contient 8 bits de données, dans l'ordre spécifié par le tableau ci-dessus.

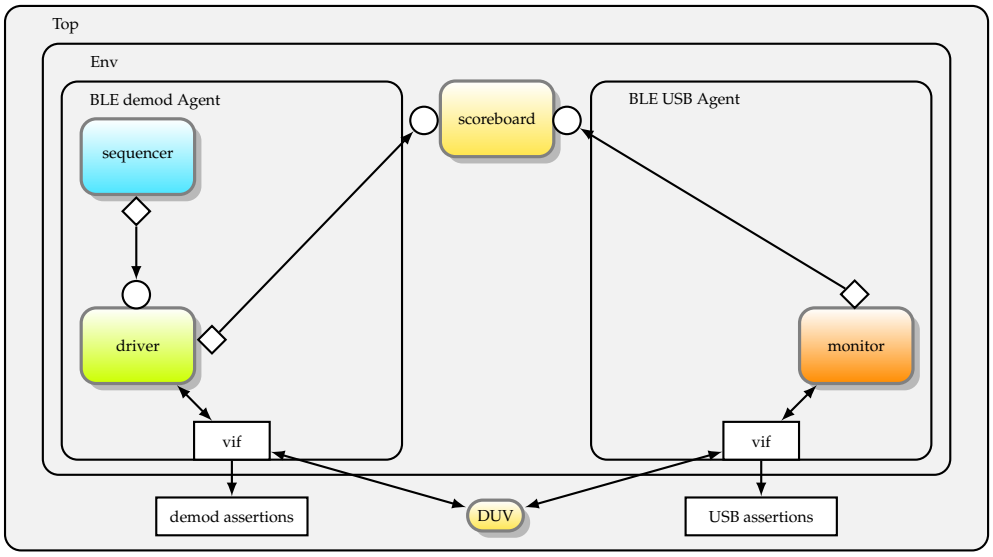
DUV

Un paramètre générique, `ERRNO`, permet d'injecter des erreurs dans le DUV. Il est également un paramètre du banc de test, et peut donc être forcé via le script de lancement de la simulation. Le DUV fonctionne ainsi, en fonction de `ERRNO` :

- 0 : Fonctionnement normal, et accepte des numéros de canaux pas nécessairement dans l'ordre ;
- 1 : Fonctionnement normal, l'ordre des numéros de canaux doit être respecté ;
- 2-6 : Fonctionnement erroné.

Architecture du banc de test

Le banc de test a été décomposé selon la méthodologie vue en cours, afin de bien différencier les responsabilités (séquenceur, driver, moniteur, scoreboard). La structure globale vous est déjà fournie, avec les instanciations de composants et les connexions via des FIFOs. Passez un peu de temps à bien comprendre cette architecture en regardant les fichiers SystemVerilog.



Vous pouvez noter les points suivants :

1. Les transactions envoyées par l'agent Input au scoreboard viennent du driver plutôt que d'un moniteur. Il serait plus propre d'avoir un moniteur d'entrée, mais ceci ajoute de la complexité au travail.
2. Il n'y a pas de driver de sortie, car rien n'est en entrée du DUV du côté sortie USB.

Le banc de test fourni offre déjà une fonctionnalité de base. Il envoie 1 paquet d'avertising suivi de 10 paquets de données.

- Le banc de test instancie le DUV et crée un objet `ble_analyzer_env` qui contient toutes les fonctionnalités de test ;
- Il instancie également deux modules qui pourraient contenir des assertions pertinentes. Pour l'instant il s'agit d'assertions qui sautent. Commentez-les pour éviter tous ces messages ;
- Le séquenceur a pour responsabilité de générer des séquences de test, et les envoyer au driver ;
- Le driver est responsable de jouer les séquences en intéragissant avec les entrées du DUV ;
- Le moniteur doit observer les sorties du DUV et reconstruire des objets USB pour les transmettre au scoreboard ;
- Le scoreboard doit récupérer les informations du driver et du moniteur afin de déterminer si tout se passe bien ou non ;
- Les interfaces du DUV sont déclarées dans des fichiers à part ;

— Les transactions (BLE et USB), ainsi que deux types mailbox associés sont déclarés dans des fichiers à part ;
— L'utilisation de mailbox pour la communication entre les éléments permet de disposer de FIFOs. La taille des mailbox est définie à leur création, et les méthodes `put` et `get` sont bloquantes ;
— ⚠ Pour l'utilisation des mailbox il y a un point auquel il faut faire attention. Il faut être sûr que l'objet mis dans la mailbox ne sera plus réutilisé par l'envoyeur. Il faut donc avoir une nouvelle instance de l'objet à chaque fois ;
— La fin de simulation peut être gérée avec les objections. Observez la mise en place du temps de drain dans le testbench, et un exemple d'utilisation dans `ble_demod_driver` ;
— Un logger est fourni et offre un *singleton* permettant de l'accéder depuis tous les fichiers. ⚠ le logger a été amélioré depuis celui du précédent labo, et exploite les fonctions systèmes de type `$error` et autres ;
— Une variable `testcase` est passée à chaque composant, via le script de lancement de la simulation ;
— La décomposition en plusieurs fichiers devrait faciliter la collaboration dans le groupe.
— Un fichier `default.rmdb` vous est fourni et permet de lancer plusieurs vérifications. Il n'est pas nécessaire d'avoir un testcase 0 pour lancer tous les tests, mais il vous est demandé de modifier le fichier `default.rmdb` de manière à ce que pour chaque ERRNO il y a au moins un testcase permettant de mettre en lumière l'erreur (pour les ERRNO avec erreur), et qu'il y ait tous les testcases pour les ERRNO valides.

Travail

Votre travail consiste à développer le banc de test et de mettre en place des tests pertinents. N'hésitez pas à exploiter de la randomisation, et potentiellement quelques assertions.
Un mini rapport est également demandé, et devra contenir (au moins) les éléments suivants :

1. Les choix liés à la modification de la classe `ble_packet` ;
2. Les choix liés à la classe `ble_usb_packet` ;
3. Les choix liés à la gestion de la fin de simulation ;
4. La manière dont vous avez envisagé la vérification au niveau du scoreboard ;
5. La manière dont vous avez géré le concept de canaux multiples ;
6. Les différents testcases que vous avez envisagés afin de valider le composant ;
7. Et tous les éléments de réflexion que vous avez eu durant le développement.

Le mini rapport et le code seront rendus sous forme d'archive générée par le script fourni.