Analyseur de réseau radiofréquence à base de STM32 et de synthétiseur numérique

Zakaria BELBACHA

1 Objectif principal

En basant sur un dispositif microsystème ainsi sur d'autre périphériques, notre objectif est de réaliser un programme capable de piloter un instrument de mesure afin de caractériser la réponse spectrale, en phase et en amplitude, des dispositifs radio-fréquences fonctionnant en transmission (résonateur à quartz, filtre...). Le microcontrôleur sera chargé de contrôler ces dispositifs.

2 Introduction et définitions

Tout d'abord, il faut bien consulter la documentation technique pour avoir les informations caractéristiques des composants numériques.

Un analyseur de réseau est un instrument de mesure qui permet de déterminer les paramètres S (*Scattering parameters : coefficients de diffraction ou de répartition)* pour décrire le comportement d'un circuit électrique .
On utilise généralement un analyseur de réseau dans le domaine des radiofréquences afin de caractériser les filtres, les câbles et les antennes[1].
On distingue deux catégories d'analyseurs de réseau: analyseur de réseau scalaire (SNA) qui mesure seulement les propriétés en amplitude, et analyseur de réseau vectoriel (VNA) qui mesure les propriétés en amplitude et en phase[1]. Ceci étant dit, l'analyseur de réseau vectoriel est le mieux adapté pour faire nos mesures.
Un analyseur de spectre est un instrument de mesure destiné à afficher les différentes fréquences contenues dans un signal ainsi que leurs amplitudes respectives.

La gamme de microcontrôleurs STM32F410 [2] avec cœur Cortex-M4 et unité de virgule flottante est le niveau d'entrée de la série STM32 F4. Ils sont conçus pour les applications nécessitant une performance équilibrée et une bonne efficacité énergétique. La famille des microprocesseurs STM32 de ST Microelectronics fournit une vaste gamme de périphériques, allant de

l'interface de communication série synchrone (x3 SPI) ou I²C, jusqu'à 2 ports USB 2.0 OTG FS/HS, d'un convertisseur ADC 12 bits (16 chaînes), x3 USART émetteur-récepteur asynchrone/synchrone universel, avec 9 Timers.

Ce processeur est compatible pour des applications faibles consommations avec un mode veille dont le réveil s'obtient par une condition sur une horloge interne ou une interruption externe.

En particulier ; un DDS (Direct Digital Synthesis) est un composant numérique (source de fréquence ajustable) dans lequel le microprocesseur va programmer une fréquence, c'est-à-dire générer plusieurs fréquences et le composant numérique va se charger de synthétiser un signal sur ses sorties différentielles I_{out} ou $I_{out\cdot Complémentaire}$ avec le signal radiofréquence qui nous intéresse. Il faudra implémenter le protocole de communication SPI (à chaque fois on ne sort qu'une fréquence, puis on balaie cette fréquence). Le contexte de ce travail est aussi de se familiariser avec le microcontrôleur, et une fois le microcontrôleur est capable de communiquer avec l'utilisateur, on va programmer le bus SPI et on essaye d'avoir des démarches rationnelles de *debuguage*. On a besoin de programmer le *timer* pour générer les signaux qui permet de cadencer le DDS, et pratiquement ; on va manipuler tous les périphériques de microcontrôleur. Le microcontrôleur a besoin de communiquer avec l'utilisateur et aussi il a besoin de programmer la liaison SPI-DDS. Le microcontrôleur fournit l'horloge au DDS *timer* et ce dernier génère un signal radiofréquence. C'est bien le principe de mesure d'un dispositif radiofréquence, ce qui nous permet par la suite d'établir la fonction de transfert de ce dernier. On a un détecteur de puissance *(diode)* en large gamme de puissance et en large gamme de fréquence dans le but de recevoir le signal issu du dispositif en mettant en marche le convertisseur analogique numérique *(ADC)*.

3 Schéma fonctionnel et principe de mesure

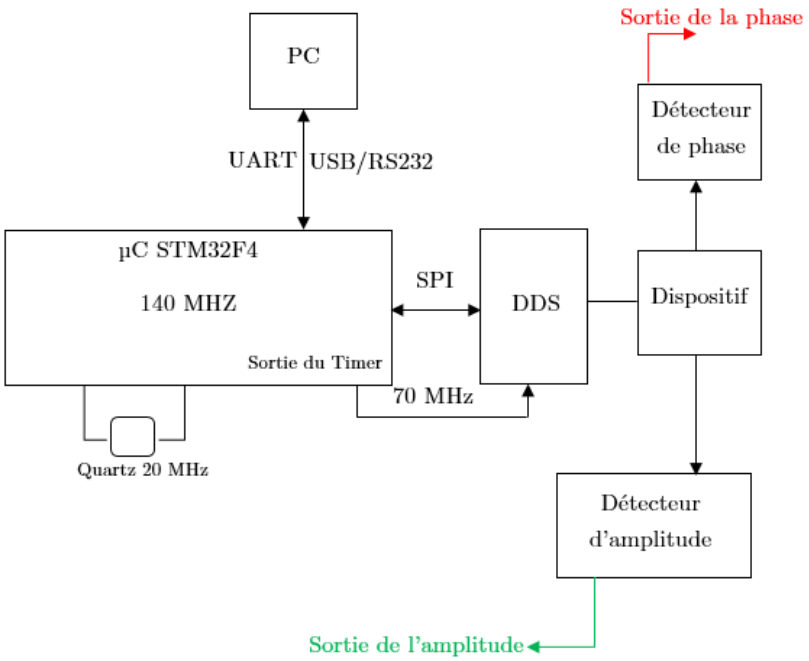

Figure 1 : Schéma fonctionnel

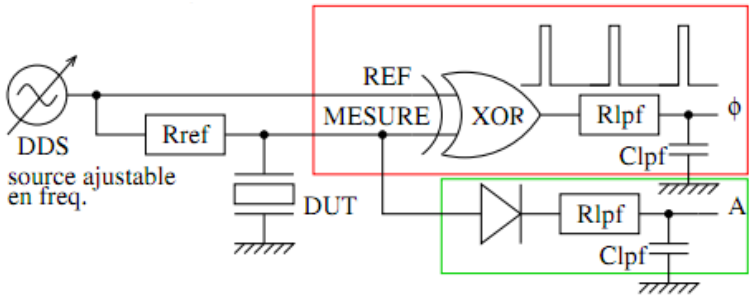

Figure 2 : Le principe de mesure (vert ☾ amplitude, rouge ☾ phase).

La communication de la carte avec le PC se fait via le port de communication USB/RS232 qui à comme rôle d'alimenter notre carte et la récupération des données.
Le STM32F410 dispose de 3 interfaces séries (UARTs), dans notre cas l'interface série 1 est connecté à un convertisseur USB-série. La communication entre le µC et le DDS se fait à travers le bus synchrone SPI.

Le signal de sortie du Timer permet de cadencer l'horloge principale du DDS. De son côté, notre synthétiseur de fréquence génère des signaux périodiques. Et pour le mesure, la carte se dispose d'un détecteur de phase basé sur un circuit logique XOR (avec deux entrées et une sorties) et un détecteur de puissance, chaque détecteur a un filtre passe-bas pour filtrer les bruits engendrés par l'ADC.
En pratique, on va mettre le dispositif en impédance connue[3], et on va faire un pont pour regarder si le dispositif absorbe l'énergie. On va voir le signal radiofréquence passe à travers d'une résistance. Donc, si les conditions de synchronisme sont vérifiées, la puissance va préférer de traverser le dispositif plutôt que d'aller dans le vecteur de puissance.
Normalement, ce qu'on va trouver c'est un signal en fonction de la fréquence dont lequel on va voir un signal qui est pratiquement constant autant qu'on ne respecte pas les conditions de synchronisme.

4 Programmation

4.1 La communication RS232 et GPIO

La communication RS232 ou communication série est une communication asynchrone c'est à dire le maître et l'esclave ne partage pas la même fréquence horloge, néanmoins si l'émetteur transmet l'information à une vitesse V et il faut que le récepteur fonctionne à cette même vitesse V ou presque, pour que les deux dispositifs puissent communiquer ensemble, contrairement au port parallèle. Le port série utilise seulement deux fils qui sont le Rx(récepteur) et Tx(émetteur).
La transmission de données commence par un bit de start qui annonce l'envoie des bits de données (8 bits). Et à la fin, un bit de stop. On envoi toujours le bit du poids faible (LSB) en premier, et finir par le bit le plus significatif (MSB). On peut ajouter un bit de parité après la fin de l'envoi pour s'assurer de la validité de l'information. On utilise la configuration 8N1.

Sur le STM32, la quasi-totalité des pattes sont accessibles en entrée-sortie tout-ou-rien. Les broches sont regroupées par paquets de 16, appelés ports. Chaque port est géré par un périphérique appelé GPIO (General Purpose Input/Output).

Notre microcontrôleur possède plusieurs ports GPIO comme GPIOA, GPIOB, GPIOC, GPIOH. Par exemple le périphérique GPIOA gère le port A, c'est à dire les pattes PA0 à PA15.
Nous allons dans un premier temps voir comment allumer des LEDs avec un STM32. Sur notre carte, il y a trois LEDs (LED1, LED2 et LED3) que l'on peut utiliser à sa guise. Ces trois LEDs sont connectées aux broches PA11, PA12 et PA13 respectivement du composant. On va donc utiliser le périphérique GPIOA.
Le STM32 étant un composant conçu dans le but de consommer le moins d'énergie possible, tous ses périphériques sont désactivés par défaut. Pour les mettre en route, il faut activer leur horloge. Il faut mettre à 1 le bit 4 du registre RCC_APB. Puisque nous allons utiliser les broches du GPIOA pour sortir une information du microcontrôleur (configuration des ports en sortie).

À noter qu'il faut toujours activer le bus de communication et l'horloge du périphérique utilisé, un tableau qui regroupe les fonctions alternatives du microcontrôleur (ci dessous), on s'intéresse particulièrement aux fonctions du GPIOA suivantes :

- AF1 (Timer1 : TIM1_CH1)
- AF5 (SPI1 : SPI1_NSS, SCK, MISO, MOSI)
- AF 7 (USART1 : USART1_TX, USART1_RX)

L'activation de l'horloge : rcc_periphclock_enable(RCC_GPIOA) dans la fonction void clock_setup().
Le périphérique GPIOB regroupe la fonctionnalité de notre convertisseur analogique-numérique ADC1, et le GPIOC notre détecteur de puissance. L'activation de leurs horloges se fait aussi dans la fonction void clock_setup().
Il faut aussi préciser la nature de la broche utilisée. Elle peut être une fonction alternative (GPIO_MODE_AF), une entrée flottante (on ne sait pas la nature de l'état haut ou bas), ou bien une sortie (open-drain, push-pull).
Il est important que les signaux doivent toujours être en état haut ou en état bas, tout simplement parce que en flottant, l'état est indéterminé, et provoque quelques de problèmes . La façon de corriger cela est de rajouter une résistance.

Table 10. Alternate function mapping

Port		AF0	AF1	AF2	AF3	AF4	AF5	AF6	AF7	AF8	AF9	AF10	AF11	AF12	AF13	AF14	AF15
		SYS_AF	TIM1/LPTIM1	TIM5	TIM9/ TIM11	I2C1/I2C2 /I2C4	SPI1/I2S1/S PI2/I2S2	SPI1/I2S1/ SPI2/I2S2/ SPI5/I2S5	USART1/ USART2	USART6	I2C2/ I2C4	-	-	-	-	-	SYS_AF
Port A	PA0	-	-	TIM5_ CH1	-	-	-	-	USART2_ CTS	-	-	-	-	-	-	-	EVENTOUT
	PA1	-	-	TIM5_ CH2	-	-	-	-	USART2_ RTS	-	-	-	-	-	-	-	EVENTOUT
	PA2	-	-	TIM5_ CH3	TIM9_ CH1	-	I2S2_ CKIN	-	USART2_ TX	-	-	-	-	-	-	-	EVENTOUT
	PA3	-	-	TIM5_ CH4	TIM9_ CH2	-	I2S2_MCK	-	USART2_ RX	-	-	-	-	-	-	-	EVENTOUT
	PA4	-	-	-	-	-	SPI1_NSS/ I2S1_WS	-	USART2_ CK	-	-	-	-	-	-	-	EVENTOUT
	PA5	-	-	-	-	-	SPI1_SCK/ I2S1_CK	-	-	-	-	-	-	-	-	-	EVENTOUT
	PA6	-	TIM1_BKIN	-	-	-	SPI1_MISO	I2S2_MCK	-	-	-	-	-	-	-	-	EVENTOUT
	PA7	-	TIM1_CH1N	-	-	-	SPI1_MOSI /I2S1_SD	-	-	-	-	-	-	-	-	-	EVENTOUT
	PA8	MCO_1	TIM1_CH1	-	-	I2C4_ SCL	-	-	USART1_ CK	-	-	-	-	-	-	-	EVENTOUT
	PA9	-	TIM1_CH2	-	-	-	-	-	USART1_ TX	-	-	-	-	-	-	-	EVENTOUT
	PA10	-	TIM1_CH3	-	-	-	-	SPI5_MOSI /I2S5_SD	USART1_ RX	-	-	-	-	-	-	-	EVENTOUT
	PA11	-	TIM1_CH4	-	-	-	-	-	USART1_ CTS	USART6 _TX	-	-	-	-	-	-	EVENTOUT
	PA12	-	TIM1_ETR	-	-	-	-	SPI5_MISO	USART1_ RTS	USART6 _RX	-	-	-	-	-	-	EVENTOUT
	PA13	JTMS-SWDIO	-	-	-	-	-	-	-	-	-	-	-	-	-	-	EVENTOUT
	PA14	JTCK-SWCLK	-	-	-	-	-	-	-	-	-	-	-	-	-	-	EVENTOUT
Port A	PA15	JTDI	-	-	-	-	SPI1_NSS/ I2S1_WS	-	USART1_ TX	-	-	-	-	-	-	-	EVENTOUT

Figure 3 : Fonctions alternatives des diverses broches numériques du microcontrôleur.

4.2 Initialisation de l'horloge du µC

Les microcontrôleurs sont, comme presque tout en électronique numérique, des composants synchrones, c'est à dire qu'ils ont besoin d'une horloge pour synchroniser toutes les opérations. Le STM32 possède de nombreuses sources d'horloge, certaines rapides, d'autres plus lentes, internes ou externes...
La façon dont on procède pour régler l'horloge principale d'un microcontrôleur, c'est d'aller dans le manuel de référence, chercher l'arborescence des horloges. Dans le manuel des STM32, il se trouve dans «Clock configuration tool»[4].

Le type d'horloge utilisée est la PLL[5] (phase-locked loop, c'est à dire boucle à verrouillage de phase), est un asservissement de phase ou de fréquence qui asservit la fréquence d'un oscillateur commandé en tension ou VCO (organe qui génère une tension variable) à un signal injecté à l'entrée.

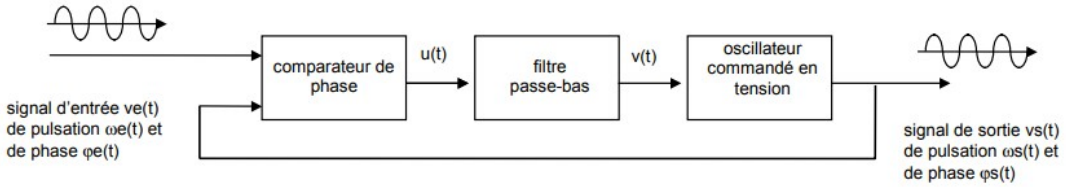

Figure 3 : Structure de base de la boucle à verrouillage de phase (PLL).

Les éléments de la PLL sont les suivants :

- l'oscillateur VCO donne une fréquence qui varie en fonction de la tension de commande v appliquée sur son entrée.
- la fréquence du VCO est comparée avec une fréquence de référence (consigne) grâce à un comparateur de phase (Ou exclusif, comparateur phase-fréquence).
- le comparateur de phase fournit à sa sortie une tension u (valeur du déphasage entre Ve et Vs) donnée par un filtre passe-bas.

Le VCO est l'organe qui génère la tension variable Vs. Le multiplieur est le comparateur de phase. Enfin, le filtre passe-bas a pour but de filtrer des fréquences élevées produites par le comparateur de phase.

=> La PLL est un composant radio dédié à la synthèse d'une fréquence.

La fréquence du microcontrôleur varie selon l'oscillateur externe implémenté sur notre carte, c'est à dire selon l'horloge externe à grande vitesse HSE[6] (L'oscillateur HSE permet de générer une horloge rapide via un quartz externe).

Utiliser ce dispositif coûte un peu plus cher que d'utiliser le HSI (horloge interne rapide), mais la précision de l'horloge est plus grande, car on sait fabriquer des quartz avec une fréquence de résonance de quelques mégahertz à quelques pulsations par minute (ppm) près, alors que l'oscillateur utilise une cellule R-C, il va donc avoir une fréquence d'oscillation dépendant de la température.
La fréquence du cœur du processeur est issue d'une multiplication par PLL fractionnaire de la fréquence du quartz comme suit :

SysCoreClock=((HSE/PLL_M)*PLL_N)/PLL_P

Pour bien configurer notre PLL, on s'est basé sur l'outil de configuration d'horloge proposé par STMicroelectronics sous forme de feuille de calcul Excel.
Le résultat d'une configuration avec un résonateur 20 MHz pour sur-cadencer le cœur du microcontrôleur à 140 MHz est représenté dans cette figure :

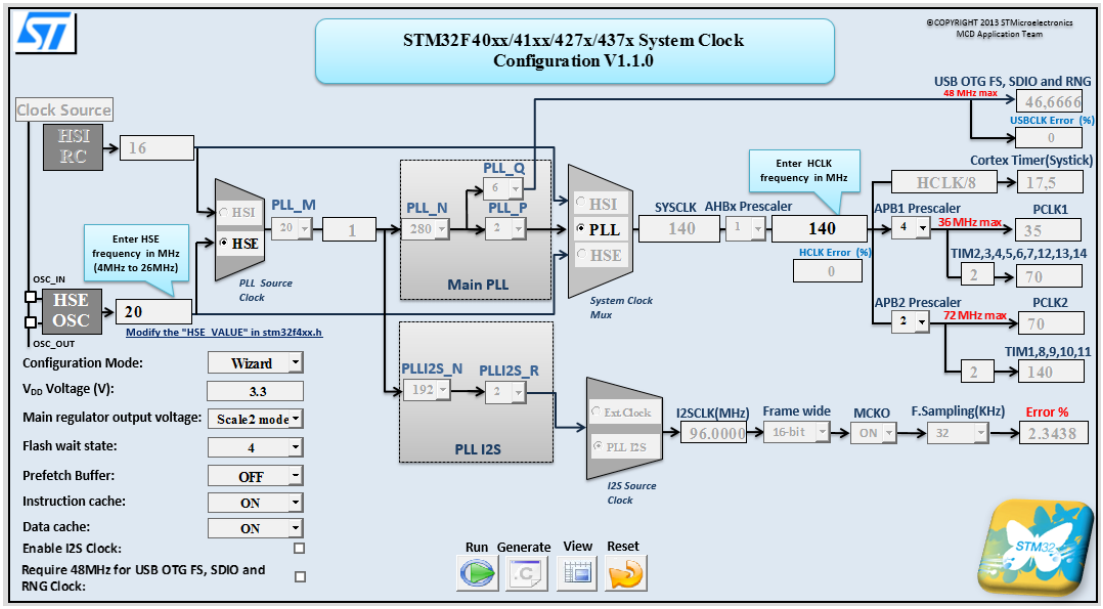

Figure 4 : Configuration des horloges pour un cadencement par un quartz à 20 MHz.

Concernant le côté programmation de cette partie, on a pu suivre les étapes suivantes :

- Choisir la fréquence du microcontrôleur
- Initialiser PLL vers le registre RCC_PLLCFGR
- Initialiser le registre RCC_CFGR
- PPRE1: APB Low speed prescaler (APB1)
- PPRE2: APB high-speed prescaler (APB2)

4.3 UART

Un UART, pour Universal Asynchronous Receiver Transmitter, est un émetteur-récepteur asynchrone universel.
C'est le composant utilisé pour faire la liaison entre l'ordinateur et le port série. L'ordinateur envoie les données en parallèle (autant de fils que de bits de données). Il faut donc transformer ces données pour les faire passer à travers une liaison série qui utilise un seul fil pour faire passer tous les bits de données.
- Étapes de programmation :

- Activer l'horloge de GPIO
- Activer l'horloge de USART
- Configurer le port GPIO9/GPIO10 (fonction en mode AF)
- Multiplexer les 2 ports en mode USART
- Initialiser le type de transmission de la donnée entre le PC et le microcontrôleur (baudrate)
- Activer USART

4.4 SPI

Le protocole SPI est un registre à décalage où on le remplie avec des mots transférés sur le bus du SPI, et à l'issu de transmission, le DDS a une petite particularité par rapport au DDS classique qui a besoin d'un signal particulier pour écrire réellement dans sa mémoire de façon à générer le signal (C'est pour éviter les états instables). Les deux caractéristiques ajustables de ce bus : l'état au repos de l'horloge (état haut) et le front d'horloge sur lequel le signal de données est échantillonné (descendant) [4].

Les transactions entre le microcontrôleur et le DDS se font au travers ce bus synchrone SPI, ou le maître et l'esclave partagent l'horloge. Cette communication utilise 4 signaux :

- SCLK : Serial Clock, horloge qui synchronise la transmission des données
- MOSI : Master Output, Slave Input, le maître va envoyer une donnée sur le bus SPI que l'esclave va récupérer.

- MISO : Master Input, Slave Output, l'esclave envoie la donnée au maître
- SS : Slave Select, Actif à l'état bas (généré par le maître)

L'esclave renvoie toujours la même information fournie par son maître. Le Chip Select CS permet au maître de choisir son esclave avec quel il veut communiquer, on distingue deux mode pour le Chip Select :

- Mode low : C'est à dire la broche du CS a 0 V comme tension, l'esclave va comprendre que la prochaine information lui appartient.
- Mode high : La broche du CS reçoit 5 V, l'esclave ne prend pas en considération les données dans le bus SPI.

Le tableau suivant présente les ports GPIO de la communication SPI :

Type du signal	Le port	Mode
SS	PA4	Output
SCK	PA5	AF7
MOSI	PA7	AF7
MISO	PA6	AF7
MCLK de 70 MHz	PA8	AF1

D'après la datasheet, le bus SPI a deux caractéristiques : l'état au repos de l'horloge (haut), et le front de l'horloge (bas) sur lequel le signal est échantillonné.

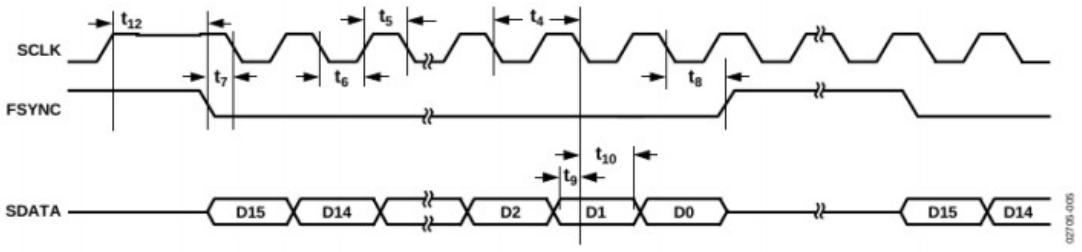

Figure 5 : Synchronisation en série.

Les deux figures suivantes représentent les signaux reçus depuis le bus SPI. En vert c'est l'horloge, et en jaune des caractères aléatoires.

Figure 6 : Signaux reçus depuis le bus SPI.

4.5 Timer

Le Timer1 est un compteur électronique à fonctionnement libre avec une fréquence de comptage qui est une fraction de son horloge source. Il a une résolution de 16 bits (0 à 65535).

La vitesse de comptage peut être réduite à l'aide d'un prescaler (un diviseur) pour notre Timer1. La formule pour trouver la fréquence en sortie du Timer dépend de la valeur du prescaler.

- Lorsque le prescaler = 0 : f_{Timer1} = Sysclock /(prescaler+1).periode

La sortie du Timer1 sera la moitié de la fréquence µC (c'est à dire 140 Mhz/2 = 70 MHz).

- Lorsque le prescaler $\neq 0$: f_{Timer1} = Sysclock /2.prescaler.periode

Dans le cas où le prescaler vaut 1, la sortie du Timer1 sera 35 MHz (Figure ci-dessous).

Figure 7 : Sortie Timer 1 - En haut avec un prescaler de 0, en bas avec un prescaler de 1.

Ce signal alimentera l'horloge principale du DDS.

Figure 8 : Sortie MOSI.

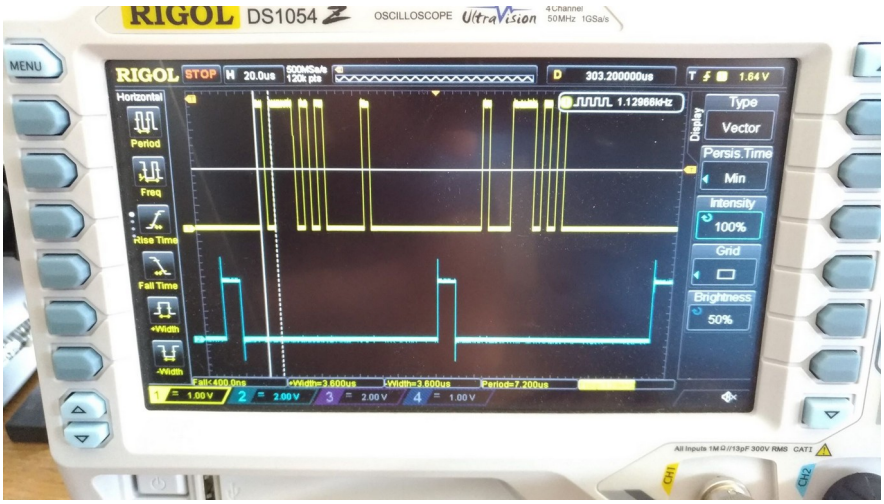

Figure 9 : Sortie CS.

4.6 ADC

D'après le schéma de notre carte, la phase et l'amplitude à mesurer sont connectées sur le port PB1 et PB0. D'après la documentation (en.DM00214043 tableau 9, page 35), les deux convertisseurs sont associés aux broches 9 et 8 du multiplexeur de ADC1. La mise en œuvre du convertisseur nécessite l'activation de l'horloge.
La fonction read_adc(k) permet de lire le contenu de l'ADC.

Table 9. STM32F410x8/B pin definitions (continued)

Pin Number			Pin name (function after reset)(1)	Pin type	I/O structure	Notes	Alternate functions	Additional functions
WLCSP36	UFQFPN48	LQFP64						
-	18	26	PB0	I/O	FT	-	TIM1_CH2N, SPI5_SCK/I2S5_CK, EVENTOUT	ADC1_8
-	19	27	PB1	I/O	TC	-	TIM1_CH3N, SPI5_NSS/I2S5_WS, EVENTOUT	ADC1_9

Figure 10 : Fonctions analogiques.

Pour vérifier le bon fonctionnement de notre ADC, on connecte un fil entre la masse (0V) et l'amplitude, puis avec un fil de 3, 3 V. En résultat, nous avons eu respectivement 0 V et 0xfff, donc cela confirmera que notre convertisseur fonctionne correctement.

4.7 DDS

On cadence le DDS par la sortie de notre Timer1. La fréquence de cadencement du microcontrôleur est le double de la fréquence maximale de commande du DDS.
Pour calculer la fréquence en sortie du DDS, on utilise cette formule :

$$f_{out} = W.70/2^{28}$$

La configuration du DDS sera enregistrée dans les registres de contrôles 16 bits (CR).
La configuration du registre de contrôle CR =0x2128 :

DB15	DB14	DB13	DB12	DB11	DB10	DB9	DB8	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
0	0	B28	HLB	FSEL	PSEL	PIN/SW	RESET	SLEEP1	SLEEP12	OPBITEN	SIGN/PIB	DIV2	0	MODE	0
0	0	1	0	0	0	0	1	0	0	1	0	1	0	0	0

- DB3 = 1 : La valeur de Sign bit out égale à la valeur en sortie du DDS
- DB5 = 1 : Activation du détecteur de phase (la porte XOR)
- DB8 = 1 : Reset du DDS (réinitialisation des registres)

- DB13 = 1 : Écrire un mot dans un registre de fréquence (14 bits du premier registre représentent LSB du mot, et les 14 bits du deuxième registre représentent MSB du mot).

En ce qui concerne le programme de l'envoie de 16 bits au DDS, on commence à mettre l'horloge du DDS à l'état bas (dds_cs_clr ()), ce la nous permet de faire l'envoie et ma transmission des données. On envoie seulement 8 bits, pour cela on commence par prendre le 8 bits du poids faible via un masque (spi_send (SPI1, (entree>>8)&0xff)) c'est à dire 0000 0000 1111 1111, puis de faire un décalage de 8 bits encore une fois pour les bits du poids fort (spi_send (SPI1, (entree&0xff))), ce qui veut dire que nous sommes limité à 8 bits de données en envoie. En fin, pour finir la transmission, on doit mettre l'horloge du DDS à l'état haut (dds_cs_set ()).

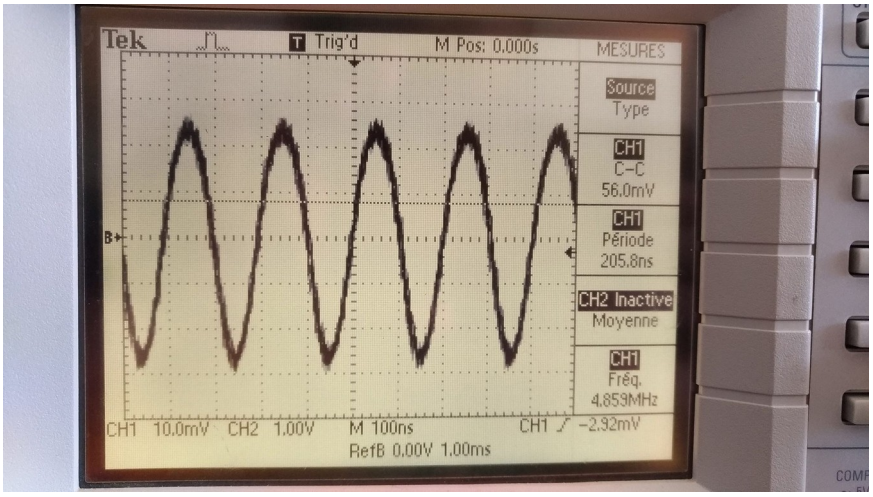

Figure 10 : Sortie DDS.

5 Cratérisation d'un quartz 4,9 MHz

Pour caractériser un dispositif radio-fréquence (dans notre cas un quartz de 4,9 MHz, il ne faut un balayage de fréquence qui s'incrémente par un pas donné. Pour cela, on a utilisé une boucle de type for en utilisant deux fréquences, une fréquence min et une fréquence max.

```
for (frequence=frequenceMin; frequence<=frequenceMax;frequence=frequence+h) {
```

Ensuite, on a calculé ces deux fréquences comme suit :

- Fréquence min : 4,9152 MHz – 1 KHz = 4,9142 MHz
  => $W_{min} = (4{,}9142/7)*2^{28} = (18842251)_{10} = 0x11F828B$
- Fréquence max : 4,9152 MHz + 1 Khz = 4,9162 MHz
  => $W_{max} = (4{,}9162/7)*2^{28} = (18845319)_{10} = 0x11F8E87$

- On a choisi 3068 points d'échantillonnage, pour trouver la valeur du pas du balayage on fait : $(f_{Max}-f_{Min})/3068$

```
h=(frequenceMax-frequenceMin)/3068;
```

- Les données sont envoyées dans l'ordre suivant: phase, amplitude puis fréquence.

```
v=read_adc(8);
put_hex(v);
jmf_putchar (' ');
v=read_adc(9);
put_hex(v);
jmf_putchar ('\n');
jmf_putchar ('\r');
```

- Pour tracer l'amplitude en fonction de la fréquence, on a utilisé 3 outils :
  - Octave : Après la récupération des données via minicom (Ctrl+A puis L puis un nom du fichier), on trace la courbe avec ces commandes .

```
octave:1> file =fopen("zb")
octave:1> donnees=fscanf(file,"%x",[3,Inf])'
octave:1> plot(donnees(:,1),donnees(:,3))
```

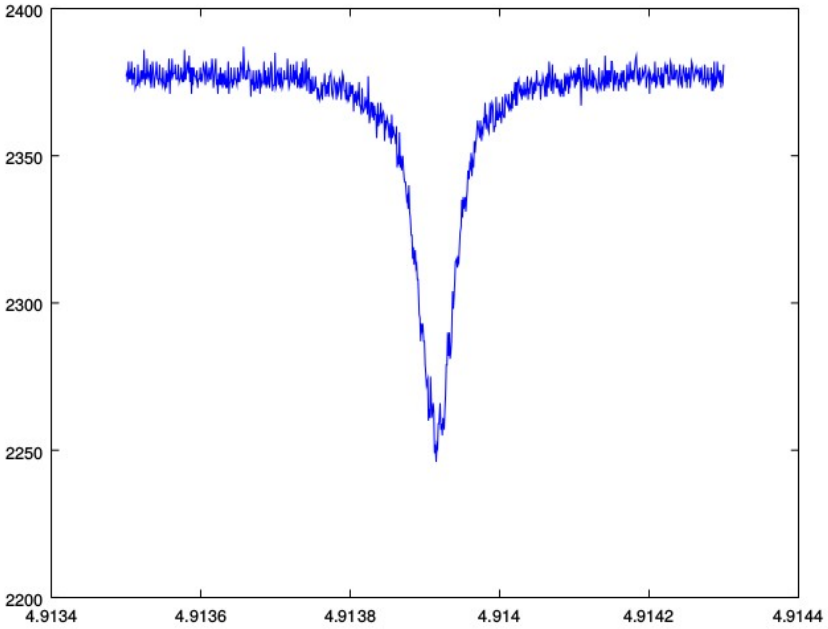

Figure 11 : Fréquence de résonance sous octave.

- Pyqtgraph : On trace deux courbes en temps réel (l'amplitude en fonction de la fréquence, et la phase en fonction de la fréquence).
  Les données sont reçues dans un tableau, donc il suffit d'indiquer le numéro de case pour avoir les valeurs.

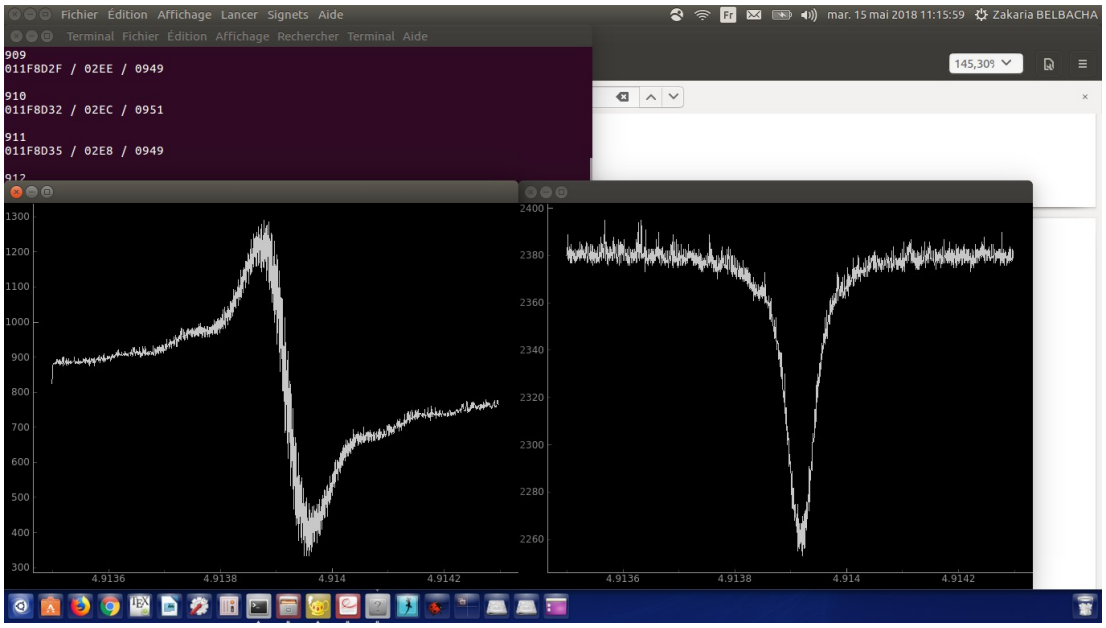

Figure 12 : Traçage de l'amplitude (à droite) et de la phase (à gauche) en fonction de la fréquence au temps réel

- Matplotlib : On trace les deux courbes avec un subplot. Le type d'affichage sera mode longueur. La figure s'affichera quand l'interface graphique ne reçoit pas de données.

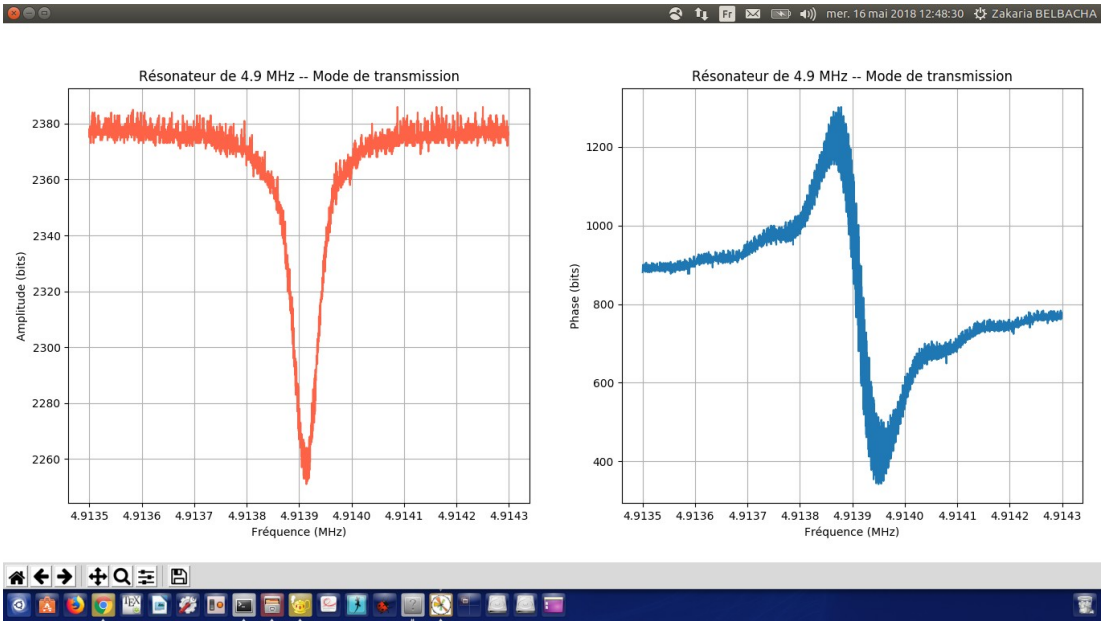

Figure 13 : Traçage de l'amplitude (à gauche) et de la phase (à droite) en fonction de la fréquence avec Matplotlib

6 Références

[1] https://fr.wikipedia.org/wiki/Analyseur_de_r%C3%A9seau
[2]http://www.st.com/content/ccc/resource/technical/document/data-sheet/6e/f3/b0/1b/05/eb/49/fe/DM00214043.pdf/files/DM00214043.pdf/jcr:content/translations/en.DM00214043.pdf
[3] http://jmfriedt.free.fr/network_analyzer.pdf
[4] http://www.st.com/content/ccc/resource/technical/document/application_note/7c/dc/ef/0a/c8/74/46/6e/CD00290434.pdf/files/CD00290434.pdf/jcr:content/translations/en.CD00290434.pdf
[5] http://www.ta-formation.com/acrobat-modules/pll.pdf
[6] http://www.atrylec.com/fr/electronique/stm32discovery/stm32discovery_page_4B.php.html#Endroit_3-0-0