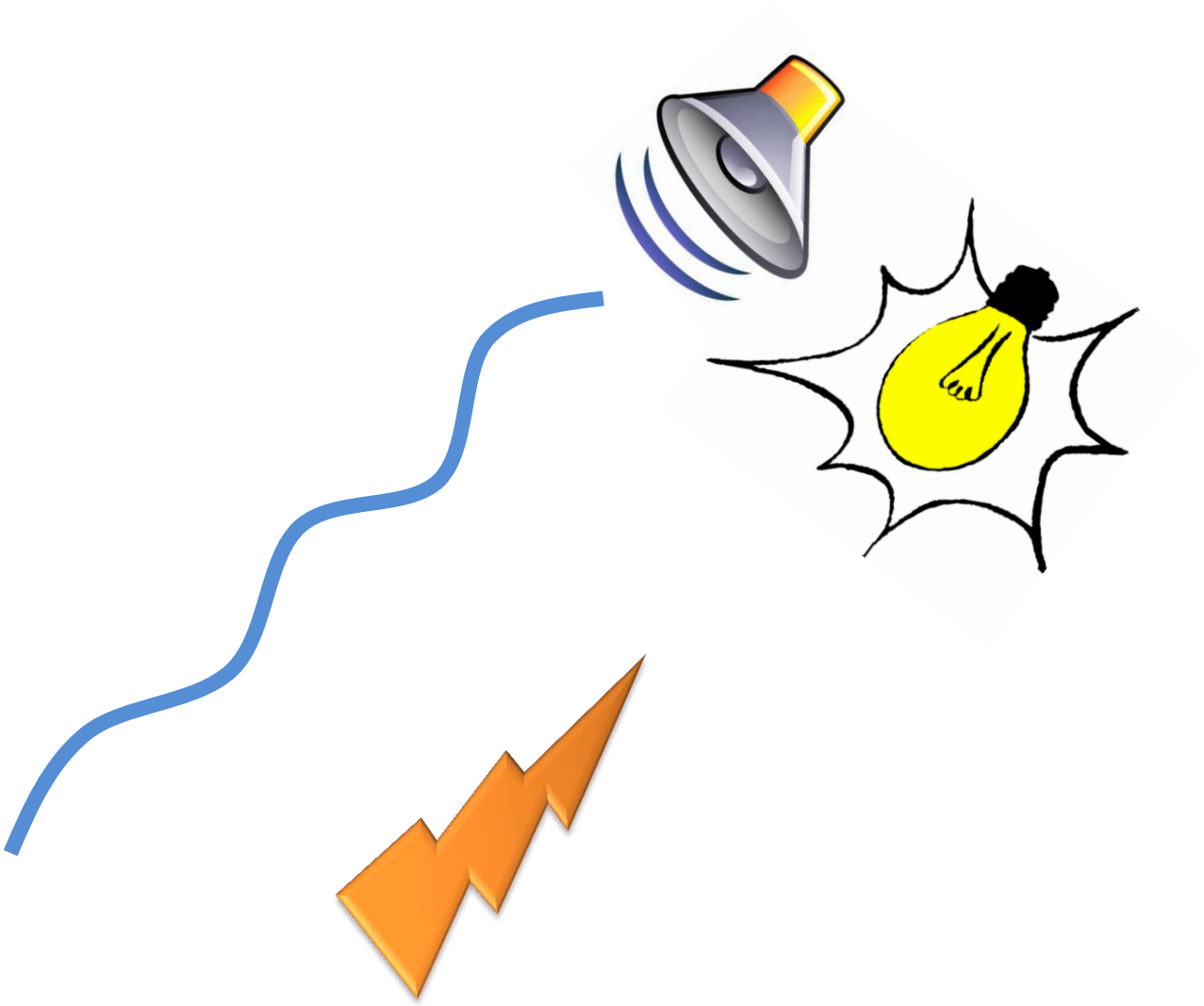
Mission 2 – Le son va moins vite que l'image.



2015

Objectifs de la mission :

- Prise en main des outils, de la cible, de l'environnement de développement
- Lecture et modification d'un programme existant
- Utilisation du débogueur
- Utilisation du périphérique IT Externe

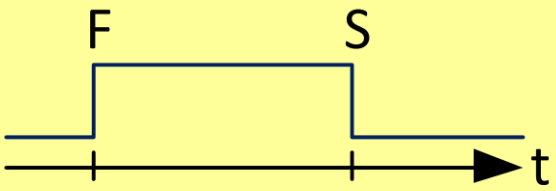
1 Le cahier des charges

.

La société FlashoWave développe un produit révolutionnaire permettant la mesure de distance entre deux points, sans contact.

D'un côté, un émetteur, qui produit un flash lumineux en même temps qu'un bruit important. De l'autre côté, un récepteur, qui est sensible au flash et au son reçu.

Le récepteur fournit le signal suivant :



L'instant F correspond à l'instant de réception du Flash. L'instant S à l'instant de réception du Son. Si l'on néglige le temps de propagation de la lumière devant celui du son, et les variations locales de température et de milieu, la durée entre ces deux instants est proportionnelle à la distance entre l'émetteur et le récepteur.

La société souhaite sous-traiter à votre bureau d'étude la mesure du temps entre les instants F et S.

Votre chef de projet est en relation avec le client, et doit lui fournir un devis très prochainement. Face au risque dans la détermination du temps à consacrer au développement, il a souhaité demander à ses développeurs d'effectuer une étude de faisabilité.

De : Yvon Developpertoutcapourmoi [yvon.developpertoutcapourmoi@super-BE-R&D.fr]
Le : 25/12/2018 à 06h37

Objet : Etude de faisabilité – devis n°76284 – client FlashoWave.
Note interne – confidentiel.

Cible : STM32F4-DISCOVERY
Constat : Nous n'avons pas encore utilisé le périphérique IT Externe sur cette cible
Objectif de l'étude : développer la brique logicielle manquante dans nos librairies et caractériser la précision obtenue. Cette étude pourra ainsi éclairer le chef de projet dans la rédaction de son devis vers le client.
Contenu de l'étude : mesurer la durée de l'appui sur un bouton poussoir. La mesure pourra être consultée directement en mode Debug. L'ordre de grandeur de la précision attendue est de 5ms.
Il est nécessaire de vérifier la mesure obtenue…

3 Quelques rappels sur le fonctionnement du périphérique IT Externe.

Qu'est-ce qu'une interruption ?

Vous êtes un microprocesseur… vous êtes en train de travailler, vous recopiez le dictionnaire… Cette tâche est longue et monotone… mais vous occupe néanmoins, c'est la « tâche de fond votre programme » (le « main »).

Le téléphone sonne.

Vous répondez au téléphone, discutez avec votre interlocuteur, raccrochez. Puis, vous reprenez votre travail…

***

Dans cet exemple, vous avez traité la sonnerie du téléphone comme une requête d'interruption, et vous avez abandonné temporairement votre tâche de fond pour répondre à l'appel en exécutant votre « routine d'interruption » (= répondre au téléphone).. Vous êtes ensuite revenu à votre tâche de fond, exactement à l'endroit où vous aviez abandonné votre recopie du dictionnaire.

Le concept d'interruption permet au microcontrôleur, lorsqu'un évènement survient, d'interrompre le déroulement de son programme pour exécuter sans plus attendre une routine d'interruption. Cette routine est une simple fonction (que l'on appelle pas nous-même). Lorsque le processeur à terminé d'exécuter la routine d'interruption, il retourne dans la tâche de fond, exactement à l'endroit où il avait interrompu son travail !

L'évènement qui déclenche l'interruption peut être la levée d'un **flag d'interruption**, en provenance d'un **périphérique interne**… ou bien un **évènement externe** qui se traduit généralement par un changement de l'état d'une broche d'entrée.

Ainsi, il n'est pas nécessaire de scruter en permanence l'état de la broche d'entrée pour réagir très rapidement en cas de changement.

Des configurations permettent d'indiquer si l'on veut surveiller des fronts montants/descendants/les deux… et quelles sont les broches concernées.

La philosophie de développement avec les interruptions est omni-présente et indispensable sur les systèmes embarqués temps-réels.

On dispose de plusieurs priorités d'interruptions. Ainsi, une interruption de haute priorité peut interrompre une interruption de moins haute priorité. La tâche de fond, en tant que tâche de plus basse priorité, ne peut s'exécuter que lorsqu'aucune interruption n'est en court.

4 Informations complémentaires sur le fonctionnement du périphérique TIMER.

« Le périphérique Timer, c'est un peu comme un type avec un chronomètre. »

On peut lui demander par exemple :

- Préviens moi dans 125 ms.
- Appelle cette fonction toutes les 10 ms.
- Combien as tu compté depuis tout à l'heure ?
- Compte les fronts montant sur la broche d'entrée RB5 !
- Je vais m'endormir pour économiser de l'énergie, réveille moi dans 4 minutes.
- Compte jusqu'à 100, recommence tout le temps. Sors sur la broche RC4 un niveau haut si ton décompte est plus grand que 40. (cela revient à produire un signal PWM dont le rapport cyclique est 40%).
- …

Mais comment ça marche ?

- Un timer, c'est avant tout un compteur (constitué de bascules D et de quelques portes logiques).
- Il est cadencé par une source (généralement une horloge, parfois un évènement externe)
    - Souvent, cet évènement externe est beaucoup trop rapide, et on ajoute un prédiviseur. Le timer compte donc un multiple d'évènements.

Ok, le timer compte… et alors ?

- Lorsque le timer a atteint la valeur demandée, il génère un évènement (on dit qu'il lève un drapeau / **flag**).
- Pour exploiter cette information, on peut
    - **Scruter** en permanence l'état du flag, pour savoir si le timer à fini de compter
    - Demander au gestionnaire d'interruption **d'interrompre temporairement le déroulement normal du programme et d'exécuter une routine d'interruption** lorsque le flag est levé. Cette fonction est appelée automatiquement. Le processeur retourne ensuite à ce qu'il faisait avant d'avoir été interrompu.
- Dans tout les cas, il faudra penser à baisser le flag (on **acquitte** la demande d'interruption)

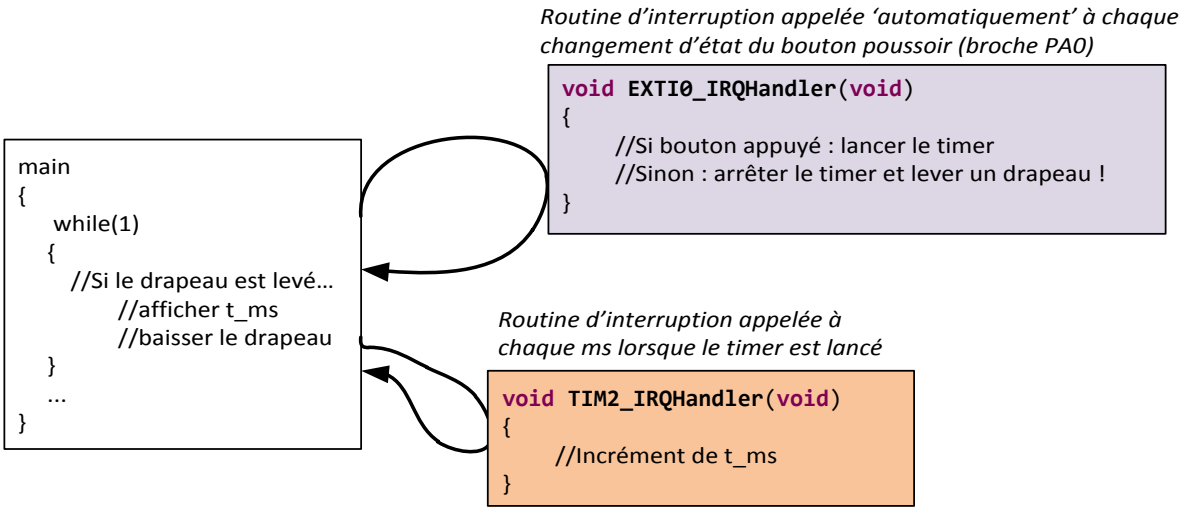
Ainsi, on peut facilement obtenir un appel périodique à une fonction ou un comptage précis du temps sans monopoliser le processeur !

5 Travail à faire

Nous allons :

- Configurer la broche RA0 avec déclenchement d'une interruption externe
- Construire un module logiciel timer qui nous permettra de compter le temps

Ces deux routines d'interruptions seront utilisées.
Le processeur les appelera automatiquement, et retournera ensuite à sa tâche de fond...



Dans la fonction d'initialisation de votre application, remplacez la ligne suivante, par une configuration de la broche en interruption :

```
~~BSP_GPIO_PinCfg(GPIOA, GPIO_PIN_0, GPIO_MODE_INPUT, ...);~~

//Configuration du port du bouton bleu en entrée, avec interruption
GPIO_InitTypeDef GPIO_InitStructure;
GPIO_InitStructure.Pin = GPIO_PIN_0;
GPIO_InitStructure.Mode = GPIO_MODE_IT_RISING_FALLING;
GPIO_InitStructure.Pull = GPIO_NOPULL;
GPIO_InitStructure.Speed = GPIO_SPEED_FAST;
GPIO_InitStructure.Alternate = 0;
HAL_GPIO_Init(GPIOA, &GPIO_InitStructure);
```

Ajoutez également cette ligne pour autoriser les interruptions

```
NVIC_EnableIRQ(EXTI0_IRQn);
```

A ce stade, vous pouvez compiler votre programme.
En revanche, si vous testez votre programme en l'état, vous allez rencontrer un problème : lorsque vous appuyez sur le bouton, le processeur va chercher à exécuter la routine d'interruption, qui n'a pas été renseignée… Si cela vous amuse, vous pouvez tester…

Ajoutez maintenant ceci :

Dans APPLI_Init :

```
TIMER2_init_for_1ms();
```

Dans appli.c :

```
static volatile uint32_t t_ms;         //Durée en ms
static volatile bool_e flag_new_value; //pour informer la tâche de fond
//Routine d'interruption pour la ligne d'interruption externe 0.
void EXTI0_IRQHandler(void)
{
	if(__HAL_GPIO_EXTI_GET_IT(GPIO_PIN_0) != RESET) //Normalement forcément vrai
	{
		__HAL_GPIO_EXTI_CLEAR_IT(GPIO_PIN_0);  //On acquitte l'interruption.
		if(HAL_GPIO_ReadPin(GPIOA, GPIO_PIN_0))      //Front montant
		{
			t_ms = 0;
			TIMER2_run();        //On lance le timer
		}
		else   //Front descendant
		{
			TIMER2_stop();             //On arrête le timer
			flag_new_value = TRUE;     //On lève le flag
		}
	}
}
```

En tâche de fond (dans la fonction **appli_process_main()**), on utilise le flag_new_value pour traiter chaque nouvelle mesure. Par exemple :

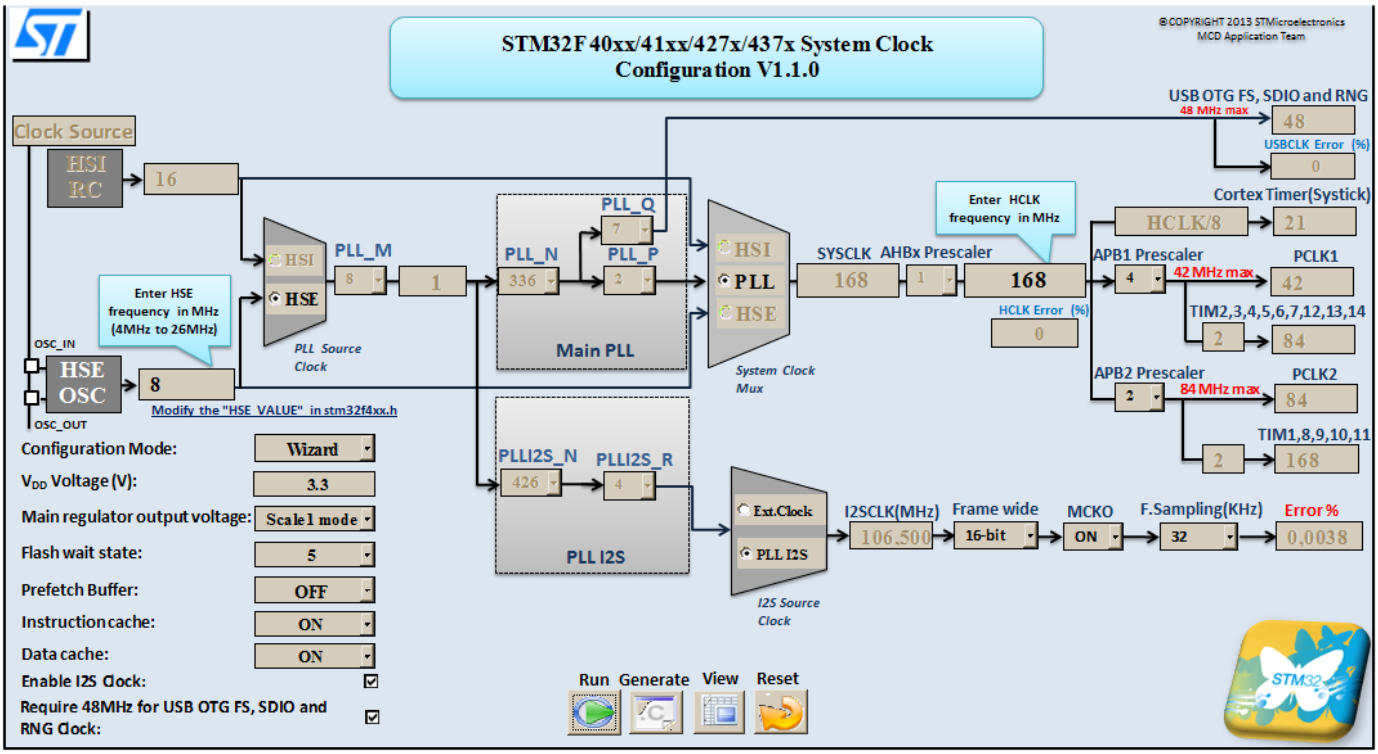
```
if(flag_new_value)
{
	flag_new_value = FALSE;
	printf("%lu\n",t_ms);
}
```

Construction du module logiciel « timer »

Le périphérique timer compte des évènements…

Ces évènements peuvent être des fronts d'horloge. Les possibilités de configuration des différentes horloges qui parcourent ce microcontrôleur sont multiples.

La configuration actuellement proposée dans stm32f4_sys.c impose l'horloge PCLK2, présentée à l'entrée des timers 2,3,4,5,6,7,12,13 et 14 à **84MHz.**



Construisez un module logiciel timer.c/h fournissant les fonctions suivantes :

```
void TIMER2_init_for_1ms(void); //initialise et paramètre le timer
void TIMER2_run(void);          //lance le timer
void TIMER2_stop(void);         //arrête le timer
void TIM2_IRQHandler(void);     //Interruption provoquée par le timer
void TIMER2_user_handler_it_1ms(void); //fonction de l'utilisateur
```

La fonction **`TIMER2_init_for_1ms`** initialise le timer 2, pour qu'il produise la levée d'une interruption toutes les ms.

Il faut définir une structure dans le fichier timer.c :

```
static TIM_HandleTypeDef Tim2_Handle;
```

Tout d'abord, on active l'horloge du périphérique timer.

```
__HAL_RCC_TIM2_CLK_ENABLE();
```

Puis, on active les interruptions pour le timer 2.

```
HAL_NVIC_SetPriority(TIM2_IRQn , 0, 1);
HAL_NVIC_EnableIRQ(TIM2_IRQn);
```

On remplit la structure Tim2_Handle avec les paramètres souhaités. Seuls les champs **Instance** et **Init** doivent être fournis : L'instance est TIM2.
Init contient les 5 champs de la structure TIM_Base_InitTypeDef :

`Prescaler`	[nombre de 0 à 65535 pour un prédiviseur de 1 à 65536] L'horloge comptée par le timer 2 est cadencée à 84MHz. Pour simplifier, et pour que le timer compte des microsecondes, il est judicieux de mettre régler le prédiviseur sur 84. Attention, pour faire cela, il faut mettre 83 dans ce champ.
`CounterMode`	On compte vers le haut. Consultez stm32f4xx_hal_tim.h pour savoir comment remplir ces champs
`Period`	1000 [µs]
`ClockDivision`	0
`RepetitionCounter`	0

.
Il faut ensuite faire appel aux fonctions ou macro suivantes :

`HAL_TIM_Base_Init`	On applique les paramètres d'initialisation
`__HAL_TIM_ENABLE_IT``(&Tim2_Handle, TIM_IT_UPDATE);`	On autorise les interruptions

La fonction `void TIMER2_run(void)` contient uniquement la ligne :

```
__HAL_TIM_ENABLE(&Tim2_Handle);
```

La fonction `void TIMER2_stop(void)` contient uniquement la ligne :

```
__HAL_TIM_DISABLE(&Tim2_Handle);
```

A ce stade, vous pouvez compiler votre programme. Pour le tester, il manque encore la routine d'interruption du timer !

Lorsque le timer arrive à la fin de son décompte, il déclenche une requête d'interruption. Le processeur se rend immédiatement exécuter la routine d'interruption correspondante, abandonnant temporairement ce qu'il était en train de faire. La voici :

```
void TIM2_IRQHandler(void)
{
	if(__HAL_TIM_GET_IT_SOURCE(&Tim2_Handle, TIM_IT_UPDATE) != RESET)
	{
		__HAL_TIM_CLEAR_IT(&Tim2_Handle, TIM_IT_UPDATE); //Acquittement
		TIMER2_user_handler_it_1ms();
	}
}
```

Cette fonction TIM2_IRQHandler NE DOIT PAS être appelée directement par l'utilisateur...
Nous n'avons PAS le choix du nom de cette fonction. (Défini dans startup_stm32.s)

Une bonne pratique de conception consiste à écrire des modules logiciels indépendants et réutilisables pour chaque projet. La brique logicielle « timer » a donc vocation à être construite sans lien avec un projet en particulier… Ainsi, il est judicieux de ne pas écrire de code lié uniquement à l'application dans timer.c. C'est la raison pour laquelle la fonction `TIMER2_user_handler_it_1ms()` est appelée. On ne sait pas ce que fait cette fonction. Elle est définie par l'utilisateur, dans son application. (dans appli.c par exemple).

Vous pouvez alors tester votre module logiciel timer. Que remarque-t-on ?