Table des matières

I) Mise en œuvre d'une interruption sur un port GPIO

Le but de ce tutoriel est la mise en œuvre d'une interruption sur un port GPIO. Dans cet exemple j'ai eu besoin qu'une interruption soit déclenchée chaque fois qu'un niveau passait de bas à haut ou de haut à bas sur le port PB8 d'une carte NUCLEO-L476RG.

Pour ce tutoriel, nous modifierons un projet déjà existant.

I-1) Recherche de la documentation sur le mode EXTI des GPIO

Voici ce que nous trouvons dans le « *user manual* UM1884 *Description of STM32L4/L4+ HAL and Low-layer drivers* »

GPIOs

GPIO HAL APIs are the following:

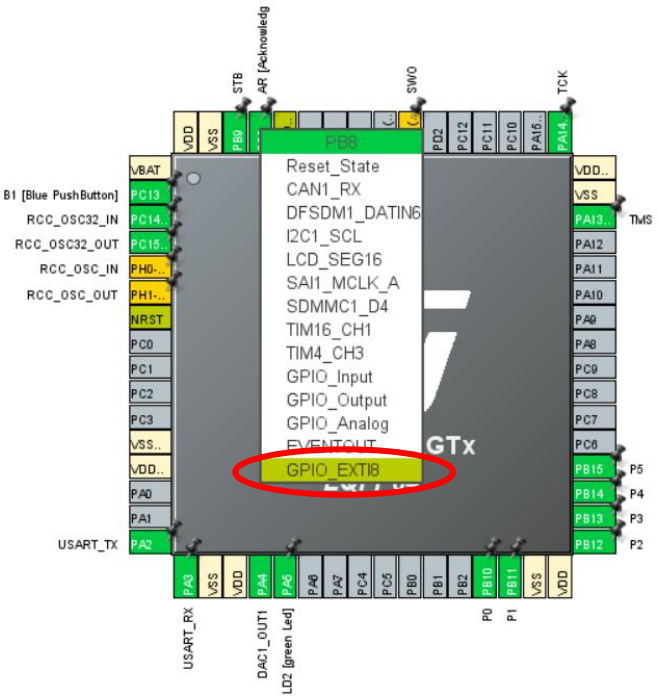
- HAL_GPIO_Init() / HAL_GPIO_DeInit()
- HAL_GPIO_ReadPin() / HAL_GPIO_WritePin()
- HAL_GPIO_TogglePin ().

In addition to standard GPIO modes (input, output, analog), the pin mode can be configured as EXTI with interrupt or event generation.

When selecting EXTI mode with interrupt generation, the user must call HAL_GPIO_EXTI_IRQHandler() from stm32l4xx_it.c and implement HAL_GPIO_EXTI_Callback()

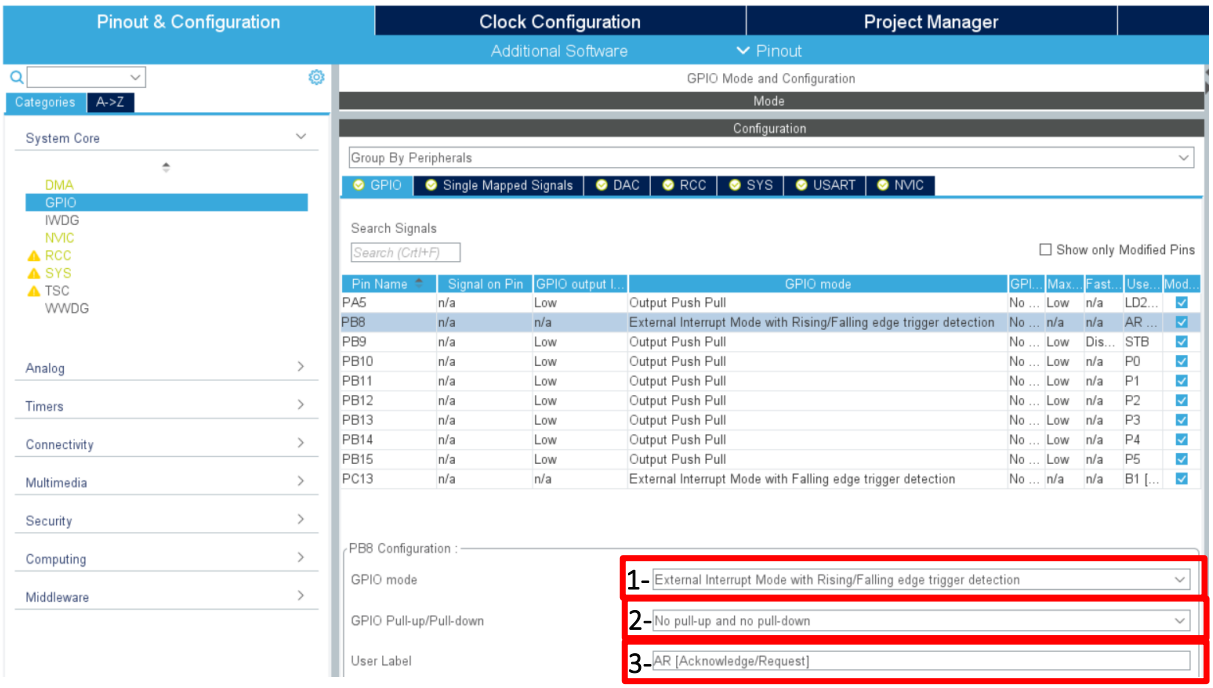
I-2) Configuration à réaliser dans STM32CubeMX

I-2-a) Choix du GPIO PB8 et de sa configuration



Il suffit de **cliquer GPIO PB8** et de sélectionner « **GPIO_EXTI8** ». On voit que c'est la ligne d'interruption **N°8** qui est affectée à ce GPIO.

I-2-b) Personnalisation du GPIO PB8



1- puisque l'on veut une interruption sur la transition bas → haut mais aussi haut→ bas, il suffit de l'indiquer dans la liste « **GPIO mode** »

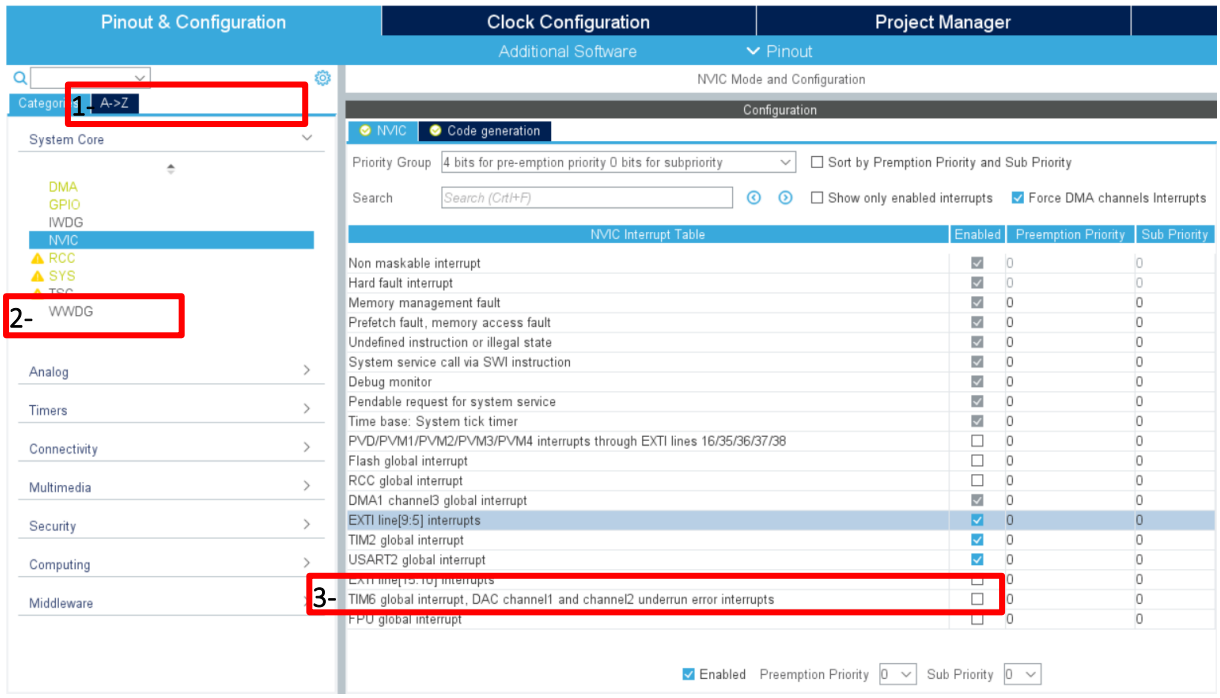
2- Le signal provient d'un circuit dont les niveaux sont parfaitement définis, nous n'avons pas besoin de résistance de *pull-up* ni *pull-down*

3- On en profite pour donner un nom représentant la fonctionnalité de ce signal. En l'occurrence, il s'agit d'un signal « ***A**cknowledge/**R**equest*», on l'appellera « **AR** ». On voit qu'il est possible de mettre un commentaire entre crochets, on ne s'en privera donc pas.

I-2-c) Activation des interruptions

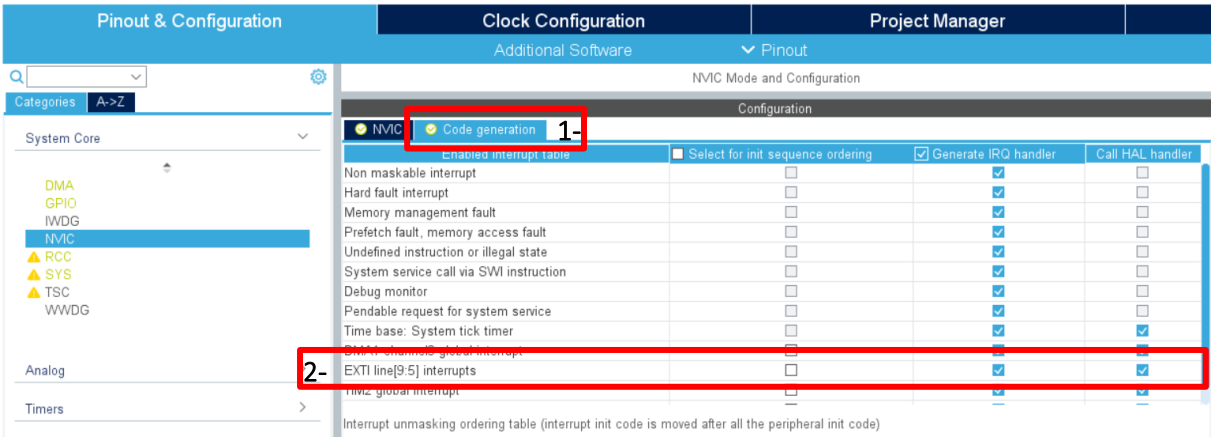
Nous avons vu lors du choix du **GPIO PB8**, que l'option **EXTI** est associée à la ligne d'interruption **N°8**. Il faut donc s'assurer de l'activation de cette ligne d'interruption. Nous allons donc demander à STM32CubeMX de générer tout le code pour cette réalisation.

1- Rendons-nous dans dans « **Pinout & Configuration** » ;
2- Cliquons sur **NVIC** ;
3- Faisons apparaitre la coche au croisement de la colonne « ***Enabled*** » et de la ligne « EXTI line [9 :5] interrupts » (*puisque l'interruption 8 est comprise dans l'intervalle [9 :5]*) ;



I-2-d) Assurons-nous que le code va être généré…

Pour que STM32CubeMX réalise le maximum de ce que nous demande le « *user manual UM1884 Description of STM32L4/L4+ HAL and Low-layer drivers* », **1-** rendons-nous dans l'onglet « ***Code generation*** » et **2-** cocher les 2 cases comme ci-dessous.



I-2-e) Demandons la génération du code

Il suffit d'appuyer sur le bouton habituel : 

I-3) Programmation dans Keil µVision

I-3-a) Vérifions que le code a bien été généré

Après avoir rechargé le nouveau code généré par STM32CubeMX dans notre IDE Keil µVision et en recherchant où se trouve la fonction « HAL_GPIO_EXTI_IRQHandler», on peut constater qu'elle est appelée dans la procédure « EXTI9_5_IRQHandler(void) » qui se trouve dans le fichier « stm32l4xx_it.c ». Tout va donc très bien, STM32CubeMX a fait le maximum pour nous !

stm32l4xx_it.c

```
215
216 /**
217   * @brief This function handles EXTI line[9:5] interrupts.
218   */
219 void EXTI9_5_IRQHandler(void)
220 {
221   /* USER CODE BEGIN EXTI9_5_IRQn 0 */
222
223   /* USER CODE END EXTI9_5_IRQn 0 */
224   HAL_GPIO_EXTI_IRQHandler(GPIO_PIN_8);
225   /* USER CODE BEGIN EXTI9_5_IRQn 1 */
226
227   /* USER CODE END EXTI9_5_IRQn 1 */
228 }
229
```

Des instructions du *user manual*, ne restera qu'à implémenter la fonction de ***callback***.

Recherchons maintenant « HAL_GPIO_EXTI_Callback » et voici ce que l'on trouve dans le fichier « stm32l4xx_hal_gpio.c » :

stm32l4xx_hal_gpio.c

```
523
524 /**
525   * @brief  EXTI line detection callback.
526   * @param  GPIO_Pin Specifies the port pin connected to corresponding EXTI line.
527   * @retval None
528   */
529 __weak void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin)
530 {
531   /* Prevent unused argument(s) compilation warning */
532   UNUSED(GPIO_Pin);
533
534   /* NOTE: This function should not be modified, when the callback is needed,
535            the HAL_GPIO_EXTI_Callback could be implemented in the user file
536    */
537 }
```

On nous le dit clairement, c'est à nous de l'implémenter dans le « user file », c'est-à-dire dans notre fichier « main.c » à l'endroit prévu pour.

I-3-b) Et écrivons ce qui nous revient

Dans ce qui suit, seul ce qui est entouré en rouge concerne notre exemple, le reste concerne le programme existant qui a été modifié pour l'occasion.

Dans l'exemple que nous avons pris, nous aurons besoin de **deux définitions** et d'un variable déclarée « **volatile** ».
Commençons pas les définitions :

```
/* Private define ------------------------------------------------------------*/
/* USER CODE BEGIN PD */
#define PI 3.1415926 // DAC1 : pour utilisation de la fonction décrite dans AN3126
#define true 1
#define false 0
/* USER CODE END PD */
```

Ici nous avons défini les constantes « true » et « false » pour des raisons de lisibilité. L'autre constante PI n'a rien à voir avec notre exemple, mais avec le programme existant que nous sommes en train de modifier.

Maintenant ajoutons la variable « **PretPourPhonemeSuivant** », inutile d'en chercher la signification, encore une fois c'est un programme existant en cours de modification.

```
/* USER CODE BEGIN PV */
char BufIndex ;           // USART2 : index qui pointera sur la position libre dans le buffer de reception
char BufRecept[100] ;     // USART2 : buffer  de réception d'une taille de 100 octets
char buffer[100];         // USART2 : buffer de manoeuvre utilisé dans la boucle principale
uint8_t BufReady=0 ;      // USART2 : booléen : 1 si une ligne terminée pa LF (0xA) a été reçue, 0 sinon
uint8_t BufUSART2[2];     // USART2 : buffer utilisé par les fonctions HAL dédiées à l'USART2
uint32_t sine_val[100];   //   DAC1 : tableau 100 positions max, pour les valeurs de la forme d'onde sinusoidale
int n = 100 ;                 //   DAC1 : Nb échantillons
volatile int PretPourPhonemeSuivant = false ;
// liste des phonèmes. Le N° d'ordre dans le tableau constitue le code à envoyer au SC-01-A
char *Phonemes_list[] = {"EH3", "EH2", "EH1", "PA0", "DT", "A2", "A1", "ZH", "AH2", "I3", "I2",
                         "I1", "M", "N", "B", "V", "CH", "SH", "Z", "AW1", "NG", "AH1", "OO1",
                         "OO", "L", "K", "J", "H", "G", "F", "D", "S", "A", "AY", "Y1", "UH3", "AH",
                         "P", "O", "I", "U", "Y", "T", "R", "E", "W", "AE", "AE1", "AW2", "UH2", "UH1",
                         "UH", "O2", "O1", "IU", "U1", "THV", "TH", "ER", "EH", "E1", "AW", "PA1", "STOP"};
/* USER CODE END PV */
```

Mais en revanche, il convient de s'attarder sur le mot-clé « volatile »… La variable « **PretPourPhonemeSuivant** » a été déclarée volatile pour que le compilateur ne cherche à réaliser aucune optimisation à son sujet (comme la mettre dans un registre), ce qui obligera le code généré à aller relire sa valeur en mémoire à chaque utilisation. En effet, cette variable est utilisée par la fonction de callback, elle est donc susceptible d'être modifiée de manière imprévisible lors de l'arrivée d'un front montant ou descendant sur GPIO B8. Il faut donc que la boucle infinie relise cette variable en mémoire à chaque fois qu'elle en a besoin.

Il se peut que cela fonctionne sans ce mot-clé et que le compilateur détecte par lui-même qu'il ne faut pas optimiser l'accès à cette variable, mais pour être certain, il vaut encore mieux être directif et l'imposer via ce mot-clé « **volatile** ».

Ecrivons maintenant notre fonction de callback :

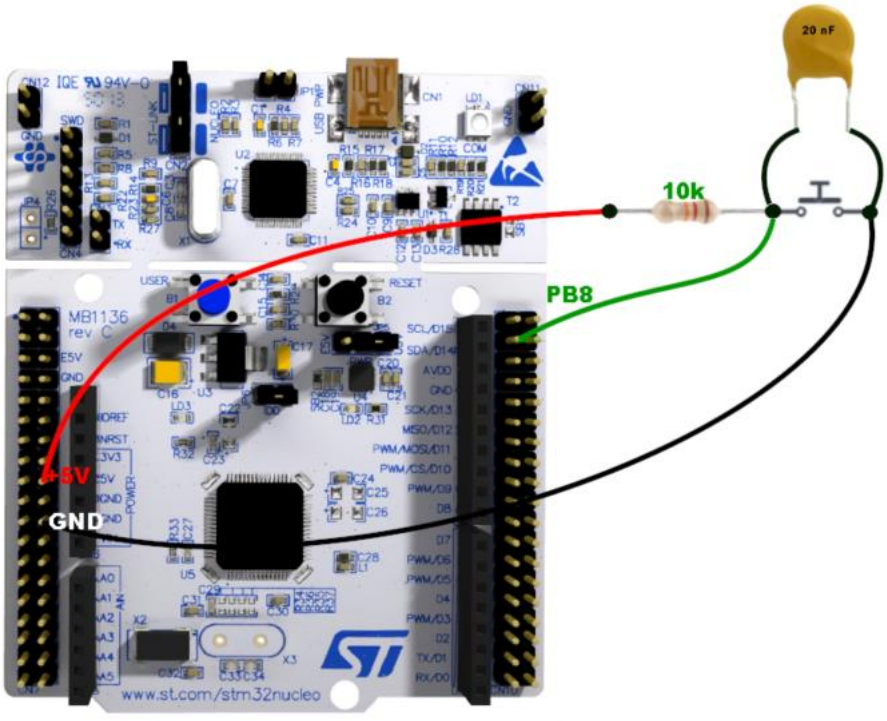
```
/* Private user code ---------------------------------------------------------*/
/* USER CODE BEGIN 0 */
void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin) // Fonction Callback gérant les interruptions sur les GPIO
{
 if (GPIO_Pin==AR_Pin)
 {
   PretPourPhonemeSuivant = true ;
 }
}

/* USER CODE END 0 */
```

Et enfin, le code dans la boucle infinie :

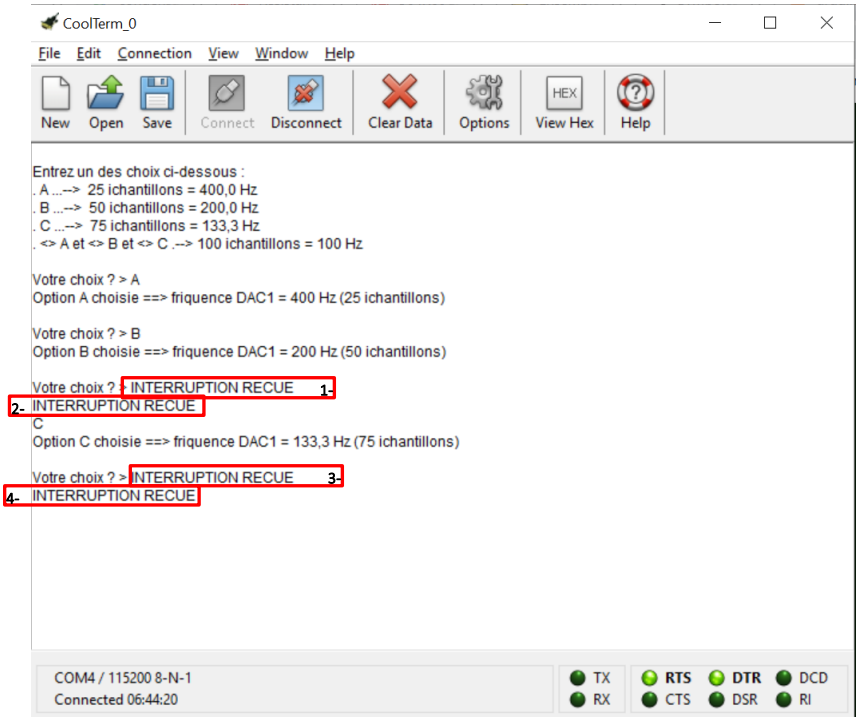
```
 while (1)
 {
   /* USER CODE END WHILE */

   /* USER CODE BEGIN 3 */
   if (PretPourPhonemeSuivant)
   {
     PretPourPhonemeSuivant=false;
     printf("INTERRUPTION RECUE\r\n");
   }
   if (BufReady)
   {
     sprintf(buffer, "%s", BufRecept);
     BufReady=0;

     //ATTENTION, pour que ce programme fonctionne, il faut que votre terminal série
     //ajoute uniquement "0x0A" après que la touche return ait été appuyée
     //Par exemple dans les options du terminal "Coolterm", il faut régler "Enter Key" Emulation sur LF
     //Si votre terminal ajoute 0X0D 0x0A, il faudra changer la condition du test "strlen(buffer)==2" etc

     HAL_DAC_Stop_DMA(&hdac1,DAC1_CHANNEL_1); // Arrêt du DAC1
     if(strlen(buffer)==1)
     {
       switch (buffer[0])
       {
         case 'A': //Option A
           printf("Option A choisie ==> fréquence DAC1 = 400 Hz (25 échantillons)\r\n");
           n = 25 ;
           break;
        case 'B': //Option B
           printf("Option B choisie ==> fréquence DAC1 = 200 Hz (50 échantillons)\r\n");
           n = 50 ;
           break;
         case 'C': //Option C
           printf("Option C choisie ==> fréquence DAC1 = 133,3 Hz (75 échantillons)\r\n");
           n = 75 ;
           break;
         default:
           // instructions à exécuter si expression n'est égale à aucune des valeurs des case
           printf("L'option choisie \" %s \" est différente de A et de B et de C ==> fréquence DAC1 = 100
           n = 100 ;
           break;
       }
     }
     else
     {
       printf("L'option choisie \" %s \" est différente de A et de B et de C ==> fréquence DAC1 = 100 Hz (
       n = 100 ;
     }
     printf("\nVotre choix ? > "); // On réaffiche la demande pour un nouveau choix
     get_sineval(); // On reconstruit la forme d'onde avec le nombre d'échantillons choisi
     HAL_DAC_Start_DMA(&hdac1,DAC1_CHANNEL_1,sine_val,n,DAC_ALIGN_12B_R); // On redémarre le DAC1
   }
 }
 /* USER CODE END 3 */
```

I-4) Test du programme

Pour tester le fonctionnement de l'interruption sur le GPIO PB8, nous avons réalisé ce petit montage avec un bouton poussoir, un condensateur de 20 nF pour éviter les rebonds et une résistance de 10 k ohms de rappel au +5V.



1- Premier appui sur le poussoir
2- Premier relâchement du poussoir
3- Deuxième appui sur le poussoir
4- Deuxième relâchement du poussoir



Petite modification permettant de tester le type de front :

```
/* USER CODE BEGIN PV */
char BufIndex ;           // USART2 : index qui pointera sur la position libre dans le buffer de reception
char BufRecept[100] ;     // USART2 : buffer  de réception d'une taille de 100 octets
char buffer[100];         // USART2 : buffer de manoeuvre utilisé dans la boucle principale
uint8_t BufReady=0 ;      // USART2 : booléen : 1 si une ligne terminée pa LF (0xA) a été reçue, 0 sinon
uint8_t BufUSART2[2];     // USART2 : buffer utilisé par les fonctions HAL dédiées à l'USART2
uint32_t sine_val[100];   //   DAC1 : tableau 100 positions max, pour les valeurs de la forme d'onde sinusoidale
int n = 100 ;                //   DAC1 : Nb échantillons
volatile int PretPourPhonemeSuivant = false ;
volatile int CodePhonemeRecu = false ;
// liste des phonèmes. Le N° d'ordre dans le tableau constitue le code à envoyer au SC-01-A
char *Phonemes_list[] = {"EH3", "EH2", "EH1", "PA0", "DT", "A2", "A1", "ZH", "AH2", "I3", "I2",
                         "I1", "M", "N", "B", "V", "CH", "SH", "Z", "AW1", "NG", "AH1", "OO1",
                         "OO", "L", "K", "J", "H", "G", "F", "D", "S", "A", "AY", "Y1", "UH3", "AH",
                         "P", "O", "I", "U", "Y", "T", "R", "E", "W", "AE", "AE1", "AW2", "UH2", "UH1",
                         "UH", "O2", "O1", "IU", "U1", "THV", "TH", "ER", "EH", "E1", "AW", "PA1", "STOP"};
/* USER CODE END PV */
```

```
104  void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin) // Fonction Callback gérant les interruptions sur les GPIO
105  {
106    if (GPIO_Pin==AR_Pin)
107    {
108      CodePhonemeRecu = !HAL_GPIO_ReadPin(GPIOB,AR_Pin) ;
109      PretPourPhonemeSuivant = !CodePhonemeRecu ;
110    }
111  }
```

```
while (1)
{
  /* USER CODE END WHILE */

  /* USER CODE BEGIN 3 */
  if (PretPourPhonemeSuivant)
  {
    PretPourPhonemeSuivant=false;
    printf("PRET POUR PHONEME SUIVANT\r\n");
  }
  if (CodePhonemeRecu)
  {
    CodePhonemeRecu=false;
    printf("CODE PHONEME RECU\r\n");
  }
```

La lecture du port nous permet d'identifier le type de transition et d'afficher le message adéquat.