Compte-rendu des TP de microprocesseur : oscilloscope

Séances 3 & 4

Adjustable blinking of the green LED:

Task 22 :

La fonction suivante permet de faire clignoter la LED à une fréquence de 1, 2 ou 4Hz.

```
6
7  void LED_SetFreqGreen(int f){
8      TIM2->ARR = 9999/f;
9      TIM2->CNT = 0;
0  }
1
```

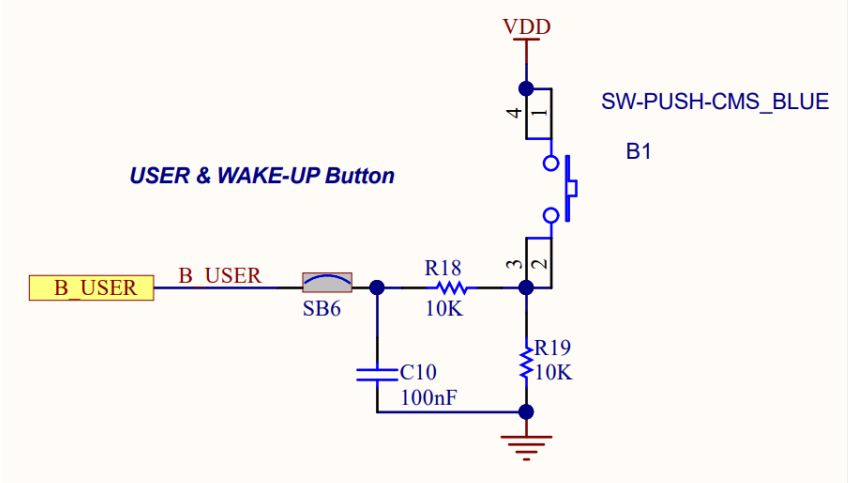
On n'oublie pas de remettre le compteur à 0 pour ne pas changer le ARR à une valeur inférieur au CNT (registre de comptage), ce qui aurait pour effet de ne plus appeler l'interruption du Timer 2.

Job Requirement 23:

```
if (g_ts.touchDetected && toggleParameter && release==1){
    release=0;
    if(g_ts.touchX[0]>240){
        if(frequence ==1){
            frequence=2;
        }else if (frequence ==2){
            frequence =4;
        }
    }else{
        if(frequence ==2){
                    frequence=1;
        }else if (frequence ==4){
                    frequence =2;
        }

    }
    GENE_ToggleSetFrequPin(g_ts.touchX[0]);
    LED_SetFreqGreen(frequence);
}else if (!g_ts.touchDetected){
    release=1;
}
```

Ce code nous permet de choisir la fréquence d'allumage de la LED au dos de la carte entre 1,2 ou 4 Hz en touchant l'écran. On a déjà ajouté le Blue button à ce moment-là du code.

Question 24/25/26:



C10 est une capacité de découplage.
On configure donc le Blue button pour que l'interruption activant la modification de la fréquence de clignotement par appui sur l'écran soit appelée lors de la pression sur le bouton. On va alors basculer du mode normal au mode paramètres et inversement par un simple appui.

Work requested 27:

L'utilisation d'interruptions pour le bouton permet au processeur de ne pas avoir à regarder tout le temps l'état du bouton libérant ainsi la capacité du processeur à gérer du code. Le bouton va se manifester de lui-même lorsqu'il est pressé.

Requested work 28:

Voir code Job Requirement 23.

Digital signal generation

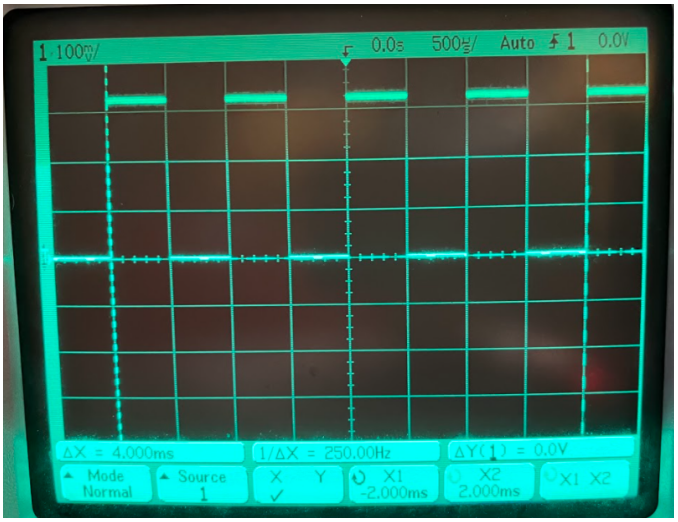
On va envoyer le signal carré de sortie sur la sortie D2 ce qui correspond au pin G6.
On configure le pin 6 du GPIOG en output. On place le bloc de bit n° 6 du registre MODER de GPIOG à 1 pour le faire.
Pour le basculement de l'état haut à l'état bas, on va utiliser le Timer 5. On configure entièrement le Timer 5 avec PSC=99 et ARR=499 afin d'avoir une fréquence de 1kHz en sortie.
Et on écrit :

```
void GENE_TogglePin(){
    if (GPIOG->ODR){
        GPIOG->ODR=GPIOG->ODR &~0x40;
    }else{
        GPIOG->ODR=GPIOG->ODR |0x40;
    }
}
```

Qui est appelé dans l'interruption du Timer 5.
Et on fait varier la fréquence avec :

```
void GENE_ToggleSetFrequPin(int freq){
        int f1Hz=499999;
        TIM5->ARR=f1Hz/freq;
        TIM5->CNT=0;
}
```

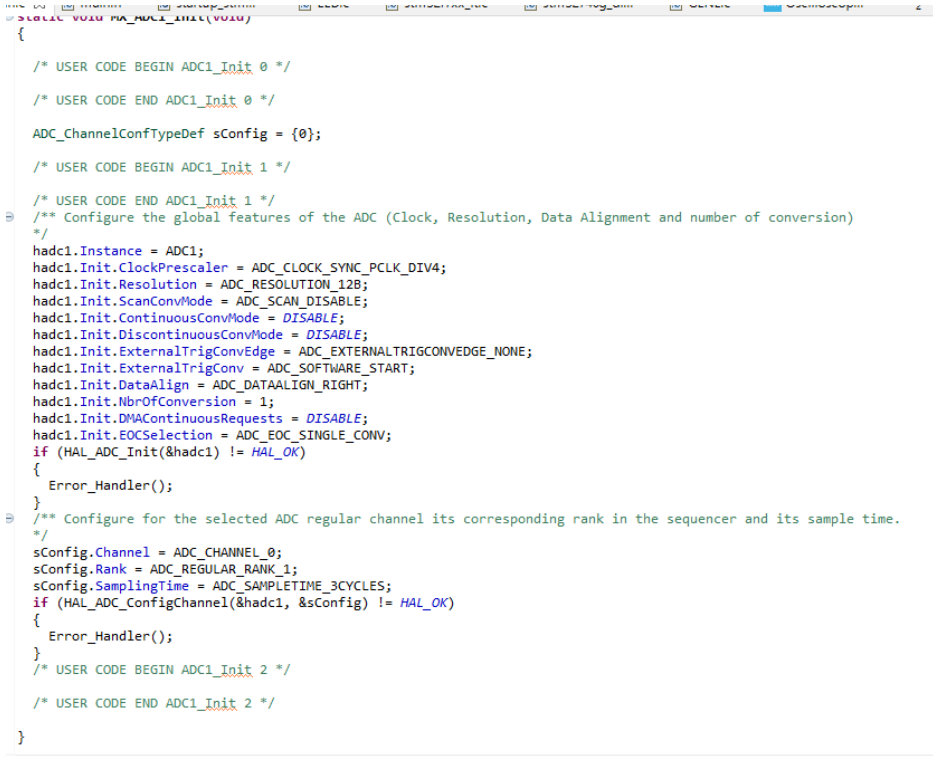
Qu'on appelle dans le Timer 4 Dans le mode parameter déjà programmé.
Ainsi, on observe le signal de sortie sur le pin D2, on trouve ceci :



On a bien le signal voulu.

Acquisition of an analog signal

On configure le Convertisseur analogique numérique n°1 sur l'entrée A0 de la carte :

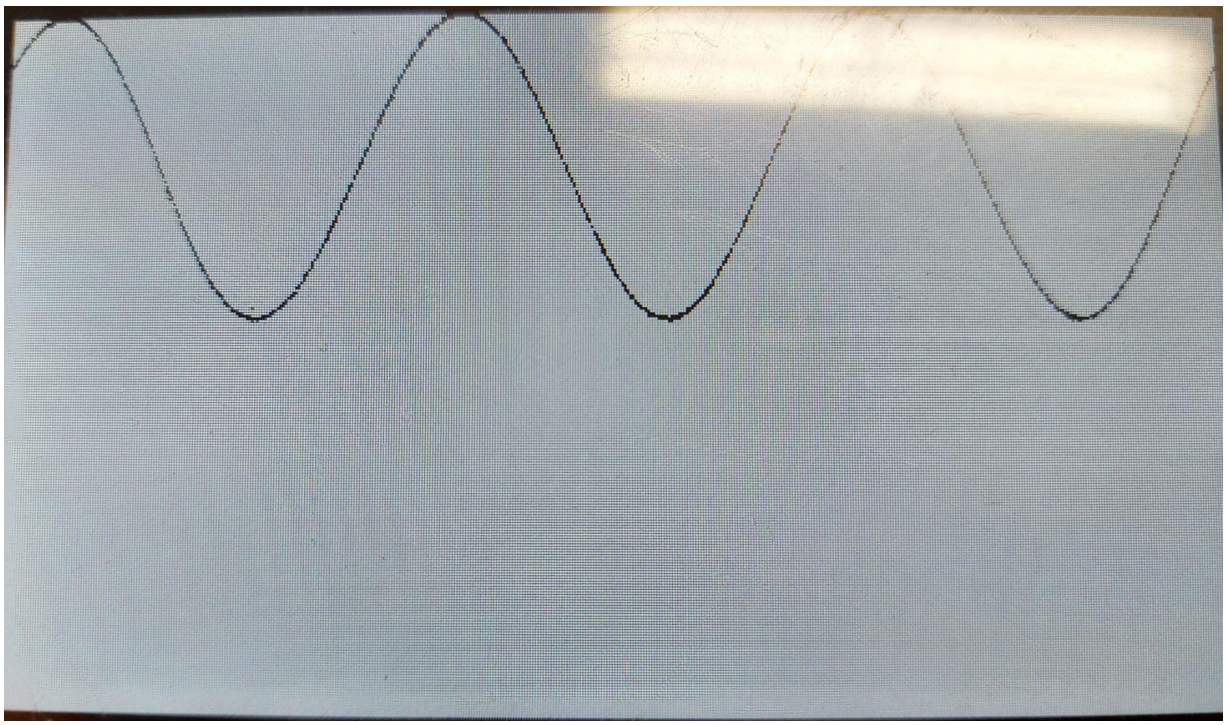


```
static void MX_ADC1_Init(void)
{

  /* USER CODE BEGIN ADC1_Init 0 */

  /* USER CODE END ADC1_Init 0 */

  ADC_ChannelConfTypeDef sConfig = {0};

  /* USER CODE BEGIN ADC1_Init 1 */

  /* USER CODE END ADC1_Init 1 */
  /** Configure the global features of the ADC (Clock, Resolution, Data Alignment and number of conversion)
  */
  hadc1.Instance = ADC1;
  hadc1.Init.ClockPrescaler = ADC_CLOCK_SYNC_PCLK_DIV4;
  hadc1.Init.Resolution = ADC_RESOLUTION_12B;
  hadc1.Init.ScanConvMode = ADC_SCAN_DISABLE;
  hadc1.Init.ContinuousConvMode = DISABLE;
  hadc1.Init.DiscontinuousConvMode = DISABLE;
  hadc1.Init.ExternalTrigConvEdge = ADC_EXTERNALTRIGCONVEDGE_NONE;
  hadc1.Init.ExternalTrigConv = ADC_SOFTWARE_START;
  hadc1.Init.DataAlign = ADC_DATAALIGN_RIGHT;
  hadc1.Init.NbrOfConversion = 1;
  hadc1.Init.DMAContinuousRequests = DISABLE;
  hadc1.Init.EOCSelection = ADC_EOC_SINGLE_CONV;
  if (HAL_ADC_Init(&hadc1) != HAL_OK)
  {
    Error_Handler();
  }
  /** Configure for the selected ADC regular channel its corresponding rank in the sequencer and its sample time.
  */
  sConfig.Channel = ADC_CHANNEL_0;
  sConfig.Rank = ADC_REGULAR_RANK_1;
  sConfig.SamplingTime = ADC_SAMPLETIME_3CYCLES;
  if (HAL_ADC_ConfigChannel(&hadc1, &sConfig) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN ADC1_Init 2 */

  /* USER CODE END ADC1_Init 2 */

}
```

Les valeurs de ARR et PSC du Timer 7, qui va lancer les différentes acquisitions de l'ADC, sont choisies pour avoir une fréquence de 480 Hz afin d'afficher une seconde entière sur l'écran de la carte à l'horizontal. On prend PSC = 2083 , ARR = 99 et on a donc f=479.84Hz.
Dans l'interruption du Timer 7 on demande à commencer une conversion de l'ADC avec la commande : "HAL_ADC_Start_IT(&hadc1)".
Et on écrit les actions à effectuer lors de la conversion dans ADC_IRQHandler():

```
t=(t+1)%480;
sortieADC=HAL_ADC_GetValue(&hadc1);
if (t>=479){
    BSP_LCD_Clear(LCD_COLOR_WHITE);
}
BSP_LCD_DrawPixel(t,sortieADC*200/4096,LCD_COLOR_BLACK);
```

La sortie ADC étant sur 12 bits, sa valeur va entre 0 et 4096. On affichera alors notre signal sur les 200 premiers pixels de la carte. L'entrée analogique ne pouvant aller qu'à 3.3V on veillera à ne pas dépasser cette valeur.
En envoyant un signal sinusoïdal de 3Hz avec Vpp=2V et un offset de 1 V sur l'entrée A0 on voit directement sur la carte :



L'accès à la valeur moyenne se fait par un simple calcul avec les valeurs du signal. Il suffit donc de stocker les valeurs pour calculer la valeur moyenne.

On écrit ensuite ce code :

```
if (t>=479){
    BSP_LCD_Clear(LCD_COLOR_WHITE);
}
```

Il permet d'effacer le signal une fois dessiné sur la carte pour éviter les dessins qui se superposent et les décalages. On a finalement un code s'aidant uniquement d'interruptions avec des timers et ne contenant rien dans le while(1) de la fonction principale libérant de la marge d'action pour le processeur.