Mark	/11

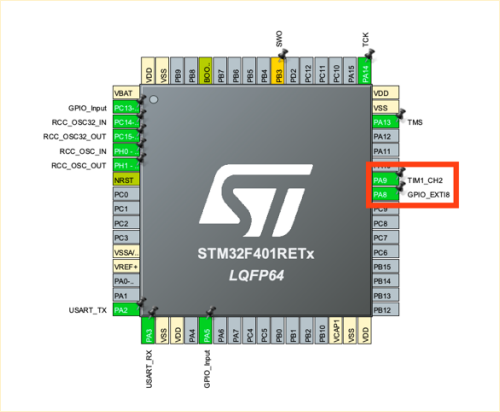
Team name:	*A1*		
Homework number:	*HOMEWORK 03*		
Due date:	06/10/23		
Contribution	NO	Partial	Full
Piombo			*x*
Fumagalli			*x*
Pierfederici			*x*
Zenoni			*x*
Ferraro			*x*
Notes: none			

Project name	Play a song		
Not done	Partially done (major problems)	Partially done (minor problems)	Completed
			x

Part 1a:

We looked for the SND_IN wire (microphone) and we found out that it is connected to microcontroller's pin PA8, while the speaker is connected to the pin PA9.

Then we set PA8 as GPIO_EXTI8 and PA9 as TIM1_CH2 to use the PWM mode of the timer 1 to generate the notes.

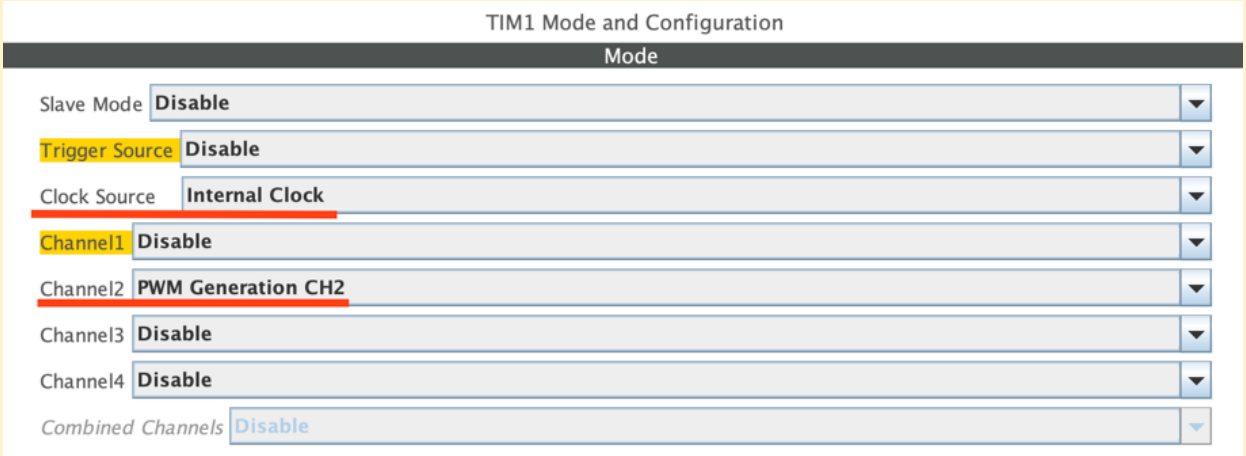


Then we needed to enable the microphone's interrupt (NVIC tab) and to correctly set it as only rising edge trigger detection (GPIO tab). We set the Preemption Priority of GPIO_EXTI8 as 1 in order to avoid conflicts with the interrupt of SysTick timer to implement the HAL_Delay function that we are going to use in the code.

NVIC Interrupt Table	Enabled	Preemption Priority	Sub Priority
Non maskable interrupt	☑	0	0
Hard fault interrupt	☑	0	0
Memory management fault	☑	0	0
Pre-fetch fault, memory access fault	☑	0	0
Undefined instruction or illegal state	☑	0	0
System service call via SWI instruction	☑	0	0
Debug monitor	☑	0	0
Pendable request for system service	☑	0	0
Time base: System tick timer	☑	0	0
PVD interrupt through EXTI line 16	☐	0	0
Flash global interrupt	☐	0	0
RCC global interrupt	☐	0	0
EXTI line[9:5] interrupts	☑	1	0
TIM1 break interrupt and TIM9 global interrupt	☐	0	0
TIM1 update interrupt and TIM10 global interrupt	☐	0	0
TIM1 trigger and commutation interrupts and TIM11 global interrupt	☐	0	0
TIM1 capture compare interrupt	☐	0	0
USART2 global interrupt	☐	0	0
FPU global interrupt	☐	0	0

The timer 1 has been set as a PWM generator on channel 2, with internal clock as clock source
The most relevant parameters are:

- Prescaler: it is set to 100-1 in order to use the excel file provided
- Counter Period (ARR): it is initially set to an arbitrary value because it is going to be changed depending on the note that is going to be played.
- Pulse (of PWM): it is set as ARR/2, thus it also changes depending of the note. The idea is to set the Duty Cycle to 50%.



We create a struct representing a note with its tone and its duration in TEMPO units. Then we defined the period of the PWM to play each note. “TEMPO” is 1/16 which corresponds to 75 ms.

```
/* Private typedef -----------------------------------------------------------*/
/* USER CODE BEGIN PTD */
struct note{
    int tone;
    int duration;
};
/* USER CODE END PTD */

/* Private define ------------------------------------------------------------*/
/* USER CODE BEGIN PD */
#define DO4  3205
#define DOD4 3031
#define RE4  2856
#define RED4 2700
#define MI4  2544
#define FA4  2406
#define FAD4 2269
#define SOL4 2142
#define SOLD4 2023
#define LA4  1908
#define LAD4 1802
#define SI4  1699

#define TEMPO  75
/* USER CODE END PD */
```

```
/* USER CODE BEGIN PV */
struct note score[]={
  {SOL4,6},
  {LA4,2},
  {SOL4,4},
  {FA4,4},
  {MI4,4},
  {FA4,4},
  {SOL4,8},
  {RE4,4},
  {MI4,4},
  {FA4,8},
  {MI4,4},
  {FA4,4},
  {SOL4,8},
  {SOL4,6},
  {LA4,2},
  {SOL4,4},
  {FA4,4},
  {MI4,4},
  {FA4,4},
  {SOL4,8},
  {RE4,8},
  {SOL4,8},
  {MI4,4},
  {DO4,12}
};
/* USER CODE END PV */
```

In this array is reported the song that is used to test the configuration, which is the same given at the laboratory session.

When the microphone sends an interrupt request, the song is started with a custom function (playsong) that calls playnote function for each note of the song.

```
void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin) {
    switch(GPIO_Pin) {
    case GPIO_PIN_8:
            playsong();
            __HAL_GPIO_EXTI_CLEAR_IT(GPIO_Pin);
        break;
    default:
        break;
    }
}
```

```
void playsong() {

    int length;
    length = sizeof(song)/sizeof(song[0]);

    for(int i = 0; i < length; i++) {
        playnote(song[i]);
    };

}
```

__HAL_GPIO_EXTI_CLEAR_IT is called when the song finishes to clear possible interrupt request risen while the song was playing.

The “playnote” function is used to set the PWM period and pulse depending on the input parameter of the note itself. This block of code is copied from MX_TIM1_Init function (the configuration function that is written by the IDE according to what we set in the GUI).

```
void playnote(struct note note_playing){

    TIM_ClockConfigTypeDef sClockSourceConfig = {0};
    TIM_MasterConfigTypeDef sMasterConfig = {0};
    TIM_OC_InitTypeDef sConfigOC = {0};
    TIM_BreakDeadTimeConfigTypeDef sBreakDeadTimeConfig = {0};

    htim1.Instance = TIM1;
    htim1.Init.Prescaler = 100-1;
    htim1.Init.CounterMode = TIM_COUNTERMODE_UP;
    htim1.Init.Period = note_playing.tone;
    htim1.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
    htim1.Init.RepetitionCounter = 0;
    htim1.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_DISABLE;
    if (HAL_TIM_Base_Init(&htim1) != HAL_OK)
    {
    Error_Handler();
    }
    if (HAL_TIM_PWM_Init(&htim1) != HAL_OK)
    {
    Error_Handler();
    }

    sConfigOC.OCMode = TIM_OCMODE_PWM1;
    sConfigOC.Pulse = note_playing.tone/2;
    sConfigOC.OCPolarity = TIM_OCPOLARITY_HIGH;
    sConfigOC.OCNPolarity = TIM_OCNPOLARITY_HIGH;
    sConfigOC.OCFastMode = TIM_OCFAST_DISABLE;
    sConfigOC.OCIdleState = TIM_OCIDLESTATE_RESET;
    sConfigOC.OCNIdleState = TIM_OCNIDLESTATE_RESET;
    if (HAL_TIMEx_ConfigBreakDeadTime(&htim1, &sBreakDeadTimeConfig) != HAL_OK)
    {
      Error_Handler();
    }
```

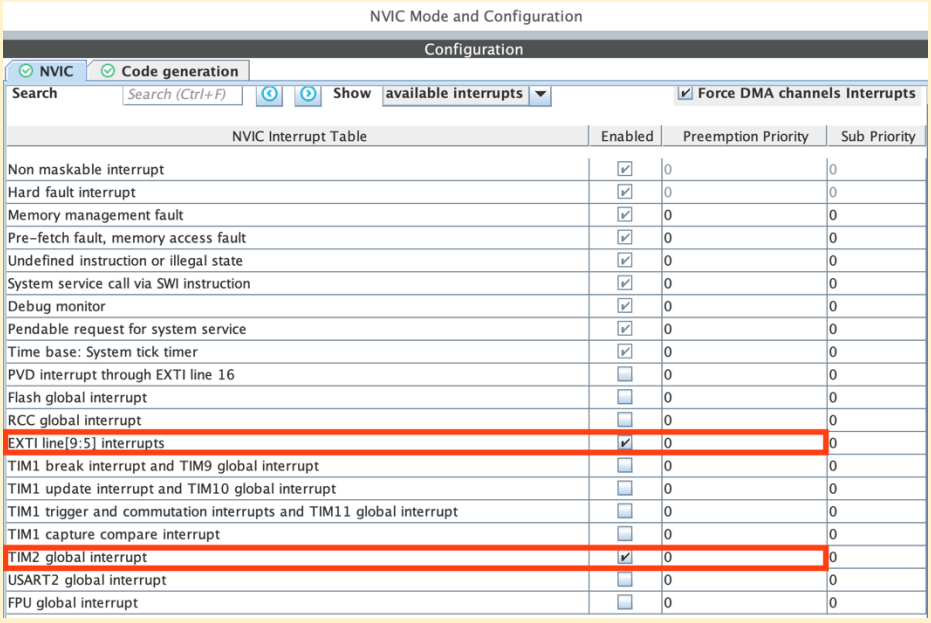
At the very end of playnote function the PWM is turned on (with the HAL_TIM_PWM_START function) for a time span of duration*TEMPO (which is the note duration in ms) and then turned off with the HAL_TIM_PWM_STOP function.

```
HAL_TIM_PWM_Start(&htim1, TIM_CHANNEL_2);
HAL_Delay(noteplaying.duration*TEMPO);
HAL_TIM_PWM_Stop(&htim1, TIM_CHANNEL_2);
```

Part 1b:

Starting from the previous project we describe here only the differences for the part b.

To remove the HAL_delay function we need to use a timer (TIM2) that triggers an interrupt to count the time flow.



NVIC Interrupt Table	Enabled	Preemption Priority	Sub Priority
Non maskable interrupt	☑	0	0
Hard fault interrupt	☑	0	0
Memory management fault	☑	0	0
Pre-fetch fault, memory access fault	☑	0	0
Undefined instruction or illegal state	☑	0	0
System service call via SWI instruction	☑	0	0
Debug monitor	☑	0	0
Pendable request for system service	☑	0	0
Time base: System tick timer	☑	0	0
PVD interrupt through EXTI line 16	☐	0	0
Flash global interrupt	☐	0	0
RCC global interrupt	☐	0	0
EXTI line[9:5] interrupts	☑	0	0
TIM1 break interrupt and TIM9 global interrupt	☐	0	0
TIM1 update interrupt and TIM10 global interrupt	☐	0	0
TIM1 trigger and commutation interrupts and TIM11 global interrupt	☐	0	0
TIM1 capture compare interrupt	☐	0	0
TIM2 global interrupt	☑	0	0
USART2 global interrupt	☐	0	0
FPU global interrupt	☐	0	0

Without the HAL_Delay function we don't need anymore the different priority level of the 2 interrupt sources.

We configured the timer 2 parameters directly in the MX_TIM2_Init function in the main.c code, so we are able to exploit the TEMPO define. This configuration triggers an interrupt every TEMPO ms:

```
static void MX_TIM2_Init(void)
{

  /* USER CODE BEGIN TIM2_Init 0 */

  /* USER CODE END TIM2_Init 0 */

  TIM_ClockConfigTypeDef sClockSourceConfig = {0};
  TIM_MasterConfigTypeDef sMasterConfig = {0};

  /* USER CODE BEGIN TIM2_Init 1 */

  /* USER CODE END TIM2_Init 1 */
  htim2.Instance = TIM2;
  htim2.Init.Prescaler = 8400-1;
  htim2.Init.CounterMode = TIM_COUNTERMODE_UP;
  htim2.Init.Period = TEMPO*10;
  htim2.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
  htim2.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_DISABLE;
```

The song starts when the microphone sends an interrupt request

```
void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin) {
    if(GPIO_Pin == GPIO_PIN_8) {
        if (gpio_interrupt_enable) {
            gpio_interrupt_enable = 0;              // flag to ignore interrupt request from the microphone till the song is finished
            playnote(song[0]);                      // start the song
        }
    }
}
```

Now our playnote function sets the PWM as the HW_03a, but only starts the note and the timer interrupt function

```
HAL_TIM_PWM_Start(&htim1, TIM_CHANNEL_2);                                   // start the note
HAL_TIM_Base_Start_IT(&htim2);                                              // start the timer interrupt function
```

We decided to use the timer to just count the time (in ms) to keep the ISR (HAL_TIM_PeriodElapsedCallback) as quick as possible, while we chose to handle the notes sequence in the while(1) of the main function.

```
void HAL_TIM_PeriodElapsedCallback (TIM_HandleTypeDef *htim) {
    if (htim == &htim2) {
        time = time + TEMPO;    // timer set as TEMPOms elapsed
    }
}
```

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

  /* USER CODE BEGIN 3 */
    if (time >= TEMPO*song[note_index].duration) {
          time = 0;
          HAL_TIM_PWM_Stop(&htim1, TIM_CHANNEL_2);
          HAL_TIM_Base_Stop_IT(&htim2);                     // timer interrupt function deactivated
          if (note_index == length-1) {                     // song is finished
              note_index = 0;
              __HAL_GPIO_EXTI_CLEAR_IT(GPIO_PIN_8);         // cancel possible interrupt request arisen when the song haven't finished yet
              gpio_interrupt_enable = 1;                    // now we are ready again to start the song
          }
          else {
              playnote(song[++note_index]);
          }
      }
}
/* USER CODE END 3 */
```

Here we check if the note duration time has passed, with the time variable incremented by the timer. Every time a note finishes we stopped the PWM and also the timer interrupt function, to avoid unnecessary interrupt requests.