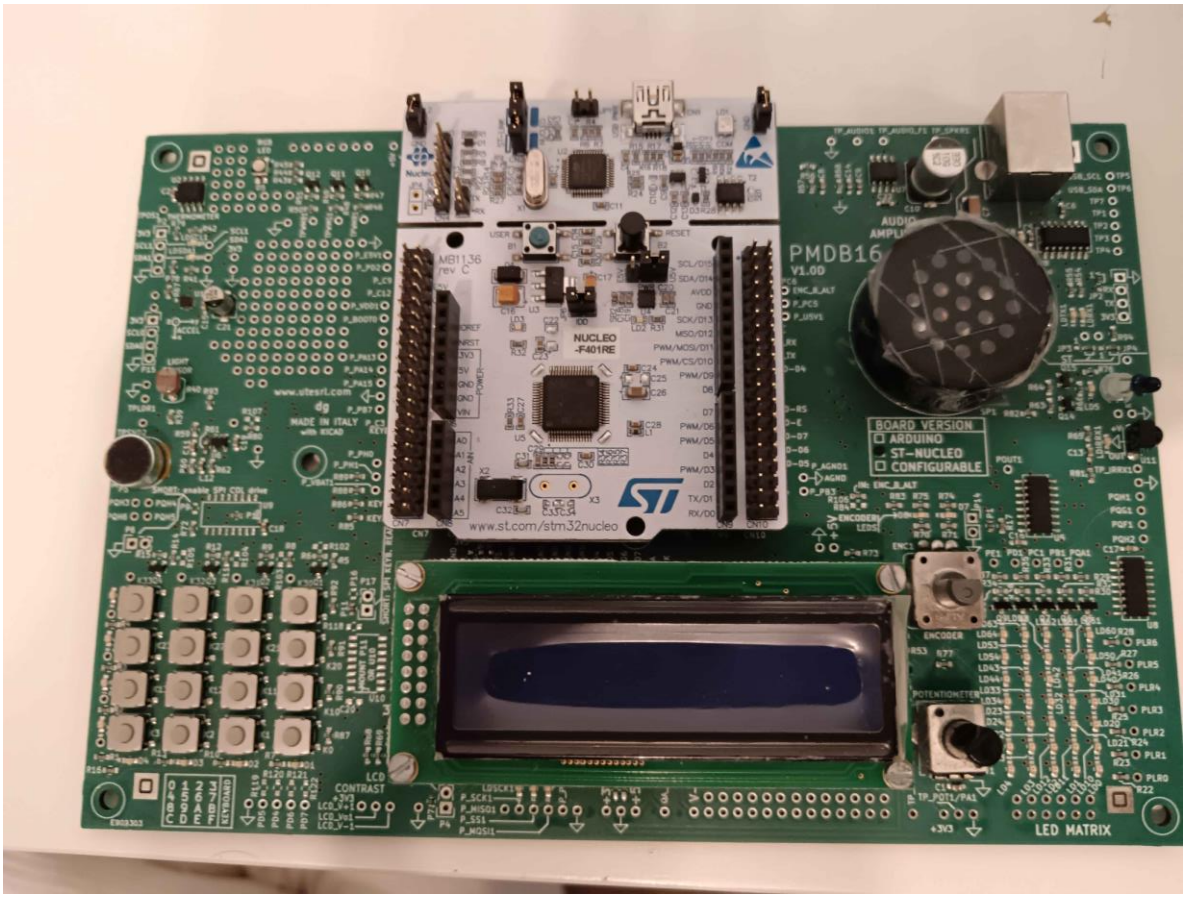
Computer Science Team B4

Sensor system

Homework Report



HOMEWORK NUMBER 3

2024 - 2025

POLITECNICO MILANO 1863

Summary

I. Members work

Members	Exercise 1	Exercise 2	General
Cazin Némo	✓		
Adrien Paliferro	✓	✓	
Heikki Leveelahti	✓	✓	Reviewed and edited report
Osmo Kieksi	✓	✓	Reviewed and edited report
Constantijn Coppers	✓	✓	Written the report

In general :

Our group strategy is that we first try to make the projects ourselves. Then, we discuss our approaches and make a consensus.

We agreed upon a rotation system where every week someone writes the report while the others critically review and edit the report.

This week the report was mainly written by Constantijn.

Link to the demonstrations: HW03

II. Exercise 1

General functionality:

In this project, the song '*London Bridge Is Falling Down*' will be played using the speaker on the NUCLEO board. Additionally, the blue push button will be used to start and stop the song. This will be implemented by an interrupt.

Implementation:

To obtain the desired functionality, the following steps are followed:

1. **PIN lookup**

The used peripherals with their corresponding PIN are displayed in the table below.

Peripheral	**PIN**	**I/O**
Speaker	PA9	Timer
Blue push button (B1)	PC13	Interrupt

2. **Board configuration**

Next, in the configuration tab, the I/O PINs are configured accordingly:

- PA9 : TIM1_CH2
- PC13 : GPIO_EXTI13

In order to generate tones using the speaker, one must apply a PWM signal to the speaker PIN. The frequency of the generated sound is then related to the frequency of the applied PWM.

Under Timers → TIM1, following settings are used to generate a C4 tone:

Mode pane

- Clock Source : Internal Clock
- Channel2 : PWM Generation CH2

Configuration pane

- Prescaler : 99
- Counter Period : 3206 - 1
- Pulse : 1603 - 1

Furthermore, pressing the blue push button should raise an interrupt and thus we must enable EXTI line[15:10] interrupts in System Core → NVIC. In the GPIO pane, we select External Interrupt Mode with Rising edge trigger detection.

3. Code implementation

First, we create a new typedef structure called ‘note’. It contains the tone and duration.

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

Secondly, we define the PINs, notes and the tempo (minimal duration of a note). A song is thus defined as a ‘sequence’ of note structures. We also define a flag that will be used for the interrupt.

```
/* Private define ------------------------------------------------------------*/
/* USER CODE BEGIN PD */
// Define the speaker PIN
#define SPKR_PIN GPIOA, GPIO_PIN_9
#define PUSHBUTTON_PIN GPIOC, GPIO_PIN_13
// Define the Notes
#define DO 3205
#define DOS 3031
#define RE 2856
#define RES 2699
#define MI 2544
#define FA 2405
#define FAS 2269
#define SOL 2141
#define SOLS 2023
#define LA 1908
#define LAS 1801
#define SI 1699
// Define the tempo
#define TEMPO 70
/* USER CODE END PD */

...

/* USER CODE BEGIN PV */
struct note song[] = { { SOL, 6 }, { LA, 2 }, { SOL, 4 }, { FA, 4 }, { MI, 4 },
{ FA, 4 }, { SOL, 8 }, { RE, 4 }, { MI, 4 }, { FA, 8 }, { MI, 4 }, { FA, 4 }, {
SOL, 8 }, { SOL, 6 }, { LA, 2 },
```

```
            { SOL, 4 }, { FA, 4 }, { MI, 4 }, { FA, 4 }, { SOL, 8 }, { RE, 8 },
{ SOL, 8 }, { MI, 4 }, { DO, 10 }};
int flag_song = 0;
/* USER CODE END PV */
```

To generate a specific tone, one must first reconfigure the TIMER settings appropriately. This is done by the function

```
void SetTone (int Tone) {
	// Reconfigure the TIMER with the correct parameters
	__HAL_TIM_SET_AUTORELOAD(&htim1, Tone);  		// Change the freq
	__HAL_TIM_SET_COMPARE(&htim1, TIM_CHANNEL_2, Tone / 2);  // DC of 0.5
	__HAL_TIM_SET_COUNTER(&htim1, 0);
}
```

A note is now generated by first setting the tone and then playing the tone for a certain duration.

```
void PlayNote (struct note Note) {
	SetTone(Note.tone);  			// Set the tone

	 // Play the tone for a certain time
	HAL_TIM_PWM_Start(&htim1, TIM_CHANNEL_2);
	HAL_Delay(TEMPO * Note.duration);
	HAL_TIM_PWM_Stop(&htim1, TIM_CHANNEL_2);
}
```

As stated earlier, a song is a sequence of notes (tone and duration). To play a song, we must simply iterate over the music sheet and play each note.

```
void PlayASong () {
	// Get the amount of iterations needed
	int length = sizeof(song) / sizeof(struct note);

	 // play the song as long as the flag is high (no int, raised)
	for (int i = 0; i < length; i++) {
			if (flag_song == 1) {
				PlayNote(song[i]);
			}
			else {
				break;
			}
		}
}
```

The starting/stopping of the song is controlled by the pushbutton. the flag_song will be used to indicate if the song should play or not. Thus, when pressing the blue pushbutton, the flag_song will be inverted (0 → 1 and 1 → 0).

```
void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin) {
	switch (GPIO_Pin) {
	case (GPIO_PIN_13):
		if (flag_song == 1) {
			flag_song = 0;
		} else {
			flag_song = 1;
		}
		break;
	}
}
```

Finally, in the while loop, the next code is used:

```
 /* Infinite loop */
 /* USER CODE BEGIN WHILE */
 while (1)
 {
	// Only play the song if flag is HIGH
	while (flag_song == 1) {
		PlayASong();
	}
   /* USER CODE END WHILE */
```

4. Execution

Link to the video: [HW03]

III. Exercise 2

General functionality:

Same functionality as Exercise 1, except for the fact that the HAL_Delay function will be replaced. More specifically, we will configure a new timer that will raise an interrupt when it overflows. The overflow time will be adjusted to the duration of the note to-be-played.

Implementation:

1. **PIN lookup**

The same PINs are used and configured as in exercise 1. We only add an extra debug LED that will toggle when the timer overflows.

Peripheral	**PIN**	**I/O**
LD2	PA5	Timer

2. **Board configuration**

The same board configuration as in exercise 1 is used. We set the LD2 PIN as output.

- PA5 : GPIO_Output

Timer 3 is configured with following settings:

Mode pane

- Clock Source : Internal Clock

Configuration pane

- Prescaler : 8399
- Counter Period : 9999
- Trigger Event Selection : Update Event

Also the 'TIM3 Global interrupt' is enabled.

3. **Code implementation**

We basically copy-paste the code of exercise 1. We will only adjust minor things.

First, we add the green LED PIN to the private defines and an extra timer flag to the private variables.

```
#define GREEN_LED_PIN GPIOA, GPIO_PIN_5
...
uint8_t flag_timer = 0;
```

The functions SetTone, PlayASong and HAL_GPIO_EXTI_Callback remain unchanged. However, the PlayNote function did change: we first set the correct tone by the SetTone function. Afterwards, we generate the tone by starting the PWM on the

speaker PIN. To stop the note, we rely on the interrupt generated by timer 3. We set the flag_timer to zero and we change the Timer 3 settings to match the note duration. We start the timer, and as long as the timer and song flag remain low and high respectively, the MCU is 'trapped' inside the while loop (doing nothing). When the timer overflows or the song flag is pulled down, the tone and timer 3 stops.

```
void PlayNote (struct note Note) {
       SetTone(Note.tone);
       HAL_TIM_PWM_Start(&htim1, TIM_CHANNEL_2);
       flag_timer = 0;
       __HAL_TIM_SET_AUTORELOAD(&htim3, (TEMPO * 10 * Note.duration) - 1);
       HAL_TIM_Base_Start_IT(&htim3);
       while (flag_timer == 0) {
       }
       HAL_TIM_PWM_Stop(&htim1, TIM_CHANNEL_2);
       HAL_TIM_Base_Stop_IT(&htim3);
}
```

When timer 3 overflows, an interrupt is generated and the flag timer is pulled up. Furthermore LD2 is toggled and is used to debug.

```
void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim)
{
   // Check if it's TIM3 that caused the interrupt
   if (htim->Instance == TIM3)
   {
       // Toggle the LED pin
       HAL_GPIO_TogglePin(GREEN_LED_PIN);
       // pull up the timer flag to exit the while loop in PlayNote
       flag_timer = 1;
   }
}
```

Finally, the timer is started and the same infinite while loop is used as in exercise 1.

```
int main(void)
{
...
/* USER CODE BEGIN 2 */
 HAL_TIM_Base_Start_IT(&htim3);
 /* USER CODE END 2 */
...
}
```

4. Execution

Link to the video: HW03

IV. Professors comments

EXERCISE 1 :
EXERCISE 2 :