|  |  |
| --- | --- |
| **Mark** |  |

|  |  |  |  |
| --- | --- | --- | --- |
| Team name: | *A2* | | |
| Homework number: | *HOMEWORK 2* | | |
| Due date: | 15/10/2023 | | |
|  |  |  |  |
| Contribution | NO | Partial | Full |
| Hui Jiang |  |  | *x* |
| Mattia Sironi |  |  | *x* |
| Gabriele Landi |  |  | *x* |
| Arturo Caliandro |  |  | *x* |
| Luigi Lizzini |  |  | *x* |
| Notes: | | | |

|  |  |  |  |
| --- | --- | --- | --- |
| Project name | MEMS Accelerometer | | |
| Not done | Partially done  (major problems) | Partially done  (minor problems) | Completed |
|  |  |  | *x* |
| We have done the whole homework, here is our explanation:  **Part 1:**  Project 2a: this project has been done during the last laboratory session and the code is the same as the one shown by the professor.  Project 2b: like Project 2a, although we made some optimizations: our version of the function playnote uses a few functions to change the AUTORELOAD register and COMPARE register and resets the timer counter every time a new tone is played.  Those functions are well-described in the document: “HAL functions.pdf”:        The code works as expected.  **Project 2c (also recognized as Homework 02 part 1):**  Starting from this project, we have encountered a lot of difficulties: it made us think a lot.  Firstly, we have configured the board as shown below:    We have configured the pin PA9 (which is connected to the speaker) as TIM1\_CH2, in order to generate a PWM signal, and the pin PA8 (which is connected to the microphone) as GPIO\_EXTI8, in order to detect sounds and in consequence generate a interrupt.  Secondly, we configured the TIM1\_CH2 as shown below:  Here it doesn’t really matter the value of the AUTORELOAD register and the pulse value (the value of the register COMPARE), because we are going to override them with different values according to the tone to play. In order to simplify calculations, we fix it to 99.  Before we explain the third step, we want to explain a curious thing that we have encountered: our first idea was based on the code of the previous project, i.e. to put the playsong function into the HAL\_GPIO\_EXTI\_Callback function, then everytime there is a sound, it would trigger the callback function and as this function contains the function playsong, therefore, it will play the sound.  This idea sounded right, but it’s not: according to our test, if done in this way, after the microphone detects the first sound from the environment, the speaker will play the first sound of the first tune forever.  After a while, we understood the problem. Inside the interrupt callback function we invoke the playsong function. The big amount of HAL\_Delay calls triggers the “Time Base: System tick” timer interrupt.  What we realized is that, by default, in the NVIC table, the EXTI interrupt and the “Time Base: System tick” timer have the same priority. It means that whenever the speaker begins to play the song, the microphone detects it and triggers the EXTI interrupt again, thus starting an infinite loop.  Therefore, to fix this problem we changed the priority of the EXTI interrupt, as shown below:    We change the Priority group to “1 bit for pre-emption priority, 4 bits for sub priority” and then in the row “EXTI line[9:5] interrupts”, we set the value 1, which means a lower priority.  Finally, in the main.c, we maintain everything as the previous project but add the callback function:    As the professor commented on the previous lecture, we add steps for the clear flags. And we have also proved it using our board.  **Homework02 2:**  As this project just consists in making again the Project2c without using the HAL\_Delay function, the configuration is almost the same as the previous project. Considering the hint, we chose the timer2 to replace the HAL\_Delay, and the timer1 channel 2 to generate the PWM signal.    We fixed the Prescaler to 8399 in order to simplify our calculation later. The value of the counter period doesn’t really matter, because we are going to override it according to the duration of the tune to play. The other settings on the board are the same as the previous project (although this time we don’t need to change the priority in NVIC, more about it later).  Secondly, we define some global variables in the main.c:  The variable song\_playing identifies if the song is over or not. The variable index identifies the order of the tune that it’s playing.  Then every time there is a sound, triggers the callback function:  The callback function checks if the previous song is over: if true, it will start the next song, otherwise it will do nothing. That’s also the reason why in this case we don’t need to change the priority on the NVIC board: the only input that starts the song is a legit external sound.  Then, the function change the value of song\_playing to 1, which means that now the song is been reproduced, and initializes the index to 0. Eventually, it calls the playnote function:  The first thing we do in this function is to stop the PWM (just in case, maybe it’s not necessary), and then it checks if the song is finished or not: if it’s finished, the program will stop the timer2, resets the flag of the timer2 and puts the song\_playing to false. Otherwise, it sets the timer1 channel 2 to generate the PWM signal according to the next tune to reproduce and sets the timer2 to count for the end of this tune: as the Prescaler is 8399, the counter increments every 0.1 ms, hence we set the period based on the duration of every tone. Thus, when the counter counts until this number, it will trigger its callback function as shown below:    Here are the descriptions of those HAL functions we have used in this project:        The code works as expected. | | | |
| Professor comments: | | | |