|  |  |
| --- | --- |
| **Mark** | **/11** |

|  |  |  |  |
| --- | --- | --- | --- |
| Team name: | *A05* | | |
| Homework number: | *HOMEWORK 03* | | |
| Due date: | 6/10/2024 | | |
|  |  |  |  |
| Contribution | NO | Partial | Full |
| Alessio Spineto |  |  | *x* |
| Riccardo Lamarca |  |  | *x* |
| Sofia Cecchetto |  |  | *x* |
| Annamaria De Togni |  |  | *x* |
| Emma Crespi |  |  | *x* |
| Notes: none | | | |

|  |  |  |  |
| --- | --- | --- | --- |
| Project name | Play a song | | |
| Not done | Partially done  (major problems) | Partially done  (minor problems) | Completed |
|  |  |  |  |
| **Part 1a:**  The first part of the assignment was to conclude the project “play a song” as in the slides: **play a song using the speaker when the microphone detects a loud sound.**  The first step was to locate on the green board schematic the corresponding pins for the microphone and the speaker, respectively: PA8 and PA9.  Using the GUI we configured the PINs: PA9 as TIM1\_CH2 and PA8 as GPIO\_EXTI8.    Next, we set the parameters for the PWM of TIM1 as shown in the image.  Since we wanted to reproduce musical notes of different frequency values, we set the PSC as the only constant parameter at 99 (reasonable value that would allow to span all the relevant frequencies with a duty cycle of 50%). This generates a square wave where the period is the one of the desired note. The values of the period and the pulse are going to change at runtime to reproduce the different notes (so we did not set them using the GUI). We set the internal clock as clock source and PWM generation on TIM1 channel 2, the one connected to the speaker.      We set the interrupt generation on the rising edge of the trigger detection, to allow the microphone to generate an IRQ (interrupt request), whenever it detects a sound. We also enabled interrupt lines between 5 and 9 since the microphone is on PIN 8.      In the main.c we defined the notes, setting their pulse value as shown in the excel file.  We defined a struct variable to store the pulse value (in the form of the defined note) and the duration of each note of the song. The duration of the note varies according to the music sheet as a multiple of 1/16. In our case 1/16 = 62 ms.    The song is stored in an array in which each note is a struct.    The start of the song is triggered by the interrupt sent by the speaker, which sets the variable *playing* = 1  In the while(1) of the main.c, the song is played using a for loop that loops through each note. The number of elements that compose the song is computed as  For each note, the *set\_note* function is called to set the correct parameters for the timer that generates the corresponding PWM. The timer is then started, and the note is played for the required amount of time using a blocking delay function. The duration of the delay is set as a multiple of 62ms, depending on the duration of the single note defined in the struct. After this delay time, the PWM is stopped.  The global variable *playing* is then set to 0 at the end of the song.  We used HAL\_Delay(5) just to separate the execution of the while(1). | | | |
| **Part 1b:**  The goal of the second part was to avoid using the HAL\_Delay function, substituting it with an interrupt.  In addition to the timer used in part 1a, we decided to use a second timer TIM2\_CHANNEL1. The purpose of this timer is to play the notes the correct amount of time.  We configured the timer so that it generates a PWM with the characteristics shown in the images below.  To compute them, we set the variable ‘pulse’ to:  Since the DC of the PWM is set to be 50%, the PWM generated by the timer will have a duty of 31ms and a period of 62 ms. We configured the other variables to accommodate this value:    According to this formula, we chose:   * ARR + 1 = 5280 * PSC + 1 = 1000   We then enabled the interrupt on the timer TIM2 to be of higher priority than the speaker’s interrupt.    In the main.c, the definition of the notes remains the same. In addition, we defined the following variables:     * *Playing* is used as a flag. It is set to 0 when the song is not playing, and set to 1 while the song is playing. * *Index\_note* is a variable used to cycle the whole song. It represents the note we are currently playing. Its minimum value is 0, while the maximum value is the length of the song - 1; * *Index\_duration* is a variable used to count the number of times the second timer (TIM2) generates an interrupt, so every 62ms. Since every note’s duration is a multiple of 62ms, we used it as a counter that gets set to 0 when the note starts playing, and increases up until the note’s duration value. * *Length* is defined as the number of notes in the song.   When the microphone detects a sound, an interrupt is generated. In the function below we controlled that the interrupt was generated by the right pin and that the speaker was not playing any sound. If this condition is true, we set the flag ‘playing’ equal to 1 and we call the function *start\_note*().    The void function start\_note() first checks if *index\_note* has reached the full length of the song. If this condition is true, we clear the interrupt line of the microphone and of the second timer, and set both *playing* and *index\_note*  to 0. If the condition in false it means that we must keep (or start) playing the song. In this case, we use the function *set\_note()* just like in the previous exercise. Then we start both timers.    When TIM2 goes in overflow, so 62ms have passed, an interrupt is generated. In the corresponding function *HAL\_TIM\_PeriodElapsedCallback,* we first check if the timer that generated the interrupt is the correct one and if the speaker is in function. If these conditions are correct, we call the function *stop\_note.*    In the void function *stop\_note()* we first check if the variable *index\_duration* has reached the duration of the note we are currently playing. If this is true, we reset *index\_duration* to 0, we increase *index\_note* and we stop the second timer. Finally, we call again *start\_note*. Instead, if the if condition is not satisfied, it means that the note has to be played for longer, so we increase *index\_duration* and exit the function.      **Notes for the Professor:**  During the tutoring hours we had the possibility to discuss with Arianna De Vecchi the use of interrupts.  She suggested to us to try using the clear function related to the interrupt request made by the microphone as an alternative to the *playing* variable, to check the status and ignore any interrupt that arrives from the speaker while the song is playing.    While trying to implement this alternative, we had some doubts about how concurrent interrupts are managed, in particular about how this "queue" is managed and what is being cleared.    For this reason, we wanted to ask if it's possible to clarify this concept during the next laboratory. | | | |
| Professor comments: | | | |