

<b>Mark</b>	<b>/11</b>
-------------	------------

Team name:	A14		
Homework number:	HOMEWORK 03		
Due date:	5/10/25		
Contribution	NO	Partial	Full
Francesca Biondi			x
Lorenzo Castelli			x
Mattia Di Mauro			x
Pietro Albriggi			x
Notes: none			

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

**Part 1:**

According to the “Hands-on Lab Schematics” document (page 3), the PWM\_SPKR pin, which is the GPIO line connected to the speaker, is assigned to CN5 pin 1. Referring to the Nucleo board user manual, CN5 pin 1 maps directly to the PA9 pin of the STM32 microcontroller.

In STM32CubeIDE, we configured the speaker GPIO (PA9) to operate as TIM1 Channel 2 (PWM Generation CH2), enabling it to drive the speaker through a PWM signal:

Homework 3 - delay.ioc - Pinout & Configuration

Pinout & Configuration
Clock Configuration
Project Manager
Tools

Categories A-Z

- System Core >
- Analog >
- Timers >
- RTC
- TIM1**
- TIM2
- TIM3
- TIM4
- TIM5
- TIM9
- TIM10
- TIM11

  

- Connectivity >
- Multimedia >
- Computing >
- Middleware and Software Pac... >

Software Packs

Tim1 Mode and Configuration

Mode

Slave Mode	Disable
Trigger Source	Disable
Clock Source	Disable
Channel1	Disable
Channel2	PWM Generation CH2
Channel3	Disable
Channel4	Disable
Combined Channels	Disable

Activate-Break-Input

Use ETR as Clearing Source

XOR activation

One Pulse Mode

Configuration

Reset Configuration

NVIC Settings    DMA Settings    GPIO Settings

Parameter Settings    User Constants

Configure the below parameters :

Search (Ctrl+F)

Counter Settings

- Prescaler (PSC - 16 bits va... 0
- Counter Mode Up
- Counter Period (AutoReloa... .65535
- Internal Clock Division (CK... No Division
- Repetition Counter (RCR - ... 0
- auto-reload preload Disable

Trigger Output (TRGO) Param...

- Master/Slave Mode (MSM...) Disable (Trigger input effect not dela...
- Trigger Event Selection ... Reset (USC bit from TIMx\_EGR)

Pinout

Pinout view

System view

We configured the microphone input pin (PA8) as GPIO\_EXTI8, setting its mode to External Interrupt Mode with rising edge trigger detection, and enabled the EXTI interrupt in the NVIC section (EXTI lines [9:5] interrupts).

We ensured that the microphone interrupt was assigned a lower priority to prevent the interrupt from being triggered again by loud sounds from the song itself or by another snap occurring while the song is playing:

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

In the private define section, we assigned each note a name corresponding to its timer period value. Additionally, we defined a constant for the tempo, which specifies the minimum note duration (corresponding to a sixteenth note):

```

35/* Private define -----
36 /* USER CODE BEGIN PD */
37 #define DO 3205
38 #define DOD 3031
39 #define RE 2856
40 #define RED 2699
41 #define MI 2544
42 #define FA 2405
43 #define FAD 2269
44 #define SOL 2141
45 #define SOLD 2023
46 #define LA 1908
47 #define LAD 1801
48 #define SI 1699
49
50 #define Tempo 75
51 /* USER CODE END PD */
52

```

We then defined a struct to store the note period and duration and an array of structs (score) to represent the specific song melody:

```

28 /* USER CODE BEGIN PTD */
29 struct note {
30     uint16_t period;
31     uint16_t duration;
32 };
33 /* USER CODE END PTD */

```

```
63 /* USER CODE BEGIN PV */
64 struct note score[] = {
65     {SOL,6},
66     {LA,2},
67     {SOL,4},
68     {FA,4},
69     {MI,4},
70     {FA,4},
71     {SOL,8},
72     {RE,4},
73     {MI,4},
74     {FA,8},
75     {MI,4},
76     {FA,4},
77     {SOL,8},
78     {SOL,6},
79     {LA,2},
80     {SOL,4},
81     {FA,4},
82     {MI,4},
83     {FA,4},
84     {SOL,8},
85     {RE,8},
86     {SOL,8},
87     {MI,4},
88     {DO,12},
89 };
90 /* USER CODE END PV */
```

We then defined the functions playnote and playsong as follows:

```

103/* Private user code -----*/
104 /* USER CODE BEGIN 0 */
105void playnote(struct note note_playing){
106    /* USER CODE BEGIN TIM1_Init_0 */
107
108    /* USER CODE END TIM1_Init_0 */
109
110    TIM_MasterConfigTypeDef sMasterConfig = {0};
111    TIM_OC_InitTypeDef sConfigOC = {0};
112    TIM_BreakDeadTimeConfigTypeDef sBreakDeadTimeConfig = {0};
113
114    /* USER CODE BEGIN TIM1_Init_1 */
115
116    /* USER CODE END TIM1_Init_1 */
117    htim1.Instance = TIM1;
118    htim1.Init.Prescaler = 99;
119    htim1.Init.CounterMode = TIM_COUNTERMODE_UP;
120    htim1.Init.Period = note_playing.period;
121    htim1.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
122    htim1.Init.RepetitionCounter = 0;
123    htim1.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_DISABLE;
124    if (HAL_TIM_PWM_Init(&htim1) != HAL_OK)
125    {
126        Error_Handler();
127    }
128    sMasterConfig.MasterOutputTrigger = TIM_TRGO_RESET;
129    sMasterConfig.MasterSlaveMode = TIM_MASTERSLAVEMODE_DISABLE;
130    if (HAL_TIMEx_MasterConfigSynchronization(&htim1, &sMasterConfig) != HAL_OK)
131    {
132        Error_Handler();
133    }
134    sConfigOC.OCMode = TIM_OCMODE_PWM1;
135    sConfigOC.Pulse = 0.5*(note_playing.period+1) -1;
136    sConfigOC.OCPolarity = TIM_OCPOLARITY_HIGH;
137    sConfigOC.OCNPolarity = TIM_OCNPOLARITY_HIGH;
138    sConfigOC.OCFastMode = TIM_OCFAST_DISABLE;
139    sConfigOC.OCIdleState = TIM_OCIDLESTATE_RESET;
140    sConfigOC.OCNIdleState = TIM_OCNIDLESTATE_RESET;
141    if (HAL_TIM_PWM_ConfigChannel(&htim1, &sConfigOC, TIM_CHANNEL_2) != HAL_OK)
142    {
143        Error_Handler();
144    }
145    sBreakDeadTimeConfig.OffStateRunMode = TIM_OSSR_DISABLE;
146    sBreakDeadTimeConfig.OffStateIDLEMode = TIM_OSSI_DISABLE;
147    sBreakDeadTimeConfig.LockLevel = TIM_LOCKLEVEL_OFF;
148    sBreakDeadTimeConfig.DeadTime = 0;
149    sBreakDeadTimeConfig.BreakState = TIM_BREAK_DISABLE;
150    sBreakDeadTimeConfig.BreakPolarity = TIM_BREAKPOLARITY_HIGH;
151    sBreakDeadTimeConfig.AutomaticOutput = TIM_AUTOMATICOUTPUT_DISABLE;
152    if (HAL_TIMEx_ConfigBreakDeadTime(&htim1, &sBreakDeadTimeConfig) != HAL_OK)
153    {
154        Error_Handler();
155    }
156    /* USER CODE BEGIN TIM1_Init_2 */
157
158    /* USER CODE END TIM1_Init_2 */
159    HAL_TIM_MspPostInit(&htim1);
160    HAL_TIM_PWM_Start(&htim1, TIM_CHANNEL_2);
161    HAL_Delay(note_playing.duration*Tempo);
162    HAL_TIM_PWM_Stop(&htim1, TIM_CHANNEL_2);
163}
164void playsong(){
165    int length = sizeof(score)/sizeof(score[0]);
166    for (int i= 0; i< length; i++){
167        playnote(score[i]);
168    }
169}

```

The playnote() function was implemented by copying the code from the MX\_TIM1\_Init function modifying it to dynamically reconfigure Timer 1 for every element of the score. Specifically, we modified the period (line 120) and pulse (line 135) parameters to adapt to the values of the note to be played. For each note, the pulse (CCRx) is recalculated according to the period, while maintaining a duty cycle of 50%:

$$\text{PWM Duty Cycle: } DC = \frac{CCRx+1}{ARR+1}$$

At the end of the function, the PWM signal is started and then stopped after a delay proportional to the desired note duration.

Lastly, we implemented the EXTI callback associated with the microphone, with debouncing (tick, last\_tick and delta were previously defined as global variables) and clearing of the interrupt flag:

```

170 void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin){
171     if(GPIO_Pin==GPIO_PIN_8){
172         _HAL_GPIO_EXTI_CLEAR_IT(GPIO_Pin);
173         tick = HAL_GetTick();
174         delta = tick - last_tick;
175         if(delta>=100){
176             playsong();
177         }
178         last_tick = tick;
179     }
180 }
181 /* USER CODE END 0 */

```

We executed the program on the board, and it functioned as intended.

## Part 2:

NVIC Interrupt Table	Enabled	Preemption Priority	Sub Priority
Non maskable interrupt	<input checked="" type="checkbox"/>	0	0
Hard fault interrupt	<input checked="" type="checkbox"/>	0	0
Memory management fault	<input checked="" type="checkbox"/>	0	0
Pre-fetch fault, memory access fault	<input checked="" type="checkbox"/>	0	0
Undefined instruction or illegal state	<input checked="" type="checkbox"/>	0	0
System service call via SWI instruction	<input checked="" type="checkbox"/>	0	0
Debug monitor	<input checked="" type="checkbox"/>	0	0
Pendable request for system service	<input checked="" type="checkbox"/>	0	0
Time base: System tick timer	<input checked="" type="checkbox"/>	0	0
PVD interrupt through EXTI line 16	<input type="checkbox"/>	0	0
Flash global interrupt	<input type="checkbox"/>	0	0
RCC global interrupt	<input type="checkbox"/>	0	0
EXTI line[9:5] interrupts	<input checked="" type="checkbox"/>	2	0
TIM1 break interrupt and TIM9 global interrupt	<input type="checkbox"/>	0	0
TIM1 update interrupt and TIM10 global interrupt	<input type="checkbox"/>	0	0
TIM1 trigger and commutation interrupts and TIM11 global int...	<input type="checkbox"/>	0	0
TIM1 capture compare interrupt	<input type="checkbox"/>	0	0
TIM3 global interrupt	<input checked="" type="checkbox"/>	1	0
USART2 global interrupt	<input type="checkbox"/>	0	0
FPU global interrupt	<input type="checkbox"/>	0	0

In Part 2 of the assignment, we based our work on the configuration used for the previous part but removed all blocking delay mechanisms and replaced them with a non-blocking, interrupt-based approach using Timer 3.

The screenshot shows the STM32CubeMX software interface for configuring Timer 3. The top half of the window displays the 'Mode' configuration, which includes settings for Slave Mode, Trigger Source, Clock Source (set to Internal Clock), and four channels (Channel1 to Channel4) each set to Disable. Below these are three checkboxes: 'Use ETR as Clearing Source', 'XOR activation', and 'One Pulse Mode'. The bottom half of the window displays the 'Configuration' tab, which is currently active. It features a 'Reset Configuration' button and tabs for 'NVIC Settings', 'DMA Settings', 'Parameter Settings' (which is selected), and 'User Constants'. A search bar at the top of this section allows for quick parameter searching. The 'Parameter Settings' section contains a list of parameters grouped under 'Counter Settings' and 'Trigger Output (TRGO) Param.'. Under Counter Settings, Prescaler (PSC - 16 bit... 8399), Counter Mode (Up), Counter Period (AutoRel... 749), Internal Clock Division (... No Division), and auto-reload preloaded are listed. Under Trigger Output (TRGO) Param., Master/Slave Mode (MS... Disable (Trigger input effect not de...), and Trinari Event Selection (Reset ITG hit from TIMx\_FGR1) are listed.

Timer 3 was configured to generate periodic interrupts, corresponding to the base tempo unit, we defined its prescaler and ARR to do so and it uses the Internal clock as source. This timer now serves as

the core timebase for counting the duration of each note without halting the main loop or blocking the processor.

```
68 uint32_t counter = 0;
69 uint32_t current_index = 0;
70 uint8_t song_playing = 0;
71
```

A global *counter* variable is incremented on each Timer 3 interrupt, and once it exceeds the duration field of the currently playing note, the PWM output is stopped and the next note in the *score* array is triggered.

```
172 void playsong(){
173     if (song_playing) return;
174     song_playing = 1;
175     current_index = 0;
176     playnote(score[current_index]);
177     HAL_TIM_Base_Start_IT(&htim3);
178 }
179 void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin){
180     if(GPIO_Pin==GPIO_PIN_8){
181         _HAL_GPIO_EXTI_CLEAR_IT(GPIO_Pin);
182         tick = HAL_GetTick();
183         delta = tick - last_tick;
184         if(delta>=100 && !(song_playing)){
185             playsong();
186         }
187         last_tick = tick;
188     }
189 }
```

Even if we modified *playsong()* and *HAL\_GPIO\_EXTI\_Callback()* this entire logic is managed inside the *HAL\_TIM\_PeriodElapsedCallback()* function, which performs the following steps:

- Stops the PWM output if the note duration has elapsed.
- Advances to the next note in the sequence using *playnote*.
- Stops Timer 3 when the end of the melody is reached.

This allows continuous playback of the melody in real time without blocking the main thread.

```

190 void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim){
191     if(htim == &htim3){
192         counter++;
193         if (counter > current_note.duration) {
194             HAL_TIM_PWM_Stop(&htim1, TIM_CHANNEL_2);
195             counter = 0;
196             current_index++;
197             if(current_index < sizeof(score)/sizeof(score[0])){
198                 playnote(score[current_index]);
199             }else {
200                 HAL_TIM_Base_Stop_IT(&htim3);
201                 song_playing = 0;
202             }
203         }
204     }
205 }
```

To avoid re-triggering the song while it is still playing, a flag *song\_playing* was introduced. The playback sequence can only be initiated from the external interrupt on pin PA8 if this flag is cleared.

Interrupt management played a crucial role in the success of the project. Timer 3 was assigned the priority 1 to guarantee a precise timing of note durations, while the GPIO external interrupt on PA8 was configured with priority 2 to prevent unintended feedback between the speaker and the microphone.

Additionally, the same debounce mechanism based on the HAL\_GetTick() function prevents false triggering from rapid successive sound events as in the previous part of the homework.

NVIC Interrupt Table	Enabled	Preemption Priority	Sub Priority
Non maskable interrupt	<input checked="" type="checkbox"/>	0	0
Hard fault interrupt	<input checked="" type="checkbox"/>	0	0
Memory management fault	<input checked="" type="checkbox"/>	0	0
Pre-fetch fault, memory access fault	<input checked="" type="checkbox"/>	0	0
Undefined instruction or illegal state	<input checked="" type="checkbox"/>	0	0
System service call via SWI instruction	<input checked="" type="checkbox"/>	0	0
Debug monitor	<input checked="" type="checkbox"/>	0	0
Pendable request for system service	<input checked="" type="checkbox"/>	0	0
Time base: System tick timer	<input checked="" type="checkbox"/>	0	0
PVD interrupt through EXTI line 16	<input type="checkbox"/>	0	0
Flash global interrupt	<input type="checkbox"/>	0	0
RCC global interrupt	<input type="checkbox"/>	0	0
EXTI line[9:5] interrupts	<input checked="" type="checkbox"/>	2	0
TIM1 break interrupt and TIM9 global interrupt	<input type="checkbox"/>	0	0
TIM1 update interrupt and TIM10 global interrupt	<input type="checkbox"/>	0	0
TIM1 trigger and commutation interrupts and TIM11 global int...	<input type="checkbox"/>	0	0
TIM1 capture compare interrupt	<input type="checkbox"/>	0	0
TIM3 global interrupt	<input checked="" type="checkbox"/>	1	0
USART2 global interrupt	<input type="checkbox"/>	0	0
FPU global interrupt	<input type="checkbox"/>	0	0

Enabled       

The code was then tested on the board and confirmed to work as intended.

Professor comments: