Personal Project

Voice Recorder/Repeater v1

Matthew Clayton Huber

December 12, 2023

Contents

List of Tables

List of Figures

1 Lab Overview

The purpose of my project is to build a simple voice repeater that records two seconds of audio, then plays it back through a speaker. The signal will be input through a sound sensor module, then amplified by an LM386 amplifier module. The signal must be converted to digital to store as binary in a code file, then converted back to analog to play through a speaker. An STM32 microcontroller is used to make the entire circuit function. Direct Memory Access (DMA) will be used to free up the CPU and store the data points in the RAM to ensure that data samples can be taken accurately. The specific microcontroller which I am using is the Nucleo-L476RG. Specific objectives are included below.

1.1 Objectives

- Construct a circuit which takes 2 seconds of voice input, then replays it through a speaker when prompted.
- Practice using external interrupts and timers in a circuit.
- Practice using an ADC in a circuit.
- Learn about using a DAC and DMA in a circuit.

2 Specifications

The final result of this project seems simple. However, a lot of parts work together to make it function. The first is the sound sensor module. The sound sensor module captures the change in pressure due to sound. Because these changes are so small, the signal must be amplified to become easily readable by the Analog-to-Digital converter. The LM386 amplifier module amplifies the signal, then outputs it to the ADC input pin (PC0) on the microcontroller.

The sound sensor module will only start to record the input if I press a ”record” button (PA0). This triggers an interrupt which activates the ADC and turns on the on-board LED (PA5).

The ADC is triggered by a timer to take a sample about 10,000 times per second. When the analog value becomes a digital value, it becomes an 8-bit binary number (which is an approximation of the original value).

Because there is a lot of information to transfer at a high rate, I use the DMA utility. If I didn’t use it, my CPU wouldn’t be able to take proper samples. Every time an interrupt service routine is called, the routine must be completed before the CPU returns to what it was previously doing. This could distort the sampling process. Using DMA allows me to quickly fill up the volatile memory of the microcontroller without burdening the CPU. With each sample trigger, an array in my code file gains the most recent converted value from the ADC. The DMA for both the ADC and the DAC is set to ”circular” mode, which means that values will be recorded until the array is completely full.

When I press a ”play” button (PC13), an interrupt is called which stops the analog-to-digital conversion and begins the digital-to-analog conversion. The now-converted analog signal goes through a buffer to ensure that it can be played through a speaker, then is played through the speaker (PA4).

2.1 Parts List

- Nucleo-L476RG board and USB cable
- LM386 amplifier module
- Small speaker
- Sound sensor module
- Several jumper wires

For specifics on the wiring setup, see the schematics in Section 4.

3 Operating Instructions and Constraints

3.1 Operating Instructions

The instructions to operate the circuit are very simple. The user must push Button 1 to start recording. When B1 is pushed, the ADC starts to transfer input data into the storage array. To hear the recorded data, the user must push Button 2. Button 2 stops the ADC and starts the DAC, which will play the recording back through the speaker.

3.2 Operating Constraints

The storage array will only record two seconds of the input. When the storage array is filled, it overwrites the old data with the most recent data. Ideally, the user pushes B2 as soon as he or she finishes recording to retain all of what was recorded. Otherwise, it will be lost.

The user may push B2 repeatedly to play back the recording multiple times. As soon as the user pushes B1 to record again, the storage array is overwritten with the incoming data and the recorded data is lost.

Because the storage array is located in the RAM memory of the microcontroller, the recorded data is volatile and will be lost as soon as the microcontroller is powered off.

4 Schematic

Below is the complete schematic which shows the wiring of the circuit. A photo of the circuit is included as well in Figure 2.

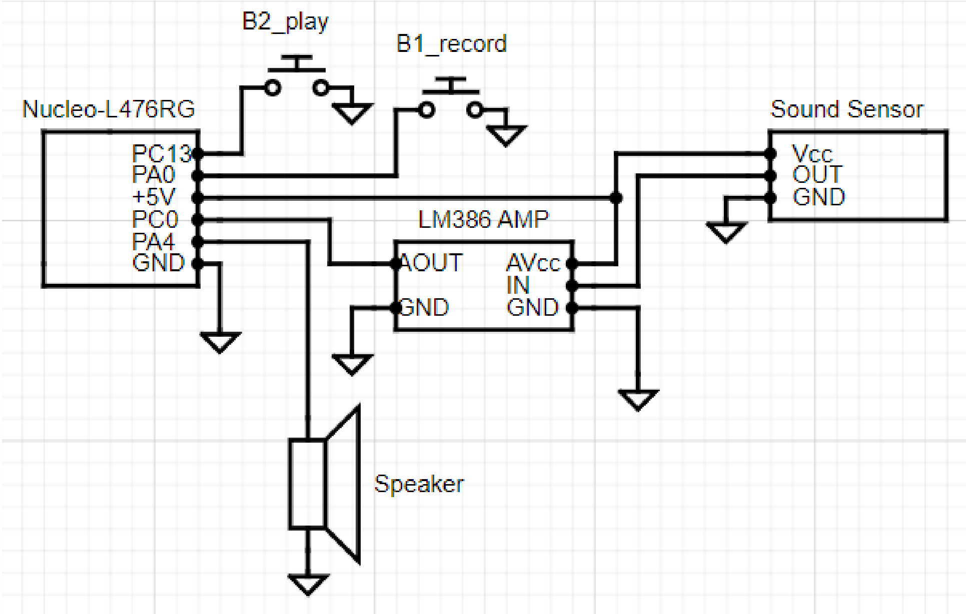

Figure 1: The complete voice repeater circuit and all necessary pins.

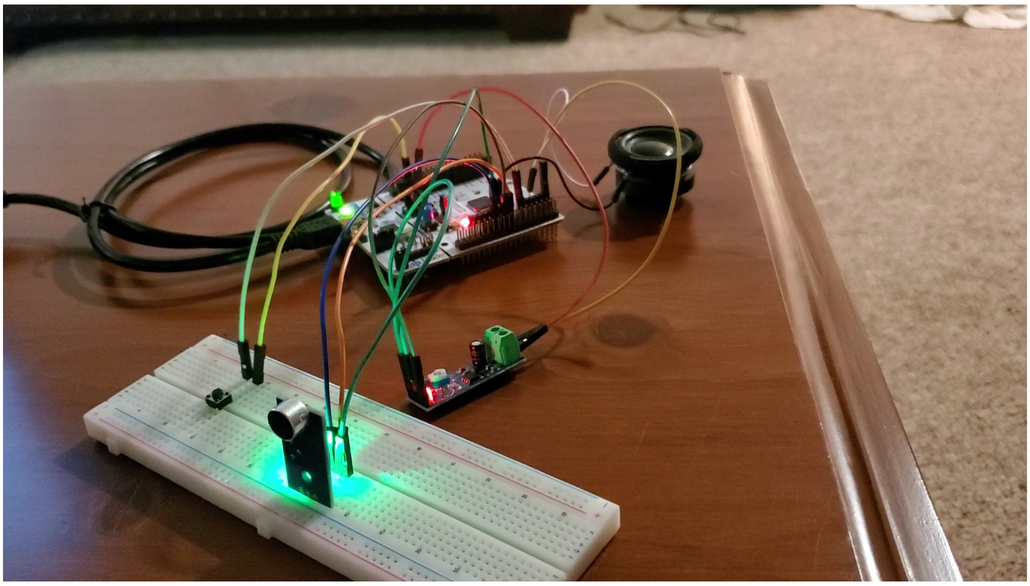

Figure 2: A photo of the complete voice repeater circuit in operation.

5 Test Plan and Test Results

This sections outlines the steps that I performed to ensure that the completed circuit functions in the same manner that I planned.

5.1 Test Plan Procedure

This test plan is meant to test the general functionality of the circuit as well as edge cases. I perform five tests in total.

- Test Scenario #1: Check ADC input.
  - Step 1: Run the program on the Nucleo-L476RG microcontroller.
  - Step 2: Push B1 to start recording.
  - Step 3: Talk into the sound sensor to create an input.
  - Step 4: Check the desired storage address in the code IDE to make sure that the array is filled with values.
- Test Scenario #2: Check DAC output.
  - Step 1: Run the program on the Nucleo-L476RG microcontroller.
  - Step 2: Push B2 to play a pre-loaded array of values.
  - Step 3: Listen if sound comes out.
- Test Scenario #3: Check ADC input to DAC output.
  - Step 1: Run the program on the Nucleo-L476RG microcontroller.
  - Step 2: Push B1 to start recording.
  - Step 3: Talk into the sound sensor to create an input.
  - Step 4: Push B2 to play the stored data through the speaker.
  - Step 5: Listen for the repeated output.
- Test Scenario #4: Record for more than two seconds.
  - Step 1: Run the program on the Nucleo-L476RG microcontroller.
  - Step 2: Push B1 to start recording.
  - Step 3: Wait for 5 seconds, then talk.
  - Step 4: Push B2 to play the stored data and listen for the output.
- Test Scenario #5: Record for less than two seconds, two times.
  - Step 1: Run the program on the Nucleo-L476RG microcontroller.
  - Step 2: Push B1 to start recording.
  - Step 3: Talk quickly and push B2 before one second has passed.
  - Step 4: Talk quickly again and push B2 before one second has passed.
  - Step 5: Listen to the output on the speaker.

5.2 Expected and Observed Results

This section should include the the expected and actual results of each test.

- Test Scenario #1
  - Expected Result: Data points get stored in the array in order.
  - Actual Result: Data points get stored in the array in order.
- Test Scenario #2
  - Expected Result: A sine wave plays through the speaker.
  - Actual Result: A sine wave plays through the speaker.
- Test Scenario #3
  - Expected Result: The voice input plays through the speaker.
  - Actual Result: The voice input plays through the speaker.
- Test Scenario #4
  - Expected Result: The array will be overfilled and will overwrite the old data with the new data. The most recent will be kept.
  - Actual Result: The array overwrites the old data with the new data continuously, keeping the most recent data.
- Test Scenario #5
  - Expected Result: The data from the first recording will be completely overwritten even though it does not occupy the whole array.
  - Actual Result: The data from the first recording does not get overwritten completely. Data points from the first recording that don't overlap with the second persist after recording a second time.

6 Code

The entirety of my main.c code file is included below in 6.1. The sample sine wave array that I define near the beginning is defined using 12-bit DAC conversion, but I eventually switched to 8-bit conversion to save space in my RAM.

6.1 Code for main.c

```c
/* USER CODE BEGIN Header */
/**
  ******************************************************************************
  * @file           : main.c
  * @brief          : Main program body
  ******************************************************************************
  * @attention
  *
  * Copyright (c) 2023 STMicroelectronics.
  * All rights reserved.
  *
  * This software is licensed under terms that can be found in the LICENSE file
  * in the root directory of this software component.
  * If no LICENSE file comes with this software, it is provided AS-IS.
  *
  ******************************************************************************
  */
/* USER CODE END Header */
/* Includes ------------------------------------------------------------------*/
#include "main.h"

/* Private includes ----------------------------------------------------------*/
/* USER CODE BEGIN Includes */

/* USER CODE END Includes */

/* Private typedef -----------------------------------------------------------*/
/* USER CODE BEGIN PTD */

/* USER CODE END PTD */

/* Private define ------------------------------------------------------------*/
/* USER CODE BEGIN PD */
```

```c

/* USER CODE END PD */

/* Private macro ------------------------------------------------------------
   */
/* USER CODE BEGIN PM */

/* USER CODE END PM */

/* Private variables ---------------------------------------------------------
   */
ADC_HandleTypeDef hadc1;
DMA_HandleTypeDef hdma_adc1;

DAC_HandleTypeDef hdac1;
DMA_HandleTypeDef hdma_dac_ch1;

TIM_HandleTypeDef htim2;

/* USER CODE BEGIN PV */

/* USER CODE END PV */

/* Private function prototypes -----------------------------------------------
   */
void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_DMA_Init(void);
static void MX_TIM2_Init(void);
static void MX_DAC1_Init(void);
static void MX_ADC1_Init(void);
/* USER CODE BEGIN PFP */

/* USER CODE END PFP */

/* Private user code ---------------------------------------------------------
   */
/* USER CODE BEGIN 0 */
#define NS  128 //This is the number of samples that are in the Sine below

uint32_t Sine[NS] = {
    2048, 2149, 2250, 2350, 2450, 2549, 2646, 2742, 2837, 2929, 3020, 3108,
   3193, 3275, 3355,
    3431, 3504, 3574, 3639, 3701, 3759, 3812, 3861, 3906, 3946, 3982, 4013,
   4039, 4060, 4076,
    4087, 4094, 4095, 4091, 4082, 4069, 4050, 4026, 3998, 3965, 3927, 3884,
   3837, 3786, 3730,
    3671, 3607, 3539, 3468, 3394, 3316, 3235, 3151, 3064, 2975, 2883, 2790,
   2695, 2598, 2500,
    2400, 2300, 2199, 2098, 1997, 1896, 1795, 1695, 1595, 1497, 1400, 1305,
   1212, 1120, 1031,
    944, 860, 779, 701, 627, 556, 488, 424, 365, 309, 258, 211, 168, 130, 97,
    69, 45, 26, 13, 4, 0, 1, 8, 19, 35, 56, 82, 113, 149, 189,
```

```c
    234, 283, 336, 394, 456, 521, 591, 664, 740, 820, 902, 987, 1075, 1166,
   1258,
    1353, 1449, 1546, 1645, 1745, 1845, 1946, 2047
};

uint32_t gatheredData[24000];


/* USER CODE END 0 */

/**
  * @brief  The application entry point.
  * @retval int
  */
int main(void)
{
  /* USER CODE BEGIN 1 */

  /* USER CODE END 1 */

  /* MCU Configuration--------------------------------------------------------
    */

  /* Reset of all peripherals, Initializes the Flash interface and the Systick
    . */
  HAL_Init();

  /* USER CODE BEGIN Init */

  /* USER CODE END Init */

  /* Configure the system clock */
  SystemClock_Config();

  /* USER CODE BEGIN SysInit */



  /* USER CODE END SysInit */

  /* Initialize all configured peripherals */
  MX_GPIO_Init();
  MX_DMA_Init();
  MX_TIM2_Init();
  MX_DAC1_Init();
  MX_ADC1_Init();
  /* USER CODE BEGIN 2 */



  /* USER CODE END 2 */

  /* Infinite loop */
  /* USER CODE BEGIN WHILE */
```

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

    /* USER CODE BEGIN 3 */

  }
  /* USER CODE END 3 */
}

/**
  * @brief System Clock Configuration
  * @retval None
  */
void SystemClock_Config(void)
{
  RCC_OscInitTypeDef RCC_OscInitStruct = {0};
  RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};

  /** Configure the main internal regulator output voltage
  */
  if (HAL_PWREx_ControlVoltageScaling(PWR_REGULATOR_VOLTAGE_SCALE1) != HAL_OK)
  {
    Error_Handler();
  }

  /** Initializes the RCC Oscillators according to the specified parameters
  * in the RCC_OscInitTypeDef structure.
  */
  RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSI;
  RCC_OscInitStruct.HSIState = RCC_HSI_ON;
  RCC_OscInitStruct.HSICalibrationValue = RCC_HSICALIBRATION_DEFAULT;
  RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
  RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSI;
  RCC_OscInitStruct.PLL.PLLM = 1;
  RCC_OscInitStruct.PLL.PLLN = 10;
  RCC_OscInitStruct.PLL.PLLP = RCC_PLLP_DIV7;
  RCC_OscInitStruct.PLL.PLLQ = RCC_PLLQ_DIV2;
  RCC_OscInitStruct.PLL.PLLR = RCC_PLLR_DIV2;
  if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
  {
    Error_Handler();
  }

  /** Initializes the CPU, AHB and APB buses clocks
  */
  RCC_ClkInitStruct.ClockType = RCC_CLOCKTYPE_HCLK|RCC_CLOCKTYPE_SYSCLK
                              |RCC_CLOCKTYPE_PCLK1|RCC_CLOCKTYPE_PCLK2;
  RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_PLLCLK;
  RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
  RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV1;
  RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV1;

  if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_4) != HAL_OK)
```

```
  {
    Error_Handler();
  }
}

/**
  * @brief ADC1 Initialization Function
  * @param None
  * @retval None
  */
static void MX_ADC1_Init(void)
{

  /* USER CODE BEGIN ADC1_Init 0 */

  /* USER CODE END ADC1_Init 0 */

  ADC_MultiModeTypeDef multimode = {0};
  ADC_ChannelConfTypeDef sConfig = {0};

  /* USER CODE BEGIN ADC1_Init 1 */

  /* USER CODE END ADC1_Init 1 */

  /** Common config
  */
  hadc1.Instance = ADC1;
  hadc1.Init.ClockPrescaler = ADC_CLOCK_SYNC_PCLK_DIV1;
  hadc1.Init.Resolution = ADC_RESOLUTION_8B;
  hadc1.Init.DataAlign = ADC_DATAALIGN_RIGHT;
  hadc1.Init.ScanConvMode = ADC_SCAN_DISABLE;
  hadc1.Init.EOCSelection = ADC_EOC_SINGLE_CONV;
  hadc1.Init.LowPowerAutoWait = DISABLE;
  hadc1.Init.ContinuousConvMode = DISABLE;
  hadc1.Init.NbrOfConversion = 1;
  hadc1.Init.DiscontinuousConvMode = DISABLE;
  hadc1.Init.ExternalTrigConv = ADC_EXTERNALTRIG_T2_TRGO;
  hadc1.Init.ExternalTrigConvEdge = ADC_EXTERNALTRIGCONVEDGE_RISING;
  hadc1.Init.DMAContinuousRequests = DISABLE;
  hadc1.Init.Overrun = ADC_OVR_DATA_PRESERVED;
  hadc1.Init.OversamplingMode = DISABLE;
  if (HAL_ADC_Init(&hadc1) != HAL_OK)
  {
    Error_Handler();
  }

  /** Configure the ADC multi-mode
  */
  multimode.Mode = ADC_MODE_INDEPENDENT;
  if (HAL_ADCEx_MultiModeConfigChannel(&hadc1, &multimode) != HAL_OK)
  {
    Error_Handler();
  }

```

```
  /** Configure Regular Channel
  */
  sConfig.Channel = ADC_CHANNEL_1;
  sConfig.Rank = ADC_REGULAR_RANK_1;
  sConfig.SamplingTime = ADC_SAMPLETIME_2CYCLES_5;
  sConfig.SingleDiff = ADC_SINGLE_ENDED;
  sConfig.OffsetNumber = ADC_OFFSET_NONE;
  sConfig.Offset = 0;
  if (HAL_ADC_ConfigChannel(&hadc1, &sConfig) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN ADC1_Init 2 */

  /* USER CODE END ADC1_Init 2 */

}

/**
  * @brief DAC1 Initialization Function
  * @param None
  * @retval None
  */
static void MX_DAC1_Init(void)
{

  /* USER CODE BEGIN DAC1_Init 0 */

  /* USER CODE END DAC1_Init 0 */

  DAC_ChannelConfTypeDef sConfig = {0};

  /* USER CODE BEGIN DAC1_Init 1 */

  /* USER CODE END DAC1_Init 1 */

  /** DAC Initialization
  */
  hdac1.Instance = DAC1;
  if (HAL_DAC_Init(&hdac1) != HAL_OK)
  {
    Error_Handler();
  }

  /** DAC channel OUT1 config
  */
  sConfig.DAC_SampleAndHold = DAC_SAMPLEANDHOLD_DISABLE;
  sConfig.DAC_Trigger = DAC_TRIGGER_T2_TRGO;
  sConfig.DAC_OutputBuffer = DAC_OUTPUTBUFFER_ENABLE;
  sConfig.DAC_ConnectOnChipPeripheral = DAC_CHIPCONNECT_DISABLE;
  sConfig.DAC_UserTrimming = DAC_TRIMMING_FACTORY;
  if (HAL_DAC_ConfigChannel(&hdac1, &sConfig, DAC_CHANNEL_1) != HAL_OK)
  {
    Error_Handler();
```

```c
  }
  /* USER CODE BEGIN DAC1_Init 2 */

  /* USER CODE END DAC1_Init 2 */

}

/**
  * @brief TIM2 Initialization Function
  * @param None
  * @retval None
  */
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 = 799;
  htim2.Init.CounterMode = TIM_COUNTERMODE_UP;
  htim2.Init.Period = 10;
  htim2.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
  htim2.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_ENABLE;
  if (HAL_TIM_Base_Init(&htim2) != HAL_OK)
  {
    Error_Handler();
  }
  sClockSourceConfig.ClockSource = TIM_CLOCKSOURCE_INTERNAL;
  if (HAL_TIM_ConfigClockSource(&htim2, &sClockSourceConfig) != HAL_OK)
  {
    Error_Handler();
  }
  sMasterConfig.MasterOutputTrigger = TIM_TRGO_UPDATE;
  sMasterConfig.MasterSlaveMode = TIM_MASTERSLAVEMODE_DISABLE;
  if (HAL_TIMEx_MasterConfigSynchronization(&htim2, &sMasterConfig) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN TIM2_Init 2 */

  /* USER CODE END TIM2_Init 2 */

}

/**
  * Enable DMA controller clock
```

```c
  */
static void MX_DMA_Init(void)
{

  /* DMA controller clock enable */
  __HAL_RCC_DMA1_CLK_ENABLE();

  /* DMA interrupt init */
  /* DMA1_Channel1_IRQn interrupt configuration */
  HAL_NVIC_SetPriority(DMA1_Channel1_IRQn, 0, 0);
  HAL_NVIC_EnableIRQ(DMA1_Channel1_IRQn);
  /* DMA1_Channel3_IRQn interrupt configuration */
  HAL_NVIC_SetPriority(DMA1_Channel3_IRQn, 0, 0);
  HAL_NVIC_EnableIRQ(DMA1_Channel3_IRQn);

}

/**
  * @brief GPIO Initialization Function
  * @param None
  * @retval None
  */
static void MX_GPIO_Init(void)
{
  GPIO_InitTypeDef GPIO_InitStruct = {0};
/* USER CODE BEGIN MX_GPIO_Init_1 */
/* USER CODE END MX_GPIO_Init_1 */

  /* GPIO Ports Clock Enable */
  __HAL_RCC_GPIOC_CLK_ENABLE();
  __HAL_RCC_GPIOH_CLK_ENABLE();
  __HAL_RCC_GPIOA_CLK_ENABLE();
  __HAL_RCC_GPIOB_CLK_ENABLE();

  /*Configure GPIO pin Output Level */
  HAL_GPIO_WritePin(GPIOA, GPIO_PIN_5, GPIO_PIN_RESET);

  /*Configure GPIO pin : PC13 */
  GPIO_InitStruct.Pin = GPIO_PIN_13;
  GPIO_InitStruct.Mode = GPIO_MODE_IT_RISING;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  HAL_GPIO_Init(GPIOC, &GPIO_InitStruct);

  /*Configure GPIO pin : PA0 */
  GPIO_InitStruct.Pin = GPIO_PIN_0;
  GPIO_InitStruct.Mode = GPIO_MODE_IT_RISING;
  GPIO_InitStruct.Pull = GPIO_PULLDOWN;
  HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);

  /*Configure GPIO pin : PA5 */
  GPIO_InitStruct.Pin = GPIO_PIN_5;
  GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
```

```c
  HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);

  /* EXTI interrupt init*/
  HAL_NVIC_SetPriority(EXTI0_IRQn, 0, 0);
  HAL_NVIC_EnableIRQ(EXTI0_IRQn);

  HAL_NVIC_SetPriority(EXTI15_10_IRQn, 0, 0);
  HAL_NVIC_EnableIRQ(EXTI15_10_IRQn);

/* USER CODE BEGIN MX_GPIO_Init_2 */
/* USER CODE END MX_GPIO_Init_2 */
}

/* USER CODE BEGIN 4 */

void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin)
{
  HAL_TIM_Base_Start(&htim2);                          //Start Timer

    if (GPIO_Pin == GPIO_PIN_13) //if button PC13 is pressed
    {

      HAL_ADC_Stop_DMA(&hadc1);
      HAL_GPIO_WritePin(GPIOA, GPIO_PIN_5, GPIO_PIN_RESET);
      HAL_DAC_Start_DMA(&hdac1, DAC_CHANNEL_1, (uint32_t*)gatheredData, 24000,
    DAC_ALIGN_8B_R);

    }
    if (GPIO_Pin == GPIO_PIN_0) //if button A0 is pressed
    {

      HAL_DAC_Stop_DMA(&hdac1, DAC_CHANNEL_1);
      HAL_GPIO_WritePin(GPIOA, GPIO_PIN_5, GPIO_PIN_SET);
      HAL_ADC_Start_DMA(&hadc1, (uint32_t*)gatheredData, 24000);
    }


}
/* USER CODE END 4 */

/**
  * @brief  This function is executed in case of error occurrence.
  * @retval None
  */
void Error_Handler(void)
{
  /* USER CODE BEGIN Error_Handler_Debug */
  /* User can add his own implementation to report the HAL error return state
    */
  __disable_irq();
  while (1)
  {
  }
  /* USER CODE END Error_Handler_Debug */
```

```
}

#ifdef  USE_FULL_ASSERT
/**
  * @brief  Reports the name of the source file and the source line number
  *         where the assert_param error has occurred.
  * @param  file: pointer to the source file name
  * @param  line: assert_param error line source number
  * @retval None
  */
void assert_failed(uint8_t *file, uint32_t line)
{
  /* USER CODE BEGIN 6 */
  /* User can add his own implementation to report the file name and line number,
     ex: printf("Wrong parameters value: file %s on line %d\r\n", file, line) */
  /* USER CODE END 6 */
}
#endif /* USE_FULL_ASSERT */
```

7 Conclusion

Overall I am satisfied with this project. The result is simple but it functions as intended. The audio quality is not ideal but it is good enough that I am able to understand it. This project has taught me quite a bit about analog-to-digital converters and digital-to-analog converters.

I believe that the most important lesson to be learned from this project is that projects are meant to be taken a step at a time. Before I began to work on this project, I did not know how I was going to do it because I didn't know enough about analog-to-digital converters and I had never used a digital-to-analog converter with an STM32 microcontroller. I started with the DAC and worked on it until it functioned correctly. Then I added the sound sensor input using the ADC.

I have often found myself impatiently trying to conquer tasks that I do not know how to conquer. However, when I set my sight on just the next step, I find that I am able to learn efficiently and that I start to make progress. As I continue to learn about embedded systems, I know that this project will be made just another step along the way.