EEE3095/6S: Practical 4

STM32 DACs

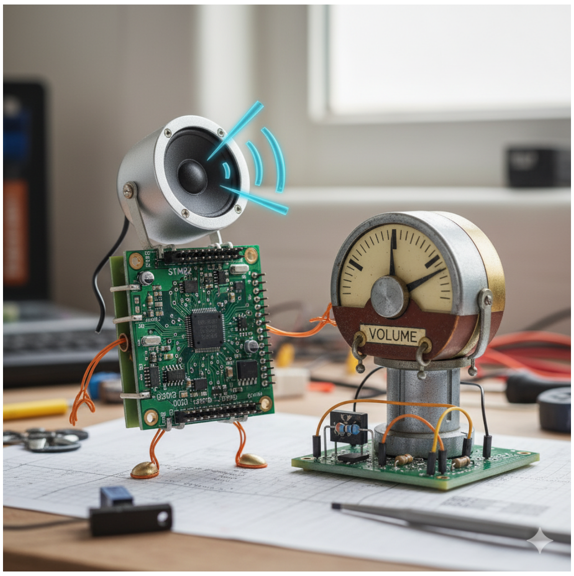
September 22, 2025

1 Overview

Finally, no more turning LEDS on and off and onto more serious stuff with real world implications.

This practical introduction introduces you to DACs. A DAC converts a digital code into an analog voltage or current and works on the general premise that $V_{OUT} = k \times DigitalInput$, where **k** is a proportionality factor and the **digital input** is a number in the range of $0 \rightarrow (2^{\text{bits}} - 1)$. DACs have a variety of uses, with the most important among them converting digital signals to analog audio in just about every single electronic audio system in the world. The STM Board already has a built-in DAC. Setting it up is trivial and there are plenty of online resources available if you wish to test it out.

In this practical we will be making use of a quick and dirty DAC, an amplifier circuit, look-up tables (LUTs) for a few different waveforms, set up two timers for generating the PWM signal and cycling through the LUT values generated from 3 wav files(Sample rate of 44.1 kHz) and 3 waveforms, and then use the small speaker to produce audible sound. Volume control will be implemented using an analog potentiometer and audio signals will be generated from lookup tables (LUTs) derived from .wav files and the waveforms.



2 Outcomes and Knowledge Areas

In this practical, you will be using C code (and the HAL libraries) to interface your microcontroller board with an amplifier circuit that you will build in the lab. You will also use look-up tables (LUTs) to represent sound waveforms, set up two timers on the dev board (one timer for generating the PWM signal and one for cycling through the LUT values periodically), and then feed the output to your amplifier. The aim of this is to use the LUT values to vary the CCR (Capture/Compare Register) value, effectively changing your duty cycle and generating sound. You will learn about the following aspects:

- DACs

- Simple Amplifier design
- PWM
- Pushbutton interrupts

3 Hardware

You will require the following external hardware (in addition to your STM dev board) to properly complete this practical:

- Breadboard
- Signal generator
- Components to build amplifier circuit such as resistors, capacitors, transistors, op-amps etc
- Speaker
- Potentiometer

4 Deliverables

For this practical, you must:

- Develop the code required to meet all objectives specified in the Tasks section
- Push your completed code to a shared repository on GitHub
- Demonstrate your working implementation to a tutor in the lab. Before calling a tutor for the demo submit a pdf file of the main.c code to Gradescope and for this prac every student should submit i.e it's not a group submission. You will be allowed to conduct your demo during any lab session before the practical submission deadline. **All group members will be required to be present at the demo as each group member has to answer two questions that count for 5 marks.** Name your pdf file as follows:

  EEE3096S 2025 Practical 4 Demo STDNUM001.pdf

- Write a short **2-page** report documenting your **main.c** code, GitHub repo link, and a brief description of the implementation of your solutions. This must be in PDF format and submitted on Gradescope with the naming convention(**If you do not adhere to the naming convention, there will be a penalty**):

  EEE3096S 2025 Practical 4 Report STDNUM001 STDNUM002.pdf

  Check the Appendix for Report Structure 7.

- Your practical mark will be based both on your demo to the tutor (i.e., completing the below tasks correctly) as well as your short report.

5 Getting Started

The procedure is as follows:

1. Clone or download the Git repository(The practical folder(s) of interest is Practical4
   git clone https://github.com/EEE3096S-UCT/EEE3096S-2025.git
2. - Open **STM32CubeIDE**, then navigate through the menus:

     `File` → `Import` → `Existing Code as Makefile Project`

   - Click `Next`.
   - In the dialog, click `Browse...` and select your project folder.
   - Under **Toolchain**, choose `MCU ARM GCC`.

- Click Finish.

6 Tasks

Complete the following tasks using the main.c file in STM32CubeIDE with the HAL libraries, and then demonstrate the working execution of each task to a tutor:

Sample Solution can be found at: Sample

Task 1

Your first task will be to:

1. Generate lookup tables (LUTs) for a single cycle of a sinusoid, a sawtooth wave and a triangular wave. Your lookup table should have a minimum of 128 values ranging from 0 to 4095(12 bit resolution). Plot your LUTs using MATLAB/Excel/Python to ensure you have the correct wave shape. Copy your LUTs into your main.c file.
2. Generate a Lookup Table (LUT) for three sound waveforms sampled at 44.1kHz. The LUT should have at least 128 values that range from 0 to 4095 (12-bit resolution). Use MATLAB or Python to process a .wav audio file, plot the waveform to confirm accuracy, and then copy the LUT values into your main.c file. The three .wav files can be found here:

   wav_files

Task 2

Assign values to NS, TIM2CLK and Fsignal. NS is the number of samples in your LUT, TIM2CLK should be set to 16 MHz (which you can see in the .ioc file), and Fsignal is the frequency that we want our analog signal to have. Find a suitable limit to the Fsignal. You will be required to explain your reasoning.

Task 3

Calculate TIM2_Ticks. This is the number of cycles that the PWM duty cycle will be changed and depends on the clock frequency TIM2CLK, number of samples NS and output frequency F signal. Note: We will be using Direct Memory Access (DMA) to change the PWM duty cycle. DMA is used to provide high-speed data transfer between peripherals and memory as well as memory to memory. Data can be quickly moved by DMA without any CPU actions. Check the .ioc file to see how the DMA is configured (as well as the other peripherals!).

Task 4

Configure the DAC to output values from the LUT periodically. Use a timer (e.g., TIM2) to cycle through the LUT and generate the waveform output. Verify that the DAC is correctly configured to produce sound through the amplifier circuit. To do this complete the following:

In the main() function:

1. Start TIM3 in PWM mode on channel 3
2. Start TIM2 in Output Compare (OC) mode on channel 1.
3. Start the DMA in interrupt (IT) mode. The source address is one of the LUTs you created earlier. The destination address is the CCR3 register for TIM3_CH3. Start with the Sine wave LUT.
4. Write the waveform type to the LCD screen, i.e., "Sine".
5. Make use of __HAL_TIM_ENABLE_DMA(htim2, TIM_DMA_CC1) to start the
6. DMA transfer.

HINT: The 3 functions you need for TASK 4 begin with either HAL_DMA or HAL_TIM. Press Ctrl + Space to see the options once you have typed out the first few letters.

Task 5

An external interrupt has been configured on the PA0 pushbutton(labelled Button0 in the code), and EXTI0_1_IRQHandler(void) is called when it is pressed. Write code that changes from one waveform/sound to the next when the button is pressed. Remember to debounce your button presses; debouncing should eliminate noise from bouncing, but not make the response seem sluggish if the button is pressed multiple times shortly after each other. Use __HAL_TIM_DISABLE_DMA(htim2, TIM_DMA_CC1) and HAL_DMA_Abort_IT to stop the DMA transfer before changing the source address, then re-enable DMA.

As part of your interrupt, write the current waveform type/ Song to the LCD screen, e.g., "Sine", "Sawtooth", or "Triangular" or "Piano" or "Guitar" or "Drum" depending on which LUT is selected after pressing the pushbutton.

Task 6

Build the amplifier circuit on your breadboard using components provided and others that you can find at White Lab if necessary. Include a brief motivation for your designed circuit explaining your design choices.Pages required to document your circuit design and motivation to be added as an appendix and will not count towards the total page length of the report.

Some helpful links for designing the amplifier:

1. Resource1
2. Resource2

You might want to build a simple filter to filter the output before passing it to the speaker, to better improve on the quality of the output(Though wont make much of a difference unless your filter is well designed). Warning be careful as you might end up filtering the sound itself.

NOTE: The quality of your sound largely depends on how good your amplifier is and the choice of F_SIGNAL.

The sample circuits are shown in the Figures 1, 2, 3, 4 below:

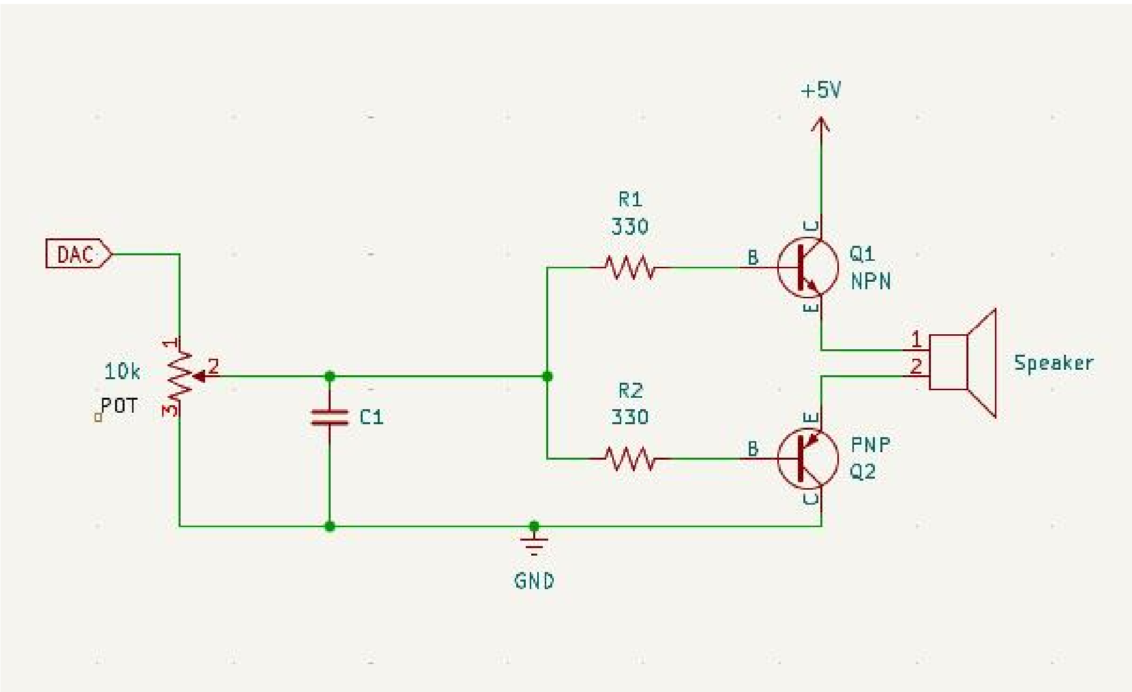


Figure 1: Amplifier Design 1

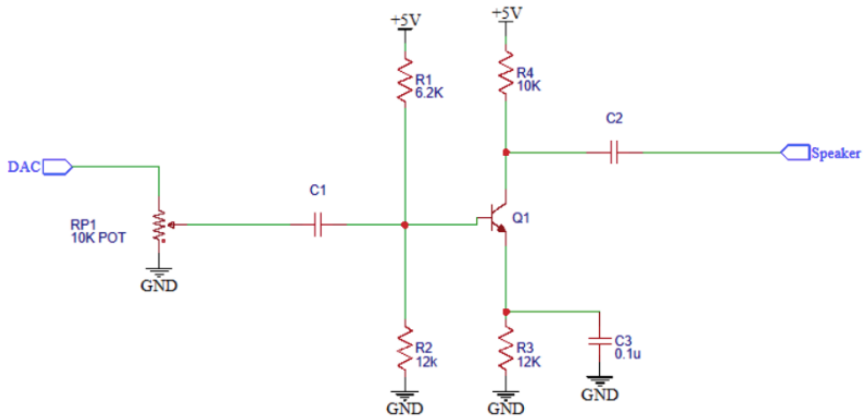


Figure 2: Amplifier Design 2

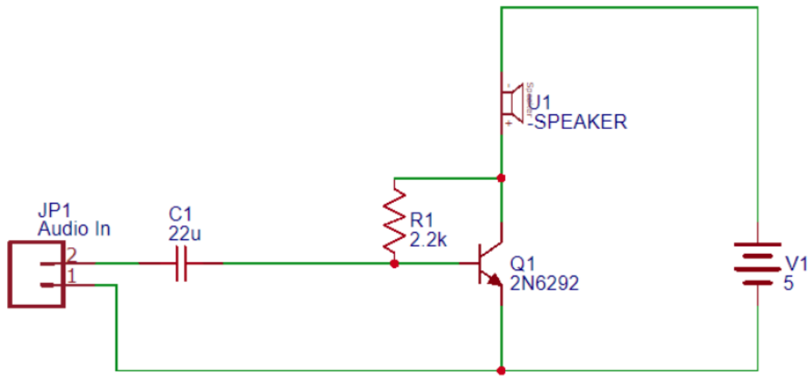


Figure 3: Amplifier Design 3

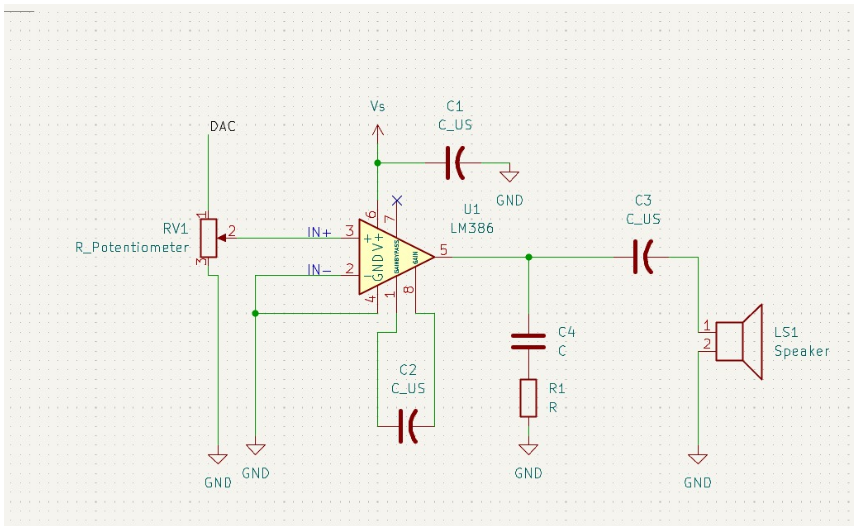


Figure 4: Amplifier Design 4

Task 7

Connect PB0 (TIM3_CH3) to the input of your amplifier. Make sure that the STM32 and your amplifier share a ground line. Test all three of your waveforms and three .wav files.

Task 8

This Task is purely optional. Attempt to redo the prac but this time making use of the inbuilt STM32 DAC instead.

A helpful resource to assist with this task can be found here: DAC

7 Submission

1. You will need to submit your main.c file in pdf format to Practical 4 Demonstration Assignment on Gradescope before your demo. It is mandatory at the top of your code to include the following(Check the Appendix 7 for example):

```
    /*
    1. Link: https://github.com/EEE3096S-UCT/EEE3096S-2025

    2. Group Number: ##

    3. Members: STDNUM001 STDNUM002
    */
    //Rest of your code
```

2. For your report once you are done submit the pdf to Practical 4 Report assignment on Gradescope. Remember to adhere to the naming convention.

If you want to replicate the code styling on Latex as the one in this prac manual do the following:

```
% Make sure to include the following packages in your document
\usepackage{xcolor}
\usepackage{minted}

Here is an example of C code:

\begin{minted}[frame=lines, linenos, bgcolor=black!5]{c}
/*
  1. Link: https://github.com/EEE3096S-UCT/EEE3096S-2025
  2. Group Number: ##
  3. Members: STDNUM001 STDNUM002
*/
// Rest of your code
\end{minted}

Rest of your report
```

That will display the C code as:

```c
/*
  1. Link: https://github.com/EEE3096S-UCT/EEE3096S-2025
  2. Group Number: ##
  3. Members: STDNUM001 STDNUM002
*/
// Rest of your code
int main(){

}
```

Appendix: Report Structure

Section	Description
Introduction	Briefly introduce the aim and objectives of the practical and summarise your work briefly.
Methodology	Detail the steps undertaken and methods used to achieve the practical task in a logical and coherent manner.
Results and Discussion	Present the results and discuss their significance.
Conclusion	Summarise the work you did for the practical and any improvements you could make to your implementation.
AI Clause	In one paragraph discuss how you used LLMs while working on the practical and if you did find them useful in an embedded systems programming context.

Appendix: Code Submission Example

```c
/*

1. Link: https://github.com/EEE3096S-UCT/EEE3096S-2025

2. Group Number: ##

3. Members: STDNUM001 STDNUM002

*/


/* USER CODE BEGIN Header */
/**
  ******************************************************************************
  * @file           : main.c
  * @brief          : Main program body
  ******************************************************************************
  * @attention
  *
  * Copyright (c) 2025 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 */

/* USER CODE END PD */

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

/* USER CODE END PM */

/* Private variables ---------------------------------------------------------*/

/* USER CODE BEGIN PV */
```

```c

/* USER CODE END PV */

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

/* USER CODE END PFP */

/* Private user code ---------------------------------------------------------*/
/* USER CODE BEGIN 0 */

/* 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();
  /* USER CODE BEGIN 2 */

  /* USER CODE END 2 */

  /* Infinite loop */
  /* USER CODE BEGIN WHILE */
  while (1)
  {

    /* USER CODE END WHILE */

    /* USER CODE BEGIN 3 */
          // Toggle the LEDs with a 1s delay
          // Code to check if the board is working
                  HAL_GPIO_TogglePin(GPIOB, GPIO_PIN_0|GPIO_PIN_1|GPIO_PIN_2|GPIO_PIN_3
```

```
                                    |GPIO_PIN_4|GPIO_PIN_5|GPIO_PIN_6|GPIO_PIN_7);

                    HAL_Delay(1000);
  }
  /* 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
  */
  __HAL_RCC_PWR_CLK_ENABLE();
  __HAL_PWR_VOLTAGESCALING_CONFIG(PWR_REGULATOR_VOLTAGE_SCALE3);

  /** 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_NONE;
  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_HSI;
  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_0) != HAL_OK)
  {
    Error_Handler();
  }
}

/**
  * @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 */
```

```c

  /* GPIO Ports Clock Enable */
  __HAL_RCC_GPIOB_CLK_ENABLE();

  /*Configure GPIO pin Output Level */
  HAL_GPIO_WritePin(GPIOB, GPIO_PIN_0|GPIO_PIN_1|GPIO_PIN_2|GPIO_PIN_3
                          |GPIO_PIN_4|GPIO_PIN_5|GPIO_PIN_6|GPIO_PIN_7, GPIO_PIN_RESET);

  /*Configure GPIO pins : PB0 PB1 PB2 PB3
                           PB4 PB5 PB6 PB7 */
  GPIO_InitStruct.Pin = GPIO_PIN_0|GPIO_PIN_1|GPIO_PIN_2|GPIO_PIN_3
                          |GPIO_PIN_4|GPIO_PIN_5|GPIO_PIN_6|GPIO_PIN_7;
  GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
  HAL_GPIO_Init(GPIOB, &GPIO_InitStruct);

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

/* USER CODE BEGIN 4 */

/* 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 */
```