الجامعة الأمريكية في رأس الخيمة
American University of Ras Al Khaimah

Hands-Free Light Switch

CENG 315

Instructor: Dr. Arfan Ghani

24 April 2026

Alaa Eddin
2023006114

Ali Alhaeri
2023006084

Mariam Hemeida
2023006110

Mustafa Turani
2023006252

Omar Al Nuaimi
2023006311

Zaineh Khawaja
2023006253

Department of Computer Science and Engineering
American University of Ras Al Khaimah
2025-26

Table of Contents

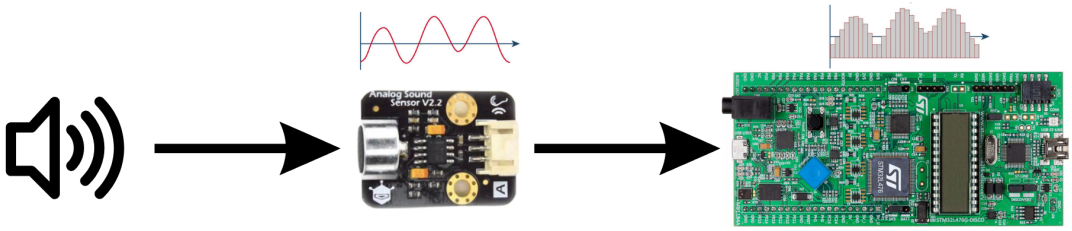
List of Figures

1 Introduction & Design Specifications

Embedded systems play an important role in modern electronic applications by allowing hardware and software to work together efficiently. Microcontrollers, such as the STM32L476VG, are widely used due to their low power consumption, flexibility, and ability to interface with various sensors and devices.

This project presents the design of a sound-controlled lighting system using the STM32L476VG microcontroller. The system responds to sound events to control a light source, demonstrating a simple real-time embedded application.

The main objective of this project is to apply key concepts learned in the course, including General Purpose Input/Output (GPIO), Analog-to-Digital Conversion (ADC), interrupt handling, and serial communication.



2 System Description

The sound sensor detects acoustic signals and produces an analog voltage proportional to the sound intensity. This signal is connected to the PA1 pin of the STM32L476VG and is converted into a digital value using the built-in ADC. The microcontroller then processes this value and compares it with a predefined threshold to detect a clap. When the threshold is exceeded, the system toggles the state of the LED connected to a GPIO pin, allowing the light to turn ON and OFF with each detected clap. This demonstrates the integration of analog input processing, digital decision-making, and output control within the system.

2.1 System Block Diagram

The STM32L476 Discovery board connects to the sound sensor and LED through GPIO and ADC interfaces. The internal processing includes signal conversion and decision logic.

To see the ADC block diagram refer to Appendix II.

3 Hardware Architecture & Interfacing

3.1 Sound Sensor Description

The Gravity Sound Sensor is an analog sensor used to detect the sound intensity of the environment. It is extendable and only requires a dedicated data line. The sensor can be extended with the Arduino board or any microcontroller platform, making it very easy to achieve sound perception for interactive applications.

3.2 STM32 Microcontroller Overview

The hardware architecture centers around the ARM Cortex-M4 core. The STM32L476VG microcontroller provides multiple integrated peripherals including GPIO, ADC, USART, and timers that enable real-time signal processing for the sound-controlled lighting system.

3.3 Hardware Pin Connections

The Sound Sensor is interfaced through the following physical connections:

Sensor Pin	**STM32 Pin**	**Configuration**
VCC (3.3V)	3.3V Output	Power Supply
GND	GND	Common Ground
Signal	PA1	ADC1_IN6 (Analog Input)
LED (Output)	PB2	GPIO Output (Red LED)

Table 1: Hardware Pin Connections

The sensor is powered using the 3.3V supply from the STM32 board. A common ground is shared between the sensor and the microcontroller for signal stability. The analog signal output from the sensor is connected to Pin PA1, which is configured as ADC1_IN6 for 12-bit analog-to-digital conversion.

3.4 Design Specifications

The system is designed with the following specifications:

- **Detection:** Continuous monitoring of ambient audio via the Gravity Analog Sound Sensor.
- **Processing:** 12-bit ADC conversion to translate analog audio signals into a digital range (0-4095).
- **Threshold:** A software trigger set at 1000 to differentiate intentional sounds from background noise.
- **Response:** Real-time toggling of the onboard Red LED (PB2) when a sound event is detected.

3.5 Hardware Calibration

Calibration of the sound sensor is essential to ensure reliable detection of intentional sounds (such as hand claps) while rejecting background noise. The following calibration steps were performed:

3.5.1 Threshold Determination

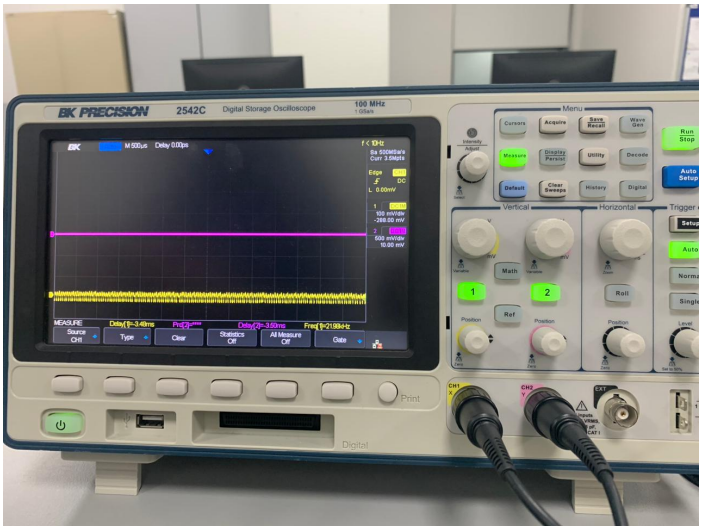
The ambient noise level was measured by reading the ADC value in a quiet environment. The typical ambient ADC reading ranged from 100 to 300. A clap produced a significantly higher ADC reading, typically between 2500 and 3800. The threshold value was set to 1000, which is approximately the midpoint above ambient noise but below typical clap peaks.

3.5.2 Testing with Oscilloscope

An oscilloscope was used to verify the analog output signal from the sound sensor. When no sound was present, the signal remained at a stable DC level (approximately 0.3V - 0.9V). When a clap was detected, the oscilloscope captured a distinct voltage spike corresponding to the sound intensity.

The oscilloscope testing confirmed that:

- The sensor responds instantaneously to acoustic events
- The output voltage amplitude is proportional to sound intensity
- The signal returns to baseline within approximately 50-100 ms after the sound event



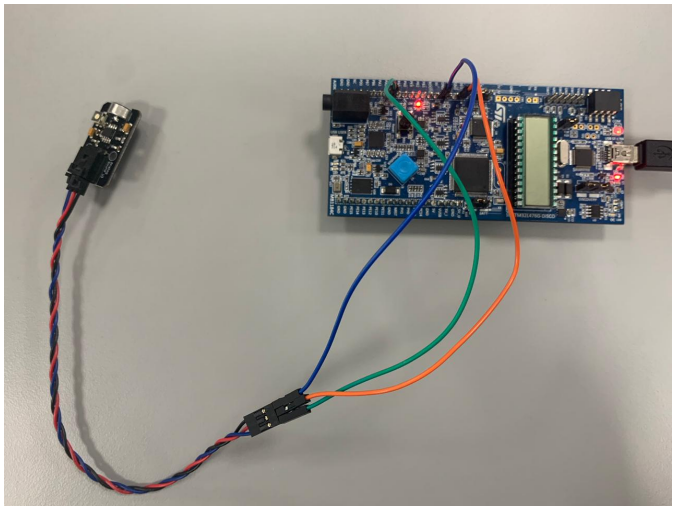
(a) Sensor reading prior to sound exposure

(b) Sensor reading after sound exposure

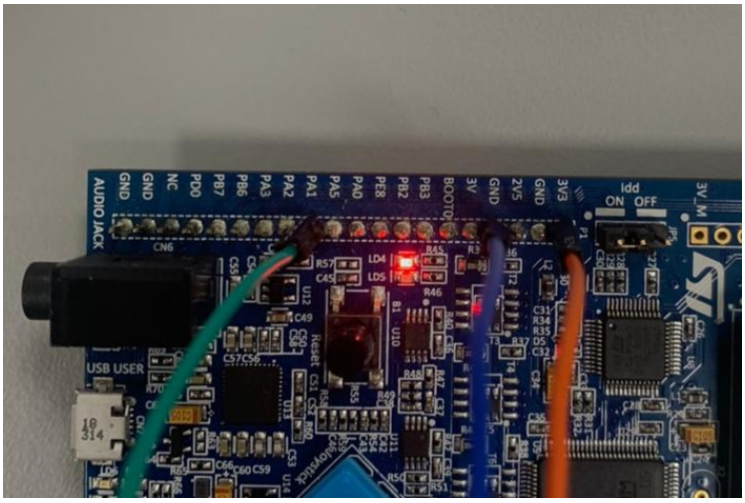
Figure 1: Oscilloscope readings before and after sound exposure

3.6 Hardware Implementation Photos

The complete hardware implementation consists of the STM32L476G-DISCO Discovery board connected to the Gravity Sound Sensor via jumper wires.



(a) Complete hardware implementation of the sound-controlled lighting system



(b) Close-up of pin connections between sound sensor and STM32 board

4 System Implementation (Hardware & Software)

4.1 Peripheral Operations

The project utilizes two core peripherals:

1. **ADC1 Configuration:** According to the STM32L476VGT6 Reference Manual, we used ADC1, which features 12-bit resolution and 19 multiplexed channels. The purpose of the ADC in our project is to convert the analog voltage from the sound sensor, which fluctuates as it detects sound. The ADC samples this voltage at pin **PA1**. We configured it in **Continuous Conversion Mode**, which ensures there is zero wait time for data. Through **quantization**, the 12-bit resolution provides 2^{12} (4095) levels, where 0V represents silence and 3.3V represents the maximum value. The result is stored in a 16-bit register using **Right Alignment**, meaning the 12 bits of data are at the end of the 16-bit space while the most significant bits are filled with zeros. When `HAL_ADC_GetValue()` is called, it utilizes the **AHB Slave Interface** to retrieve the 32-bit audio samples instantly. This ensures real-time sound detection and immediate LED toggling without lag.

   For detailed configuration parameters and manual references, see Appendix I.

2. **GPIO Configuration:** We configured pin **PB2** as a GPIO output to control the LED. First, we set the `GPIOBEN` bit in the **AHB2ENR** register to enable the peripheral clock, which is disabled by default to save power. The **GPIOx_MODER** register was used to set the pin to output mode (01). For the LED output, we configured the **GPIOx_PUPDR** register to **No Pull-up/Pull-down** (00) to prevent internal resistors from interfering with the current. Finally, the **NVIC** and **SYS** peripherals managed the system's background timing and the debug connection, which allowed us to monitor the `sound_level` values in real-time during testing.

   For detailed configuration parameters and manual references, see Appendix III.

4.2 Software Flow

The software is written in C using the **HAL (Hardware Abstraction Layer)**. The logic follows a polling method inside a `while(1)` loop, where the system constantly reads the `sound_level` from the ADC. Instead of just looking for any sound, we implemented a threshold of 1000.

When a sound exceeds this threshold, the code enters a logic block that toggles the state of the LED on pin PB2. To prevent the system from toggling multiple times for a single clap (which can last several milliseconds), we added a short HAL_Delay. This acts as a "software debounce" ensuring the lock only triggers once per sound. By using this polling method combined with the high-speed AHB interface, the system can respond to triggers instantly without missing any data.

5 Group Contributions

Team Members	Contribution
Alaa Eddin	Hardware wiring, sensor signal testing, and code implementation.
Ali Alhaeri	Datasheet analysis and Report Writing
Mariam Hemeida	Hardware selection, threshold calibration testing, and code implementation.
Mustafa Turani	LED configuration, technical writing, and reference compilation.
Omar Al Nuaimi	Datasheet analysis and hardware troubleshooting.
Zaineh Khawaja	Hardware wiring, Code Implementation, and Report Writing.

Table 2: Group Contribution Matrix

6 Github repository

GitHub: https://github.com/meoweng/Hands-free-Switch/blob/main/README.md

References

[1] DFRobot. (2017). *Gravity: Analog sound sensor for Arduino (SKU: DFR0034).* https://www.dfrobot.com/product-83.html

[2] Onweagba, V. E. (2023). *Construction and testing of sound activated switch.* Michael Okpara University of Agriculture, Umudike, Nigeria.

[3] RobotShop. (n.d.). *Gravity: Analog sound sensor.* RobotShop. https://www.robotshop.com/products/gravity-sound-sensor

[4] STMicroelectronics. (2015). *RM0351 Reference manual: STM32L4x6 advanced ARM-based 32-bit MCUs.* https://www.se.rit.edu/~swen-563/resources/STM32L476/STM32L476VGT6%20Reference%20manual.pdf

[5] STMicroelectronics. (2018). *UM1879 User manual: Discovery kit with STM32L476VG MCU.* https://www.st.com/resource/en/user_manual/um1879-discovery-kit-with-stm32l476vg-mcu-stmicroelectronics.pdf

[6] STMicroelectronics. (2024). *HAL ADC how to use, in STM32CubeU5 HAL driver description.* https://dev.st.com/stm32cube-docs/

7 Appendix I

ADC1 Mode & Configuration

- IN6: Single Ended. (On every GPIO pin that has ADC capability is assigned a channel number. On STM32L476, **PA1** is connected to **ADC1_IN6**)
- Mode: Independent Mode.
- Clock Prescaler: Asynchronous clock mode divided by 1.
- Resolution: ADC 12-bit Resolution.
- Data Alignment: Right Alignment.
- Continuous Conversion Mode: Enabled.
- Sampling Time: 47.5 Cycles

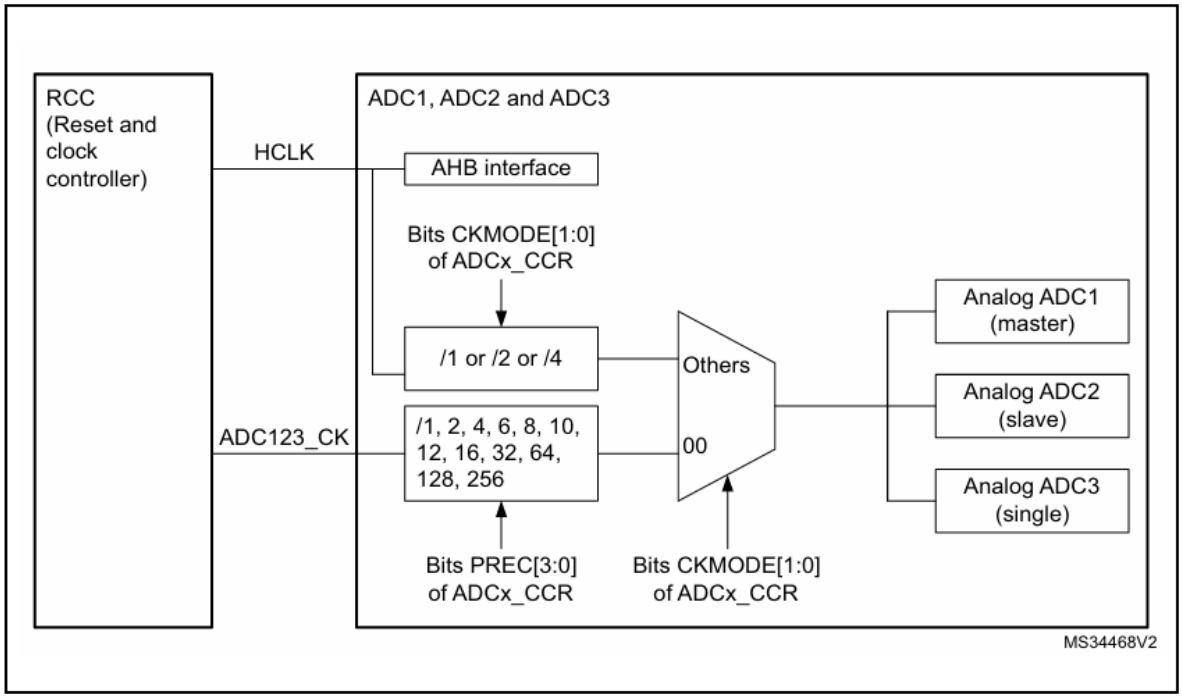


Figure 3: ADC Clock Scheme

To give a little insight into the diagram above, our project utilizes Asynchronous Clock Mode. This means the ADC operates on its own dedicated clock source and is not affected by the HCLK (the main system synchronous clock).

Setting CKMODE[1:0] to 00 in the configuration registers is what enables this asynchronous behavior. In this mode, we utilize a prescaler (a frequency divider) to adjust the clock speed. This ensures the ADC operates at the optimal frequency required for accurate audio sampling without exceeding the hardware's maximum timing specifications.

In the synchronous case, where the HCLK can reach high frequencies (up to 80 MHz), a 1/2/4 prescaler is necessary to reduce the clock speed. This prevents the ADC's internal circuitry from running too fast, which would otherwise compromise signal integrity and result in inaccurate digital conversions. By slowing the clock down to a supported range, we ensure the ADC has sufficient time to accurately capture the analog voltage levels from the sound sensor.

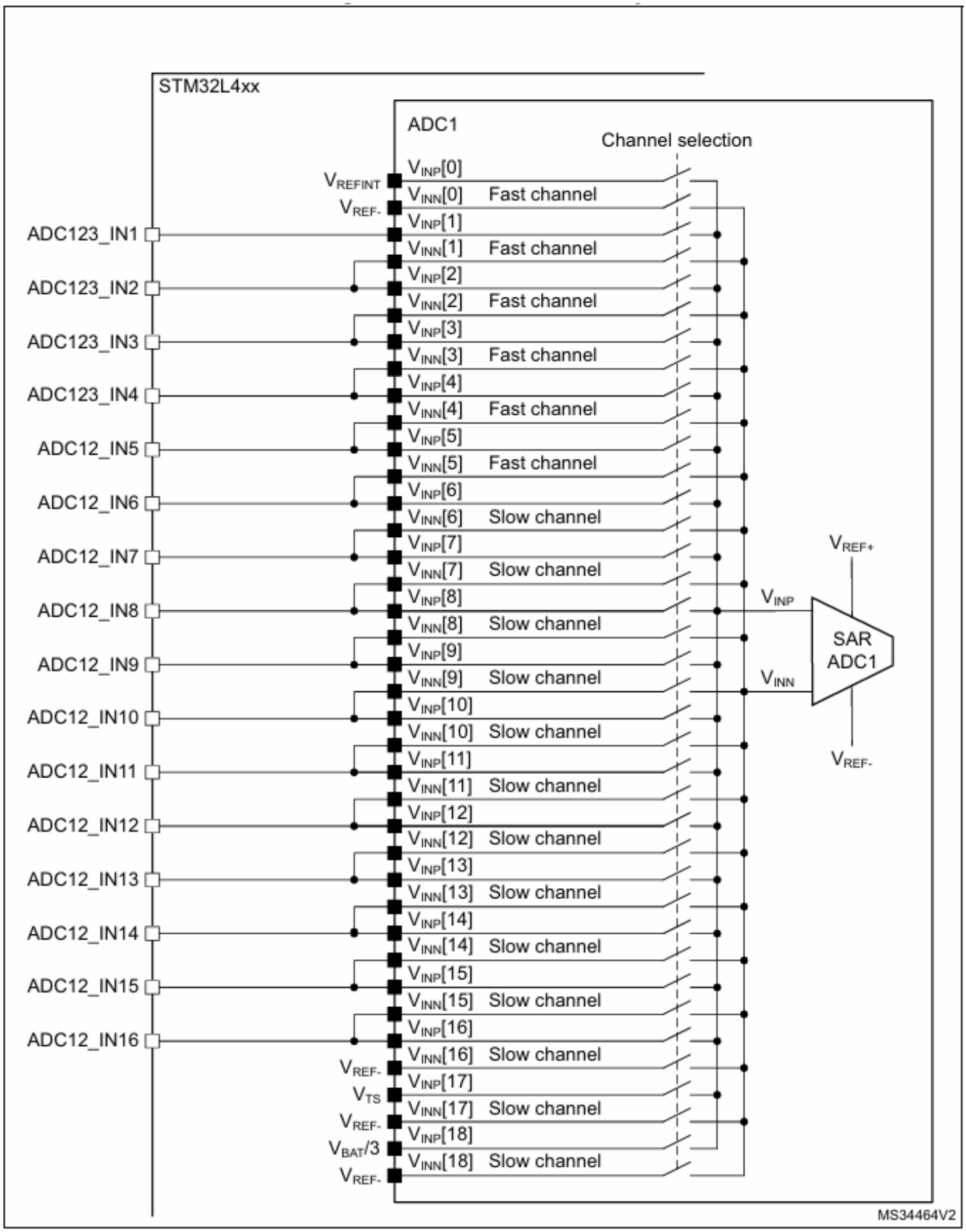


Figure 4: ADC1 Connectivity

16.2 ADC main features

- High-performance features
  - Up to 3x ADCs, out of which two of them can operate in dual mode
  - ADC1 is connected to 16 external channels + 3 internal channels
  - ADC2 is connected to 16 external channels + 2 internal channels
  - ADC3 is connected to 12 external channels + 4 internal channels
  - 12, 10, 8 or 6-bit configurable resolution
  - ADC conversion time:

    Fast channels: 0.188 µs for 12-bit resolution (5.33 Ms/s)

    Slow channels: 0.238 µs for 12-bit resolution (4.21 Ms/s)
  - ADC conversion time is independent from the AHB bus clock frequency
  - Faster conversion time by lowering resolution: 0.16 µs for 10-bit resolution
  - Can manage Single-ended or differential inputs (programmable per channels)
  - AHB slave bus interface to allow fast data handling
  - Self-calibration
  - Channel-wise programmable sampling time
  - Up to four injected channels (analog inputs assignment to regular or injected channels is fully configurable)
  - Hardware assistant to prepare the context of the injected channels to allow fast context switching
  - Data alignment with in-built data coherency
  - Data can be managed by GP-DMA for regular channel conversions
  - 4 dedicated data registers for the injected channels
- Oversampler
  - 16-bit data register
  - Oversampling ratio adjustable from 2 to 256x
  - Programmable data shift up to 8-bit
- Low-power features
  - Speed adaptive low-power mode to reduce ADC consumption when operating at low frequency
  - Allows slow bus frequency application while keeping optimum ADC performance (0.188 µs conversion time for fast channels can be kept whatever the AHB bus clock frequency)
  - Provides automatic control to avoid ADC overrun in low AHB bus clock frequency application (auto-delayed mode)
- Each ADC features an external analog input channel
  - Up to 5 fast channels from dedicated GPIO pads
  - Up to 11 slow channels from dedicated GPIO pads
- In addition, there are five internal dedicated channels
  - The internal reference voltage (V_{REFINT}), connected to ADC1
  - The internal temperature sensor (V_{TS}), connected to ADC1 and ADC3
  - The V_{BAT} monitoring channel ($V_{BAT}/3$), connected to ADC1 and ADC3
  - DAC1 and DAC2 internal channels, connected to ADC2 and ADC3

Figure 5: Key features of the STM32L476 ADC architecture

8 Appendix II

ADC Block Diagram

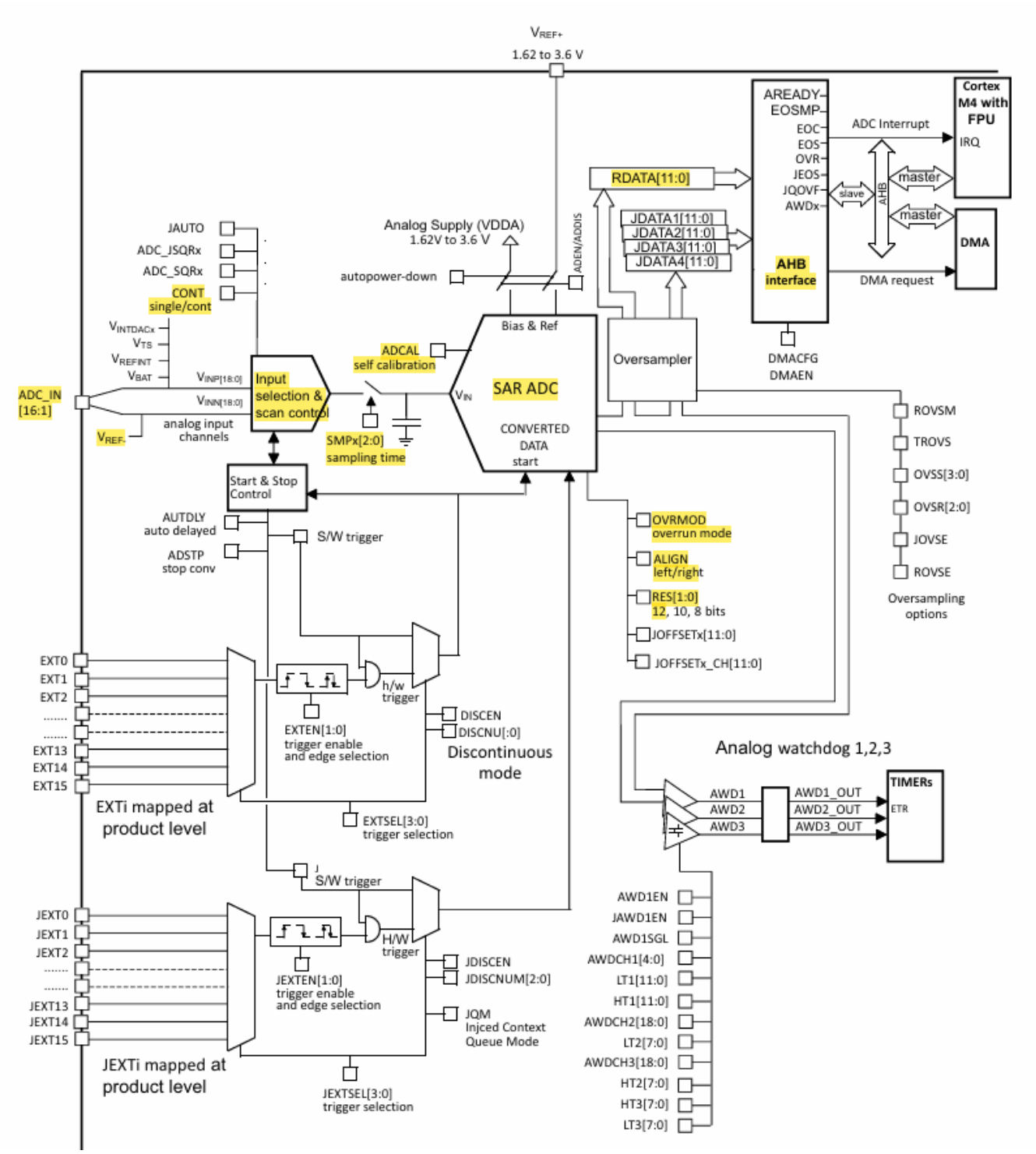


Figure 6: ADC Block Diagram

9 Appendix III

LED Configuration Tables

Register	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
AHB2ENR														RNGEN	HASHEN	AESEN		DCMIEN	ADCEN	OTGFSEN				GPIOIEN	GPIOHEN	GPIOGEN	GPIOFEN	GPIOEEN	GPIODEN	GPIOCEN	GPIOBEN	GPIOAEN
Value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	1	0

Value in hex: 0x00002002

Table 3: AHB2 Peripheral Clock Enable Register (AHB2ENR)

Register	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RCC-CCIPR	DFSDMSEL	SWPMI1SEL	ADCSEL[1:0]		CLK48SEL[1:0]		SAI2SEL[1:0]		SAI1SEL[1:0]		LPTIM2SEL[1:0]		LPTIM1SEL[1:0]		I2C3SEL[1:0]		I2C2SEL[1:0]		I2C1SEL[1:0]		LPUART1SEL[1:0]		UART5SEL[1:0]		UART4SEL[1:0]		UART3SEL[1:0]		UART2SEL[1:0]		UART1SEL[1:0]	
Value	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0

Value in hex: 0x10000004

Table 4: RCC Peripheral Independent Clock Configuration Register (RCC-CCIPR)

Register	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
MODER	MODER15		MODER14		MODER13		MODER12		MODER11		MODER10		MODER9		MODER8		MODER7		MODER6		MODER5		MODER4		MODER3		MODER2		MODER1		MODER0	
Mask	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0
Value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0

Mask in hex: 0x00000030 **Value in hex:** 0x00000010

Table 5: GPIOB Mode Register(MODER)

Register	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
OTYPER																	OT15	OT14	OT13	OT12	OT11	OT10	OT9	OT8	OT7	OT6	OT5	OT4	OT3	OT2	OT1	OT0
Mask	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
Value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Mask in hex: 0x00000004 **Value in hex:** 0x00000000

Table 6: GPIOB Output Type Register (OTYPER)

Register	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
PUPDR	PUPDR15		PUPDR14		PUPDR13		PUPDR12		PUPDR11		PUPDR10		PUPDR9		PUPDR8		PUPDR7		PUPDR6		PUPDR5		PUPDR4		PUPDR3		PUPDR2		PUPDR1		PUPDR0	
Mask	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0
Value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Mask in hex: 0x00000030 **Value in hex:** 0x00000000

Table 7: GPIOB Pull-up/Pull-down Register (PUPDR)

Register	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
BSRR	BR15	BR14	BR13	BR12	BR11	BR10	BR9	BR8	BR7	BR6	BR5	BR4	BR3	BR2	BR1	BR0	BS15	BS14	BS13	BS12	BS11	BS10	BS9	BS8	BS7	BS6	BS5	BS4	BS3	BS2	BS1	BS0
LED ON	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
LED OFF	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

LED ON: 0x00000004 **LED OFF:** 0x00040000

Table 8: GPIOB Bit Set/Reset Register (BSRR)

10 Appendix IV

Main Program Source Code

The following listing presents the complete `main.c` source code used in the implementation of the sound-activated LED control system. The program continuously samples the analog signal from the sound sensor through ADC1 and toggles the onboard LED connected to pin **PB2** whenever the detected sound level exceeds the predefined threshold.

```c
/* USER CODE BEGIN Header */
/**
  ******************************************************************************
  * @file           : main.c
  * @brief          : Sound Sensor Toggle Project
  ******************************************************************************
  */
/* USER CODE END Header */

/* Includes ------------------------------------------------------------------*/
#include "main.h"
#include <stdint.h>

/* Private variables ---------------------------------------------------------*/
ADC_HandleTypeDef hadc1; // Global declaration so main can see it

/* USER CODE BEGIN PV */
uint32_t sound_level = 0;
uint32_t threshold = 1000; // Adjust this if the LED toggles too easily
/* USER CODE END PV */

/* Private function prototypes -----------------------------------------------*/
void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_ADC1_Init(void);

/**
  * @brief  The application entry point.
  */
int main(void)
{
  /* MCU Configuration */
  HAL_Init();
  SystemClock_Config();

  /* Initialize all configured peripherals */
  MX_GPIO_Init();
  MX_ADC1_Init();

  /* USER CODE BEGIN 2 */
  // Calibrate and Start the ADC
  HAL_ADCEx_Calibration_Start(&hadc1, ADC_SINGLE_ENDED);
  HAL_ADC_Start(&hadc1);
  /* USER CODE END 2 */

  /* Infinite loop */
  while (1)
  {
    /* Check for ADC Conversion */
    if (HAL_ADC_PollForConversion(&hadc1, 10) == HAL_OK)
    {
        sound_level = HAL_ADC_GetValue(&hadc1);

        // Toggle LED if sound level is above threshold
        if (sound_level > threshold)
        {
            HAL_GPIO_TogglePin(GPIOB, GPIO_PIN_2);
            HAL_Delay(250); // Pause to avoid double-triggers
        }
    }
  }
}
```

```c

/**
  * @brief System Clock Configuration
  */
void SystemClock_Config(void)
{
  RCC_OscInitTypeDef RCC_OscInitStruct = {0};
  RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};

  if (HAL_PWREx_ControlVoltageScaling(PWR_REGULATOR_VOLTAGE_SCALE1) != HAL_OK)
  {
    Error_Handler();
  }

  RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_MSI;
  RCC_OscInitStruct.MSIState = RCC_MSI_ON;
  RCC_OscInitStruct.MSICalibrationValue = 0;
  RCC_OscInitStruct.MSIClockRange = RCC_MSIRANGE_6;
  RCC_OscInitStruct.PLL.PLLState = RCC_PLL_NONE;
  if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
  {
    Error_Handler();
  }

  RCC_ClkInitStruct.ClockType = RCC_CLOCKTYPE_HCLK|RCC_CLOCKTYPE_SYSCLK
                              |RCC_CLOCKTYPE_PCLK1|RCC_CLOCKTYPE_PCLK2;
  RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_MSI;
  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 ADC1 Initialization Function
  */
static void MX_ADC1_Init(void)
{
  ADC_ChannelConfTypeDef sConfig = {0};

  hadc1.Instance = ADC1;
  hadc1.Init.ClockPrescaler = ADC_CLOCK_ASYNC_DIV1;
  hadc1.Init.Resolution = ADC_RESOLUTION_12B;
  hadc1.Init.DataAlign = ADC_DATAALIGN_RIGHT;
  hadc1.Init.ScanConvMode = ADC_SCAN_DISABLE;
  hadc1.Init.EOCSelection = ADC_EOC_SINGLE_CONV;
  hadc1.Init.ContinuousConvMode = ENABLE;
  hadc1.Init.NbrOfConversion = 1;
  hadc1.Init.ExternalTrigConv = ADC_SOFTWARE_START;
  hadc1.Init.Overrun = ADC_OVR_DATA_OVERWRITTEN;
  if (HAL_ADC_Init(&hadc1) != HAL_OK)
  {
    Error_Handler();
  }

  sConfig.Channel = ADC_CHANNEL_6; // This is Pin PA1
  sConfig.Rank = ADC_REGULAR_RANK_1;
  sConfig.SamplingTime = ADC_SAMPLETIME_47CYCLES_5;
  if (HAL_ADC_ConfigChannel(&hadc1, &sConfig) != HAL_OK)
  {
    Error_Handler();
  }
}

/**
  * @brief GPIO Initialization Function
  */
static void MX_GPIO_Init(void)
{
```

```
  GPIO_InitTypeDef GPIO_InitStruct = {0};

  __HAL_RCC_GPIOA_CLK_ENABLE();
  __HAL_RCC_GPIOB_CLK_ENABLE();

  HAL_GPIO_WritePin(GPIOB, GPIO_PIN_2, GPIO_PIN_RESET);

  GPIO_InitStruct.Pin = GPIO_PIN_2;
  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);
}

void Error_Handler(void)
{
  __disable_irq();
  while (1) {}
}
```

Listing 1: Main program source code for sound-activated LED control

11 Appendix V

Sensor Datasheet

The Gravity Sound Sensor datasheet was used as reference for pin configuration. Key specifications from the datasheet include:

- Power supply: 5VDC (tolerates 3.3V for signal logic)
- Output: Analog voltage 0-VCC
- Interface: PH2.0 connector
- Pin 1: GND
- Pin 2: Power (VCC)
- Pin 3: Signal Output

Sensor Specifications:

- Power supply: 3.3VDC
- Output: Analog voltage proportional to sound intensity
- Interface: PH2.0 socket

Sensor Pin Connection Reference

The sensor interface uses a PH2.0 socket. Based on the datasheet attached in Appendix A, the following connection scheme was implemented:

1. The datasheet specifies Pin 1 as GND → Connected to STM32 Ground
2. The datasheet specifies Pin 2 as Power (VCC) → Connected to STM32 3.3V Output
3. The datasheet specifies Pin 3 as Signal Output → Connected to STM32 PA1 (ADC1_IN6)

Verification Results

After calibration and testing, the system demonstrated the following performance:

Parameter	Measured Value
Ambient ADC Reading	100 - 300
Clap ADC Reading	2500 - 3800
Threshold Setting	1000
Response Time	< 10 ms
Signal Return to Baseline	50 - 100 ms

Table 9: System Verification Results

The system successfully detects sound events and toggles the LED state with each valid detection, confirming proper hardware integration and calibration.

12 Appendix VI

STM32L476 Discovery Board Features

The STM32L476 Discovery board provides multiple integrated peripherals and interfaces that support embedded system development. Table 10 summarizes the main features relevant to this project.

Table 10: STM32L476 Discovery Board Key Features

Feature	Description
Microcontroller	STM32L476VGT6 (ARM Cortex-M4)
Flash Memory	1 MB
SRAM	128 KB
ADC	12-bit Analog-to-Digital Converter
GPIO Pins	Multiple configurable input/output pins
Power Supply	3.3V operation
Debugging Tool	(add debugging tool if we have or remove this row)
User LEDs	PB2 Red

These features enable the implementation of real-time embedded applications such as the sound-controlled lighting system.