Function Generator

Kieran Valino

May 2, 2023

Behavior Description

This function generator outputs sinusoidal, triangle, sawtooth, and square waveforms with a variable duty cycle at 3V. These 3V waveforms range from 100 Hz to 500 Hz with 100 Hz increments. The duty cycle of the square waveform ranges from 10% to 90% with 10% increments. The default output is a square wave at 100 Hz with a 50% duty cycle. The 12-button keypad adjusts the output of the function generator.

1 - Output a 100 Hz wave
2 - Output a 200 Hz wave
3 - Output a 300 Hz wave
4 - Output a 400 Hz wave
5 - Output a 500 Hz wave
6 - Change waveform to sinusoidal wave
7 - Change waveform to triangle wave
8 - Change waveform to sawtooth wave
9 - Change waveform to square wave
0 - Reset square wave duty cycle to 50%
* - Decrease square wave duty cycle by 10%
# - Increase square wave duty cycle by 10%

System Specifications:

Board:

TYPE	DESCRIPTION
Name	STM32L476RG
Manufacturer	STMicroelectronics
Series	STM32L4
Core	ARM Cortex M4
Mounting Type	MCU 32 bit
Operating Frequency	32 MHz
Flash Memory Size	1 Mbyte
Operating Supply Voltage	3.3V
Package Pin Count	64 pins
Interface Type	USB
Unit Weight	10.582189 oz.

Table 1: Board specifications

Keypad:

TYPE	DESCRIPTION
Name	COM-14662
Manufacturer	SparkFun Electronics
Number of Keys	12
Matrix (Columns x Rows)	3 x 4
Switch Type	Conductive Rubber
Illumination	Non-illuminated
Legend Type	Fixed
Key Type	Polymer
Legend	Telephone Format
Mounting Type	Panel Mount, Front

Table 2: Keypad specifications

DAC:

TYPE	**DESCRIPTION**
Name	MCP4921-E/P
Manufacturer	Microchip Technology
Number of Bits	12
Number of D/A Converters	1
Setting Type	4.5µs (Typ)
Output Type	Voltage - Buffered
Data Interface	SPI
Voltage - Supply, Analog	2.7 V ~ 5.5 V
Voltage - Supply, Digital	2.7 V ~ 5.5 V
INL - Integral Nonlinearity	+/- 4 LSB
DNL - Differential Nonlinearity	+/- 0.25 LSB
Architecture	String DAC
Operating Temperature	-40°C ~ 125°C
Mounting Type	Through Hole

Table 1: DAC specifications

System Schematic:

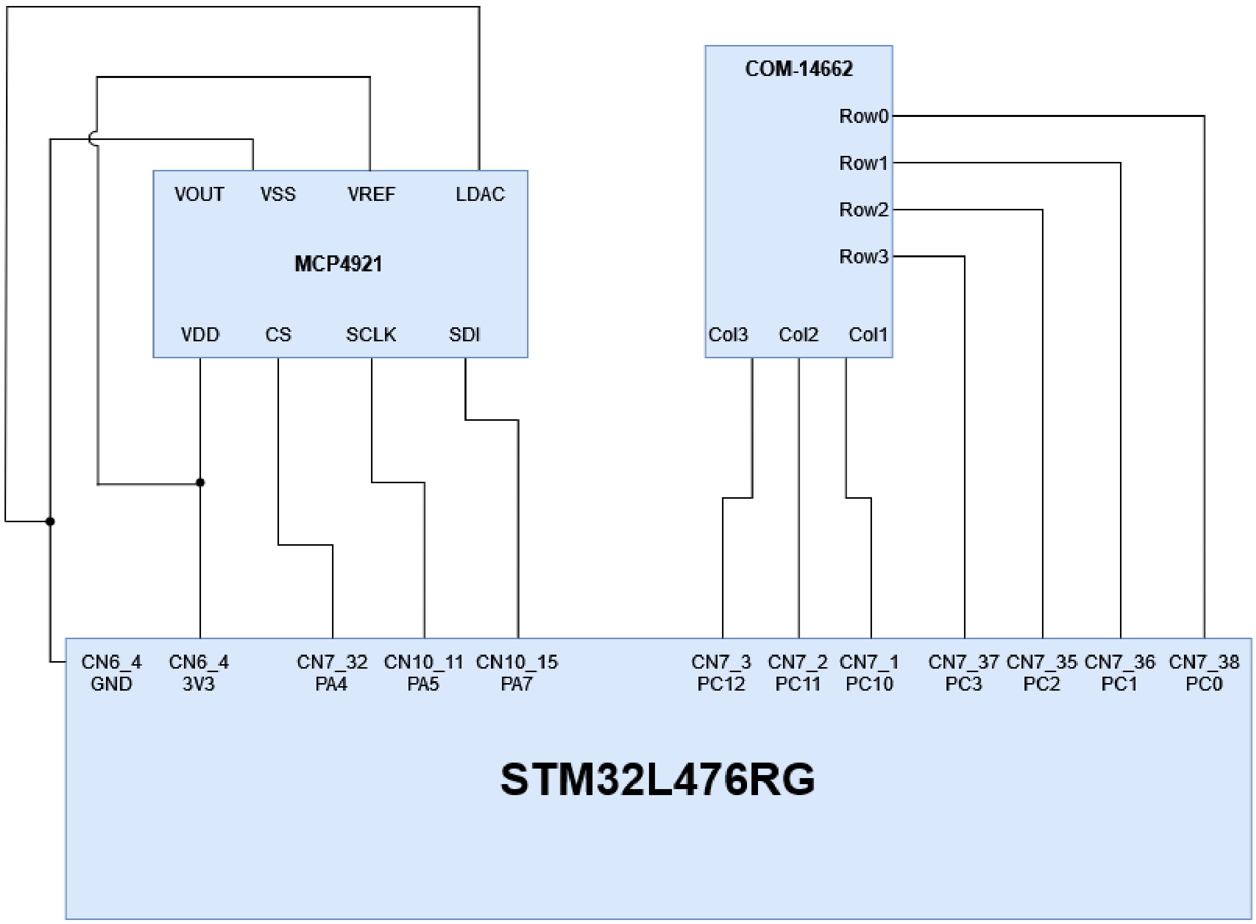

Figure 1: Function Generator Schematic showcasing how the STM32L476RG board is connected to the COM-14662 keypad and MCP4921 DAC

Software Architecture:

Main function:

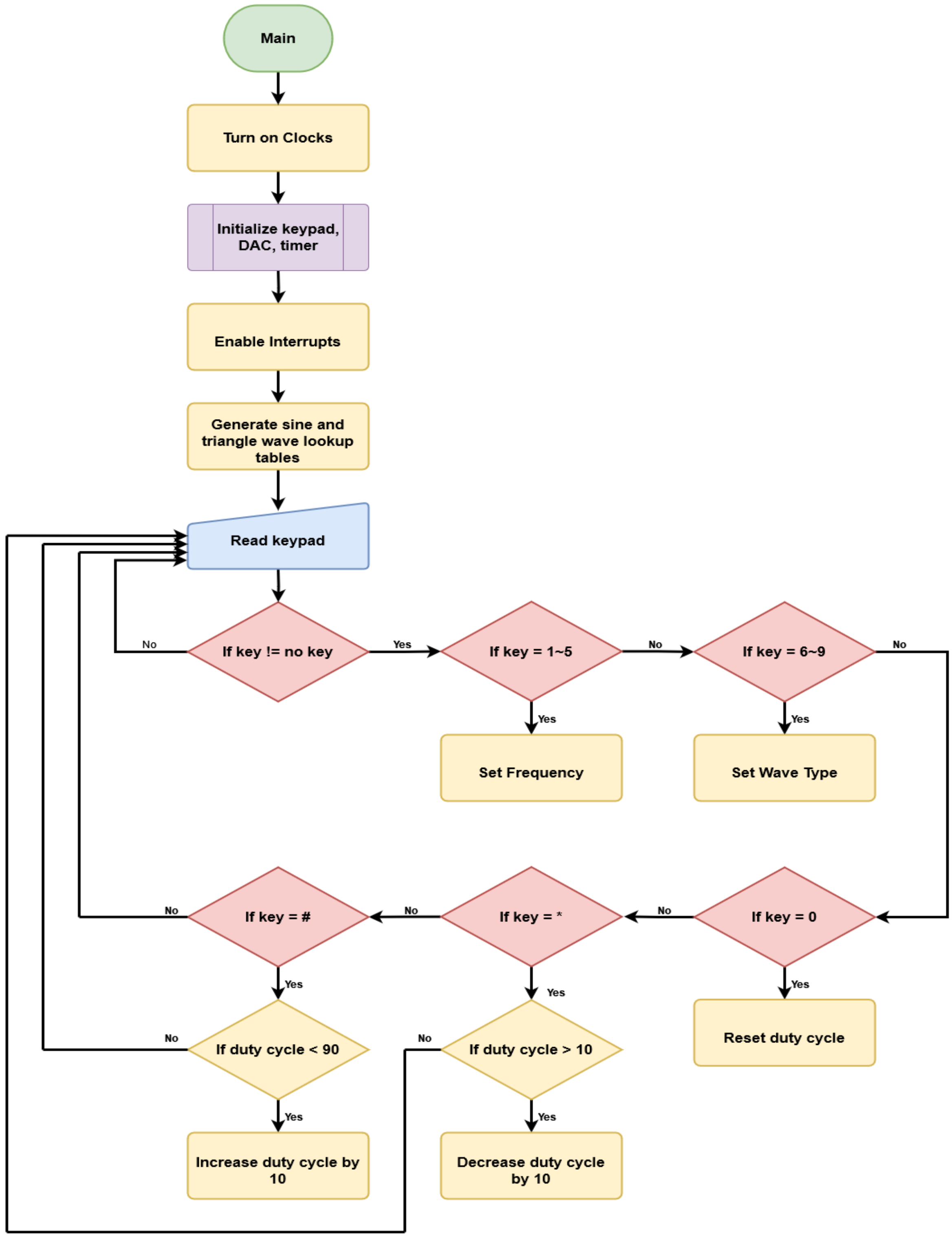

Figure 2: Main function flowchart demonstrating the main process of the Function Generator (see appendix for code details)

Interrupt Handler function:

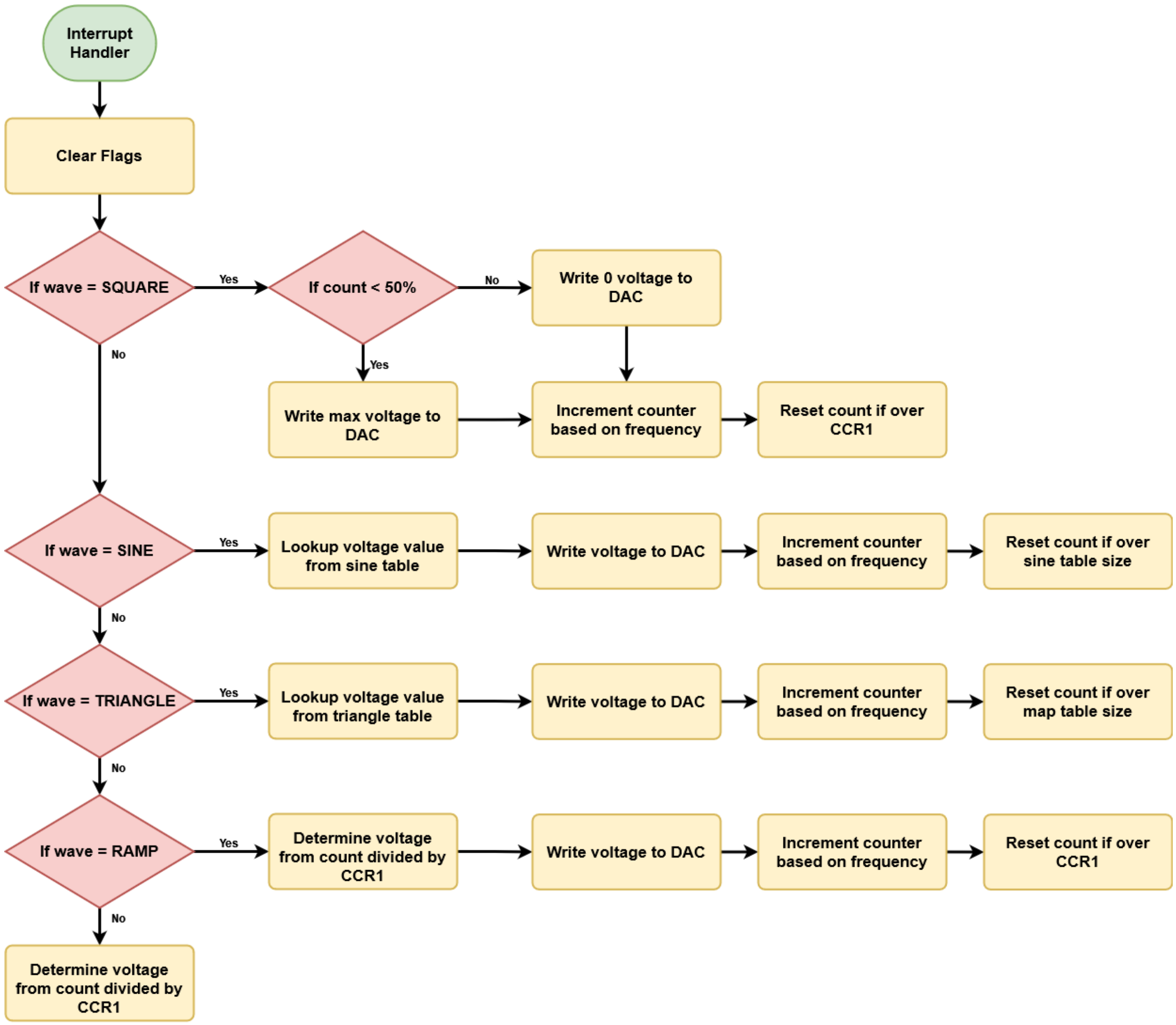

Figure 3: Interrupt handler function flowchart for Timer 2 demonstrating the interrupt process of the Function Generator (see appendix for code details)

Generate Sine Table:

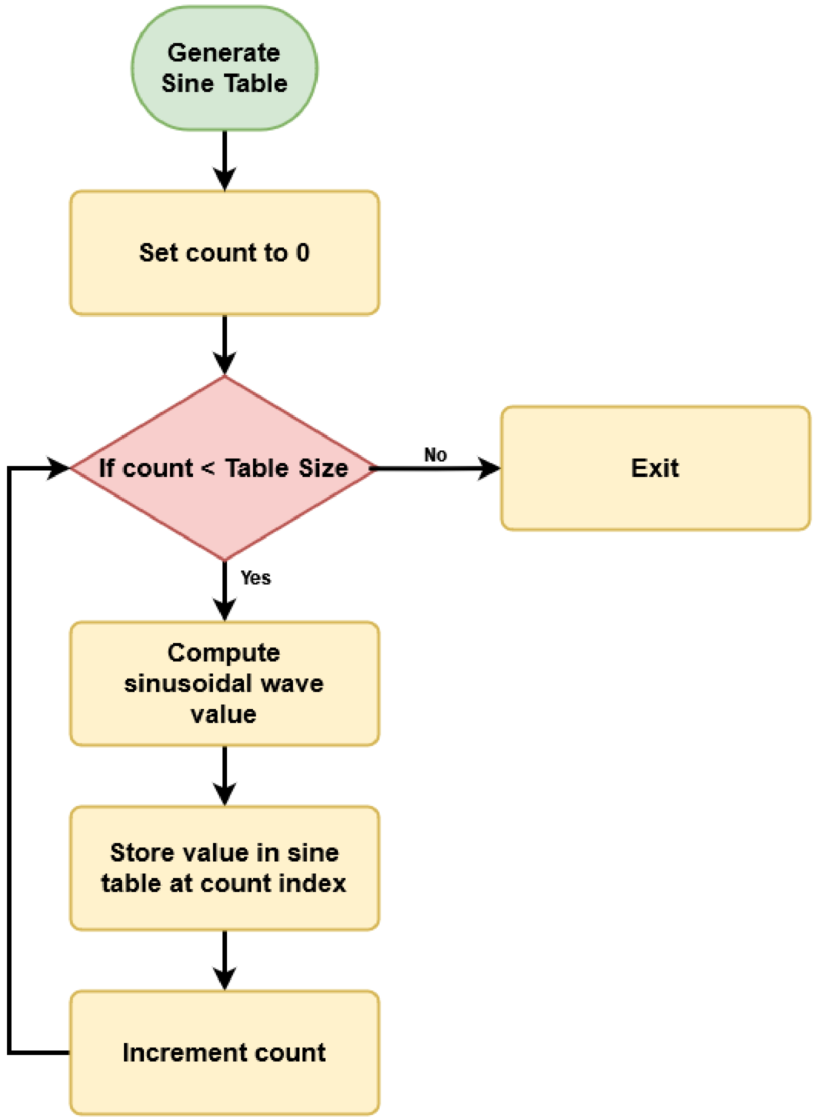

Figure 4: Sinusoidal lookup table generation function flowchart to assist in the generation of sinusoidal waves for the interrupt process of the Function Generator (see appendix for code details)

Generate Triangle Table:

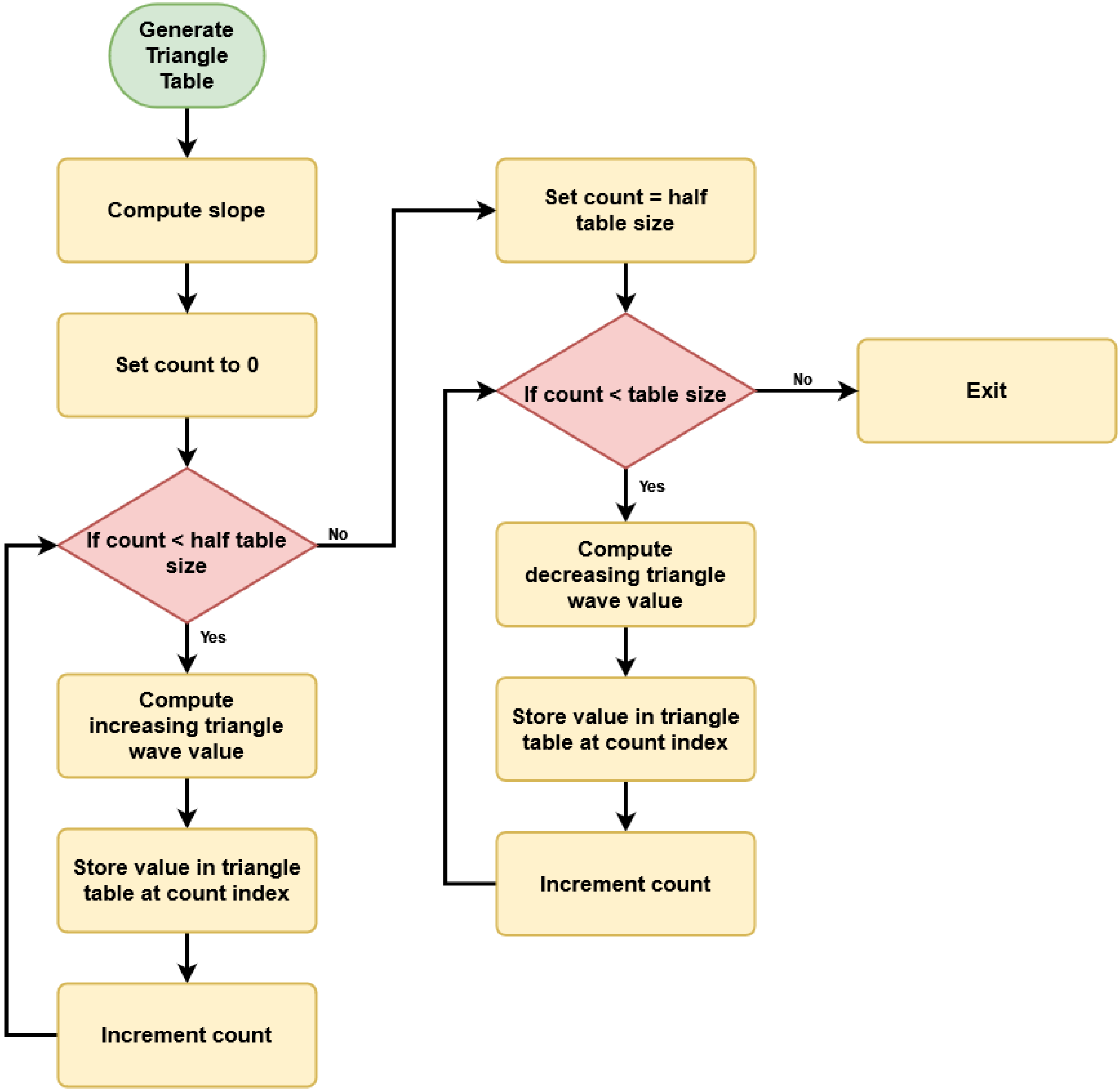

Figure 5: Triangle lookup table generation function flowchart to assist in the generation of triangular waves for the interrupt process of the Function Generator (see appendix for code details)

Appendix:

main.c

```
#include "main.h"
#include "keypad.h"
#include "dac.h"
#include <stdlib.h>
#include <stdint.h>
#include <math.h>
#include <unistd.h>
#include <time.h>

#define arr 0xFFFFFFFF

void TIM2_init(void);

uint16_t sine_table[TABLE_SIZE];
uint16_t triangle_table[TABLE_SIZE];

void SystemClock_Config(void);

WaveformMode waveform = SQUARE;
uint16_t duty_cycle = 50;
uint32_t ccr1 = 567;
uint32_t count = 0;
uint32_t dac_value;
uint16_t frequency = 1;

int main(void)
{

  HAL_Init();
  SystemClock_Config();

  // Set Up Clocks

  // Enable SPI1 clock
  RCC->APB2ENR |= RCC_APB2ENR_SPI1EN;
  // Turn on the clock for GPIOA
  RCC->AHB2ENR |= RCC_AHB2ENR_GPIOAEN; // clock is meant for SPI Pins
  // Turn on the clock for GPIOC
  RCC->AHB2ENR |= RCC_AHB2ENR_GPIOCEN; // clock is meant Keypad

  keypad_init();
  DAC_init();
  TIM2_init();

  __enable_irq();

  generate_sine_table();
  generate_triangle_table();
```

```c
int8_t keypad_val = 0;


while(1)
{
    keypad_val = keypad_read();

    // Button debounce delay
    uint8_t debounce = keypad_val;
    for (uint32_t i = 0; i < DELAY_TIME; i++);
    if(keypad_val != -1)
    {

        if (keypad_val == debounce)
        {
                switch(keypad_val)
                {
                        case 1:
                                // 100 Hz
                                frequency = 1;
                                break;

                        case 2:
                                // 200 Hz
                                frequency = 2;
                                break;

                        case 3:
                                // 300 Hz
                                frequency = 3;
                                break;

                        case 4:
                                // 400 Hz
                                frequency = 4;
                                break;

                        case 5:
                                // 500 Hz
                                frequency = 5;
                                break;

                        case 6:
                                // Sine wave output
                                waveform = SINE;
                                break;

                        case 7:
                                // Triangle wave output
                                waveform = TRIANGLE;
                                break;
```

```
                        case 8:
                                // Sawtooth/Ramp wave output
                                waveform = RAMP;
                                break;

                        case 9:
                                // Square wave output
                                waveform = SQUARE;
                                break;

                        case 0:
                                // Reset duty_cycle to 50 %
                                duty_cycle = 50;
                                break;

                        case 10:                    // * key
                                // Decrement duty_cycle by 10 %
                                if (duty_cycle > 10)
                                        duty_cycle -= 10;
                                break;

                        case 12:                    // # key
                                // Increment duty_cycle by 10 %
                                if (duty_cycle < 90)
                                        duty_cycle += 10;
                                break;
                }
        }
    }
  }

}

// Define TIM2 ISR
void TIM2_IRQHandler(void)
{
    TIM2->SR &= ~((TIM_SR_UIF) | (TIM_SR_CC1IF));    // Clear UIF and CC1IF flag

    switch(waveform)
    {
        case SQUARE:

                if(count < (duty_cycle * ccr1) / 100)    // Check if count is less than duty
cycle percentage of CCR1
                {
                        DAC_write(VOLT_MAX);
                }
                else{
                        DAC_write(0);
                }
                count = (count + frequency);        // Increment count based on frequency
```

```
                if(count > (ccr1 - frequency))      // subtracting frequency is for some
instrument fine tuning
                {
                        count = 0;
                }
                break;

        case SINE:
               dac_value = sine_table[count];
               DAC_write(dac_value);
               count = (count + frequency) % TABLE_SIZE;    // Increment count based on
frequency, reset if over table size
                break;

        case TRIANGLE:
               dac_value = triangle_table[count];
               DAC_write(dac_value);
               count = (count + frequency) % TABLE_SIZE;    // Increment count based on
frequency, reset if over table size
               break;

        case RAMP:
               DAC_write(VOLT_MAX * count / ccr1);
               count += (frequency);                // Increment count based on frequency
               if (count > ccr1 - 3)                // subtracting 3 is for some instrument
fine tuning
               {
               count = 0;
               }
               break;

    }
    TIM2->CCR1 += ccr1;                      // Increment CCR1 by previous CCR1
}



void TIM2_init(void)
{
       // Configure Timer 2
       RCC->APB1ENR1     |=     RCC_APB1ENR1_TIM2EN;
       TIM2->DIER      |=     (TIM_DIER_UIE | TIM_DIER_CC1IE);    // enable update event and
compare/capture 1 interrupt
       TIM2->SR        &=  ~(TIM_SR_UIF | TIM_SR_CC1IF);          // clear update event and
compare/capture 1 flag
       TIM2->ARR       =     arr - 1;
       TIM2->CCR1      =     ccr1 - 1;
       TIM2->CCMR1     &=  ~(TIM_CCMR1_CC1S_0 | TIM_CCMR1_CC1S_1);    // set output compare
mode to Toggle
       TIM2->CCER      |=     TIM_CCER_CC1E;                              // enable
capture/compare channel 1 output
       TIM2->CR1       |=     TIM_CR1_ARPE;                               // enable
```

```
auto-reload preload
	TIM2->CR1       |=      TIM_CR1_CEN;                                        // start
timer

	// Enable interrupt in NVIC
	NVIC->ISER[0] = (1 << (TIM2_IRQn & 0x1F));
}


void generate_sine_table(void) {
	for (int i = 0; i < TABLE_SIZE; i++) {
	 sine_table[i] = (uint16_t) (VOLT_MAX / 2.0 * sin(2 * M_PI * i / TABLE_SIZE) +
VOLT_MAX / 2.0);
	}
}

void generate_triangle_table(void) {
  uint16_t half_size = TABLE_SIZE / 2;
  float slope = (float)VOLT_MAX / half_size;

  // Generate first half of the triangle wave
  for (uint16_t i = 0; i < half_size; i++) {
	triangle_table[i] = (uint16_t)(slope * i);
  }

  // Generate second half of the triangle wave
  for (uint16_t i = half_size; i < TABLE_SIZE; i++) {
	triangle_table[i] = (uint16_t)(VOLT_MAX - slope * (i - half_size));
  }
}


/**
  * @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_MSI;
```

```
  RCC_OscInitStruct.MSIState = RCC_MSI_ON;
  RCC_OscInitStruct.MSICalibrationValue = 0;
  RCC_OscInitStruct.MSIClockRange = RCC_MSIRANGE_10; // Set frequency to 32 MHz
  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_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();
  }
}

/* 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 */
```

main.h

```
#ifndef __MAIN_H
#define __MAIN_H

#include <stdint.h>

#ifdef __cplusplus
extern "C" {
#endif

#define DELAY_TIME 250000
#define VOLT_MAX 3300
#define TABLE_SIZE 567

void SystemClock_Config(void);
void generate_sine_table(void);
void generate_triangle_table(void);

typedef enum {
    SQUARE,
    SINE,
    TRIANGLE,
    RAMP
} WaveformMode;


/* Includes ------------------------------------------------------------------*/
#include "stm32l4xx_hal.h"

/* Private includes ----------------------------------------------------------*/

void Error_Handler(void);

#ifdef __cplusplus
}
#endif

#endif /* __MAIN_H */
```

DAC.c

```
///*
// * dac.c
// *
// *  Created on: Apr 18, 2023
// *  Authors: Luis D. Garcia and Kieran Valino
// *  Description: MCP4922 DAC Library Implementation File that has the
// *                   functions DAC_init, DAC_write, and DAC_volt_conv, that
// *                   all work to setup the pins, convert a voltage to a 12-bit
// *                   value, and write the 12-bit value to the DAC
//*/

#include "dac.h"
#include "stm32l4xx_hal.h"
#include <stdint.h>

// Function to write initialize the PA4, PA5, and PA7 Pins in SPI Mode
void DAC_init(void)
{
    // GPIO Configuration of PA4 - PA7 to Alternate Function for SPI
    GPIOA->MODER &= ~(GPIO_MODER_MODE4 | GPIO_MODER_MODE5 | GPIO_MODER_MODE7);
    GPIOA->MODER |= (GPIO_MODER_MODE4_0 | GPIO_MODER_MODE5_1 | GPIO_MODER_MODE7_1);

    // User manual requires all alternate functions to be set to 5 for SPI Usage
    GPIOA->AFR[0] |= ((SPI_AF5 << GPIO_AFRL_AFSEL5_Pos)| (SPI_AF5 << GPIO_AFRL_AFSEL7_Pos));

    // Configure SPI1 CR to set MSTR, Baudrate: fCLK/2, SPI Enable, SSI, and SSM
    SPI1->CR1 |= SPI_CR1_MSTR | SPI_CR1_SSI | SPI_CR1_SSM;

    //Configure SPI CR2 to set NSSP, TXEIE, DS = 1111
    SPI1->CR2 |= (SPI_CR2_DS_3| SPI_CR2_DS_2 | SPI_CR2_DS_1 | SPI_CR2_DS_0);

    SPI1->CR1 |= SPI_CR1_SPE;
}

// Function to write a 12-bit value to the DAC
void DAC_write(uint16_t packet)
{
    //SPI1->CR1 &= ~SPI_CR1_SSI;                            // turn off CS when writing
    GPIOA->ODR &= ~GPIO_ODR_OD4;

    while (!(SPI1->SR & SPI_SR_TXE));         // wait until transmit buffer is empty

//    SPI1->DR = (packet & ~(DAC_CONFIG_MASK)) | (DAC_CONFIG_MOD1 << 12);
    SPI1->DR = packet | (DAC_CONFIG_MOD1 << 12);

    while ((SPI1->SR & SPI_SR_BSY));          // wait until transmit buffer is empty

    // set the configurations bits and send packet
    GPIOA->ODR |= GPIO_ODR_OD4;
    //SPI1->CR1 |= SPI_CR1_SSI;                                    // turn on CS when done
```

```
}
// Function to convert inputed voltage in keypad to 12-bit value
uint16_t DAC_volt_conv(uint16_t voltage)
{
       /*
       Description of Conversion Function:
         The 0 is 1000's place, 1 is the 100's place, and 2 is the 10's place in the buffer.
         The voltage inputed into keypad gets buffered and then transformed into a 12 bit
         value that the DAC is able to process and output as a voltage.
       */

       uint16_t first = voltage / 100;
       uint16_t second = (voltage % 100) / 10;
       uint16_t third = voltage % 10;
    uint16_t volt_val_12_bits = first * 1000 +
                                      second * 100 +
                                                     third * 10;

       // When the voltage inputed is higher than reference voltage, output the reference
voltage
       if (volt_val_12_bits > VREF)
         volt_val_12_bits = VREF;


       // Utilize Determined Calibrated Equation to Account for Non-Ideal Equation
       volt_val_12_bits = CALIBRATED_MULT * volt_val_12_bits - CALIBRATED_OFFSET;


       return volt_val_12_bits;
}
```

DAC.h

```
/*
 * dac.h
 *
 * Created on: Apr 18, 2023
 * Authors: Luis D. Garcia and Kieran Valino
 * Description: MCP4922 DAC Library Header File that has the
 * functions DAC_init, DAC_write, and DAC_volt_conv.
*/

#ifndef SRC_DAC_H_
#define SRC_DAC_H_

#include "stm32l4xx_hal.h"
#include <stdint.h>

#define VOLTAGE_BUF_SIZE     3                          // Voltage Buffer Size is 3
#define DAC_CONFIG_MASK     0xF000                      // Mask to Clear Configuration Bits
#define DAC_CONFIG_MOD1     3                           // Mode enables 12-bit DAC to output
Voltage
#define VREF                         4096                   // 3.30 V to 3300 mV
#define TOTAL_BITS                   12                     // DAC Requires 12 Bits
#define SPI_AF5               5                         // Permits use of pins for SPI CS, POCI,
SCK
#define CALIBRATED_MULT     1.251587463     // Calculated Multiplier for Calibration
Equation
#define CALIBRATED_OFFSET     12                        // Calculated Offset for Calibration
Equation

void DAC_init(void);
void DAC_write(uint16_t dac_value);
uint16_t DAC_volt_conv(uint16_t voltage);
```

keypad.c

```c
#include "keypad.h"
#include <stdint.h>
#include <stdlib.h>



void keypad_init()
{

    // columns Set Up
    // clear GPIOC PB10-PB13
    GPIOC->MODER       &=    ~(GPIO_MODER_MODE10
                                  | GPIO_MODER_MODE11
                                  | GPIO_MODER_MODE12
                                  | GPIO_MODER_MODE13);

    // set MODE = 01 (output)
    GPIOC->MODER       |=    ( (1 << GPIO_MODER_MODE10_Pos)
                                  | (1 << GPIO_MODER_MODE11_Pos)
                                  | (1 << GPIO_MODER_MODE12_Pos)
                                  | (1 << GPIO_MODER_MODE13_Pos));

    // set to push-pull output
    GPIOC->OTYPER      &=    ~(GPIO_OTYPER_OT10
                                  | GPIO_OTYPER_OT11
                                  | GPIO_OTYPER_OT12
                                  | GPIO_OTYPER_OT13);

    // no pull-up or pull-down resistors
    GPIOC->PUPDR       &=    ~(GPIO_PUPDR_PUPD10
                                  | GPIO_PUPDR_PUPD11
                                  | GPIO_PUPDR_PUPD12
                                  | GPIO_PUPDR_PUPD13);

    // low speed
    GPIOC->OSPEEDR   &=    ~(GPIO_OSPEEDR_OSPEED10
                                  | GPIO_OSPEEDR_OSPEED11
                                  | GPIO_OSPEEDR_OSPEED12
                                  | GPIO_OSPEEDR_OSPEED13);

    // set GPIOAumns to high
    GPIOC->ODR       |= (GPIO_ODR_OD10 | GPIO_ODR_OD11 | GPIO_ODR_OD12| GPIO_ODR_OD13);

    // Rows Set Up
    // clear GPIOC PB0-PB3 and set to input
    GPIOC->MODER      &=    ~(GPIO_MODER_MODE0
                                  | GPIO_MODER_MODE1
                                  | GPIO_MODER_MODE2
                                  | GPIO_MODER_MODE3);
    // set MODE = 01 (output)
    GPIOC->MODER       |=    ( (0 << GPIO_MODER_MODE0_Pos)
                                  | (0 << GPIO_MODER_MODE1_Pos)
```

```
                                                | (0 << GPIO_MODER_MODE2_Pos)
                                                | (0 << GPIO_MODER_MODE3_Pos));
    // clear pupdr
    GPIOC->PUPDR        &=    ~(GPIO_PUPDR_PUPD0_1
                                                | GPIO_PUPDR_PUPD1_1
                                                | GPIO_PUPDR_PUPD2_1
                                                | GPIO_PUPDR_PUPD3_1);
    // set pupdr to pull-down
    GPIOC->PUPDR        |=    (2 << GPIO_PUPDR_PUPD0_Pos
                                                | 2 << GPIO_PUPDR_PUPD1_Pos
                                                | 2 << GPIO_PUPDR_PUPD2_Pos
                                                | 2 << GPIO_PUPDR_PUPD3_Pos);
}

uint16_t keypad_read(void)
{
    uint8_t rows;
//    uint8_t cols;
    uint16_t bits;
    COL_PORT |= COL_MASK;

    uint8_t row_debug = ROW_PORT;

        rows = ROW_PORT & (ROW_MASK);

    if(rows != 0){
        bits = 0;
        // set output
        COL_PORT &= ~COL_MASK;
        COL_PORT |= COL_MASK_HIGH0;
        // read
        rows = ROW_PORT & ROW_MASK;
        // scan rows and return first found value
        if(rows != 0){
                if (rows == 8){
                        return STAR;
                }
                for (int i = 0; i < 3; i++){
                        if ((rows & 1) == 0){
                                rows = rows >> 1;
                        } else {
                                return 1 + 3 * i;
                        }

                }
        }
        // set output
        COL_PORT &= ~COL_MASK;
        COL_PORT |= COL_MASK_HIGH1;
        // read
        rows = ROW_PORT & ROW_MASK;
        // if valid, reset and return
```

```
        // scan rows and return first found value
        if(rows != 0){
                if (rows == 8){
                        return 0;
                }
                for (int i = 0; i < 3; i++){
                        if ((rows & 1) == 0){
                                rows = rows >> 1;
                        } else {
                                return 2 + 3 * i;
                        }

                }
        }
        // set output
        COL_PORT &= ~COL_MASK;
        COL_PORT |= COL_MASK_HIGH2;
        // read
        rows = ROW_PORT & ROW_MASK;
        // add and shift
        // scan rows and return first found value
        if(rows != 0){
                if (rows == 8){
                        return POUND;
                }
                for (int i = 0; i < 3; i++){
                        if ((rows & 1) == 0){
                                rows = rows >> 1;
                        } else {
                                return 3 + 3 * i;
                        }
                }
        }
    }
    return NOKEY;
}
```

keypad.h

```
#include "main.h"

#define COL_PORT GPIOC->ODR          // output
#define ROW_PORT GPIOC->IDR          // input


#define COL_MASK (GPIO_ODR_OD10 | GPIO_ODR_OD11 | GPIO_ODR_OD12 | GPIO_ODR_OD13)
#define COL_MASK_HIGH0 (GPIO_ODR_OD10)
#define COL_MASK_HIGH1 (GPIO_ODR_OD11)
#define COL_MASK_HIGH2 (GPIO_ODR_OD12)
#define COL_MASK_HIGH3 (GPIO_ODR_OD13)
#define COL_OFFSET 4

#define COL_0 GPIO_ODR_OD10
#define COL_1 GPIO_ODR_OD11
#define COL_2 GPIO_ODR_OD12

#define ROW_MASK (GPIO_IDR_ID0 | GPIO_IDR_ID1 | GPIO_IDR_ID2 | GPIO_IDR_ID3)
#define ROW_0 GPIO_IDR_ID0;
#define ROW_1 GPIO_IDR_ID1;
#define ROW_2 GPIO_IDR_ID2;
#define ROW_3 GPIO_IDR_ID3;

#define STAR 10
#define POUND 12
#define NOKEY 0xFF

void keypad_init();
uint16_t keypad_read();
```