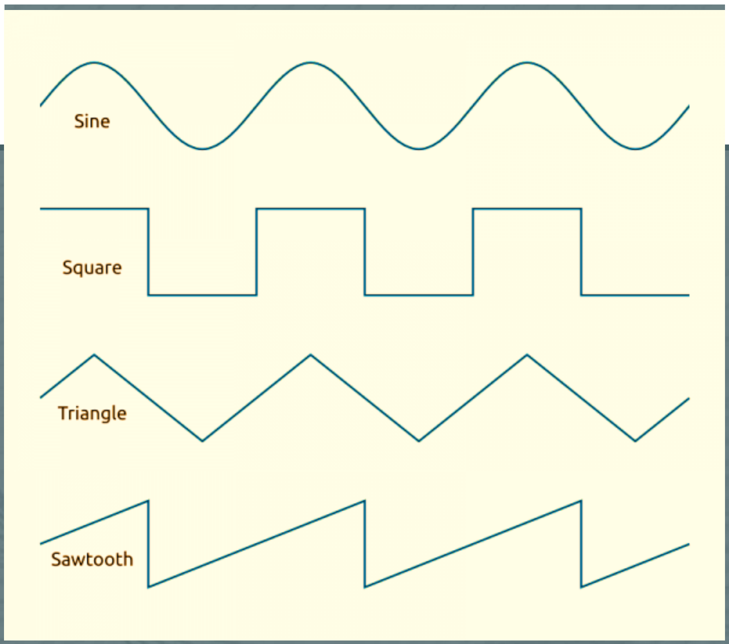
Function Generator

Shawheen Ghezavat

Spring, May 1, 2024

Behavior Description

The Function Generator is designed to produce various waveforms using the Nucleo-L476RG microcontroller interfaced with an external MCP4921 DAC. It can generate sine, square, sawtooth, and triangle waveforms at frequencies ranging from 100 Hz to 500 Hz. Users can select and modify waveform types and settings through a keypad, which allows easy adjustments of waveform, frequency and square wave duty cycle. The device utilizes timers and interrupts to ensure accurate waveform outputs.



System Specifications

Specification	Details
Power Supply Voltage	3.3V DC
Power Consumption	Approx. 100 mA
Battery Life	N/A (powered via USB)
DAC Bit Resolution	12 bits (MCP4921)
Time Resolution/Waveform Accuracy	Up to 300,000 samples per second
Clock Frequency	40 MHz
Physical Dimensions	75 mm x 54 mm x 15 mm
Weight	Approx. 50 grams
Environmental Tolerance	Operating temperature 0°C to 70°C
Interface	4x4 Membrane Keypad
Output Waveform Types	Sine, Square, Triangle, Sawtooth
Frequency Range	100 Hz-500 Hz (can be adjusted)
Output Voltage Peak-to-Peak	3.0 V
DC Offset	1.5 V
Duty Cycle Range	10%-90%, ±10% increments

Table 1

System Schematic

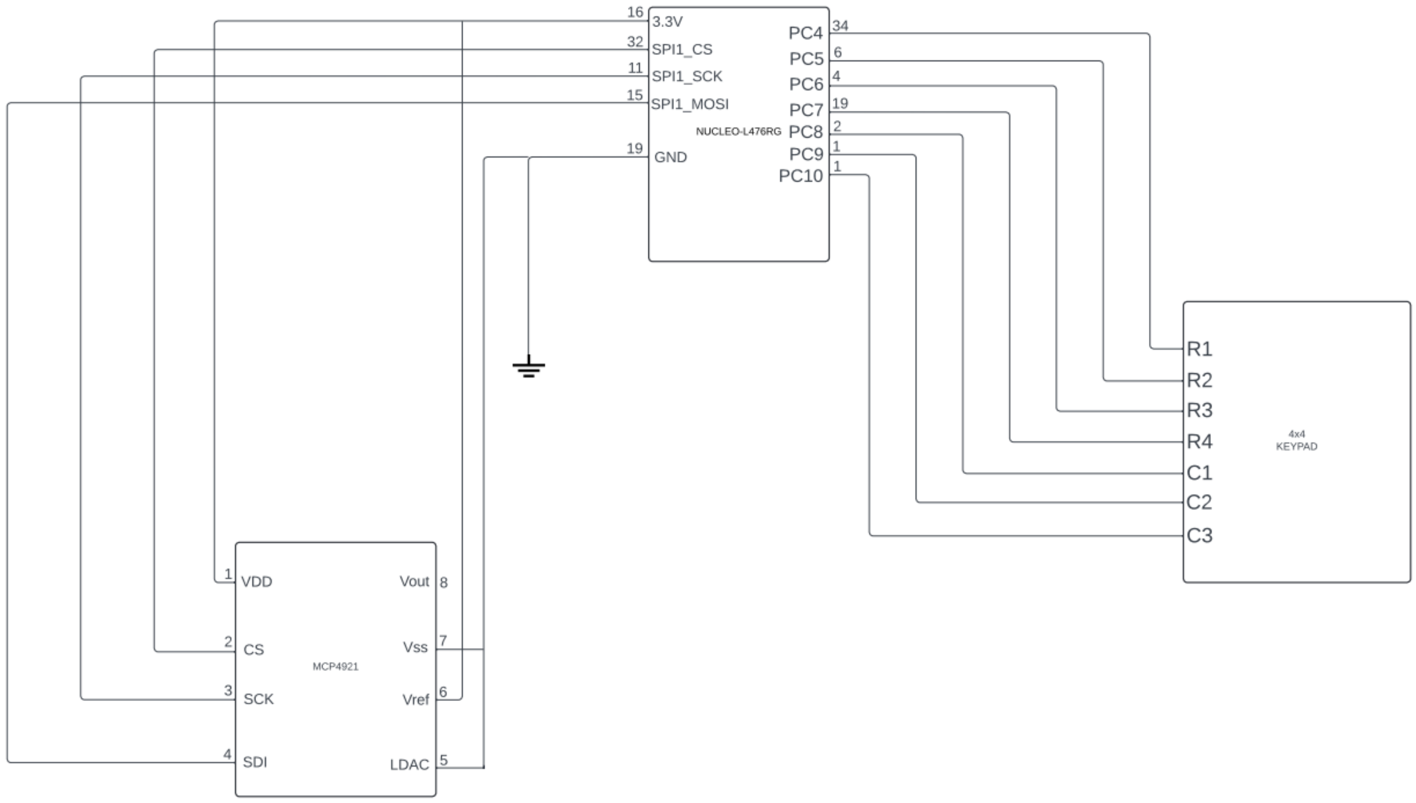


Figure 1

Software Architecture

$$\textbf{DAC\_Write()} = 6.68\ \mu\text{sec with overhead}$$

$$\textbf{Clock Cycle Time} = \frac{1}{40\text{MHz}} = 25\ \text{ns}$$

$$\textbf{Max resolution}\text{: } \frac{1}{6.68\mu\text{sec}} = 150{,}000\ \text{samples per second}$$

$$\textbf{Min resolution}\text{: } 150{,}000 \times 0.6 = 90{,}000\ \text{samples per second}$$

$$\textbf{ARR}\text{: } \frac{6.68\mu\text{sec per ISR}}{25\text{ns per clock cycle}} \approx 266\ \text{clock cycles per ISR}$$

$$\textbf{Array Size} = 1500$$

The maximum resolution is about 150,000 samples per second. This is with an MCU clock set at 40MHz, giving it a period of 25 nanoseconds. The ISR timing for a single DAC_Write() was 6.68 microseconds including the extra buffer. Given this, the calculated maximum resolution is 150,000 samples per second and the minimum resolution is 60% of that at 90,000 samples per second. The ARR was calculated to be 266 (the amount of clock cycles for one ISR). For a 100Hz waveform with a period of .01 milliseconds, 1500 samples are used for the array (1/100th of the calculated maximum samples per second).

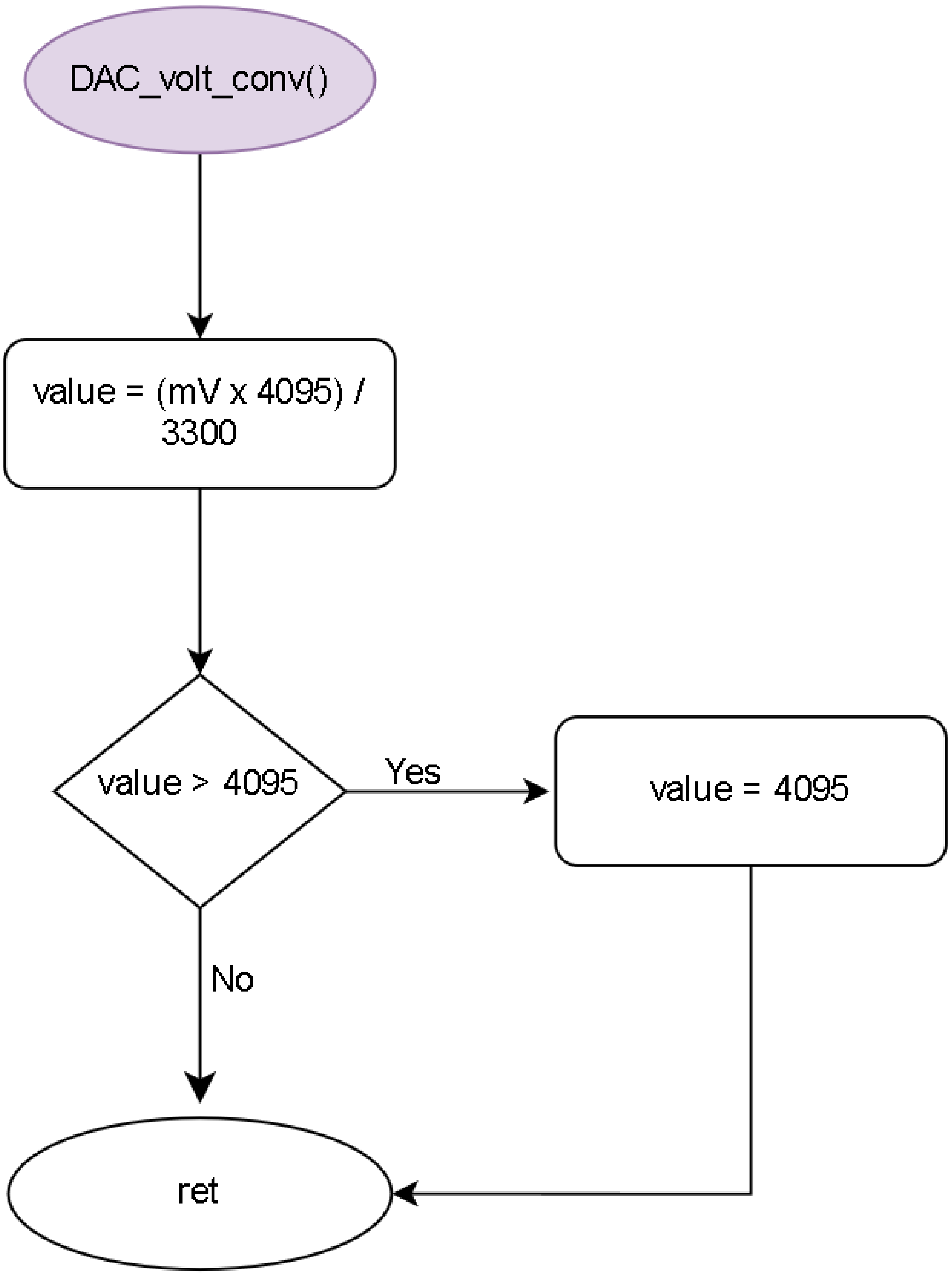

DAC_volt_conv()
value = (mV x 4095) / 3300
value > 4095
Yes
value = 4095
No
ret

Figure 2

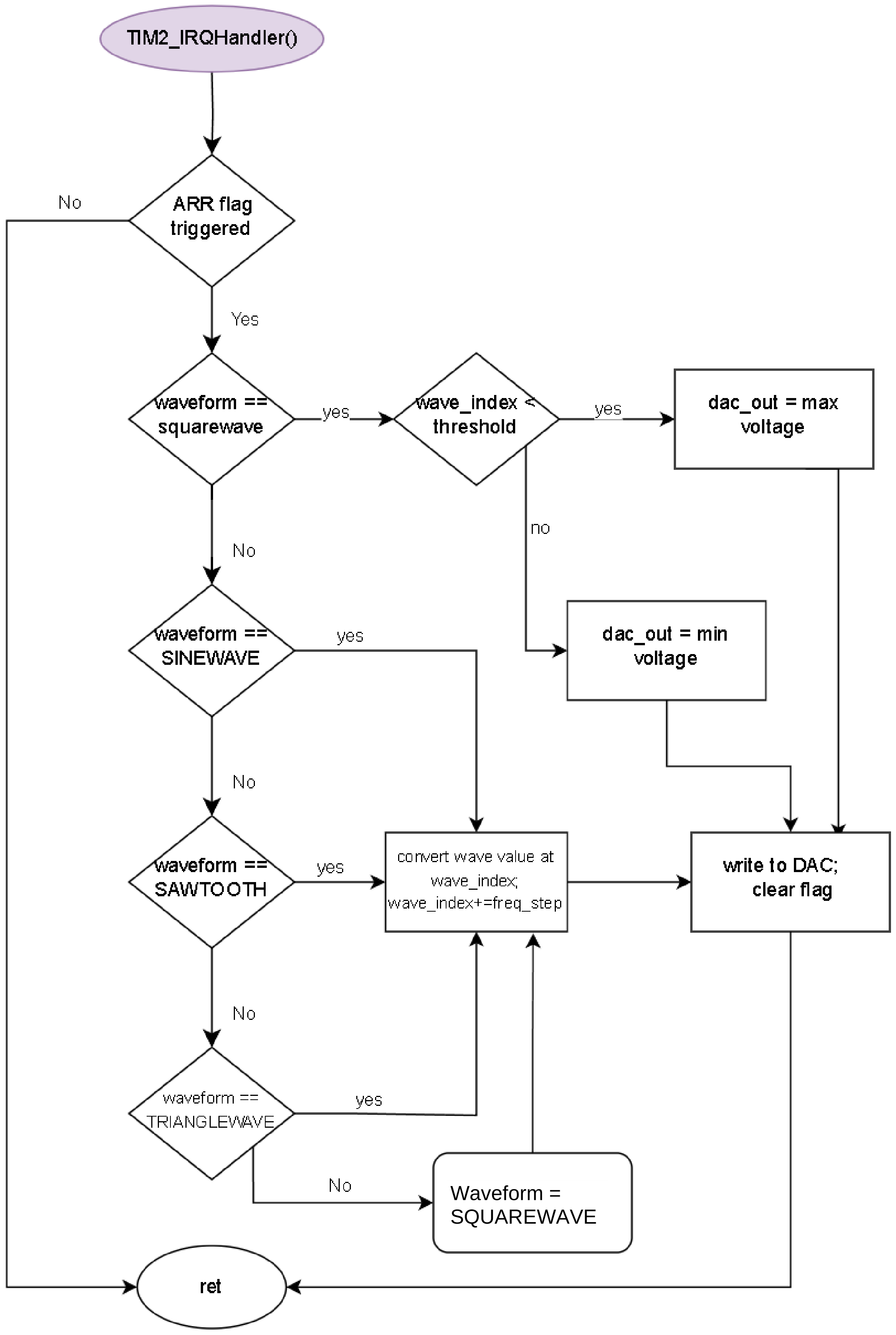

TIM2_IRQHandler()
ARR flag triggered
No
Yes
waveform == squarewave
yes
wave_index < threshold
yes
dac_out = max voltage
no
dac_out = min voltage
No
waveform == SINEWAVE
yes
No
waveform == SAWTOOTH
yes
convert wave value at wave_index; wave_index+=freq_step
write to DAC; clear flag
No
waveform == TRIANGLEWAVE
yes
No
Waveform = SQUAREWAVE
ret

Figure 3

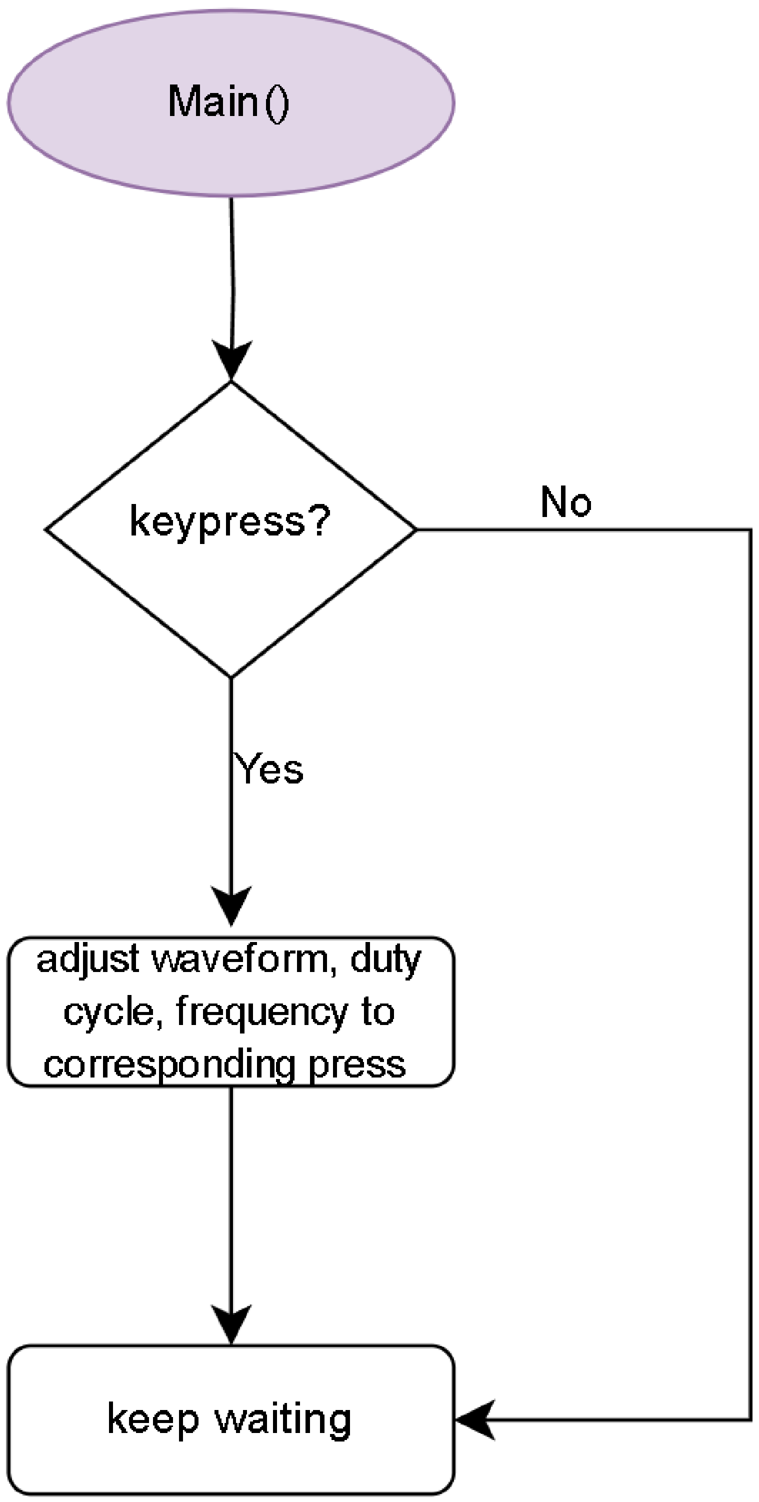

Main()
keypress?
No
Yes
adjust waveform, duty cycle, frequency to corresponding press
keep waiting

Figure 4

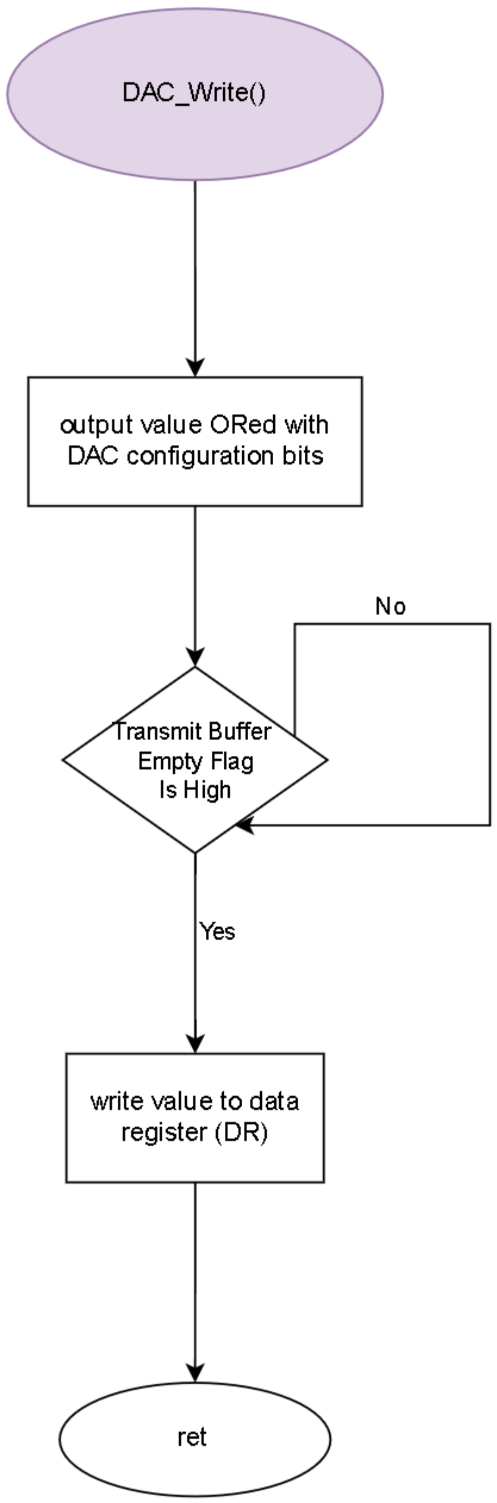

DAC_Write()
output value ORed with DAC configuration bits
No
Transmit Buffer Empty Flag Is High
Yes
write value to data register (DR)
ret

Figure 5

How It Works

The program flow and logic behind the approach that I took is straightforward. I utilize switch case statements and assign each keypad press to a different case. Keypad buttons 1, 2, 3, 4, and 5 will adjust the frequency of the waveform to 100Hz,200Hz,300Hz,400Hz, and 500Hz, respectively. Keypad buttons 6,7,8, and 9 adjust the waveform to a sine wave, triangle wave, sawtooth wave, and square wave, respectively. Keypad buttons *,0, and # adjust the square wave duty cycle to -10%, 50%, and +10%, respectively. The main while loop is continuous and once a keypress is made, the appropriate action will take place whether it's a frequency adjustment, waveform adjustment, or duty cycle adjustment. The frequency adjustment just alters how the corresponding waveform array is indexed. The waveform adjustment alters the waveform variable which is used to decide which waveform array to index, as there is a different array for each waveform. The duty cycle adjustment increments the duty cycle variable by +/- 10% which adjusts a threshold value. This threshold value is just the array size multiplied by the duty cycle percentage. Once the array index reaches this threshold value, the voltage outputted is the minimum voltage (0V), but if it is less than this threshold value, then the voltage outputted is the maximum voltage (3.0V). Every time the interrupt flag is triggered, the interrupt handler makes a check to see which waveform was selected and then converts the voltage value from the appropriate waveform array at the current index. The index is now incremented based on the selected frequency for the next interrupt trigger and the converted voltage is written to the DAC. Finally, the interrupt flag is disabled to indicated that the interrupt service routine has concluded.

main.c

```
-----------------------------------------------------------------------------------------------
#include "WaveformData.h"
#include "main.h"
#include "keypad.h"
#include "dac.h"

#define ARRAY_SIZE 1500 // my array size
#define INITIAL_DUTY_CYCLE 50 // initial duty cycle 50/100 = 50%
#define MIN_DUTY_CYCLE 10 // min duty cycle
#define MAX_DUTY_CYCLE 90 // max duty cycle
#define MAX_VOLTAGE 3722 // max digital value to output 3V
#define MIN_VOLTAGE 0 // min voltage 0V


#define SINEWAVE 6
#define TRIANGLEWAVE 7
#define SAWTOOTHWAVE 8
#define SQUAREWAVE 9


void timer_config(void);
void ISR_Handler(void);
void TIM2_IRQHandler(void);
void SystemClock_Config(void);


uint16_t waveform = SQUAREWAVE; // initial startup wave
uint16_t duty_cycle = INITIAL_DUTY_CYCLE; // start w/ 50% duty cycle
uint16_t wave_index = 0; // index in wave array
uint16_t freq_step = 1; // step increment within wave array


int main(void)
{
 // config
 HAL_Init();
 SystemClock_Config();
 DAC_Init();
 keypad_init();
 timer_config();
 __enable_irq();


 int16_t key_pressed = -1;
 /* Infinite loop reading keypad press */
 while (1){
       key_pressed = keypad_read();
       while(key_pressed == NO_PRESS){ //check for initial press
             key_pressed = keypad_read();
       }
       while(keypad_read() != NO_PRESS);

       if (key_pressed >= 0){
       switch(key_pressed){
       case 0:
             duty_cycle = 50; // initial duty cycle of square wave
             break;
      // incrementing step to alter frequency
       case 1:
             freq_step = 1;
             break;
       case 2:
             freq_step = 2;
```

```c
			break;
		case 3:
			freq_step = 3;
			break;
		case 4:
			freq_step = 4;
			break;
		case 5:
			freq_step = 5;
			break;
		// waveform based on keypress
		case 6:
			waveform = SINEWAVE;
			break;
		case 7:
			waveform = TRIANGLEWAVE;
			break;
		case 8:
			waveform = SAWTOOTHWAVE;
			break;
		case 9:
			waveform = SQUAREWAVE;
			break;
		// duty cycle increments
		case 10:
			// decrease duty cycle
			duty_cycle = duty_cycle - 10;
			if(duty_cycle <= 10){
				duty_cycle = 10;
			}
			break;
		case 11:
			// increase duty cycle
			duty_cycle = duty_cycle + 10;
			if(duty_cycle >= MAX_DUTY_CYCLE){
				duty_cycle = 90;
			}
			break;
		default:
			break;
		}

		}
  }

  }


void timer_config(void){


		RCC->APB1ENR1 |= RCC_APB1ENR1_TIM2EN; // configure TIM2
		TIM2->CR1 &= ~(TIM_CR1_CEN);		// disable counter
		TIM2->CR1 &= ~(TIM_CR1_DIR); // set upcounting
		TIM2->CR1 &= ~(TIM_CR1_CMS);		// set to edge-aligned mode
		TIM2->SR &= ~TIM_SR_UIF; // disable flag
		TIM2->DIER |= TIM_DIER_TIE | TIM_DIER_UIE; // enable interrupt trigger
		TIM2->ARR = 266; // ARR value
		NVIC->ISER[0]= (1 << (TIM2_IRQn & 0x1F)); //enable interrupt
		TIM2->CR1 |= TIM_CR1_CEN;		// enable counter

}

void TIM2_IRQHandler(void) {
      // if flag triggered
    if(TIM2->SR & TIM_SR_UIF){
        uint16_t dac_output; // stores value to write
        uint16_t threshold = (ARRAY_SIZE * duty_cycle) / 100;  //threshold is based on
current duty cycle
        if (waveform == SQUAREWAVE) {
            //0V or 3V based on the wave index & threshold
            if (wave_index < threshold) {
                dac_output = MAX_VOLTAGE;
            } else {
                dac_output = MIN_VOLTAGE;
            }

            // increment index and wrap back around
            wave_index = (wave_index + freq_step) % ARRAY_SIZE;
        } else {
            // other waveforms: sine, sawtooth, triangle
            switch (waveform) {
            // outputs converted mV value from wave array
                case SINEWAVE:
                    dac_output = DAC_volt_conv(sine_wave[wave_index]);
                    break;
                case SAWTOOTHWAVE:
                    dac_output = DAC_volt_conv(sawtooth_wave[wave_index]);
                    break;
                case TRIANGLEWAVE:
                    dac_output = DAC_volt_conv(triangle_wave[wave_index]);
                    break;
                default:
                  // never reaches this, just for debugging
                    dac_output = DAC_volt_conv(sine_wave[wave_index]);
                    break;
            }
            // increment index (based on freq) and wrap around
            wave_index = (wave_index + freq_step) % ARRAY_SIZE;
        }

        // write DAC value
        DAC_write(dac_output);

        // clear interrupt flag
        TIM2->SR &= ~(TIM_SR_UIF);
    }
}



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_ON;
  RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_MSI;
  RCC_OscInitStruct.PLL.PLLM = 1;
  RCC_OscInitStruct.PLL.PLLN = 20;
  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();
  }

  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_2) != HAL_OK)
  {
    Error_Handler();
  }
}

void Error_Handler(void)
{

 __disable_irq();
 while (1)
 {
 }
}

#ifdef  USE_FULL_ASSERT

void assert_failed(uint8_t *file, uint32_t line)
{

}
#endif
```

keypad.c

```c
#include "keypad.h"

void keypad_init(void)
{
	RCC->AHB2ENR |= RCC_AHB2ENR_GPIOCEN; // enable gpioc clock
	config_rows();
	config_col();
}
void config_rows(void)
{
	GPIOC->MODER &= ~(GPIO_MODER_MODE4_Msk);		// set mode to input
	GPIOC->PUPDR &= ~(GPIO_PUPDR_PUPD4_Msk);		// pull-down
	GPIOC->PUPDR |= (GPIO_PUPDR_PUPD4_1);
	GPIOC->MODER &= ~(GPIO_MODER_MODE5_Msk);		// set mode to input
	GPIOC->PUPDR &= ~(GPIO_PUPDR_PUPD5_Msk);		// pull-down
	GPIOC->PUPDR |= (GPIO_PUPDR_PUPD5_1);
	GPIOC->MODER &= ~(GPIO_MODER_MODE6_Msk);		// set mode to input
	GPIOC->PUPDR &= ~(GPIO_PUPDR_PUPD6_Msk);		// pull-down
	GPIOC->PUPDR |= (GPIO_PUPDR_PUPD6_1);
	GPIOC->MODER &= ~(GPIO_MODER_MODE7_Msk);		// set mode to input
	GPIOC->PUPDR &= ~(GPIO_PUPDR_PUPD7_Msk);		// pull-down
    GPIOC->PUPDR |= (GPIO_PUPDR_PUPD7_1);
}
void config_col(void)
{
	// columns
	GPIOC->MODER &= ~(GPIO_MODER_MODE8_Msk);		// set mode to output
	GPIOC->MODER |= (1 << GPIO_MODER_MODE8_Pos);
	GPIOC->OTYPER &= ~(GPIO_OTYPER_OT8_Msk);		// push-pull output
	GPIOC->OSPEEDR &= ~(GPIO_OSPEEDR_OSPEED8_Msk); // low speed
	GPIOC->PUPDR &= ~(GPIO_PUPDR_PUPD8_Msk);	// no resistor connected
	GPIOC->MODER &= ~(GPIO_MODER_MODE9_Msk);		// set mode to output
	GPIOC->MODER |= (1 << GPIO_MODER_MODE9_Pos);
	GPIOC->OTYPER &= ~(GPIO_OTYPER_OT9_Msk);		// push-pull output
	GPIOC->OSPEEDR &= ~(GPIO_OSPEEDR_OSPEED9_Msk); // low speed
	GPIOC->PUPDR &= ~(GPIO_PUPDR_PUPD9_Msk);	// no resistor connected
	GPIOC->MODER &= ~(GPIO_MODER_MODE10_Msk);		// set mode to output
	GPIOC->MODER |= (1 << GPIO_MODER_MODE10_Pos);
	GPIOC->OTYPER &= ~(GPIO_OTYPER_OT10_Msk);		// push-pull output
	GPIOC->OSPEEDR &= ~(GPIO_OSPEEDR_OSPEED10_Msk); // low speed
	GPIOC->PUPDR &= ~(GPIO_PUPDR_PUPD10_Msk);		// no resistor connected
}
uint8_t keypad_read(void)
{
	uint8_t row0_keys[NUM_COL] = {1, 2, 3};
	uint8_t row1_keys[NUM_COL] = {4, 5, 6};
	uint8_t row2_keys[NUM_COL] = {7, 8, 9};
	uint8_t row3_keys[NUM_COL] = {STAR, 0, POUND};
	// set all columns to high
	GPIOC->ODR |= GPIO_ODR_OD8_Msk;
	GPIOC->ODR |= (GPIO_ODR_OD9_Msk);
	GPIOC->ODR |= (GPIO_ODR_OD10_Msk);
	// read the rows
	uint8_t row0 = GPIOC->IDR & GPIO_IDR_ID4_Msk;
	uint8_t row1 = GPIOC->IDR & GPIO_IDR_ID5_Msk;
	uint8_t row2 = GPIOC->IDR & GPIO_IDR_ID6_Msk;
	uint8_t row3 = GPIOC->IDR & GPIO_IDR_ID7_Msk;
	// see if any rows were pressed
	if ((row0 == 0) && (row1 == 0) && (row2 == 0) && (row3 == 0)){
		return NO_PRESS;
	}
	for (int i = 0; i < NUM_COL; i++){
```

```
            // turn off all columns
            GPIOC->ODR &= ~(GPIO_ODR_OD8_Msk);
            GPIOC->ODR &= ~(GPIO_ODR_OD9_Msk);
            GPIOC->ODR &= ~(GPIO_ODR_OD10_Msk);
            // turn on column you want
            GPIOC->ODR |= (1 << (i+8));
            // small delay
            delay(10);
            // read the rows
            row0 = GPIOC->IDR & GPIO_IDR_ID4_Msk;
            row1 = GPIOC->IDR & GPIO_IDR_ID5_Msk;
            row2 = GPIOC->IDR & GPIO_IDR_ID6_Msk;
            row3 = GPIOC->IDR & GPIO_IDR_ID7_Msk;
            if(row0 != 0){
                  return row0_keys[i];
            }
            else if (row1 != 0){
                  return row1_keys[i];
            }
            else if (row2 != 0){
                  return row2_keys[i];
            }
            else if (row3 != 0){
                  return row3_keys[i];
            }
      }
      return NO_PRESS;
}
void delay(uint32_t milliseconds)
{
      for (int i = 0; i < milliseconds; i++);
}
```

dac.c

```
----------------------------------------------------------------------------------------------
#include "dac.h"

void DAC_Init(void)
{
      config_SPI_GPIO();
      config_SPI_CR();
}

void config_SPI_GPIO(void)
{
      RCC->AHB2ENR |= RCC_AHB2ENR_GPIOAEN; // enable GPIOA peripheral clock
      // Configure CS
      GPIOA->MODER &= ~(GPIO_MODER_MODE4_Msk);        // alternate function mode
      GPIOA->MODER |= GPIO_MODER_MODE4_1;
      GPIOA->AFR[0] |= (0x5 << GPIO_AFRL_AFSEL4_Pos);
      // Configure SCK
      GPIOA->MODER &= ~(GPIO_MODER_MODE5_Msk);        // alternate function mode
      GPIOA->MODER |= GPIO_MODER_MODE5_1;
      GPIOA->AFR[0] |= (0x5 << GPIO_AFRL_AFSEL5_Pos);
      // Configure PICO
      GPIOA->MODER &= ~(GPIO_MODER_MODE7_Msk);        // alternate function mode
      GPIOA->MODER |= GPIO_MODER_MODE7_1;
      GPIOA->AFR[0] |= (0x5 << GPIO_AFRL_AFSEL7_Pos);
}

void config_SPI_CR(void)
{
      RCC->APB2ENR |= RCC_APB2ENR_SPI1EN; // enable SPI peripheral clock
    SPI1->CR1 &= ~( SPI_CR1_RXONLY );            // recv-only OFF
    SPI1->CR1 &= ~( SPI_CR1_LSBFIRST );          // data bit order MSb:LSb
    SPI1->CR1 &= ~( SPI_CR1_CPOL | SPI_CR1_CPHA ); // SCLK polarity:phase = 0:0
      SPI1->CR1 |= SPI_CR1_MSTR;                 // Set as master mode
      SPI1->CR1 &= ~(SPI_CR1_BR);
      SPI1->CR1 &= ~(SPI_CR1_SSM);
      SPI1->CR2 &= ~( SPI_CR2_TXEIE | SPI_CR2_RXNEIE ); // disable FIFO intrpts
    SPI1->CR2 &= ~(SPI_CR2_FRF);                    // Moto frame format
    SPI1->CR2 |=(SPI_CR2_NSSP);                  // auto-generate NSS pulse
    SPI1->CR2 |=(SPI_CR2_DS);                    // 16-bit data
      SPI1->CR2 |= SPI_CR2_SSOE;
      SPI1->CR1 |= SPI_CR1_SPE;                  // Enable SPI1
}

void DAC_write(uint16_t value)
{
  value &= 0xFFF;  // make sure lower 12 bits used for data
  uint16_t message = (0x3<<12) | value; // store value and configure DAC register (0011)
  // transmit data over SPI
  while(!(SPI1->SR & SPI_SR_TXE));// wait for transmission to complete
  SPI1->DR = message;        // send the 16-bit data word to DAC
}

uint16_t DAC_volt_conv(uint16_t mV)
{
 // scale the input value to fit within 12 bits
      uint16_t value = (mV * 4095) / 3300;
      if(value > 4095)
      {
            value = 4095;
      }
      return value;
}
```

WaveformData.h

```
// Array for sine wave values

const int sine_wave[1500] = {

      1500,1506,1512,1518,1525,1531,1537,1543,1550,1556,1562,1569,1575,1581,1587,1594,1600,16
06,1612,1619,1625,1631,1638,1644,1650,1656,1663,1669,1675,1681,1687,1694,1700,1706,1712,1719,
1725,1731,1737,1743,1750,1756,1762,1768,1774,1781,1787,1793,1799,1805,1811,1818,1824,1830,183
6,1842,1848,1854,1860,1866,1873,1879,1885,1891,1897,1903,1909,1915,1921,1927,1933,1939,1945,1
951,1957,1963,1969,1975,1981,1987,1993,1999,2005,2011,2016,2022,2028,2034,2040,2046,2052,2058
,2063,2069,2075,2081,2087,2092,2098,2104,2110,2115,2121,2127,2132,2138,2144,2150,2155,2161,21
66,2172,2178,2183,2189,2194,2200,2206,2211,2217,2222,2228,2233,2239,2244,2250,2255,2260,2266,
2271,2277,2282,2287,2293,2298,2303,2309,2314,2319,2324,2330,2335,2340,2345,2350,2356,2361,236
6,2371,2376,2381,2386,2391,2396,2401,2406,2411,2416,2421,2426,2431,2436,2441,2446,2451,2456,2
460,2465,2470,2475,2480,2484,2489,2494,2499,2503,2508,2512,2517,2522,2526,2531,2535,2540,2544
,2549,2553,2558,2562,2567,2571,2576,2580,2584,2589,2593,2597,2602,2606,2610,2614,2618,2623,26
27,2631,2635,2639,2643,2647,2651,2655,2659,2663,2667,2671,2675,2679,2683,2687,2690,2694,2698,
2702,2706,2709,2713,2717,2720,2724,2728,2731,2735,2738,2742,2745,2749,2752,2756,2759,2763,276
6,2769,2773,2776,2779,2783,2786,2789,2792,2795,2799,2802,2805,2808,2811,2814,2817,2820,2823,2
826,2829,2832,2835,2837,2840,2843,2846,2849,2851,2854,2857,2859,2862,2865,2867,2870,2872,2875
,2877,2880,2882,2885,2887,2889,2892,2894,2896,2899,2901,2903,2905,2908,2910,2912,2914,2916,29
18,2920,2922,2924,2926,2928,2930,2932,2934,2935,2937,2939,2941,2943,2944,2946,2948,2949,2951,
2952,2954,2955,2957,2958,2960,2961,2963,2964,2965,2967,2968,2969,2971,2972,2973,2974,2975,297
6,2977,2978,2980,2981,2982,2982,2983,2984,2985,2986,2987,2988,2988,2989,2990,2991,2991,2992,2
993,2993,2994,2994,2995,2995,2996,2996,2997,2997,2997,2998,2998,2998,2998,2999,2999,2999,2999
,2999,2999,2999,2999,3000,2999,2999,2999,2999,2999,2999,2999,2999,2998,2998,2998,2998,2997,29
97,2997,2996,2996,2995,2995,2994,2994,2993,2993,2992,2991,2991,2990,2989,2988,2988,2987,2986,
2985,2984,2983,2982,2982,2981,2980,2978,2977,2976,2975,2974,2973,2972,2971,2969,2968,2967,296
5,2964,2963,2961,2960,2958,2957,2955,2954,2952,2951,2949,2948,2946,2944,2943,2941,2939,2937,2
935,2934,2932,2930,2928,2926,2924,2922,2920,2918,2916,2914,2912,2910,2908,2905,2903,2901,2899
,2896,2894,2892,2889,2887,2885,2882,2880,2877,2875,2872,2870,2867,2865,2862,2859,2857,2854,28
51,2849,2846,2843,2840,2837,2835,2832,2829,2826,2823,2820,2817,2814,2811,2808,2805,2802,2799,
2795,2792,2789,2786,2783,2779,2776,2773,2769,2766,2763,2759,2756,2752,2749,2745,2742,2738,273
5,2731,2728,2724,2720,2717,2713,2709,2706,2702,2698,2694,2690,2687,2683,2679,2675,2671,2667,2
663,2659,2655,2651,2647,2643,2639,2635,2631,2627,2623,2618,2614,2610,2606,2602,2597,2593,2589
,2584,2580,2576,2571,2567,2562,2558,2553,2549,2544,2540,2535,2531,2526,2522,2517,2512,2508,25
03,2499,2494,2489,2484,2480,2475,2470,2465,2460,2456,2451,2446,2441,2436,2431,2426,2421,2416,
2411,2406,2401,2396,2391,2386,2381,2376,2371,2366,2361,2356,2350,2345,2340,2335,2330,2324,231
9,2314,2309,2303,2298,2293,2287,2282,2277,2271,2266,2260,2255,2250,2244,2239,2233,2228,2222,2
217,2211,2206,2200,2194,2189,2183,2178,2172,2166,2161,2155,2150,2144,2138,2132,2127,2121,2115
,2110,2104,2098,2092,2087,2081,2075,2069,2063,2058,2052,2046,2040,2034,2028,2022,2016,2011,20
05,1999,1993,1987,1981,1975,1969,1963,1957,1951,1945,1939,1933,1927,1921,1915,1909,1903,1897,
1891,1885,1879,1873,1866,1860,1854,1848,1842,1836,1830,1824,1818,1811,1805,1799,1793,1787,178
1,1774,1768,1762,1756,1750,1743,1737,1731,1725,1719,1712,1706,1700,1694,1687,1681,1675,1669,1
663,1656,1650,1644,1638,1631,1625,1619,1612,1606,1600,1594,1587,1581,1575,1569,1562,1556,1550
,1543,1537,1531,1525,1518,1512,1506,1500,1493,1487,1481,1474,1468,1462,1456,1449,1443,1437,14
30,1424,1418,1412,1405,1399,1393,1387,1380,1374,1368,1361,1355,1349,1343,1336,1330,1324,1318,
1312,1305,1299,1293,1287,1280,1274,1268,1262,1256,1249,1243,1237,1231,1225,1218,1212,1206,120
0,1194,1188,1181,1175,1169,1163,1157,1151,1145,1139,1133,1126,1120,1114,1108,1102,1096,1090,1
084,1078,1072,1066,1060,1054,1048,1042,1036,1030,1024,1018,1012,1006,1000,994,988,983,977,971
,965,959,953,947,941,936,930,924,918,912,907,901,895,889,884,878,872,867,861,855,849,844,838,
833,827,821,816,810,805,799,793,788,782,777,771,766,760,755,750,744,739,733,728,722,717,712,7
06,701,696,690,685,680,675,669,664,659,654,649,643,638,633,628,623,618,613,608,603,598,593,58
8,583,578,573,568,563,558,553,548,543,539,534,529,524,519,515,510,505,500,496,491,487,482,477
,473,468,464,459,455,450,446,441,437,432,428,423,419,415,410,406,402,397,393,389,385,381,376,
372,368,364,360,356,352,348,344,340,336,332,328,324,320,316,312,309,305,301,297,293,290,286,2
82,279,275,271,268,264,261,257,254,250,247,243,240,236,233,230,226,223,220,216,213,210,207,20
4,200,197,194,191,188,185,182,179,176,173,170,167,164,162,159,156,153,150,148,145,142,140,137
,134,132,129,127,124,122,119,117,114,112,110,107,105,103,100,98,96,94,91,89,87,85,83,81,79,77
,75,73,71,69,67,65,64,62,60,58,56,55,53,51,50,48,47,45,44,42,41,39,38,36,35,34,32,31,30,28,27
,26,25,24,23,22,21,19,18,17,17,16,15,14,13,12,11,11,10,9,8,8,7,6,6,5,5,4,4,3,3,2,2,2,1,1,1,1,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,2,2,2,3,3,4,4,5,5,6,6,7,8,8,9,10,11,11,12,13,14,15,
16,17,17,18,19,21,22,23,24,25,26,27,28,30,31,32,34,35,36,38,39,41,42,44,45,47,48,50,51,53,55,
56,58,60,62,64,65,67,69,71,73,75,77,79,81,83,85,87,89,91,94,96,98,100,103,105,107,110,112,114
,117,119,122,124,127,129,132,134,137,140,142,145,148,150,153,156,159,162,164,167,170,173,176,
179,182,185,188,191,194,197,200,204,207,210,213,216,220,223,226,230,233,236,240,243,247,250,2
54,257,261,264,268,271,275,279,282,286,290,293,297,301,305,309,312,316,320,324,328,332,336,34
0,344,348,352,356,360,364,368,372,376,381,385,389,393,397,402,406,410,415,419,423,428,432,437
,441,446,450,455,459,464,468,473,477,482,487,491,496,500,505,510,515,519,524,529,534,539,543,
548,553,558,563,568,573,578,583,588,593,598,603,608,613,618,623,628,633,638,643,649,654,659,6
64,669,675,680,685,690,696,701,706,712,717,722,728,733,739,744,749,755,760,766,771,777,782,78
8,793,799,805,810,816,821,827,833,838,844,849,855,861,867,872,878,884,889,895,901,907,912,918
,924,930,936,941,947,953,959,965,971,977,983,988,994,1000,1006,1012,1018,1024,1030,1036,1042,
1048,1054,1060,1066,1072,1078,1084,1090,1096,1102,1108,1114,1120,1126,1133,1139,1145,1151,115
7,1163,1169,1175,1181,1188,1194,1200,1206,1212,1218,1225,1231,1237,1243,1249,1256,1262,1268,1
274,1280,1287,1293,1299,1305,1312,1318,1324,1330,1336,1343,1349,1355,1361,1368,1374,1380,1387
,1393,1399,1405,1412,1418,1424,1430,1437,1443,1449,1456,1462,1468,1474,1481,1487,1493
};

const int sawtooth_wave[1500] = {

	0,1,4,6,7,10,12,13,16,18,19,22,24,25,27,30,31,33,36,37,39,42,43,45,48,49,51,54,55,57,60
,62,63,66,68,69,72,74,75,78,80,81,84,86,87,90,92,93,96,98,99,101,104,105,107,110,111,113,116,
118,119,122,124,125,128,130,131,134,136,137,139,142,143,145,148,149,151,154,155,157,160,161,1
63,166,167,169,172,174,175,178,180,181,184,186,187,190,192,193,196,198,199,202,204,205,208,21
0,211,214,216,217,220,222,223,226,228,229,232,234,236,237,240,242,243,246,248,249,251,254,255
,257,260,261,263,266,267,269,272,273,275,278,279,281,284,286,287,290,292,293,296,298,299,302,
304,305,308,310,311,314,316,317,320,322,323,326,328,329,332,334,335,338,340,341,344,346,348,3
50,352,354,355,358,360,361,364,366,367,369,372,373,375,378,379,381,384,385,387,390,391,393,39
6,397,399,402,403,405,408,410,411,414,416,417,420,422,423,426,428,429,432,434,435,438,440,441
,444,446,447,450,452,453,456,458,459,462,464,465,468,469,472,474,475,478,480,481,484,486,487,
490,492,493,496,497,499,502,503,505,508,509,511,514,516,517,520,522,523,526,528,529,532,534,5
35,538,540,541,544,546,547,550,552,553,556,558,559,562,564,566,568,570,572,574,576,578,580,58
2,584,586,587,590,592,593,596,598,599,602,604,605,608,610,611,614,616,617,620,621,623,626,628
,629,632,634,635,638,640,641,644,646,647,650,652,653,656,658,659,662,664,665,668,670,671,674,
676,677,680,682,683,686,688,690,692,694,696,698,700,702,704,706,708,710,711,714,716,717,720,7
22,723,726,728,729,732,734,735,738,739,741,744,745,747,750,752,753,756,758,759,762,764,765,76
8,770,771,774,776,777,780,782,783,786,788,789,792,794,795,798,800,801,804,806,807,810,812,814
,815,818,820,821,824,826,827,830,832,833,836,838,839,842,844,845,848,850,851,854,856,857,860,
862,863,866,868,869,872,874,876,877,880,882,883,886,888,889,892,894,895,898,900,901,904,906,9
07,910,912,913,916,918,919,922,924,925,928,930,931,933,936,938,939,942,944,945,948,950,951,95
4,956,957,960,962,963,966,968,969,972,974,975,978,980,981,984,986,987,990,992,993,995,998,100
0,1001,1004,1006,1007,1010,1012,1013,1016,1018,1019,1022,1024,1026,1028,1030,1032,1034,1036,1
038,1040,1042,1044,1046,1048,1050,1052,1054,1056,1057,1060,1062,1064,1066,1068,1070,1072,1074
,1076,1078,1080,1082,1084,1086,1088,1090,1092,1094,1096,1098,1100,1102,1104,1106,1108,1110,11
12,1114,1116,1118,1119,1122,1124,1126,1128,1130,1132,1134,1136,1138,1140,1142,1144,1146,1148,
1150,1152,1154,1156,1158,1160,1162,1164,1166,1168,1170,1172,1174,1175,1178,1180,1181,1184,118
6,1188,1190,1192,1194,1196,1198,1200,1202,1204,1206,1208,1210,1212,1214,1216,1218,1220,1222,1
224,1226,1228,1230,1232,1234,1236,1237,1240,1242,1243,1246,1248,1250,1252,1254,1256,1258,1260
,1262,1264,1266,1268,1270,1272,1274,1276,1278,1280,1282,1284,1286,1288,1290,1292,1294,1296,12
98,1299,1302,1304,1305,1308,1310,1312,1314,1316,1318,1320,1322,1324,1326,1328,1330,1332,1334,
1336,1338,1340,1342,1344,1346,1348,1350,1352,1354,1355,1358,1360,1361,1364,1366,1367,1370,137
2,1374,1376,1378,1380,1382,1384,1386,1388,1390,1392,1394,1396,1398,1400,1402,1404,1406,1408,1
410,1412,1414,1416,1417,1420,1422,1423,1426,1428,1429,1432,1434,1436,1438,1440,1442,1444,1446
,1448,1450,1452,1454,1456,1458,1460,1462,1464,1466,1468,1470,1472,1474,1476,1478,1479,1482,14
84,1485,1488,1490,1491,1494,1496,1498,1500,1502,1504,1506,1507,1510,1512,1513,1516,1518,1519,
1522,1524,1525,1528,1530,1531,1534,1536,1537,1540,1542,1543,1546,1548,1549,1552,1554,1555,155
8,1560,1561,1564,1566,1567,1570,1572,1573,1576,1578,1579,1582,1584,1585,1588,1590,1591,1594,1
596,1597,1600,1602,1603,1606,1608,1609,1612,1614,1615,1618,1620,1622,1624,1626,1628,1630,1631
,1634,1636,1637,1640,1642,1643,1646,1648,1649,1652,1654,1655,1658,1660,1661,1664,1666,1667,16
70,1672,1673,1676,1678,1679,1682,1684,1685,1688,1690,1691,1694,1696,1697,1700,1702,1703,1706,
1708,1709,1712,1714,1715,1718,1720,1721,1724,1726,1727,1730,1732,1733,1736,1738,1739,1742,174
4,1745,1748,1749,1752,1754,1755,1758,1760,1761,1764,1766,1767,1770,1772,1773,1776,1778,1779,1
782,1784,1785,1788,1790,1791,1794,1796,1797,1800,1802,1803,1806,1808,1809,1812,1814,1815,1818
,1820,1821,1824,1826,1827,1830,1832,1833,1836,1838,1839,1842,1844,1845,1848,1850,1851,1854,18
56,1857,1860,1862,1863,1866,1867,1869,1872,1873,1876,1878,1879,1882,1884,1885,1888,1890,1891,
1894,1896,1897,1900,1902,1903,1906,1908,1909,1912,1914,1915,1918,1920,1921,1924,1926,1927,193
```

```
0,1932,1933,1936,1938,1939,1942,1944,1945,1948,1950,1951,1954,1956,1957,1960,1962,1963,1966,1
968,1969,1972,1974,1975,1978,1980,1981,1984,1986,1987,1990,1991,1993,1996,1997,2000,2002,2003
,2006,2008,2009,2012,2014,2015,2018,2020,2021,2024,2026,2027,2030,2032,2033,2036,2038,2039,20
42,2044,2045,2048,2050,2052,2054,2056,2058,2060,2062,2064,2066,2068,2070,2072,2074,2076,2078,
2080,2082,2084,2086,2088,2090,2092,2094,2096,2098,2100,2102,2104,2106,2108,2109,2112,2114,211
5,2118,2120,2122,2124,2126,2128,2130,2132,2134,2136,2138,2140,2142,2144,2146,2148,2150,2152,2
154,2156,2158,2160,2162,2164,2166,2168,2170,2172,2174,2176,2178,2180,2182,2184,2186,2188,2190
,2192,2194,2196,2198,2200,2202,2204,2206,2208,2210,2212,2214,2216,2218,2220,2222,2224,2226,22
28,2230,2232,2233,2236,2238,2239,2242,2244,2246,2248,2250,2252,2254,2256,2258,2260,2262,2264,
2266,2268,2270,2272,2274,2276,2278,2280,2282,2284,2286,2288,2290,2292,2294,2296,2298,2300,230
2,2304,2306,2308,2310,2312,2314,2316,2318,2320,2322,2324,2326,2328,2330,2332,2334,2336,2338,2
340,2342,2344,2346,2348,2350,2351,2354,2356,2357,2360,2362,2363,2366,2368,2370,2372,2374,2376
,2378,2380,2382,2384,2386,2388,2390,2392,2394,2396,2398,2400,2402,2404,2406,2408,2410,2412,24
14,2416,2418,2420,2422,2424,2426,2428,2430,2432,2434,2436,2438,2440,2442,2444,2446,2448,2450,
2452,2454,2456,2458,2460,2462,2464,2466,2468,2469,2472,2474,2475,2478,2480,2481,2484,2486,248
7,2490,2492,2494,2496,2498,2500,2502,2504,2506,2508,2510,2512,2514,2516,2518,2520,2522,2524,2
526,2528,2530,2532,2534,2536,2538,2540,2542,2544,2546,2548,2550,2552,2554,2556,2558,2560,2562
,2564,2566,2568,2570,2572,2574,2576,2578,2580,2582,2584,2586,2588,2590,2592,2593,2596,2598,25
99,2602,2604,2605,2608,2610,2611,2614,2616,2617,2620,2622,2624,2626,2628,2630,2632,2634,2636,
2638,2640,2642,2644,2646,2648,2650,2652,2654,2656,2658,2660,2662,2664,2666,2668,2670,2672,267
4,2676,2678,2680,2682,2684,2686,2688,2690,2692,2694,2696,2698,2700,2702,2704,2706,2708,2710,2
711,2714,2716,2717,2720,2722,2723,2726,2728,2729,2732,2734,2735,2738,2740,2741,2744,2746,2748
,2750,2752,2754,2756,2758,2760,2762,2764,2766,2768,2770,2772,2774,2776,2778,2780,2782,2784,27
86,2788,2790,2792,2794,2796,2798,2800,2802,2804,2806,2808,2810,2812,2814,2816,2818,2820,2822,
2824,2826,2828,2830,2832,2834,2835,2838,2840,2841,2844,2846,2847,2850,2852,2853,2856,2858,285
9,2862,2864,2865,2868,2870,2872,2874,2876,2878,2880,2882,2884,2886,2888,2890,2892,2894,2896,2
898,2900,2902,2904,2906,2908,2910,2912,2914,2916,2918,2920,2922,2924,2926,2928,2930,2932,2934
,2936,2938,2940,2942,2944,2946,2948,2950,2952,2953,2956,2958,2959,2962,2964,2965,2968,2970,29
71,2974,2976,2977,2980,2982,2983,2986,2988,2989,2992,2994,2996,2998
};

const int triangle_wave[1500] = {

      1500,1504,1508,1512,1516,1520,1524,1528,1532,1536,1540,1544,1548,1552,1556,1560,1564,15
68,1572,1576,1580,1584,1588,1592,1596,1600,1604,1608,1612,1616,1620,1624,1628,1632,1636,1640,
1644,1648,1652,1656,1660,1664,1668,1672,1676,1680,1684,1688,1692,1696,1700,1704,1708,1712,171
6,1720,1724,1728,1732,1736,1740,1744,1748,1752,1756,1760,1764,1768,1772,1776,1780,1784,1788,1
792,1796,1800,1804,1808,1812,1816,1820,1824,1828,1832,1836,1840,1844,1848,1852,1856,1860,1864
,1868,1872,1876,1880,1884,1888,1892,1896,1900,1904,1908,1912,1916,1920,1924,1928,1932,1936,19
40,1944,1948,1952,1956,1960,1964,1968,1972,1976,1980,1984,1988,1992,1996,2000,2004,2008,2012,
2016,2020,2024,2028,2032,2036,2040,2044,2048,2052,2056,2060,2064,2068,2072,2076,2080,2084,208
8,2092,2096,2100,2104,2108,2112,2116,2120,2124,2128,2132,2136,2140,2144,2148,2152,2156,2160,2
164,2168,2172,2176,2180,2184,2188,2192,2196,2200,2204,2208,2212,2216,2220,2224,2228,2232,2236
,2240,2244,2248,2252,2256,2260,2264,2268,2272,2276,2280,2284,2288,2292,2296,2300,2304,2308,23
12,2316,2320,2324,2328,2332,2336,2340,2344,2348,2352,2356,2360,2364,2368,2372,2376,2380,2384,
2388,2392,2396,2400,2404,2408,2412,2416,2420,2424,2428,2432,2436,2440,2444,2448,2452,2456,246
0,2464,2468,2472,2476,2480,2484,2488,2492,2496,2500,2504,2508,2512,2516,2520,2524,2528,2532,2
536,2540,2544,2548,2552,2556,2560,2564,2568,2572,2576,2580,2584,2588,2592,2596,2600,2604,2608
,2612,2616,2620,2624,2628,2632,2636,2640,2644,2648,2652,2656,2660,2664,2668,2672,2676,2680,26
84,2688,2692,2696,2700,2704,2708,2712,2716,2720,2724,2728,2732,2736,2740,2744,2748,2752,2756,
2760,2764,2768,2772,2776,2780,2784,2788,2792,2796,2800,2804,2808,2812,2816,2820,2824,2828,283
2,2836,2840,2844,2848,2852,2856,2860,2864,2868,2872,2876,2880,2884,2888,2892,2896,2900,2904,2
908,2912,2916,2920,2924,2928,2932,2936,2940,2944,2948,2952,2956,2960,2964,2968,2972,2976,2980
,2984,2988,2992,2996,3000,2996,2992,2988,2984,2980,2976,2972,2968,2964,2960,2956,2952,2948,29
44,2940,2936,2932,2928,2924,2920,2916,2912,2908,2904,2900,2896,2892,2888,2884,2880,2876,2872,
2868,2864,2860,2856,2852,2848,2844,2840,2836,2832,2828,2824,2820,2816,2812,2808,2804,2800,279
6,2792,2788,2784,2780,2776,2772,2768,2764,2760,2756,2752,2748,2744,2740,2736,2732,2728,2724,2
720,2716,2712,2708,2704,2700,2696,2692,2688,2684,2680,2676,2672,2668,2664,2660,2656,2652,2648
,2644,2640,2636,2632,2628,2624,2620,2616,2612,2608,2604,2600,2596,2592,2588,2584,2580,2576,25
72,2568,2564,2560,2556,2552,2548,2544,2540,2536,2532,2528,2524,2520,2516,2512,2508,2504,2500,
2496,2492,2488,2484,2480,2476,2472,2468,2464,2460,2456,2452,2448,2444,2440,2436,2432,2428,242
4,2420,2416,2412,2408,2404,2400,2396,2392,2388,2384,2380,2376,2372,2368,2364,2360,2356,2352,2
348,2344,2340,2336,2332,2328,2324,2320,2316,2312,2308,2304,2300,2296,2292,2288,2284,2280,2276
,2272,2268,2264,2260,2256,2252,2248,2244,2240,2236,2232,2228,2224,2220,2216,2212,2208,2204,22
```

```
00,2196,2192,2188,2184,2180,2176,2172,2168,2164,2160,2156,2152,2148,2144,2140,2136,2132,2128,
2124,2120,2116,2112,2108,2104,2100,2096,2092,2088,2084,2080,2076,2072,2068,2064,2060,2056,205
2,2048,2044,2040,2036,2032,2028,2024,2020,2016,2012,2008,2004,2000,1996,1992,1988,1984,1980,1
976,1972,1968,1964,1960,1956,1952,1948,1944,1940,1936,1932,1928,1924,1920,1916,1912,1908,1904
,1900,1896,1892,1888,1884,1880,1876,1872,1868,1864,1860,1856,1852,1848,1844,1840,1836,1832,18
28,1824,1820,1816,1812,1808,1804,1800,1796,1792,1788,1784,1780,1776,1772,1768,1764,1760,1756,
1752,1748,1744,1740,1736,1732,1728,1724,1720,1716,1712,1708,1704,1700,1696,1692,1688,1684,168
0,1676,1672,1668,1664,1660,1656,1652,1648,1644,1640,1636,1632,1628,1624,1620,1616,1612,1608,1
604,1600,1596,1592,1588,1584,1580,1576,1572,1568,1564,1560,1556,1552,1548,1544,1540,1536,1532
,1528,1524,1520,1516,1512,1508,1504,1500,1496,1492,1488,1484,1480,1476,1472,1468,1464,1460,14
56,1452,1448,1444,1440,1436,1432,1428,1424,1420,1416,1412,1408,1404,1400,1396,1392,1388,1384,
1380,1376,1372,1368,1364,1360,1356,1352,1348,1344,1340,1336,1332,1328,1324,1320,1316,1312,130
8,1304,1300,1296,1292,1288,1284,1280,1276,1272,1268,1264,1260,1256,1252,1248,1244,1240,1236,1
232,1228,1224,1220,1216,1212,1208,1204,1200,1196,1192,1188,1184,1180,1176,1172,1168,1164,1160
,1156,1152,1148,1144,1140,1136,1132,1128,1124,1120,1116,1112,1108,1104,1100,1096,1092,1088,10
84,1080,1076,1072,1068,1064,1060,1056,1052,1048,1044,1040,1036,1032,1028,1024,1020,1016,1012,
1008,1004,1000,996,992,988,984,980,976,972,968,964,960,956,952,948,944,940,936,932,928,924,92
0,916,912,908,904,900,896,892,888,884,880,876,872,868,864,860,856,852,848,844,840,836,832,828
,824,820,816,812,808,804,800,796,792,788,784,780,776,772,768,764,760,756,752,748,744,740,736,
732,728,724,720,716,712,708,704,700,696,692,688,684,680,676,672,668,664,660,656,652,648,644,6
40,636,632,628,624,620,616,612,608,604,600,596,592,588,584,580,576,572,568,564,560,556,552,54
8,544,540,536,532,528,524,520,516,512,508,504,500,496,492,488,484,480,476,472,468,464,460,456
,452,448,444,440,436,432,428,424,420,416,412,408,404,400,396,392,388,384,380,376,372,368,364,
360,356,352,348,344,340,336,332,328,324,320,316,312,308,304,300,296,292,288,284,280,276,272,2
68,264,260,256,252,248,244,240,236,232,228,224,220,216,212,208,204,200,196,192,188,184,180,17
6,172,168,164,160,156,152,148,144,140,136,132,128,124,120,116,112,108,104,100,96,92,88,84,80,
76,72,68,64,60,56,52,48,44,40,36,32,28,24,20,16,12,8,4,0,4,8,12,16,20,24,28,32,36,40,44,48,52
,56,60,64,68,72,76,80,84,88,92,96,100,104,108,112,116,120,124,128,132,136,140,144,148,152,156
,160,164,168,172,176,180,184,188,192,196,200,204,208,212,216,220,224,228,232,236,240,244,248,
252,256,260,264,268,272,276,280,284,288,292,296,300,304,308,312,316,320,324,328,332,336,340,3
44,348,352,356,360,364,368,372,377,381,385,389,393,397,401,405,409,413,417,421,425,429,433,43
7,441,445,449,453,457,461,465,469,473,477,481,485,489,493,497,501,505,509,513,517,521,525,529
,533,537,541,545,549,553,557,561,565,569,573,577,581,585,589,593,597,601,605,609,613,617,621,
625,629,633,637,641,645,649,653,657,661,665,669,673,677,681,685,689,693,697,701,705,709,713,7
17,721,725,729,733,737,741,745,750,754,758,762,766,770,774,778,782,786,790,794,798,802,806,81
0,814,818,822,826,830,834,838,842,846,850,854,858,862,866,870,874,878,882,886,890,894,898,902
,906,910,914,918,922,926,930,934,938,942,946,950,954,958,962,966,970,974,978,982,986,990,994,
998,1002,1006,1010,1014,1018,1022,1026,1030,1034,1038,1042,1046,1050,1054,1058,1062,1066,1070
,1074,1078,1082,1086,1090,1094,1098,1102,1106,1110,1114,1118,1122,1127,1131,1135,1139,1143,11
47,1151,1155,1159,1163,1167,1171,1175,1179,1183,1187,1191,1195,1199,1203,1207,1211,1215,1219,
1223,1227,1231,1235,1239,1243,1247,1251,1255,1259,1263,1267,1271,1275,1279,1283,1287,1291,129
5,1299,1303,1307,1311,1315,1319,1323,1327,1331,1335,1339,1343,1347,1351,1355,1359,1363,1367,1
371,1375,1379,1383,1387,1391,1395,1399,1403,1407,1411,1415,1419,1423,1427,1431,1435,1439,1443
,1447,1451,1455,1459,1463,1467,1471,1475,1479,1483,1487,1491,1495,1500
};
```

dac.h

```
-----------------------------------------------------------------------------------------------
#ifndef SRC_DAC_H_
#define SRC_DAC_H_
#include "main.h"
#include <math.h>


#endif /* SRC_DAC_H_ */

void DAC_Init(void);
void config_SPI_GPIO(void);
void config_SPI_CR(void);
void DAC_write(uint16_t value);
uint16_t DAC_volt_conv(uint16_t mV);
```

keypad.h

```c
#ifndef SRC_KEYPAD_H_
#define SRC_KEYPAD_H_

#include "main.h"
#include <math.h>

#define NUM_COL 3
#define STAR 10
#define POUND 11
#define NO_PRESS 15
#endif /* SRC_KEYPAD_H_ */

void keypad_init(void);
void config_rows(void);
void config_col(void);
uint8_t keypad_read(void);
void delay(uint32_t milliseconds);
```

waveforms.py

```python
import numpy as np

#sine wave
def generate_sine_wave(num_samples):
    x = np.linspace(0, 2 * np.pi, num_samples, endpoint=False)
    y = 1.5 + 1.5 * np.sin(x)
    scaled_y = y * 1000
    return scaled_y.astype(np.uint16)
#triangle wave
def generate_triangle_wave(num_samples):
    y = np.empty(num_samples)
    half_cycle = num_samples // 4
    y[:half_cycle] = np.linspace(1500, 3000, half_cycle, endpoint=False)
    y[half_cycle:3*half_cycle] = np.linspace(3000, 0, 2*half_cycle, endpoint=False)
    y[3*half_cycle:] = np.linspace(0, 1500, num_samples - 3*half_cycle)
    return y.astype(np.uint16)
#sawtooth wave
def generate_sawtooth_wave(num_samples, max_value):
    x = np.linspace(0, 1, num_samples, endpoint=False)
    y = x * 2 - 1
    normalized_y = (y + 1) / 2 * max_value
    return normalized_y.astype(np.uint16)
#save
def save_waveform_to_file(waveform, filename):
    with open(filename, 'w') as file:
        file.write(','.join(map(str, waveform)))

# save waveforms
num_samples_sine = 1500
save_waveform_to_file(generate_sine_wave(num_samples_sine), 'sine_wave_lookup_table.csv')

num_samples_triangle = 1500
save_waveform_to_file(generate_triangle_wave(num_samples_triangle),
'triangle_wave_lookup_table.csv')

num_samples_sawtooth = 1500
max_value_sawtooth = 3000
save_waveform_to_file(generate_sawtooth_wave(num_samples_sawtooth, max_value_sawtooth),
'sawtooth_wave_lookup_table.csv')
```

References

STMicroelectronics. “UM1724 User manual - STM32 Nucleo-64 boards (MB1136).” STMicroelectronics, https://www.st.com/resource/en/user_manual/um1724-stm32-nucleo64-boards-mb1136-stmicroelectronics.pdf.

STMicroelectronics. “STM32L476xx Ultra-low-power Arm® Cortex®-M4 32-bit MCU+FPU, 100DMIPS, up to 1MB Flash, 128 KB SRAM, USB OTG FS, LCD, ext. SMPS.” STMicroelectronics, https://www.st.com/resource/en/datasheet/stm32l476rg.pdf.

STMicroelectronics. “RM0351 Reference manual - STM32L47xxx, STM32L48xxx, STM32L49xxx, and STM32L4Axxx advanced Arm®-based 32-bit MCUs.” STMicroelectronics, https://www.st.com/resource/en/reference_manual/rm0351-stm32l47xxx-stm32l48xxx-stm32l49xxx-and-stm32l4axxx-advanced-armbased-32bit-mcus-stmicroelectronics.pdf.

Microchip Technology Inc. MCP4921/4922 12-Bit DAC with SPI Interface. 2007. 21897B.pdf (microchip.com)