P2 - Function Generator

Sereen Benchohra

EE 329-3

Prof. Paul Hummel

Table of Contents

Behavior Description:

The function generator is designed to be multifunctional. The generator uses a keypad to select different output settings. There are three different waveform options: square wave, sinusoidal, and sawtooth. The square wave can be selected as the output by selecting **7** on the keypad while the sine and sawtooth waves can be selected by pressing **8** or **9** respectively. The frequency options are 100 Hz, 200 Hz, 300 Hz, 400 Hz, and 500 Hz. The frequency of the output can be selected by pressing **1** for 100 Hz, **2** for 200 Hz, **3** for 300 Hz, **4** for 400 Hz, or **5** for 500 Hz. The duty cycle of the squarewave can be adjusted between 10% and 90% by selecting ***** to decrease the duty cycle by 10% or by selecting **#** to increase the duty cycle by 10%. If the **0** key is selected, the duty cycle will result in its default 50% duty cycle. Note that on power up, the function generator will default to output a 100 Hz square wave with a 50% duty cycle.

System Specification:

The system specification can be found in Table 1, and indicate the components and relative information for the Function Generator

STM32L476RG	Power Supply	MiniUSB entry, 5.0 V
	Maximum Frequency	32 MHz
	PIN capability	4-digit, numerical 0-9
MCP4921	Operating Voltage	3.3 V
	Resolution	12 bit
	Pin Capability	Up to 6 pins
Keypad	Size	16 keys, 4x3 Matrix
	Available Keys	0-9, *, #
Oscilloscope	Wiring Capability	2, one for output and GND

Table 1: System Specs

<u>Measurements and Calculations</u>

The system is configured to operate with the 32 Mhz clock, which takes about $4.316\ \mu s$to handle the ISR. With this, the maximum resolution can be calculated

$$Resolution\ = \frac{1s}{t_{ISR}}$$

$$Resolution\ = \frac{1s}{4.316\ \mu s}$$

$$Resolution\ = 231,\ 696\ samples$$

To produce an waveform of 100 Hz (T = 10 ms), it is necessary to get the maximum samples per period:

$$Samples\ per\ period\ = \frac{10\ ms}{4.316\ \mu s}$$

$$Samples\ = \frac{10\ ms}{4\ \mu s}\ = \ 2,316\ samples/\ period$$

75% of the max sample per period is 1737.
For simplicity, the sample/period will be 1740 is used since it can smoothly generate the frequencies, 100 Hz, 200 Hz, 300 Hz, 400 Hz, and 500 Hz.

System Schematic:

The system schematic as shown in Figure 1, includes the connections between the Keypad , Board, DAC, and Oscilloscope .

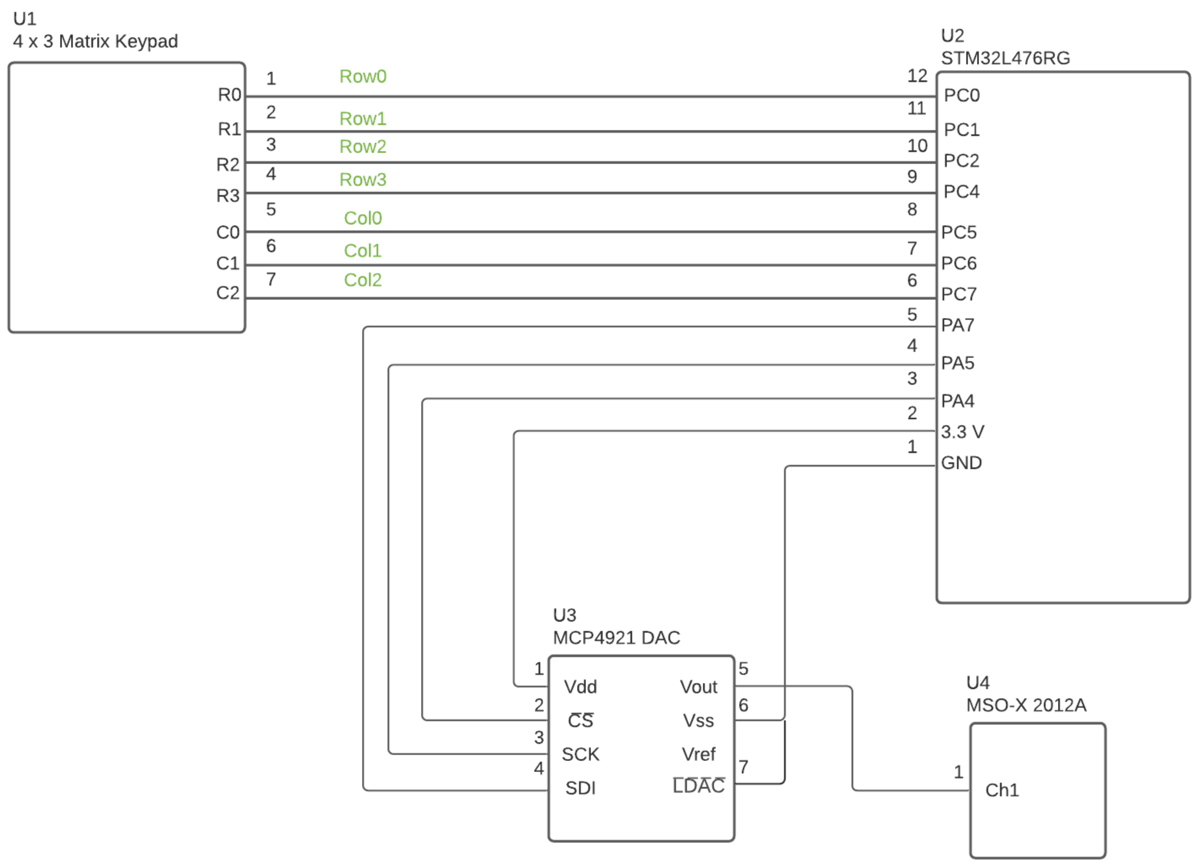

Figure 1: System Schematic Diagram for Function Generator

Software Architecture:

The overall layout of the program (Figure 2) is based on primarily conditional statements checking for the keypad and setting the appropriate states based on the keys. The program first initializes the keypad, Digital to Analog Converter(DAC), and the timer. The program's default Shape state is set to Square where the handler checks for the type of Shape state and displays it. Once the program starts, it first checks if the keys 1- 5 have been pressed. If so, the freq would be set to the key pressed. The program also checks if the keys 6-9 have been pressed, and if the condition has been met it sets the Shape state to the key that has been pressed. Finally, the program has conditional statement statements that check for the duty cycle if the SQUARE state is set. The star key decrements the duty cycle by ten percent. Zero resets the duty cycle back to fifty percent. The pound key increments the duty cycle by ten percent

Flowchart for main Function Generator

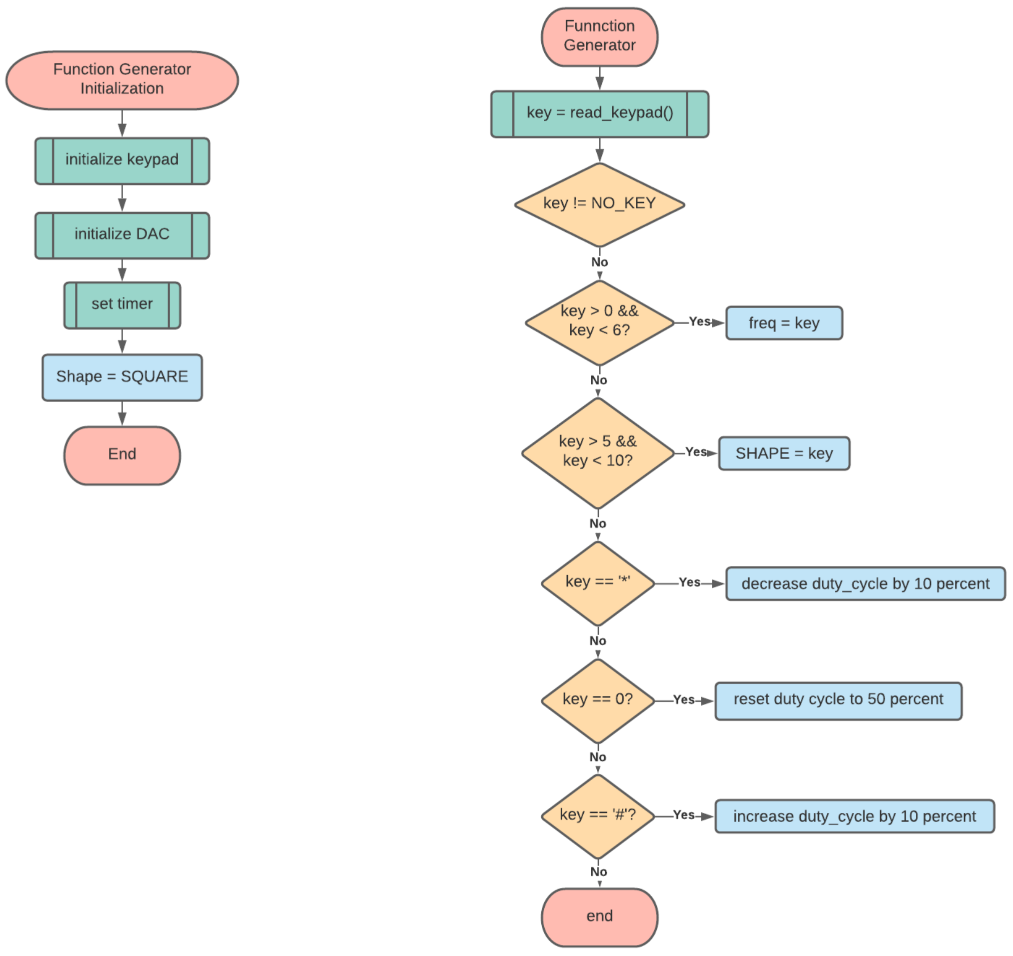

Figure 2 Function Generator initialization and execution

<u>Timer AND IRQ Handler FLOWCHART:</u>

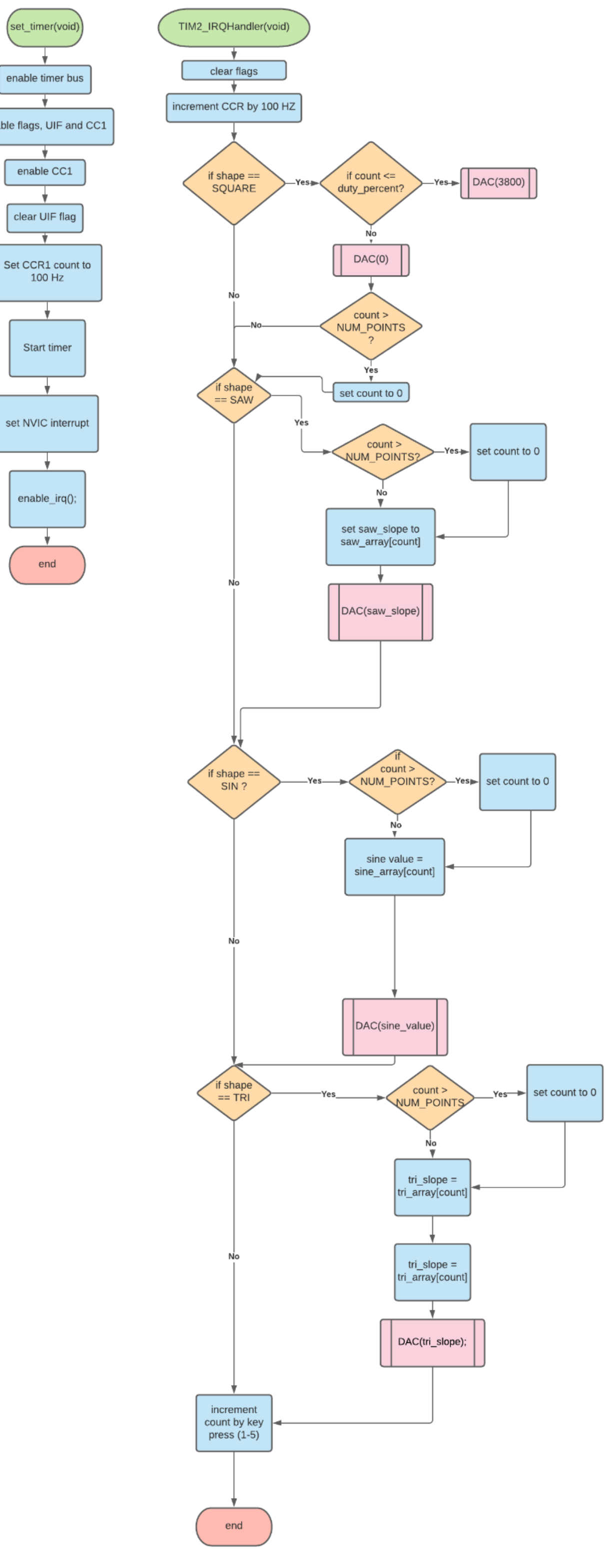

Figure 3 Flowchart for Timer and Handler

Figure 3 shows the flowchart for the setup and execution of the Timer module, where the actual wave is generated. The function, set_timer sets up the registers, enables the interrupts, and the modes to set up the wave. It is in this function where the base frequency is set to 100 Hz, this is important because when the program in the IRQ handler is incrementing, by the FREQ (values 1-5) , it will output frequencies 100 - 500 Hz. In the TIM2_IRQhandler, the CCFIG flag is cleared, then the program checks what type of shape it is (set in the conditional statements see Figure 2). If the shape is a square then the program checks if the count is less than the duty percent , if it is then the input the number 3800 in the DAC which is the equivalent of 3.0 V. Otherwise 0 is sent to the DAC, making the square wave. The program checks if the count is greater than the number of points , if so the count is reseted to 0. If the shape is a SAWTOOTH then the program checks if count is greater than the number of points based in the array, if it is then the count is reseted to 0, otherwise it directly sets the saw_slope to the saw_array[count] and the saw_slope is inputted into the DAC which draws the sawtooth wave. Like the SAWTOOTH, the program checks for the SINE and TRIANGLE wave and sees if the count is greater than the number of points based in the array, if so sets the count to zero. Both cases would lead them to setting the sine_slope and tri_slope to their respective arrays with the count as the index. After the tri_slope and sine_slope is inputted into the DAC, which creates the TRIANGLE and SINE waves. After writing into the DAC, the count would be incremented by the appropriate frequency determined by the keypad in Figure 2.

DAC init and DAC_write FLOWCHART:

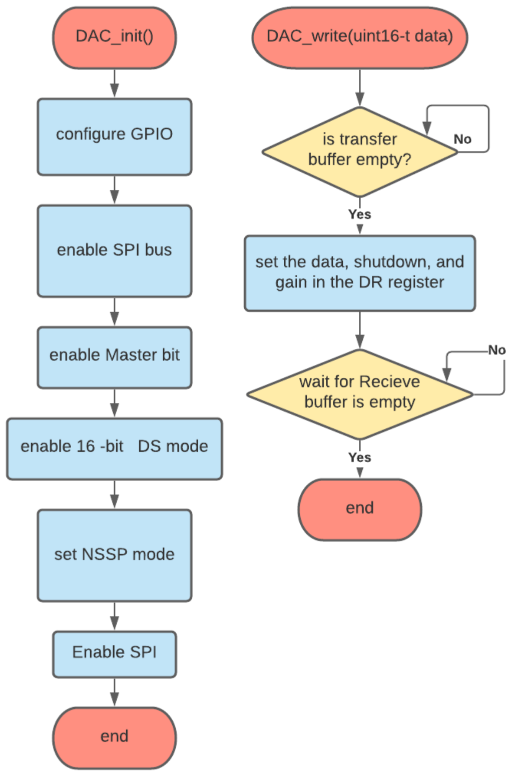

Figure 4, Flowcharts for DAC_write and DAC_init

To actually output the waves into the oscilloscope, the Function generator utilizes the Digital to Analog Converter (DAC) shown in Figure 4 to display the Waves as Voltages. The DAC _init function implemented in the beginning of the program sets the appropriate pins and buses that are needed to activate the DAC. In the DAC_init, the master bit is set, 16-bit is used, and NSSP mode is set. Once initialized, the DAC can be used primarily through the DAC_write function. In the DAC_write function, it checks if the transfer buffer is empty before it sets data into the DAC, with the gain and shutdown bits set. The function then waits for the receive buffer to be empty and ends the function.

Appendix - Code:
Main.c

```
#include "main.h"
#include "DAC.h"
#include "lookup.h"
#include "keypad.h"

/* The program  reads the keypad and checks which keys are pressed if keys
1-5 are pressed, it changes the frequency of the waves, keys 6-9 changes
the sha
 * shape of the waves. If 6 is pressed , a sine wave is displayed. If 7 is
pressed, a triangle is displayed. If 8 is pressed, SAWTOOTH is displayed.
 * In addition if 9 is pressed, SQUARE is displayed . If STAR is pressed,
the duty cycle of the square wave is decreased by 10 percent.
 * If POUND is pressed, the duty cycle of the square wave is inreased by 10
percent. If Zero is pressed the duty cycle is reset to 50 percent.
(Other waves not affected by duty cycles   */

#define DUTY_CYCLE_50 510
#define TEN_PERCENT 100

#define NINETY_PERCENT_BOUND 910
#define TEN_PERCENT_BOUND 110

#define NUM_POINTS (1020 - 1)
#define CCR_FREQ_100_HZ 315

// shape states

#define SIN 6
#define TRI 7
#define SAW 8
#define SQR 9


uint16_t count = 0; // counter for shapes

// variable that record each shape point in the lookup table
int sine_slope;
int tri_slope;
int saw_slope;

int SHAPE;
int freq = 1; // initializes frequency to 100 Hz
int duty_percent = DUTY_CYCLE_50;
int ten_percent = TEN_PERCENT;

void set_timer(void);
void SystemClock_Config(void);


int main(void)
{
	// initializations
	HAL_Init();
	SystemClock_Config();
	DAC_init();
	keypad_init();
	set_timer();

	uint8_t key;

	SHAPE = SQR; // sets initial shape and state to SQUARE


	while (1)
	{
		key = read_keypad();
		if(key != NO_KEY)
		{
			// checks if keys 1 - 5 are pressed, if so it sets the
frequency to the keys. The IRQ will then utilize the information to set the
frequency
			if((key > 0) && (key < 6) )
				freq = key;

			if((key > 5) && (key < 10)) // if keys 6 - 9 are pressed
set the SHAPE state to key ( 6 - SINE, 7 - TRIANGLE, 8 - SAWTOOTH, 9 -
SQUARE
				SHAPE = key;
```

```
                  if(key == STAR) // decrease duty_cycle by ten_percent
                  {
                        if(duty_percent > TEN_PERCENT_BOUND)
                              duty_percent -=TEN_PERCENT;
                  }

                  if(key == 0) // resets the Duty cycle to 50 percent
                        duty_percent = DUTY_CYCLE_50;
                  if(key == POUND) // increase duty_cycle by ten_percent
                  {
                        if(duty_percent < NINETY_PERCENT_BOUND)
                              duty_percent +=TEN_PERCENT;
                  }

                  while(read_keypad() != NO_KEY);  // helps debounce the
keys
            }
      }


}

void set_timer(void)
{
      RCC->APB1ENR1 |= (RCC_APB1ENR1_TIM2EN);// enable timer
      TIM2->DIER |= (TIM_DIER_UIE |TIM_DIER_CC1IE); // enable flags

      TIM2->CCER |= (TIM_CCER_CC1E); //enable CC1
      TIM2->SR &= ~(TIM_SR_UIF); // clear UIF flag
      TIM2->CCR1 = CCR_FREQ_100_HZ - 1; // set CCR1 to 100 Hz

      TIM2->CR1 |= TIM_CR1_CEN; //start timer

      NVIC->ISER[0]= (1<< (TIM2_IRQn & 0x1F)); //NVIC enable

      __enable_irq(); //global enable

}


void TIM2_IRQHandler (void)
{
	if(SHAPE == SIN) // check if SINE
	{
		if(count > NUM_POINTS) // check if bounds is exceeded reset if
so
			count = 0;
		sine_slope = sine_array[count]; // retrieve point from Sine
look-up table
		DAC_write(sine_slope);   // write point to DAC
	}

	else if(SHAPE == TRI) // check if TRIANGLE
	{
		if(count > NUM_POINTS) // check if bounds is exceeded reset if
so
			count = 0;
		tri_slope = tri_array[count]; // retrieve point from triangle
look-up table
		DAC_write(tri_slope);// write point to DAC
	}


	else if(SHAPE == SAW) //  check if SAWTOOTH
	{
		if(count > NUM_POINTS) // check if bounds is exceeded, reset if
so
			count = 0;
		saw_slope = saw_array[count]; // retrieve point from Sawtooth
look-up table
		DAC_write(saw_slope);  // write point to DAC
	}

	else if(SHAPE == SQR)  // check if SQUARE
	{
		if(count <= duty_percent)  // Square function checks how long
when it is high with duration of the duty_percent
			DAC_write(3800);        // write 3 V  when high
		else
		{
			DAC_write(0);  // if bounds of duty cycle exceeded, write
low
			if(count > NUM_POINTS) // check if bounds is exceeded,
reset if so
                        count = 0;
            }

      }


      /*increment count based on the frequency pressed in keypad. Counting
based on the keypad gives us frequencies needed for the generator outputted
as
       (freq * 100 Hz = Wave Frequency) */
      count += freq;

      TIM2->CCR1 += CCR_FREQ_100_HZ; // keeps our frequency base as 100 Hz
      TIM2->SR &= ~(TIM_SR_UIF | TIM_SR_CC1IF); // clears flags

}

void SystemClock_Config(void)
{
      RCC_OscInitTypeDef RCC_OscInitStruct = {0};
      RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};

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

/************************ (C) COPYRIGHT STMicroelectronics *****END OF
FILE****/
```

Lookup.h

```
/*
 * lookup.h
 *
 *  Created on: Nov 1, 2021
 *      Author: Sereen
 */

#ifndef SRC_LOOKUP_H_
#define SRC_LOOKUP_H_

uint16_t saw_array[1020] = { 0, 3.729, 7.458, 11.19, 14.92, 18.65, 22.38,
26.1,
            29.83, 33.56, 37.29, 41.02, 44.75, 48.48, 52.21, 55.94, 59.67,
63.4,
            67.13, 70.85, 74.58, 78.31, 82.04, 85.77, 89.5, 93.23, 96.96,
100.7,
            104.4, 108.1, 111.9, 115.6, 119.3, 123.1, 126.8, 130.5, 134.2,
138,
            141.7, 145.4, 149.2, 152.9, 156.6, 160.4, 164.1, 167.8, 171.5,
175.3,
            179, 182.7, 186.5, 190.2, 193.9, 197.6, 201.4, 205.1, 208.8,
212.6,
            216.3, 220, 223.7, 227.5, 231.2, 234.9, 238.7, 242.4, 246.1,
249.9,
            253.6, 257.3, 261, 264.8, 268.5, 272.2, 276, 279.7, 283.4,
287.1, 290.9,
            294.6, 298.3, 302.1, 305.8, 309.5, 313.2, 317, 320.7, 324.4,
328.2,
            331.9, 335.6, 339.4, 343.1, 346.8, 350.5, 354.3, 358, 361.7,
365.5,
            369.2, 372.9, 376.6, 380.4, 384.1, 387.8, 391.6, 395.3, 399,
402.7,
            406.5, 410.2, 413.9, 417.7, 421.4, 425.1, 428.9, 432.6, 436.3,
440,
            443.8, 447.5, 451.2, 455, 458.7, 462.4, 466.1, 469.9, 473.6,
477.3,
            481.1, 484.8, 488.5, 492.2, 496, 499.7, 503.4, 507.2, 510.9,
514.6,
            518.4, 522.1, 525.8, 529.5, 533.3, 537, 540.7, 544.5, 548.2,
551.9,
            555.6, 559.4, 563.1, 566.8, 570.6, 574.3, 578, 581.7, 585.5,
589.2,
            592.9, 596.7, 600.4, 604.1, 607.9, 611.6, 615.3, 619, 622.8,
626.5,
            630.2, 634, 637.7, 641.4, 645.1, 648.9, 652.6, 656.3, 660.1,
663.8,
            667.5, 671.2, 675, 678.7, 682.4, 686.2, 689.9, 693.6, 697.4,
701.1,
            704.8, 708.5, 712.3, 716, 719.7, 723.5, 727.2, 730.9, 734.6,
738.4,
            742.1, 745.8, 749.6, 753.3, 757, 760.7, 764.5, 768.2, 771.9,
775.7,
            779.4, 783.1, 786.9, 790.6, 794.3, 798, 801.8, 805.5, 809.2,
813, 816.7,
            820.4, 824.1, 827.9, 831.6, 835.3, 839.1, 842.8, 846.5, 850.2,
854,
            857.7, 861.4, 865.2, 868.9, 872.6, 876.3, 880.1, 883.8, 887.5,
891.3,
            895, 898.7, 902.5, 906.2, 909.9, 913.6, 917.4, 921.1, 924.8,
928.6,
            932.3, 936, 939.7, 943.5, 947.2, 950.9, 954.7, 958.4, 962.1,
965.8,
            969.6, 973.3, 977, 980.8, 984.5, 988.2, 992, 995.7, 999.4,
1003, 1007,
            1011, 1014, 1018, 1022, 1026, 1029, 1033, 1037, 1040, 1044,
1048, 1052,
            1055, 1059, 1063, 1067, 1070, 1074, 1078, 1081, 1085, 1089,
1093, 1096,
            1100, 1104, 1108, 1111, 1115, 1119, 1122, 1126, 1130, 1134,
1137, 1141,
            1145, 1149, 1152, 1156, 1160, 1163, 1167, 1171, 1175, 1178,
1182, 1186,
            1190, 1193, 1197, 1201, 1205, 1208, 1212, 1216, 1219, 1223,
1227, 1231,
            1234, 1238, 1242, 1246, 1249, 1253, 1257, 1260, 1264, 1268,
1272, 1275,
            1279, 1283, 1287, 1290, 1294, 1298, 1301, 1305, 1309, 1313,
1316, 1320,
            1324, 1328, 1331, 1335, 1339, 1342, 1346, 1350, 1354, 1357,
1361, 1365,
                    1369, 1372, 1376, 1380, 1384, 1387, 1391, 1395, 1398, 1402,
1406, 1410,
                    1413, 1417, 1421, 1425, 1428, 1432, 1436, 1439, 1443, 1447,
1451, 1454,
                    1458, 1462, 1466, 1469, 1473, 1477, 1480, 1484, 1488, 1492,
1495, 1499,
                    1503, 1507, 1510, 1514, 1518, 1521, 1525, 1529, 1533, 1536,
1540, 1544,
                    1548, 1551, 1555, 1559, 1563, 1566, 1570, 1574, 1577, 1581,
1585, 1589,
                    1592, 1596, 1600, 1604, 1607, 1611, 1615, 1618, 1622, 1626,
1630, 1633,
                    1637, 1641, 1645, 1648, 1652, 1656, 1659, 1663, 1667, 1671,
1674, 1678,
                    1682, 1686, 1689, 1693, 1697, 1700, 1704, 1708, 1712, 1715,
1719, 1723,
                    1727, 1730, 1734, 1738, 1742, 1745, 1749, 1753, 1756, 1760,
1764, 1768,
                    1771, 1775, 1779, 1783, 1786, 1790, 1794, 1797, 1801, 1805,
1809, 1812,
                    1816, 1820, 1824, 1827, 1831, 1835, 1838, 1842, 1846, 1850,
1853, 1857,
                    1861, 1865, 1868, 1872, 1876, 1879, 1883, 1887, 1891, 1894,
1898, 1902,
                    1906, 1909, 1913, 1917, 1921, 1924, 1928, 1932, 1935, 1939,
1943, 1947,
                    1950, 1954, 1958, 1962, 1965, 1969, 1973, 1976, 1980, 1984,
1988, 1991,
                    1995, 1999, 2003, 2006, 2010, 2014, 2017, 2021, 2025, 2029,
2032, 2036,
                    2040, 2044, 2047, 2051, 2055, 2058, 2062, 2066, 2070, 2073,
2077, 2081,
                    2085, 2088, 2092, 2096, 2100, 2103, 2107, 2111, 2114, 2118,
2122, 2126,
                    2129, 2133, 2137, 2141, 2144, 2148, 2152, 2155, 2159, 2163,
2167, 2170,
                    2174, 2178, 2182, 2185, 2189, 2193, 2196, 2200, 2204, 2208,
2211, 2215,
                    2219, 2223, 2226, 2230, 2234, 2237, 2241, 2245, 2249, 2252,
2256, 2260,
                    2264, 2267, 2271, 2275, 2279, 2282, 2286, 2290, 2293, 2297,
2301, 2305,
                    2308, 2312, 2316, 2320, 2323, 2327, 2331, 2334, 2338, 2342,
2346, 2349,
                    2353, 2357, 2361, 2364, 2368, 2372, 2375, 2379, 2383, 2387,
2390, 2394,
                    2398, 2402, 2405, 2409, 2413, 2416, 2420, 2424, 2428, 2431,
2435, 2439,
                    2443, 2446, 2450, 2454, 2458, 2461, 2465, 2469, 2472, 2476,
2480, 2484,
                    2487, 2491, 2495, 2499, 2502, 2506, 2510, 2513, 2517, 2521,
2525, 2528,
                    2532, 2536, 2540, 2543, 2547, 2551, 2554, 2558, 2562, 2566,
2569, 2573,
                    2577, 2581, 2584, 2588, 2592, 2595, 2599, 2603, 2607, 2610,
2614, 2618,
                    2622, 2625, 2629, 2633, 2637, 2640, 2644, 2648, 2651, 2655,
2659, 2663,
                    2666, 2670, 2674, 2678, 2681, 2685, 2689, 2692, 2696, 2700,
2704, 2707,
                    2711, 2715, 2719, 2722, 2726, 2730, 2733, 2737, 2741, 2745,
2748, 2752,
                    2756, 2760, 2763, 2767, 2771, 2774, 2778, 2782, 2786, 2789,
2793, 2797,
                    2801, 2804, 2808, 2812, 2816, 2819, 2823, 2827, 2830, 2834,
2838, 2842,
                    2845, 2849, 2853, 2857, 2860, 2864, 2868, 2871, 2875, 2879,
2883, 2886,
                    2890, 2894, 2898, 2901, 2905, 2909, 2912, 2916, 2920, 2924,
2927, 2931,
                    2935, 2939, 2942, 2946, 2950, 2953, 2957, 2961, 2965, 2968,
2972, 2976,
                    2980, 2983, 2987, 2991, 2995, 2998, 3002, 3006, 3009, 3013,
3017, 3021,
                    3024, 3028, 3032, 3036, 3039, 3043, 3047, 3050, 3054, 3058,
3062, 3065,
                    3069, 3073, 3077, 3080, 3084, 3088, 3091, 3095, 3099, 3103,
3106, 3110,
                    3114, 3118, 3121, 3125, 3129, 3132, 3136, 3140, 3144, 3147,
3151, 3155,
                    3159, 3162, 3166, 3170, 3174, 3177, 3181, 3185, 3188, 3192,
3196, 3200,
                    3203, 3207, 3211, 3215, 3218, 3222, 3226, 3229, 3233, 3237,
3241, 3244,
            3248, 3252, 3256, 3259, 3263, 3267, 3270, 3274, 3278, 3282,
3285, 3289,
            3293, 3297, 3300, 3304, 3308, 3311, 3315, 3319, 3323, 3326,
3330, 3334,
            3338, 3341, 3345, 3349, 3353, 3356, 3360, 3364, 3367, 3371,
3375, 3379,
            3382, 3386, 3390, 3394, 3397, 3401, 3405, 3408, 3412, 3416,
3420, 3423,
            3427, 3431, 3435, 3438, 3442, 3446, 3449, 3453, 3457, 3461,
3464, 3468,
            3472, 3476, 3479, 3483, 3487, 3490, 3494, 3498, 3502, 3505,
3509, 3513,
            3517, 3520, 3524, 3528, 3532, 3535, 3539, 3543, 3546, 3550,
3554, 3558,
            3561, 3565, 3569, 3573, 3576, 3580, 3584, 3587, 3591, 3595,
3599, 3602,
            3606, 3610, 3614, 3617, 3621, 3625, 3628, 3632, 3636, 3640,
3643, 3647,
            3651, 3655, 3658, 3662, 3666, 3669, 3673, 3677, 3681, 3684,
3688, 3692,
            3696, 3699, 3703, 3707, 3711, 3714, 3718, 3722, 3725, 3729,
3733, 3737,
            3740, 3744, 3748, 3752, 3755, 3759, 3763, 3766, 3770, 3774,
3778, 3781,
            3785, 3789, 3793, 3796, 3800 };

uint16_t tri_array[1020] = { 0x7, 0xf, 0x16, 0x1e, 0x25, 0x2d, 0x34, 0x3c,
0x43,
            0x4b, 0x52, 0x59, 0x61, 0x68, 0x70, 0x77, 0x7f, 0x86, 0x8e,
0x95, 0x9c,
            0xa4, 0xab, 0xb3, 0xba, 0xc2, 0xc9, 0xd1, 0xd8, 0xe0, 0xe7,
0xee, 0xf6,
            0xfd, 0x105, 0x10c, 0x114, 0x11b, 0x123, 0x12a, 0x131, 0x139,
0x140,
            0x148, 0x14f, 0x157, 0x15e, 0x166, 0x16d, 0x175, 0x17c, 0x183,
0x18b,
            0x192, 0x19a, 0x1a1, 0x1a9, 0x1b0, 0x1b8, 0x1bf, 0x1c7, 0x1ce,
0x1d5,
            0x1dd, 0x1e4, 0x1ec, 0x1f3, 0x1fb, 0x202, 0x20a, 0x211, 0x218,
0x220,
            0x227, 0x22f, 0x236, 0x23e, 0x245, 0x24d, 0x254, 0x25c, 0x263,
0x26a,
                0x272, 0x279, 0x281, 0x288, 0x290, 0x297, 0x29f, 0x2a6, 0x2ad,
0x2b5,
                0x2bc, 0x2c4, 0x2cb, 0x2d3, 0x2da, 0x2e2, 0x2e9, 0x2f1, 0x2f8,
0x2ff,
                0x307, 0x30e, 0x316, 0x31d, 0x325, 0x32c, 0x334, 0x33b, 0x343,
0x34a,
                0x351, 0x359, 0x360, 0x368, 0x36f, 0x377, 0x37e, 0x386, 0x38d,
0x394,
                0x39c, 0x3a3, 0x3ab, 0x3b2, 0x3ba, 0x3c1, 0x3c9, 0x3d0, 0x3d8,
0x3df,
                0x3e6, 0x3ee, 0x3f5, 0x3fd, 0x404, 0x40c, 0x413, 0x41b, 0x422,
0x429,
                0x431, 0x438, 0x440, 0x447, 0x44f, 0x456, 0x45e, 0x465, 0x46d,
0x474,
                0x47b, 0x483, 0x48a, 0x492, 0x499, 0x4a1, 0x4a8, 0x4b0, 0x4b7,
0x4bf,
                0x4c6, 0x4cd, 0x4d5, 0x4dc, 0x4e4, 0x4eb, 0x4f3, 0x4fa, 0x502,
0x509,
                0x510, 0x518, 0x51f, 0x527, 0x52e, 0x536, 0x53d, 0x545, 0x54c,
0x554,
                0x55b, 0x562, 0x56a, 0x571, 0x579, 0x580, 0x588, 0x58f, 0x597,
0x59e,
                0x5a5, 0x5ad, 0x5b4, 0x5bc, 0x5c3, 0x5cb, 0x5d2, 0x5da, 0x5e1,
0x5e9,
                0x5f0, 0x5f7, 0x5ff, 0x606, 0x60e, 0x615, 0x61d, 0x624, 0x62c,
0x633,
                0x63b, 0x642, 0x649, 0x651, 0x658, 0x660, 0x667, 0x66f, 0x676,
0x67e,
                0x685, 0x68c, 0x694, 0x69b, 0x6a3, 0x6aa, 0x6b2, 0x6b9, 0x6c1,
0x6c8,
                0x6d0, 0x6d7, 0x6de, 0x6e6, 0x6ed, 0x6f5, 0x6fc, 0x704, 0x70b,
0x713,
                0x71a, 0x721, 0x729, 0x730, 0x738, 0x73f, 0x747, 0x74e, 0x756,
0x75d,
                0x765, 0x76c, 0x773, 0x77b, 0x782, 0x78a, 0x791, 0x799, 0x7a0,
0x7a8,
                0x7af, 0x7b7, 0x7be, 0x7c5, 0x7cd, 0x7d4, 0x7dc, 0x7e3, 0x7eb,
0x7f2,
                0x7fa, 0x801, 0x808, 0x810, 0x817, 0x81f, 0x826, 0x82e, 0x835,
0x83d,
                0x844, 0x84c, 0x853, 0x85a, 0x862, 0x869, 0x871, 0x878, 0x880,
0x887,
                0x88f, 0x896, 0x89d, 0x8a5, 0x8ac, 0x8b4, 0x8bb, 0x8c3, 0x8ca,
0x8d2,
                0x8d9, 0x8e1, 0x8e8, 0x8ef, 0x8f7, 0x8fe, 0x906, 0x90d, 0x915,
0x91c,
                0x924, 0x92b, 0x933, 0x93a, 0x941, 0x949, 0x950, 0x958, 0x95f,
0x967,
                0x96e, 0x976, 0x97d, 0x984, 0x98c, 0x993, 0x99b, 0x9a2, 0x9aa,
0x9b1,
                0x9b9, 0x9c0, 0x9c8, 0x9cf, 0x9d6, 0x9de, 0x9e5, 0x9ed, 0x9f4,
0x9fc,
                0xa03, 0xa0b, 0xa12, 0xa19, 0xa21, 0xa28, 0xa30, 0xa37, 0xa3f,
0xa46,
                0xa4e, 0xa55, 0xa5d, 0xa64, 0xa6b, 0xa73, 0xa7a, 0xa82, 0xa89,
0xa91,
                0xa98, 0xaa0, 0xaa7, 0xaaf, 0xab6, 0xabd, 0xac5, 0xacc, 0xad4,
0xadb,
                0xae3, 0xaea, 0xaf2, 0xaf9, 0xb00, 0xb08, 0xb0f, 0xb17, 0xb1e,
0xb26,
                0xb2d, 0xb35, 0xb3c, 0xb44, 0xb4b, 0xb52, 0xb5a, 0xb61, 0xb69,
0xb70,
                0xb78, 0xb7f, 0xb87, 0xb8e, 0xb95, 0xb9d, 0xba4, 0xbac, 0xbb3,
0xbbb,
                0xbc2, 0xbca, 0xbd1, 0xbd9, 0xbe0, 0xbe7, 0xbef, 0xbf6, 0xbfe,
0xc05,
                0xc0d, 0xc14, 0xc1c, 0xc23, 0xc2b, 0xc32, 0xc39, 0xc41, 0xc48,
0xc50,
                0xc57, 0xc5f, 0xc66, 0xc6e, 0xc75, 0xc7c, 0xc84, 0xc8b, 0xc93,
0xc9a,
                0xca2, 0xca9, 0xcb1, 0xcb8, 0xcc0, 0xcc7, 0xcce, 0xcd6, 0xcdd,
0xce5,
                0xcec, 0xcf4, 0xcfb, 0xd03, 0xd0a, 0xd11, 0xd19, 0xd20, 0xd28,
0xd2f,
                0xd37, 0xd3e, 0xd46, 0xd4d, 0xd55, 0xd5c, 0xd63, 0xd6b, 0xd72,
0xd7a,
                0xd81, 0xd89, 0xd90, 0xd98, 0xd9f, 0xda7, 0xdae, 0xdb5, 0xdbd,
0xdc4,
                0xdcc, 0xdd3, 0xddb, 0xde2, 0xdea, 0xdf1, 0xdf8, 0xe00, 0xe07,
0xe0f,
                0xe16, 0xe1e, 0xe25, 0xe2d, 0xe34, 0xe3c, 0xe43, 0xe4a, 0xe52,
0xe59,
                0xe61, 0xe68, 0xe70, 0xe77, 0xe7f, 0xe86, 0xe8d, 0xe95, 0xe9c,
0xea4,
                0xeab, 0xeb3, 0xeba, 0xec2, 0xec9, 0xed1, 0xed8, 0xed1, 0xec9,
0xec2,
                0xeba, 0xeb3, 0xeab, 0xea4, 0xe9c, 0xe95, 0xe8d, 0xe86, 0xe7f,
0xe77,
                0xe70, 0xe68, 0xe61, 0xe59, 0xe52, 0xe4a, 0xe43, 0xe3c, 0xe34,
0xe2d,
                0xe25, 0xe1e, 0xe16, 0xe0f, 0xe07, 0xe00, 0xdf8, 0xdf1, 0xdea,
0xde2,
                0xddb, 0xdd3, 0xdcc, 0xdc4, 0xdbd, 0xdb5, 0xdae, 0xda7, 0xd9f,
0xd98,
                0xd90, 0xd89, 0xd81, 0xd7a, 0xd72, 0xd6b, 0xd63, 0xd5c, 0xd55,
0xd4d,
                0xd46, 0xd3e, 0xd37, 0xd2f, 0xd28, 0xd20, 0xd19, 0xd11, 0xd0a,
0xd03,
                0xcfb, 0xcf4, 0xcec, 0xce5, 0xcdd, 0xcd6, 0xcce, 0xcc7, 0xcc0,
0xcb8,
                0xcb1, 0xca9, 0xca2, 0xc9a, 0xc93, 0xc8b, 0xc84, 0xc7c, 0xc75,
0xc6e,
                0xc66, 0xc5f, 0xc57, 0xc50, 0xc48, 0xc41, 0xc39, 0xc32, 0xc2b,
0xc23,
                0xc1c, 0xc14, 0xc0d, 0xc05, 0xbfe, 0xbf6, 0xbef, 0xbe7, 0xbe0,
0xbd9,
                0xbd1, 0xbca, 0xbc2, 0xbbb, 0xbb3, 0xbac, 0xba4, 0xb9d, 0xb95,
0xb8e,
                0xb87, 0xb7f, 0xb78, 0xb70, 0xb69, 0xb61, 0xb5a, 0xb52, 0xb4b,
0xb44,
                0xb3c, 0xb35, 0xb2d, 0xb26, 0xb1e, 0xb17, 0xb0f, 0xb08, 0xb00,
0xaf9,
                0xaf2, 0xaea, 0xae3, 0xadb, 0xad4, 0xacc, 0xac5, 0xabd, 0xab6,
0xaaf,
                0xaa7, 0xaa0, 0xa98, 0xa91, 0xa89, 0xa82, 0xa7a, 0xa73, 0xa6b,
0xa64,
                0xa5d, 0xa55, 0xa4e, 0xa46, 0xa3f, 0xa37, 0xa30, 0xa28, 0xa21,
0xa19,
                0xa12, 0xa0b, 0xa03, 0x9fc, 0x9f4, 0x9ed, 0x9e5, 0x9de, 0x9d6,
0x9cf,
                0x9c8, 0x9c0, 0x9b9, 0x9b1, 0x9aa, 0x9a2, 0x99b, 0x993, 0x98c,
0x984,
                0x97d, 0x976, 0x96e, 0x967, 0x95f, 0x958, 0x950, 0x949, 0x941,
0x93a,
                0x933, 0x92b, 0x924, 0x91c, 0x915, 0x90d, 0x906, 0x8fe, 0x8f7,
0x8ef,
            0x8e8, 0x8e1, 0x8d9, 0x8d2, 0x8ca, 0x8c3, 0x8bb, 0x8b4, 0x8ac,
0x8a5,
            0x89d, 0x896, 0x88f, 0x887, 0x880, 0x878, 0x871, 0x869, 0x862,
0x85a,
            0x853, 0x84c, 0x844, 0x83d, 0x835, 0x82e, 0x826, 0x81f, 0x817,
0x810,
            0x808, 0x801, 0x7fa, 0x7f2, 0x7eb, 0x7e3, 0x7dc, 0x7d4, 0x7cd,
0x7c5,
            0x7be, 0x7b7, 0x7af, 0x7a8, 0x7a0, 0x799, 0x791, 0x78a, 0x782,
0x77b,
            0x773, 0x76c, 0x765, 0x75d, 0x756, 0x74e, 0x747, 0x73f, 0x738,
0x730,
            0x729, 0x721, 0x71a, 0x713, 0x70b, 0x704, 0x6fc, 0x6f5, 0x6ed,
0x6e6,
            0x6de, 0x6d7, 0x6d0, 0x6c8, 0x6c1, 0x6b9, 0x6b2, 0x6aa, 0x6a3,
0x69b,
            0x694, 0x68c, 0x685, 0x67e, 0x676, 0x66f, 0x667, 0x660, 0x658,
0x651,
            0x649, 0x642, 0x63b, 0x633, 0x62c, 0x624, 0x61d, 0x615, 0x60e,
0x606,
            0x5ff, 0x5f7, 0x5f0, 0x5e9, 0x5e1, 0x5da, 0x5d2, 0x5cb, 0x5c3,
0x5bc,
            0x5b4, 0x5ad, 0x5a5, 0x59e, 0x597, 0x58f, 0x588, 0x580, 0x579,
0x571,
            0x56a, 0x562, 0x55b, 0x554, 0x54c, 0x545, 0x53d, 0x536, 0x52e,
0x527,
            0x51f, 0x518, 0x510, 0x509, 0x502, 0x4fa, 0x4f3, 0x4eb, 0x4e4,
0x4dc,
            0x4d5, 0x4cd, 0x4c6, 0x4bf, 0x4b7, 0x4b0, 0x4a8, 0x4a1, 0x499,
0x492,
            0x48a, 0x483, 0x47b, 0x474, 0x46d, 0x465, 0x45e, 0x456, 0x44f,
0x447,
            0x440, 0x438, 0x431, 0x429, 0x422, 0x41b, 0x413, 0x40c, 0x404,
0x3fd,
            0x3f5, 0x3ee, 0x3e6, 0x3df, 0x3d8, 0x3d0, 0x3c9, 0x3c1, 0x3ba,
0x3b2,
            0x3ab, 0x3a3, 0x39c, 0x394, 0x38d, 0x386, 0x37e, 0x377, 0x36f,
0x368,
            0x360, 0x359, 0x351, 0x34a, 0x343, 0x33b, 0x334, 0x32c, 0x325,
0x31d,
            0x316, 0x30e, 0x307, 0x2ff, 0x2f8, 0x2f1, 0x2e9, 0x2e2, 0x2da,
0x2d3,
            0x2cb, 0x2c4, 0x2bc, 0x2b5, 0x2ad, 0x2a6, 0x29f, 0x297, 0x290,
0x288,
            0x281, 0x279, 0x272, 0x26a, 0x263, 0x25c, 0x254, 0x24d, 0x245,
0x23e,
            0x236, 0x22f, 0x227, 0x220, 0x218, 0x211, 0x20a, 0x202, 0x1fb,
0x1f3,
            0x1ec, 0x1e4, 0x1dd, 0x1d5, 0x1ce, 0x1c7, 0x1bf, 0x1b8, 0x1b0,
0x1a9,
            0x1a1, 0x19a, 0x192, 0x18b, 0x183, 0x17c, 0x175, 0x16d, 0x166,
0x15e,
            0x157, 0x14f, 0x148, 0x140, 0x139, 0x131, 0x12a, 0x123, 0x11b,
0x114,
            0x10c, 0x105, 0xfd, 0xf6, 0xee, 0xe7, 0xe0, 0xd8, 0xd1, 0xc9,
0xc2,
            0xba, 0xb3, 0xab, 0xa4, 0x9c, 0x95, 0x8e, 0x86, 0x7f, 0x77,
0x70, 0x68,
            0x61, 0x59, 0x52, 0x4b, 0x43, 0x3c, 0x34, 0x2d, 0x25, 0x1e,
0x16, 0xf,
            0x7, 0x0 };

uint16_t sine_array[1020] = { 0x76c, 0x778, 0x783, 0x78f, 0x79b, 0x7a7,
0x7b2,
            0x7be, 0x7ca, 0x7d5, 0x7e1, 0x7ed, 0x7f8, 0x804, 0x810, 0x81b,
0x827,
            0x833, 0x83e, 0x84a, 0x855, 0x861, 0x86d, 0x878, 0x884, 0x88f,
0x89b,
            0x8a7, 0x8b2, 0x8be, 0x8c9, 0x8d5, 0x8e0, 0x8ec, 0x8f7, 0x902,
0x90e,
            0x919, 0x925, 0x930, 0x93b, 0x947, 0x952, 0x95d, 0x969, 0x974,
0x97f,
            0x98a, 0x996, 0x9a1, 0x9ac, 0x9b7, 0x9c2, 0x9cd, 0x9d8, 0x9e3,
0x9ef,
            0x9fa, 0xa04, 0xa0f, 0xa1a, 0xa25, 0xa30, 0xa3b, 0xa46, 0xa51,
0xa5b,
            0xa66, 0xa71, 0xa7b, 0xa86, 0xa91, 0xa9b, 0xaa6, 0xab0, 0xabb,
0xac5,
            0xad0, 0xada, 0xae5, 0xaef, 0xaf9, 0xb03, 0xb0e, 0xb18, 0xb22,
0xb2c,
            0xb36, 0xb40, 0xb4a, 0xb54, 0xb5e, 0xb68, 0xb72, 0xb7c, 0xb85,
0xb8f,
            0xb99, 0xba3, 0xbac, 0xbb6, 0xbbf, 0xbc9, 0xbd2, 0xbdc, 0xbe5,
0xbee,
            0xbf8, 0xc01, 0xc0a, 0xc13, 0xc1c, 0xc25, 0xc2e, 0xc37, 0xc40,
0xc49,
            0xc52, 0xc5b, 0xc63, 0xc6c, 0xc75, 0xc7d, 0xc86, 0xc8e, 0xc97,
0xc9f,
            0xca7, 0xcb0, 0xcb8, 0xcc0, 0xcc8, 0xcd0, 0xcd8, 0xce0, 0xce8,
0xcf0,
            0xcf8, 0xd00, 0xd07, 0xd0f, 0xd16, 0xd1e, 0xd25, 0xd2d, 0xd34,
0xd3c,
            0xd43, 0xd4a, 0xd51, 0xd58, 0xd5f, 0xd66, 0xd6d, 0xd74, 0xd7b,
0xd82,
            0xd88, 0xd8f, 0xd95, 0xd9c, 0xda2, 0xda9, 0xdaf, 0xdb5, 0xdbb,
0xdc2,
            0xdc8, 0xdce, 0xdd4, 0xdd9, 0xddf, 0xde5, 0xdeb, 0xdf0, 0xdf6,
0xdfb,
            0xe01, 0xe06, 0xe0c, 0xe11, 0xe16, 0xe1b, 0xe20, 0xe25, 0xe2a,
0xe2f,
            0xe34, 0xe38, 0xe3d, 0xe42, 0xe46, 0xe4b, 0xe4f, 0xe53, 0xe58,
0xe5c,
            0xe60, 0xe64, 0xe68, 0xe6c, 0xe70, 0xe74, 0xe77, 0xe7b, 0xe7f,
0xe82,
            0xe86, 0xe89, 0xe8c, 0xe8f, 0xe93, 0xe96, 0xe99, 0xe9c, 0xe9f,
0xea1,
            0xea4, 0xea7, 0xea9, 0xeac, 0xeae, 0xeb1, 0xeb3, 0xeb5, 0xeb8,
0xeba,
            0xebc, 0xebe, 0xec0, 0xec2, 0xec3, 0xec5, 0xec7, 0xec8, 0xeca,
0xecb,
            0xecc, 0xece, 0xecf, 0xed0, 0xed1, 0xed2, 0xed3, 0xed4, 0xed4,
0xed5,
            0xed6, 0xed6, 0xed7, 0xed7, 0xed7, 0xed8, 0xed8, 0xed8, 0xed8,
0xed8,
            0xed8, 0xed8, 0xed7, 0xed7, 0xed7, 0xed6, 0xed6, 0xed5, 0xed4,
0xed4,
            0xed3, 0xed2, 0xed1, 0xed0, 0xecf, 0xece, 0xecc, 0xecb, 0xeca,
0xec8,
            0xec7, 0xec5, 0xec3, 0xec2, 0xec0, 0xebe, 0xebc, 0xeba, 0xeb8,
0xeb5,
            0xeb3, 0xeb1, 0xeae, 0xeac, 0xea9, 0xea7, 0xea4, 0xea1, 0xe9f,
0xe9c,
            0xe99, 0xe96, 0xe93, 0xe8f, 0xe8c, 0xe89, 0xe86, 0xe82, 0xe7f,
0xe7b,
            0xe77, 0xe74, 0xe70, 0xe6c, 0xe68, 0xe64, 0xe60, 0xe5c, 0xe58,
0xe53,
                0xe4f, 0xe4b, 0xe46, 0xe42, 0xe3d, 0xe38, 0xe34, 0xe2f, 0xe2a,
0xe25,
                0xe20, 0xe1b, 0xe16, 0xe11, 0xe0c, 0xe06, 0xe01, 0xdfb, 0xdf6,
0xdf0,
                0xdeb, 0xde5, 0xddf, 0xdd9, 0xdd4, 0xdce, 0xdc8, 0xdc2, 0xdbb,
0xdb5,
                0xdaf, 0xda9, 0xda2, 0xd9c, 0xd95, 0xd8f, 0xd88, 0xd82, 0xd7b,
0xd74,
                0xd6d, 0xd66, 0xd5f, 0xd58, 0xd51, 0xd4a, 0xd43, 0xd3c, 0xd34,
0xd2d,
                0xd25, 0xd1e, 0xd16, 0xd0f, 0xd07, 0xd00, 0xcf8, 0xcf0, 0xce8,
0xce0,
                0xcd8, 0xcd0, 0xcc8, 0xcc0, 0xcb8, 0xcb0, 0xca7, 0xc9f, 0xc97,
0xc8e,
                0xc86, 0xc7d, 0xc75, 0xc6c, 0xc63, 0xc5b, 0xc52, 0xc49, 0xc40,
0xc37,
                0xc2e, 0xc25, 0xc1c, 0xc13, 0xc0a, 0xc01, 0xbf8, 0xbee, 0xbe5,
0xbdc,
                0xbd2, 0xbc9, 0xbbf, 0xbb6, 0xbac, 0xba3, 0xb99, 0xb8f, 0xb85,
0xb7c,
                0xb72, 0xb68, 0xb5e, 0xb54, 0xb4a, 0xb40, 0xb36, 0xb2c, 0xb22,
0xb18,
                0xb0e, 0xb03, 0xaf9, 0xaef, 0xae5, 0xada, 0xad0, 0xac5, 0xabb,
0xab0,
                0xaa6, 0xa9b, 0xa91, 0xa86, 0xa7b, 0xa71, 0xa66, 0xa5b, 0xa51,
0xa46,
                0xa3b, 0xa30, 0xa25, 0xa1a, 0xa0f, 0xa04, 0x9fa, 0x9ef, 0x9e3,
0x9d8,
                0x9cd, 0x9c2, 0x9b7, 0x9ac, 0x9a1, 0x996, 0x98a, 0x97f, 0x974,
0x969,
                0x95d, 0x952, 0x947, 0x93b, 0x930, 0x925, 0x919, 0x90e, 0x902,
0x8f7,
                0x8ec, 0x8e0, 0x8d5, 0x8c9, 0x8be, 0x8b2, 0x8a7, 0x89b, 0x88f,
0x884,
                0x878, 0x86d, 0x861, 0x855, 0x84a, 0x83e, 0x833, 0x827, 0x81b,
0x810,
                0x804, 0x7f8, 0x7ed, 0x7e1, 0x7d5, 0x7ca, 0x7be, 0x7b2, 0x7a7,
0x79b,
                0x78f, 0x783, 0x778, 0x76c, 0x760, 0x755, 0x749, 0x73d, 0x731,
0x726,
                0x71a, 0x70e, 0x703, 0x6f7, 0x6eb, 0x6e0, 0x6d4, 0x6c8, 0x6bd,
0x6b1,
            0x6a5, 0x69a, 0x68e, 0x683, 0x677, 0x66b, 0x660, 0x654, 0x649,
0x63d,
            0x631, 0x626, 0x61a, 0x60f, 0x603, 0x5f8, 0x5ec, 0x5e1, 0x5d6,
0x5ca,
            0x5bf, 0x5b3, 0x5a8, 0x59d, 0x591, 0x586, 0x57b, 0x56f, 0x564,
0x559,
            0x54e, 0x542, 0x537, 0x52c, 0x521, 0x516, 0x50b, 0x500, 0x4f5,
0x4e9,
            0x4de, 0x4d4, 0x4c9, 0x4be, 0x4b3, 0x4a8, 0x49d, 0x492, 0x487,
0x47d,
            0x472, 0x467, 0x45d, 0x452, 0x447, 0x43d, 0x432, 0x428, 0x41d,
0x413,
            0x408, 0x3fe, 0x3f3, 0x3e9, 0x3df, 0x3d5, 0x3ca, 0x3c0, 0x3b6,
0x3ac,
            0x3a2, 0x398, 0x38e, 0x384, 0x37a, 0x370, 0x366, 0x35c, 0x353,
0x349,
            0x33f, 0x335, 0x32c, 0x322, 0x319, 0x30f, 0x306, 0x2fc, 0x2f3,
0x2ea,
            0x2e0, 0x2d7, 0x2ce, 0x2c5, 0x2bc, 0x2b3, 0x2aa, 0x2a1, 0x298,
0x28f,
            0x286, 0x27d, 0x275, 0x26c, 0x263, 0x25b, 0x252, 0x24a, 0x241,
0x239,
            0x231, 0x228, 0x220, 0x218, 0x210, 0x208, 0x200, 0x1f8, 0x1f0,
0x1e8,
            0x1e0, 0x1d8, 0x1d1, 0x1c9, 0x1c2, 0x1ba, 0x1b3, 0x1ab, 0x1a4,
0x19c,
            0x195, 0x18e, 0x187, 0x180, 0x179, 0x172, 0x16b, 0x164, 0x15d,
0x156,
            0x150, 0x149, 0x143, 0x13c, 0x136, 0x12f, 0x129, 0x123, 0x11d,
0x116,
            0x110, 0x10a, 0x104, 0xff, 0xf9, 0xf3, 0xed, 0xe8, 0xe2, 0xdd,
0xd7,
            0xd2, 0xcc, 0xc7, 0xc2, 0xbd, 0xb8, 0xb3, 0xae, 0xa9, 0xa4,
0xa0, 0x9b,
            0x96, 0x92, 0x8d, 0x89, 0x85, 0x80, 0x7c, 0x78, 0x74, 0x70,
0x6c, 0x68,
            0x64, 0x61, 0x5d, 0x59, 0x56, 0x52, 0x4f, 0x4c, 0x49, 0x45,
0x42, 0x3f,
            0x3c, 0x39, 0x37, 0x34, 0x31, 0x2f, 0x2c, 0x2a, 0x27, 0x25,
0x23, 0x20,
            0x1e, 0x1c, 0x1a, 0x18, 0x16, 0x15, 0x13, 0x11, 0x10, 0xe, 0xd,
0xc,
            0xa, 0x9, 0x8, 0x7, 0x6, 0x5, 0x4, 0x4, 0x3, 0x2, 0x2, 0x1,
0x1, 0x1,
            0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x1, 0x1, 0x1, 0x2, 0x2,
0x3, 0x4,
            0x4, 0x5, 0x6, 0x7, 0x8, 0x9, 0xa, 0xc, 0xd, 0xe, 0x10, 0x11,
0x13,
            0x15, 0x16, 0x18, 0x1a, 0x1c, 0x1e, 0x20, 0x23, 0x25, 0x27,
0x2a, 0x2c,
            0x2f, 0x31, 0x34, 0x37, 0x39, 0x3c, 0x3f, 0x42, 0x45, 0x49,
0x4c, 0x4f,
            0x52, 0x56, 0x59, 0x5d, 0x61, 0x64, 0x68, 0x6c, 0x70, 0x74,
0x78, 0x7c,
            0x80, 0x85, 0x89, 0x8d, 0x92, 0x96, 0x9b, 0xa0, 0xa4, 0xa9,
0xae, 0xb3,
            0xb8, 0xbd, 0xc2, 0xc7, 0xcc, 0xd2, 0xd7, 0xdd, 0xe2, 0xe8,
0xed, 0xf3,
            0xf9, 0xff, 0x104, 0x10a, 0x110, 0x116, 0x11d, 0x123, 0x129,
0x12f,
            0x136, 0x13c, 0x143, 0x149, 0x150, 0x156, 0x15d, 0x164, 0x16b,
0x172,
            0x179, 0x180, 0x187, 0x18e, 0x195, 0x19c, 0x1a4, 0x1ab, 0x1b3,
0x1ba,
            0x1c2, 0x1c9, 0x1d1, 0x1d8, 0x1e0, 0x1e8, 0x1f0, 0x1f8, 0x200,
0x208,
            0x210, 0x218, 0x220, 0x228, 0x231, 0x239, 0x241, 0x24a, 0x252,
0x25b,
            0x263, 0x26c, 0x275, 0x27d, 0x286, 0x28f, 0x298, 0x2a1, 0x2aa,
0x2b3,
            0x2bc, 0x2c5, 0x2ce, 0x2d7, 0x2e0, 0x2ea, 0x2f3, 0x2fc, 0x306,
0x30f,
            0x319, 0x322, 0x32c, 0x335, 0x33f, 0x349, 0x353, 0x35c, 0x366,
0x370,
            0x37a, 0x384, 0x38e, 0x398, 0x3a2, 0x3ac, 0x3b6, 0x3c0, 0x3ca,
0x3d5,
            0x3df, 0x3e9, 0x3f3, 0x3fe, 0x408, 0x413, 0x41d, 0x428, 0x432,
0x43d,
            0x447, 0x452, 0x45d, 0x467, 0x472, 0x47d, 0x487, 0x492, 0x49d,
0x4a8,
            0x4b3, 0x4be, 0x4c9, 0x4d4, 0x4de, 0x4e9, 0x4f5, 0x500, 0x50b,
0x516,
            0x521, 0x52c, 0x537, 0x542, 0x54e, 0x559, 0x564, 0x56f, 0x57b,
0x586,
            0x591, 0x59d, 0x5a8, 0x5b3, 0x5bf, 0x5ca, 0x5d6, 0x5e1, 0x5ec,
0x5f8,
            0x603, 0x60f, 0x61a, 0x626, 0x631, 0x63d, 0x649, 0x654, 0x660,
0x66b,
            0x677, 0x683, 0x68e, 0x69a, 0x6a5, 0x6b1, 0x6bd, 0x6c8, 0x6d4,
0x6e0,
            0x6eb, 0x6f7, 0x703, 0x70e, 0x71a, 0x726, 0x731, 0x73d, 0x749,
0x755,
            0x760 };

#endif /* SRC_LOOKUP_H_ */
```

Keypad.h

```
/*
 * keypad.h
 *
 *  Created on: Sep 29, 2021
 *      Author: Sereen
 */

#ifndef SRC_KEYPAD_H_
#define SRC_KEYPAD_H_

void keypad_init(void);
uint8_t read_keypad(void);

#define STAR   '*'
#define POUND  '#'
#define NO_KEY 0xFF

// Ports PC4 - PC7
#define ROW1 GPIO_IDR_ID4
#define ROW2 GPIO_IDR_ID5
#define ROW3 GPIO_IDR_ID6
#define ROW4 GPIO_IDR_ID7

#define ROW_PORT GPIOC
#define COL_PORT GPIOC
#define ROW_PORT_IDR (ROW_PORT->IDR)
#define COL_PORT_ODR (COL_PORT->ODR)

// change to PC 0, 1 , 2
#define COL1 GPIO_ODR_OD0
#define COL2 GPIO_ODR_OD1
#define COL3 GPIO_ODR_OD2

#define COL_MASK (COL1|COL2|COL3)
#define ROW_MASK (ROW1|ROW2|ROW3|ROW4)


#endif /* SRC_KEYPAD_H_ */
```

Keypad.c

```
#include "main.h"
#include "keypad.h"

void keypad_init(void)
{
      RCC->AHB2ENR |= (RCC_AHB2ENR_GPIOBEN); // enable GPIOB clock on bus
      RCC->AHB2ENR |= (RCC_AHB2ENR_GPIOCEN); // enable GPIOC clock on bus

   // clear GPIOB PA4-PA7 (also sets them for input
      ROW_PORT->MODER &= ~( GPIO_MODER_MODE4
                              | GPIO_MODER_MODE5
                              | GPIO_MODER_MODE6
                              | GPIO_MODER_MODE7);
      // clear MODE PC5 - PC7  bits for keypad and use as columns
      COL_PORT->MODER &= ~(GPIO_MODER_MODE0
                              | GPIO_MODER_MODE1
                              | GPIO_MODER_MODE2);

      // set PC5-PC7 as outputs for columns
      COL_PORT->MODER |= ( (1 << GPIO_MODER_MODE0_Pos)
                          | (1 << GPIO_MODER_MODE1_Pos)
                          | (1 << GPIO_MODER_MODE2_Pos) );

      // enable pulldown resistor for rows on PA4-7

      // clear pupdr
      ROW_PORT->PUPDR &=  ~(    GPIO_PUPDR_PUPD4_1
                              | GPIO_PUPDR_PUPD5_1
                              | GPIO_PUPDR_PUPD6_1
                              | GPIO_PUPDR_PUPD7_1);


      ROW_PORT->PUPDR |=  (    GPIO_PUPDR_PUPD4_1
                              | GPIO_PUPDR_PUPD5_1
                              | GPIO_PUPDR_PUPD6_1
                              | GPIO_PUPDR_PUPD7_1);

      // enable push-pull for columns on PC5-PC7
```

```
	// (check later if there is problems)
	COL_PORT->OTYPER &= ~(  GPIO_OTYPER_OT0
	                      | GPIO_OTYPER_OT1
	                      | GPIO_OTYPER_OT2);


	// set slow speed for columns (PC5-PC7)

	COL_PORT->OSPEEDR &= ~(GPIO_OSPEEDR_OSPEED0
	                     | GPIO_OSPEEDR_OSPEED1
	                     | GPIO_OSPEEDR_OSPEED2); // PA0
slow speed

	// set columns to high
	COL_PORT_ODR |= COL_MASK;

}


uint8_t calculate_key(uint16_t col, uint8_t row)
{
	uint8_t key;
	uint8_t rows;

	rows = row >> 4; // right shift rows 4 places for easiers calc

	if(rows == 4)
		rows = 3;
	if(rows == 8)
		rows = 4;
	// calculate key based on col
	switch(col)
	{
		case 0:
			key = (3 * rows) - 2;
			break;
		case 1:
			key = (3 * rows) - 1;
			break;
		case 2:
			key = (3 * rows);
			break;
	}


	if(key == 10)   // leave as 10 so it can output to LED (change with
LCD)
		key  = '*'; // leave it in ASCII for later use
	if(key == 11)
		key = 0;
	if(key == 12)  // leave as 12 so so it can output to LED (change with
LCD)
		key = '#';

	return key;
}

void check_columns(uint8_t cur_col)
{
	COL_PORT_ODR  &= ~(COL_MASK); // turn all colums off
	switch(cur_col)  // set a col high depending on the row
	{
		case 0:
			COL_PORT_ODR |= COL1;
			break;
		case 1:
			COL_PORT_ODR |= COL2;
			break;
		case 2:
			COL_PORT_ODR |= COL3;
			break;

	}
}


uint8_t read_keypad(void)
{
	uint8_t rows;
	uint8_t cur_col ;
	uint8_t key;
	// Read the rows PB4-PB7 only
	rows = ROW_PORT_IDR & ROW_MASK ;
	if(rows == 0) // check to see if all the rows are low (is so return
NO KEY)
		return NO_KEY;

	for(cur_col = 0; cur_col < 3; cur_col++)
	{

		// set current columns high others low
		check_columns(cur_col);
		// read the rows
	    rows = ROW_PORT_IDR & ROW_MASK;

		if(rows != 0)
		{
			// calculate button from row and col
			key = calculate_key(cur_col, rows);
			// set columns to high
			COL_PORT_ODR |= COL_MASK;
			return key;
		}

	}

	// set columns to high
	COL_PORT_ODR |= COL_MASK;

	return NO_KEY;


}
```

<u>DAC.h</u>

```
/*
 * DAC.h
 *
 *  Created on: Oct 27, 2021
 *      Author: Sereen
 */

#ifndef SRC_DAC_H_
#define SRC_DAC_H_

void DAC_init(void);
void DAC_write(uint16_t);
void DAC_volt_conv(uint16_t);

#define SHDN 0x2000     // Bit 12
#define GAIN 0x1000            // Bit 13
#define DAC_PORT GPIOA
#define DAC_RES 4095
#define VREF 3.3

#endif /* SRC_DAC_H_ */
```

DAC.c

```
/*
 * DAC.c
 *
 *  Created on: Oct 27, 2021
 *      Author: Sereen
 */

#include "main.h"
#include "DAC.h"


void DAC_GPIO_config()
{
      // PA4 = SPI1_CS, PA5 = SPI1_SCK, PA7 = SPI1_MOSI
      RCC->AHB2ENR |= (RCC_AHB2ENR_GPIOAEN);    // IO port A clock enable

      // Clear MODE bits for PA4-PA7
      DAC_PORT->MODER &= ~(GPIO_MODER_MODE4
                        | GPIO_MODER_MODE5
                        | GPIO_MODER_MODE7);

      DAC_PORT->MODER |= (GPIO_MODER_MODE4_1
                        |GPIO_MODER_MODE5_1
                        | GPIO_MODER_MODE7_1);  // alternate function
mode


      DAC_PORT->OTYPER &= ~(GPIO_OTYPER_OT4
                           | GPIO_OTYPER_OT5
                           | GPIO_OTYPER_OT7);  // Push-pull output

      DAC_PORT->PUPDR &= ~(GPIO_PUPDR_PUPD4
                               | GPIO_PUPDR_PUPD5
                               | GPIO_PUPDR_PUPD7);   // no pu/pd
resistor

      DAC_PORT->OSPEEDR |= (GPIO_OSPEEDR_OSPEED4
                              | GPIO_OSPEEDR_OSPEED5
```

```
                                  | GPIO_OSPEEDR_OSPEED7);      // high
speed

      DAC_PORT->AFR[0] |= (5 << GPIO_AFRL_AFSEL4_Pos
                            | 5 << GPIO_AFRL_AFSEL5_Pos
                            | 5 << GPIO_AFRL_AFSEL7_Pos);
}


void DAC_init(void)
{

      DAC_GPIO_config();

      RCC->APB2ENR |= (RCC_APB2ENR_SPI1EN);     //enable SP1

      SPI1 -> CR1 = (SPI_CR1_MSTR); // enable Master bit

      SPI1 ->CR2 = ( SPI_CR2_DS  // enable 16 bit DS mode
                            | SPI_CR2_NSSP); // NSSP mode

      SPI1 -> CR1 |= (SPI_CR1_SPE); // Enable SPI
}


void DAC_write(uint16_t data)
{
      while(!(SPI1->SR & SPI_SR_TXE));    // Check to Make Sure Buffer is
Empty
      SPI1->DR  = (SHDN | GAIN| data);         // enable HIGH and SHDN bits
      while(!(SPI1->SR & SPI_SR_RXNE));   // Wait for RXIFG to be Set
(RXBUF Empty)

}

// converts milliVolts taken in to be converted into count for DAC
void DAC_volt_conv(uint16_t mVolt)
{
    int DAC_count;
    DAC_count = (mVolt * DAC_RES)/ (VREF * 1000);
    DAC_write(DAC_count);
}
```