Problem 1:

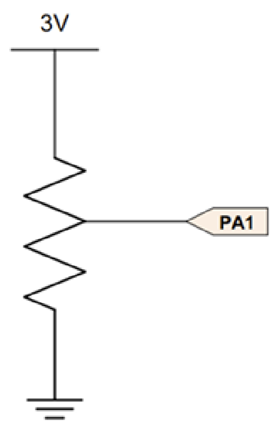


Fig: Measurement of voltage from a potentiometer-based divider at pin **PA1**.

Use the ADC of STM32 board to control the brightness of an LED connected at pin5. Now write a code such that the brightness of the on-board green LED (LD2, connected to PA5) can be controlled by turning the potentiometer. . It is important to note that our ADC has a resolution of 12 bits and gives us a result ranging from 0 (Input voltage = 0) and 4095 (Input voltage = Vcc).

Solution:

```
#include "stm32f446xx.h"
#include "stdio.h"
#define LED_PIN     5
#define PUSH_BUTTON 13

#define VECT_TAB_OFFSET  0x00
static void enable_HSI(){

        RCC->CR |= ((uint32_t)RCC_CR_HSION);
        while ((RCC->CR & RCC_CR_HSIRDY) == 0); // Wait until HSI ready

        // Store calibration value
        PWR->CR |= (uint32_t)(16 << 3);

        // Reset CFGR register
        RCC->CFGR = 0x00000000;

        // Reset HSEON, CSSON and PLLON bits
        RCC->CR &= ~(RCC_CR_HSEON | RCC_CR_CSSON | RCC_CR_PLLON);
        while ((RCC->CR & RCC_CR_PLLRDY) != 0); // Wait until PLL disabled
```

```
	RCC->PLLCFGR = 0;
	RCC->PLLCFGR &= ~(RCC_PLLCFGR_PLLSRC); // PLLSRC = 0 (HSI 16 Mhz clock
//selected as clock source)
	RCC->PLLCFGR |= 16 << RCC_PLLCFGR_PLLN_Pos;  // PLLM = 16, VCO input
	RCC->PLLCFGR |= 336 << RCC_PLLCFGR_PLLN_Pos;		// PLLN = 336, VCO
	RCC->PLLCFGR |= 4 << RCC_PLLCFGR_PLLP_Pos;	// PLLP = 4, PLLCLK = 336
	RCC->PLLCFGR |= 7 << RCC_PLLCFGR_PLLQ_Pos;	// PLLQ = 7, USB Clock =

	// Enable Main PLL Clock
	RCC->CR |= RCC_CR_PLLON;
	while ((RCC->CR & RCC_CR_PLLRDY) == 0);  // Wait until PLL ready
	FLASH->ACR |= FLASH_ACR_ICEN | FLASH_ACR_PRFTEN |
	FLASH_ACR_LATENCY_2WS;
	RCC->CFGR &= ~RCC_CFGR_HPRE; // 84 MHz, not divided

	// PPRE1: APB Low speed prescaler (APB1)
	RCC->CFGR &= ~RCC_CFGR_PPRE1;
	RCC->CFGR |= RCC_CFGR_PPRE1_DIV2; // 42 MHz, divided by 2
	// PPRE2: APB high-speed prescaler (APB2)
	RCC->CFGR &= ~RCC_CFGR_PPRE2; // 84 MHz, not divided

	// 01: HSE oscillator selected as system clock
	// 10: PLL selected as system clock
	RCC->CFGR &= ~RCC_CFGR_SW;
	RCC->CFGR |= RCC_CFGR_SW_1;
	// while ((RCC->CFGR & RCC_CFGR_SWS_PLL) != RCC_CFGR_SWS_PLL);

	// Configure the Vector Table location add offset address
//	VECT_TAB_OFFSET  = 0x00UL; // Vector Table base offset field.
			// This value must be a multiple of 0x200.
	SCB->VTOR = FLASH_BASE | VECT_TAB_OFFSET; // Vector Tab

}


static void configure_ADC(){
  // Enable the clock to GPIO Port A
  RCC->AHB1ENR |= RCC_AHB1ENR_GPIOAEN;

	// Set mode as Alternative Function 1
	GPIOA->MODER	&= ~(0x03 << (2*LED_PIN));			// Clear bits
	GPIOA->MODER	|=   0x02 << (2*LED_PIN);		// Input(00),
//Output(01), //AlterFunc(10), Analog(11)
```

```
	// GPIO Speed: Low speed (00), Medium speed (01), Fast speed (10), High speed (11)
	GPIOA->OSPEEDR &= ~(3<<(2*LED_PIN));
	GPIOA->OSPEEDR |=   2<<(2*LED_PIN);  // Fast speed

	// GPIO Output Type: Output push-pull (0, reset), Output open drain (1)
 GPIOA->OTYPER &= ~(1<<LED_PIN);      // Push-pull
	//GPIOA->OTYPER |= ((unsigned int)1<<LED_PIN);

	// GPIO Push-Pull: No pull-up, pull-down (00), Pull-up (01), Pull-down (10), Reserve
	GPIOA->PUPDR  &= ~(3<<(2*LED_PIN));  // No pull-up, no pull-down

	//configure PA1 as analog pin (MODER[1:0] = 11)
	GPIOA->MODER |= 0xC;

	/*setup ADC1*/
	RCC->APB2ENR |= 0x00000100; /* enable ADC1 clock */

	ADC1->CR2 = 0; /* SW trigger */

	ADC1->SQR3 = 1; /* conversion sequence starts at ch 1 */

	ADC1->SQR1 = 0; /* conversion sequence length 1 */
	ADC1->CR2 |= 1; /* enable ADC1 */

}
static void TIM2_CH1_Init(){

   RCC->APB1ENR              |= RCC_APB1ENR_TIM2EN;                 // Enable TIMER clock

		// Counting direction: 0 = up-counting, 1 = down-counting
		TIM2->CR1 &= ~TIM_CR1_DIR;

   TIM2->PSC = 16;       // Prescaler = 23
   TIM2->ARR = 1000-1;  // Auto-reload: Upcouting (0..ARR), Downcouting (ARR..0)
TIM2->CCMR1 &= ~TIM_CCMR1_OC1M;  // Clear ouput compare mode bits for channel 1
   TIM2->CCMR1 |= TIM_CCMR1_OC1M_1 | TIM_CCMR1_OC1M_2; // OC1M = 110 for
//PWM Mode 1 output on ch1
   TIM2->CCMR1 |= TIM_CCMR1_OC1PE;                    // Output 1 preload enable

		// Select output polarity: 0 = active high, 1 = active low
		TIM2->CCMR1 &= ~TIM_CCER_CC1P; // select active high
```

```
    // Enable output for ch1
                TIM2->CCER |= TIM_CCER_CC1E;

    // Main output enable (MOE): 0 = Disable, 1 = Enable
                TIM2->BDTR |= TIM_BDTR_MOE;

                TIM2->CCR1  = 500;         // Output Compare Register for channel 1
                TIM2->CR1  |= TIM_CR1_CEN; // Enable counter
}


static void LED_Pin_Init(){
        RCC->AHB1ENR    |= RCC_AHB1ENR_GPIOAEN;           // Enable GPIOA clock

        // Set mode as Alternative Function 1
                GPIOA->MODER      &= ~(0x03 << (2*LED_PIN));   // Clear bits
                GPIOA->MODER    |=   0x02 << (2*LED_PIN);      // Input(00), Output(01),
//AlterFunc(10), Analog(11)

                GPIOA->AFR[0] &= ~(0xF << (4*LED_PIN));         //     AF 1 = TIM2_CH1
                GPIOA->AFR[0]      |=   0x1 << (4*LED_PIN);          //     AF 1 = TIM2_CH1

                //Set I/O output speed value as very high speed
                GPIOA->OSPEEDR  &= ~(0x03<<(2*LED_PIN));
       // Speed mask
                GPIOA->OSPEEDR  |=   0x03<<(2*LED_PIN);
                GPIOA->PUPDR     &= ~(0x03<<(2*LED_PIN));                              // No
                //Set I/O as push pull
        //LED_PORT->OTYPER   &=  ~(1<<LED_PIN) ; // Push-Pull(0, reset), Open-Drain(1)
}

static void turn_on_LED(){
       GPIOA->ODR |= ((unsigned int)1) << LED_PIN;
}

static void turn_off_LED(){
       GPIOA->ODR &= ~((unsigned int)1 << LED_PIN);
}

static void toggle_LED(){
       GPIOA->ODR ^= (1 << LED_PIN);
}
```

```
int main(void){
	int i;
	uint32_t result;
	uint32_t brightness = 0;
	enable_HSI();
	configure_ADC();
	LED_Pin_Init();
	TIM2_CH1_Init(); // Timer to control LED

  // Dead loop & program hangs here
	while(1)
		{

		ADC1->CR2 |= 0x40000000; /* start a conversion */

		while(!(ADC1->SR & 2));  /* wait for conv complete */
		result = ADC1->DR; /* read conversion result */
		brightness = result * 199 / (4096);
		TIM2->CCR1 = brightness;

		for(i=0; i<1000; i++);

	}
}
```

Problem 2 (a):

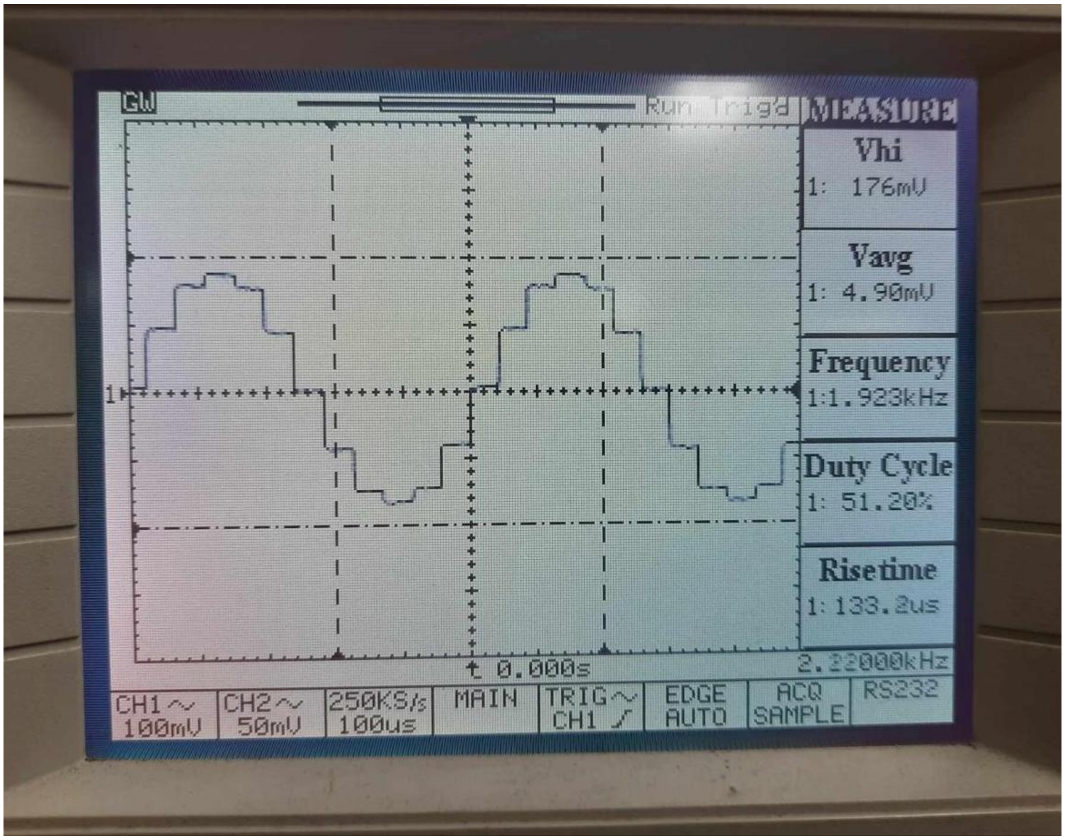
Use the on-board DAC to produce a sine wave that is continuous in time but can be discrete in amplitude. The different voltage levels will correspond to different values loaded into the DAC. The time for which the output stays constant is determined by the delay before the next conversion. Determine the fundamental frequency of the sine wave that you observe and devise a method to change this frequency of the apparent sinusoid by modifying the code.

Solution:

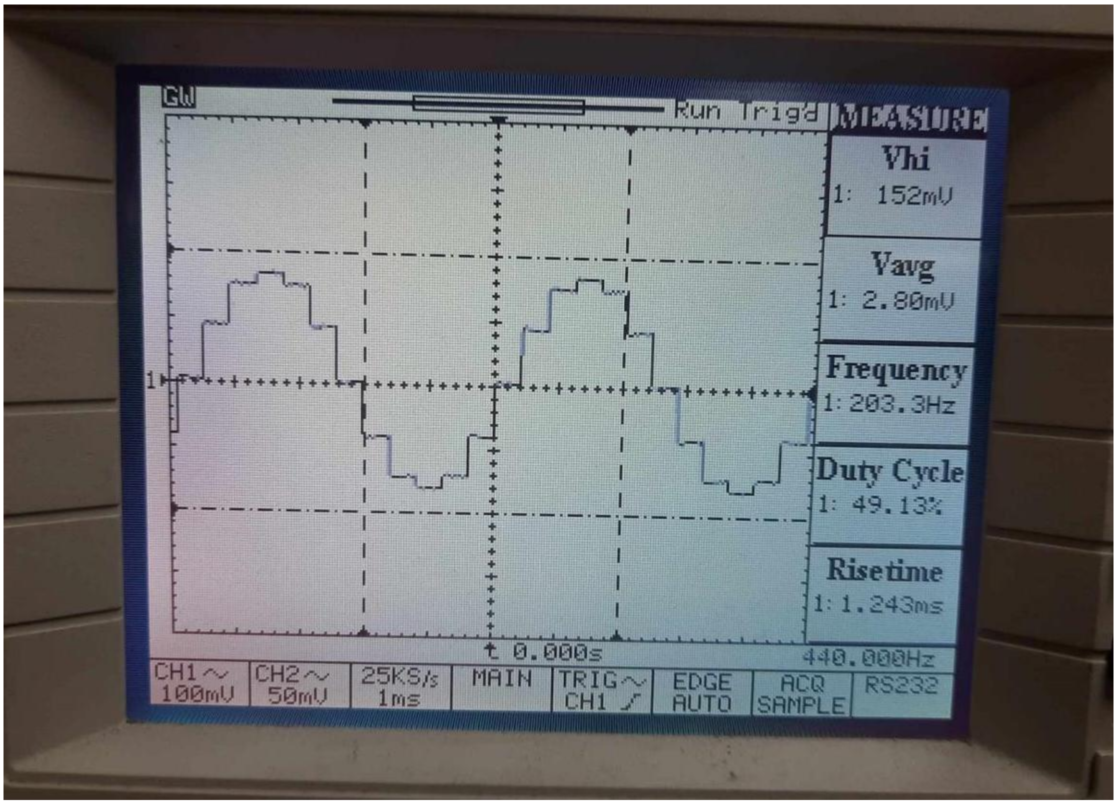
```
#include "stm32f446xx.h"
#include "stdio.h"

#define LED_PIN    4
void delayUs(int n);

static void configure_pin(){
  // Enable the clock to GPIO Port A
  RCC->AHB1ENR |= RCC_AHB1ENR_GPIOAEN;

        //configure PA4 as analog pin (MODER[1:0] = 11)
        GPIOA->MODER |= 0x00000300; /* PA4 analog */

        /* setup DAC */
        RCC->APB1ENR |= 1 << 29; /* enable DAC clock */
        DAC->CR |= 1; /* enable DAC */

}

void delayUs(int n)
{
int i;
for (; n > 0; n--)
for (i = 0; i < 3; i++) ;
}

int main(void){
        int i;
  int points = 12;
const static int sinewave[12]={2048,3071,3821,4095,3821,3071,2048,1024,274,0,274,1024};
        static int TriWave[12]={0,341,683,1023,1365,1706,1706,1365,1023,683,341,0};
        static int SawWave[12]={0,341,683,1023,1365,1706,2048,2389,2730,3071,3412,3754};
        configure_pin();
```

```
 // Dead loop & program hangs here
        while(1){

                for (i = 0; i < sizeof(sineWave)/sizeof(int); i++)
                {
                        DAC->DHR12R1 = sineWave[i];
                        delayUs(100);
                }
        }
}
/*for (i = 0; i<points/2; i++){
        TriWave[i] = 4095/points * i;
        TriWave[points - 1 - i] = 4095/points * i;
}*/

        /*for (i = 0; i < points; i++){
        SawWave[i] = 4095/points * i;
}*/
```

Oscilloscope output1 (Sine wave with fundamental frequency=2.2kHz):



Oscilloscope output2 (Sine wave with fundamental frequency=400Hz almost 10fold reduced):



Changing the argument go delayUs(int n), we can devise a method to change the frequency of the sinusoid.

delayUs(10) introduces $10\mu s$ of delay, thus total 12 samples for one period of sinusoid accumulates total
$120\mu s$ of delay.
Thus, frequency is $1/(n \times number\ sample\ per\ period) = 2.33kHz$.

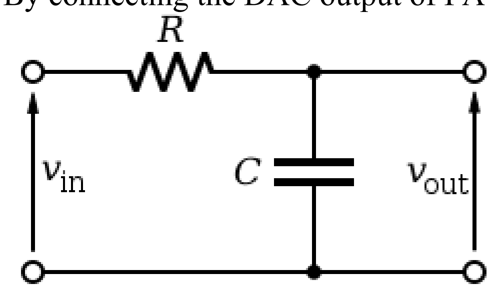
When we set delayUs(100) of 100us delay,
Frequency becomes 440 Hz, almost 10 times reduced but not 230 Hz because of lack of calibration of the oscilloscope.

Problem 2 (b):

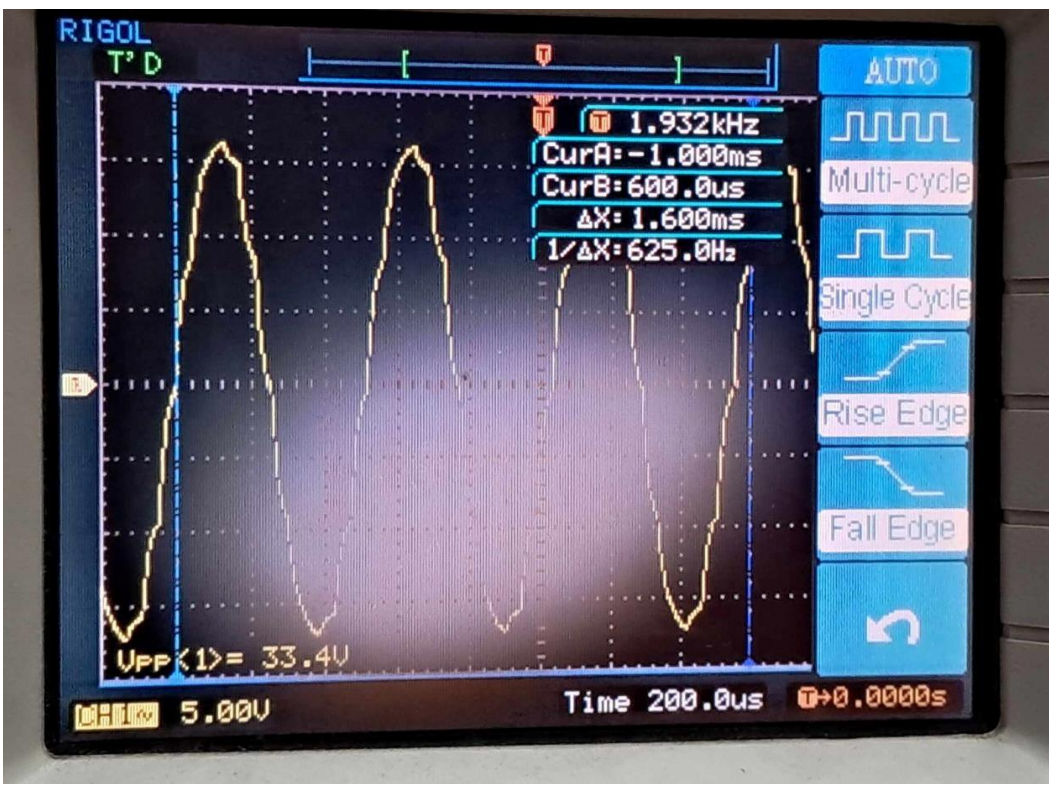
Construct a simple low-pass filter circuit to obtain a smoother sine wave from the staircase-like sine wave (DAC output). Use resistors/capacitors/inductors as required. Construct a simple low-pass filter circuit to obtain a smoother sine wave from the staircase-like sine wave (DAC output). Use resistors/capacitors/inductors as required.

Solution:

By connecting the DAC output of PA4 to the RC low pass filter, we have smooth output:



Oscilloscope output3 (Smooth Sine Wave)



Problem 3

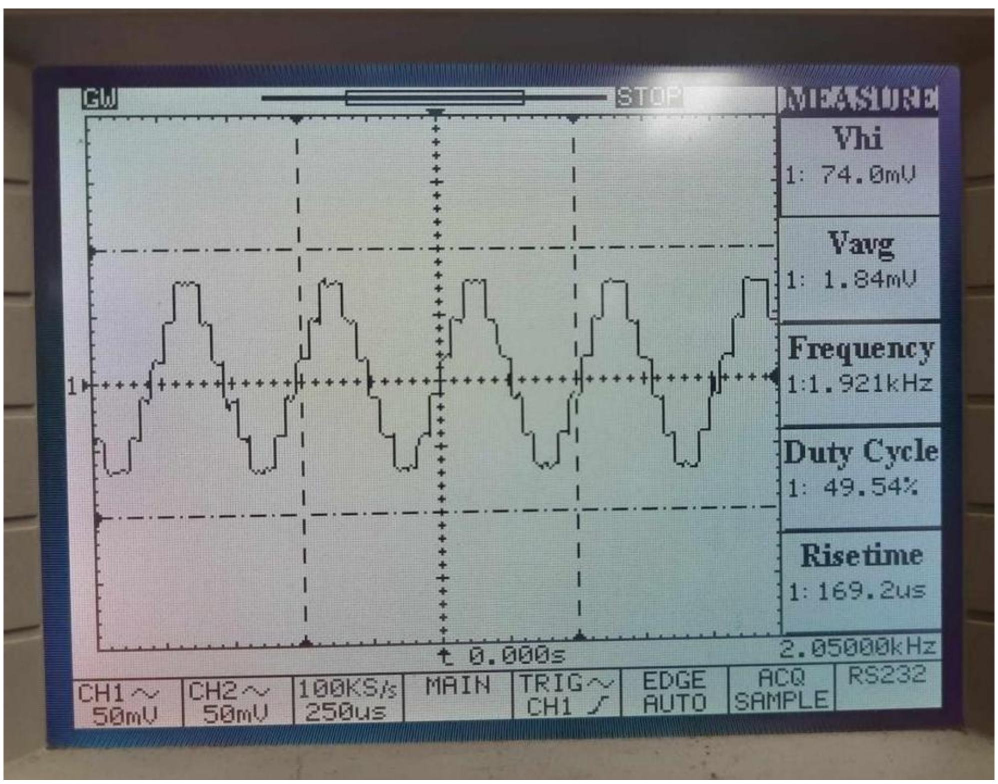
Update the DAC code that will generate the following analog signals:
(a) triangular wave, (b) saw- tooth wave, (c) a full-wave rectified sine wave, (d) Random noise.

Solution

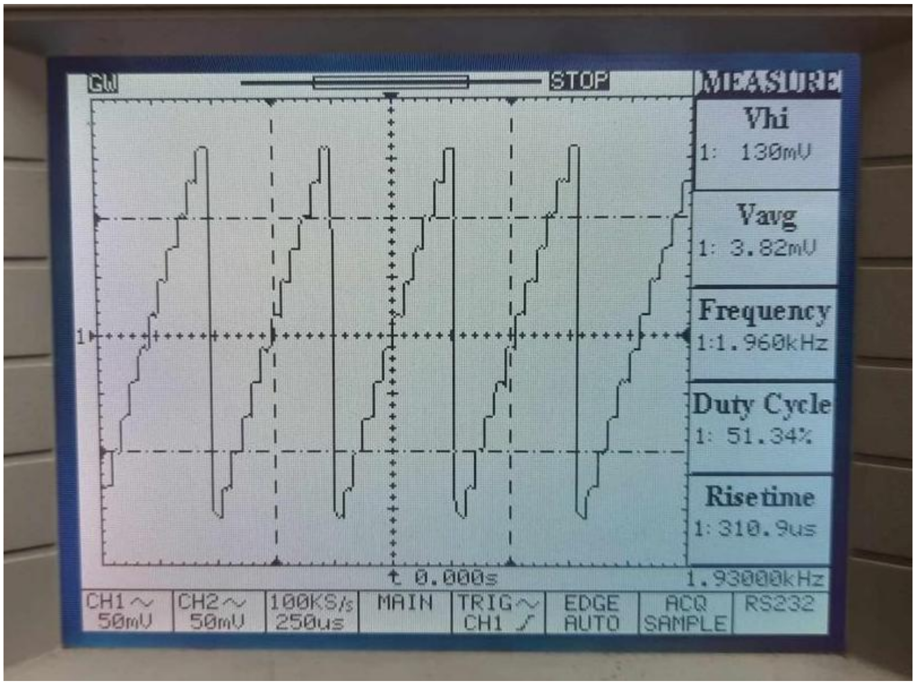
```
#include "stm32f446xx.h"
#include "stdio.h"

#define LED_PIN    4
void delayUs(int n);

static void configure_pin(){
  // Enable the clock to GPIO Port A
  RCC->AHB1ENR |= RCC_AHB1ENR_GPIOAEN;

        //configure PA4 as analog pin (MODER[1:0] = 11)
        GPIOA->MODER |= 0x00000300; /* PA4 analog */

        /* setup DAC */
        RCC->APB1ENR |= 1 << 29; /* enable DAC clock */
        DAC->CR |= 1; /* enable DAC */

}

void delayUs(int n)
{
int i;
for (; n > 0; n--)
for (i = 0; i < 3; i++) ;
}
```

```
int main(void){
        int i;
  int points = 12;
const static int sinewave[12]={2048,3071,3821,4095,3821,3071,2048,1024,274,0,274,1024};
        static int TriWave[12]={0,341,683,1023,1365,1706,1706,1365,1023,683,341,0};
        static int SawWave[12]={0,341,683,1023,1365,1706,2048,2389,2730,3071,3412,3754};
        static int Random[12]={0,341,68,102.3,365,1706,20,238,2730,301,3412,4095};
        configure_pin();

  // Dead loop & program hangs here
        while(1){

                for (i = 0; i < sizeof(sineWave)/sizeof(int); i++)
                {
                        DAC->DHR12R1 = sineWave[i];
                        delayUs(100);
                }
        }
}
```

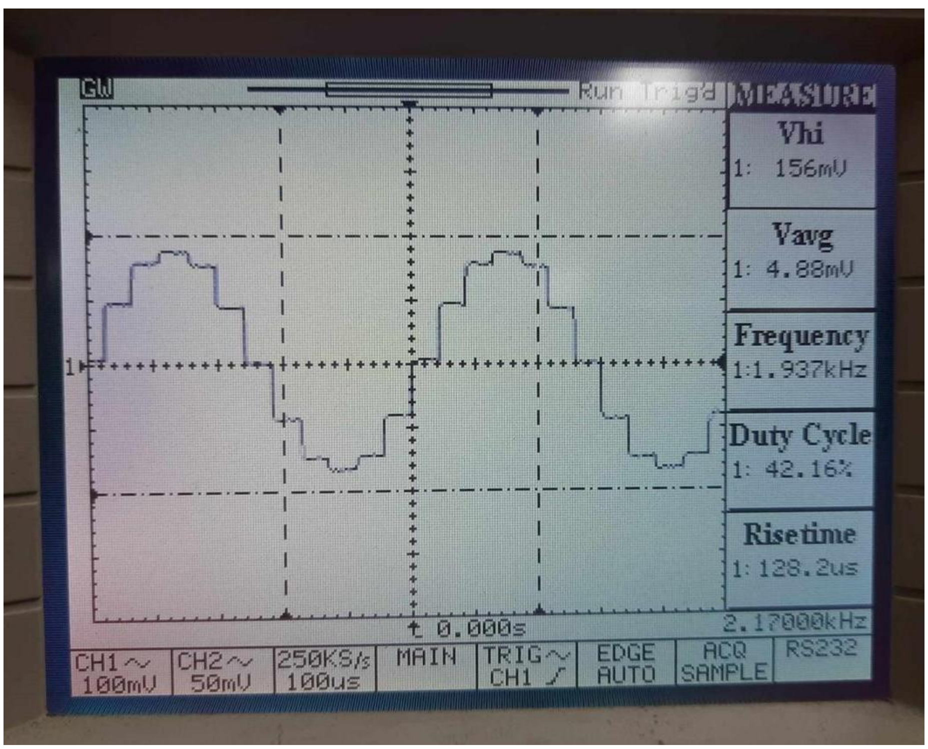
Oscilloscope output1 (Triangular Wave)



Oscilloscope output2 (Sawtooth wave)



Oscilloscope output3 (Full Wave Rectified Sine Wave)



As this sine wave has no negative component, that's why it can be considered as a full wave rectified sine wave. Of course, for better output the downward sine portion should be flipped.