C.V RAMAN GLOBAL UNIVERSITY

BHUBANESWER, ODISHA

SUBJECT:- EMBEDDED SYSTEM

SUBMITED BY:-

NAME	REGISTRATION NO.
NITISH PARAMANIK	2201020433
JAIJEET PAUL	2201020357

Project-Report

Problem Statement:

Implement signal generator system.

1. Define lookup tables for various waveforms.

2. Use interrupt-based push button as user input to change wave shape.

3. Also provide option to change frequency using another external push button.

4. Use LEDs as indicators of operations.

Algorithm:

All logical steps involved in implementing the project.

I. Initialization:

Enable the clocks for GPIOA, GPIOC, and DAC.

Configure GPIO pins:

- Set PA4 to analog mode.
- Set PA5 and PA6 to output mode (for LED indicators).
- Set PA7 and PC13 to input mode with pull-up resistors (for buttons).

Initialize the DAC (Channel 1):

- Enable DAC and disable its buffer.
- Set DAC trigger to software trigger.
- Reset DAC value.

Initialize EXTI for PC13 and PA7:

- Enable SYSCFG clock.
- Configure EXTI13 to use PC13 and EXTI7 to use PA7.
- Unmask EXTI13 and EXTI7.
- Set falling edge trigger for both EXTI13 and EXTI7.
- Enable interrupts for EXTI13 and EXTI7 in the NVIC.

II. Interrupt Handlers:

EXTI15_10_IRQHandler:

- Check if EXTI13 triggered.
- Clear pending bit for EXTI13.
- Increment waveform_mode.
- Reset waveform_mode if it exceeds 2.
- Toggle PA5 to indicate waveform change.

EXTI9_5_IRQHandler:

- Check if EXTI7 triggered.
- Clear pending bit for EXTI7.
- Increase frequency by 10,000.
- Reset frequency to 50,000 if it exceeds 200,000.
- Toggle PA6 to indicate frequency change.

III. Waveform Generation Functions:

SineWave:

- Calculate DAC value using sine function.
- Load calculated value into DAC.
- Trigger DAC.
- Increment angle for sine calculation.
- Reset angle if it reaches 360 degrees.
- Call Delay function with frequency.

SquareWave:

- Calculate DAC value based on step (high or low).
- Load calculated value into DAC.
- Trigger DAC.
- Increment step.
- Reset step if it reaches 4096.
- Call Delay function with frequency.

TriangleWave:

- Calculate DAC value based on step (rising or falling).
- Load calculated value into DAC.
- Trigger DAC.
- Increment step.
- Reset step if it reaches 4096.
- Call Delay function with frequency.

IV. Delay Function:

- Configure SysTick for the specified delay.
- Start SysTick.
- Wait for the COUNTFLAG to be set.
- Stop SysTick.

V. Main Loop:

- Initialize GPIO, DAC, and EXTI.
- Initialize SysTick for 1 ms tick.
- Enter infinite loop:

Generate waveform based on waveform_mode:

- SineWave if waveform_mode is 0.
- SquareWave if waveform_mode is 1.
- TriangleWave if waveform_mode is 2.

Connection details:

Sl No	Peripheral	Port & Pin No	Type	Description	Mode
1	LED1	PA5	OUTPUT	Used to drive LEDs to indicate changes in waveform mode.	OUTPUT
2	LED2	PA6	OUTPUT	Used to drive LEDs to indicate changes in waveform frequency.	OUTPUT
3	ON-BOARD PUSH BUTTON	PC13	INPUT	Button to change the waveform type (sine, square, triangle).	INPUT
4	EXTERNAL PUSH BUTTON	PA7	INPUT	Button to change the frequency of the waveform	INPUT
5	DAC (Digital-to-Analog Converter)	PA4		Converts digital values to analog signals to generate the desired waveform.	ANALOG INPUT
6	EXTI (External Interrupt)	PC13 & PA7		Triggers an interrupt when the button on PC13 is pressed to change the waveform type. Triggers an interrupt when the button on PA7 is pressed to change the frequency.	EXTERNAL INTERRUPT ON FALLING EDGE
7	SysTick TIMER			Provides a delay mechanism to control the timing of waveform generation.	TIMER

Source Code:

1)Main.c

```c
/*
 * This file initializes the GPIO, DAC, and EXTI peripherals. It contains the main loop
 * which continuously generates different waveforms based on the current mode. The mode
 * and frequency of the waveforms can be changed using external push buttons.
 *
 * - PA4: DAC output
 * - PC13: Change waveform mode (Sine, Square, Triangle)
 * - PA7: Change waveform frequency
 * - PA5: LED indicator for waveform mode change
 * - PA6: LED indicator for frequency change
 */
#include "stm32f446xx.h"
#include "gpio.h"
#include "waveforms.h"
#include "sysTick.h"

// Define SystemCoreClock for this example
#define SystemCoreClock 180000000U  // 180 MHz

extern uint8_t waveform_mode;
extern uint32_t frequency;

int main() {
    // Initialize GPIO, DAC, and EXTI
    GPIO_Init();

    // Initialize SysTick timer
    SysTick_Init(SystemCoreClock / 1000); // 1 ms tick

    // Main loop
    while (1) {
        switch (waveform_mode) {
            case 0: SineWave(); break;
            case 1: SquareWave(); break;
            case 2: TriangleWave(); break;
        }
    }
```

2)gpio.c

```c
/*
 * gpio.c
 *
 *  Created on: Jul 30, 2024
 *      Author: Nitish Paramanik
 */

/**
 * @file gpio.c
 * @brief GPIO and EXTI initialization and interrupt handling.
 *
 * This file initializes the GPIO pins for DAC output, push buttons, and LEDs. It also
 * sets up the EXTI (External Interrupt) for the push buttons to handle waveform mode
 * and frequency changes. The interrupt handlers are implemented to toggle the LEDs
 * and update the mode or frequency accordingly.
 *
 * - GPIOA: PA4 (DAC output), PA5 (LED for mode change), PA6 (LED for frequency change), PA7 (Frequency change button)
 * - GPIOC: PC13 (Waveform mode change button)
 */
#include "stm32f446xx.h"
#include "gpio.h"

// External variables to control the waveform mode and frequency
uint8_t waveform_mode = 0;
uint32_t frequency = 50000; // Base delay for waveform generation

void GPIO_Init() {
    // Enable GPIOA and GPIOC clock
    RCC->AHB1ENR |= (1 << 0) | (1 << 2);

    // Set PA4 to Analog mode
    GPIOA->MODER &= ~(3 << 8);  // Reset PA4
    GPIOA->MODER |= (3 << 8);   // Set PA4 to Analog mode

    // Set PA5 and PA6 to output mode (LED indicators)
    GPIOA->MODER |= (1 << (5 * 2)) | (1 << (6 * 2));
```

```
    // Set PA7 to input mode (Frequency change button)
    GPIOA->MODER &= ~(3 << (7 * 2));
    GPIOA->PUPDR &= ~(3 << (7 * 2));   // Reset pull-up/pull-down
    GPIOA->PUPDR |= (1 << (7 * 2));    // Set pull-up for PA7

    // Set PC13 to input mode (Waveform change button)
    GPIOC->MODER &= ~(3 << (13 * 2));
    GPIOC->PUPDR &= ~(3 << (13 * 2));   // Reset pull-up/pull-down
    GPIOC->PUPDR |= (1 << (13 * 2));    // Set pull-up for PC13

    // Enable DAC clock
    RCC->APB1ENR |= (1 << 29);

    // Configure DAC Channel 1
    DAC->CR |= (1 << 0);   // Enable DAC
    DAC->CR &= ~(1 << 1); // Disable DAC buffer
    DAC->CR |= (7 << 3);   // Set Trigger to Software trigger
    DAC->DHR12R1 = 0;      // Reset DAC value

    // Initialize EXTI for PC13 and PA7
    EXTI_Init();
}

void EXTI_Init() {
    // Enable SYSCFG clock
    RCC->APB2ENR |= (1 << 14);

    // Configure EXTI13 for PC13 (Waveform change button)
    SYSCFG->EXTICR[3] &= ~(0xF << (13 % 4 * 4)); // Reset EXTI13 configuration
    SYSCFG->EXTICR[3] |= (2 << (13 % 4 * 4));    // Set EXTI13 to use PC13

    // Configure EXTI7 for PA7 (Frequency change button)
    SYSCFG->EXTICR[1] &= ~(0xF << (7 % 4 * 4)); // Reset EXTI7 configuration
    SYSCFG->EXTICR[1] |= (0 << (7 % 4 * 4));    // Set EXTI7 to use PA7

    // Configure EXTI lines
    EXTI->IMR |= (1 << 13) | (1 << 7); // Unmask EXTI13 and EXTI7
    EXTI->FTSR |= (1 << 13) | (1 << 7); // Falling edge trigger

    // Enable EXTI line interrupts
    NVIC_EnableIRQ(EXTI15_10_IRQn); // For PC13
    NVIC_EnableIRQ(EXTI9_5_IRQn);   // For PA7
}

void EXTI15_10_IRQHandler(void) {
    if (EXTI->PR & (1 << 13)) {  // Check if EXTI13 triggered
        EXTI->PR |= (1 << 13);   // Clear pending bit

        // Change waveform mode
        waveform_mode++;
        if (waveform_mode > 2) {
            waveform_mode = 0;
        }

        // Update LED indicators
        GPIOA->ODR ^= (1 << 5); // Toggle PA5 to indicate waveform change
    }
}

void EXTI9_5_IRQHandler(void) {
    if (EXTI->PR & (1 << 7)) {  // Check if EXTI7 triggered
        EXTI->PR |= (1 << 7);    // Clear pending bit

        // Change frequency (adjust the base delay for waveform generation)
        frequency += 10000; // Increase frequency
        if (frequency > 200000) { // Reset frequency after reaching maximum
            frequency = 50000;
        }

        // Update LED indicators
        GPIOA->ODR ^= (1 << 6); // Toggle PA6 to indicate frequency change
    }
```

3)waveforms.c

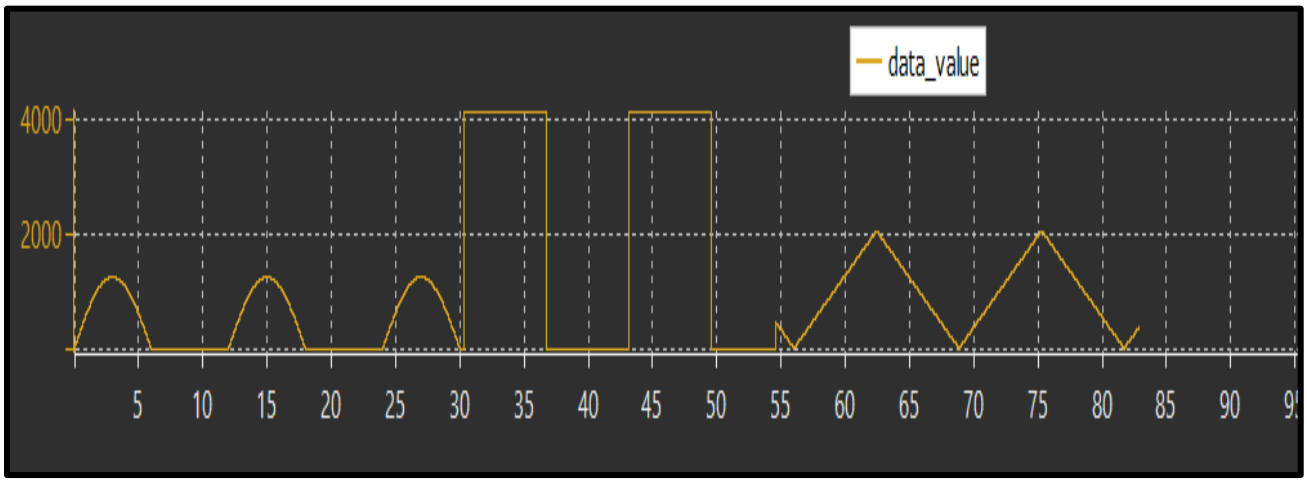
```c
/**
 * @file waveforms.c
 * @brief Waveform generation functions.
 *
 * This file contains functions to generate different waveforms using the DAC.
 * The available waveforms are Sine, Square, and Triangle. These functions use
 * a global delay variable to control the frequency of the waveforms.
 *
 * - SineWave: Generates a sine waveform.
 * - SquareWave: Generates a square waveform.
 * - TriangleWave: Generates a triangle waveform.
 */
#include "stm32f446xx.h"
#include "waveforms.h"
#include <math.h>
#include "sysTick.h"

// Define global variables for waveform generation
float val = 0.0;
uint32_t step = 0;
uint32_t data_value = 0;

extern uint32_t frequency;

void SineWave() {
    data_value = (uint32_t)(sin(val * M_PI / 180.0) * (4095.0 / 3.3));
    DAC->DHR12R1 = data_value;
    DAC->SWTRIGR |= (1 << 0);

    val += 0.1;
    if (val >= 360.0) {
        val = 0.0;
    }

    Delay(frequency);
}

void SquareWave() {
    data_value = (step < 2048) ? 4095 : 0;
    DAC->DHR12R1 = data_value;
    DAC->SWTRIGR |= (1 << 0);

    step++;
    if (step >= 4096) {
        step = 0;
    }

    Delay(frequency);
}

void TriangleWave() {
    data_value = (step < 2048) ? (step * (4095 / 2048)) : ((4095 - step) * (4095 / 2048));
    DAC->DHR12R1 = data_value;
    DAC->SWTRIGR |= (1 << 0);

    step++;
    if (step >= 4096) {
        step = 0;
    }

    Delay(frequency);
}
```

4)sysTick.

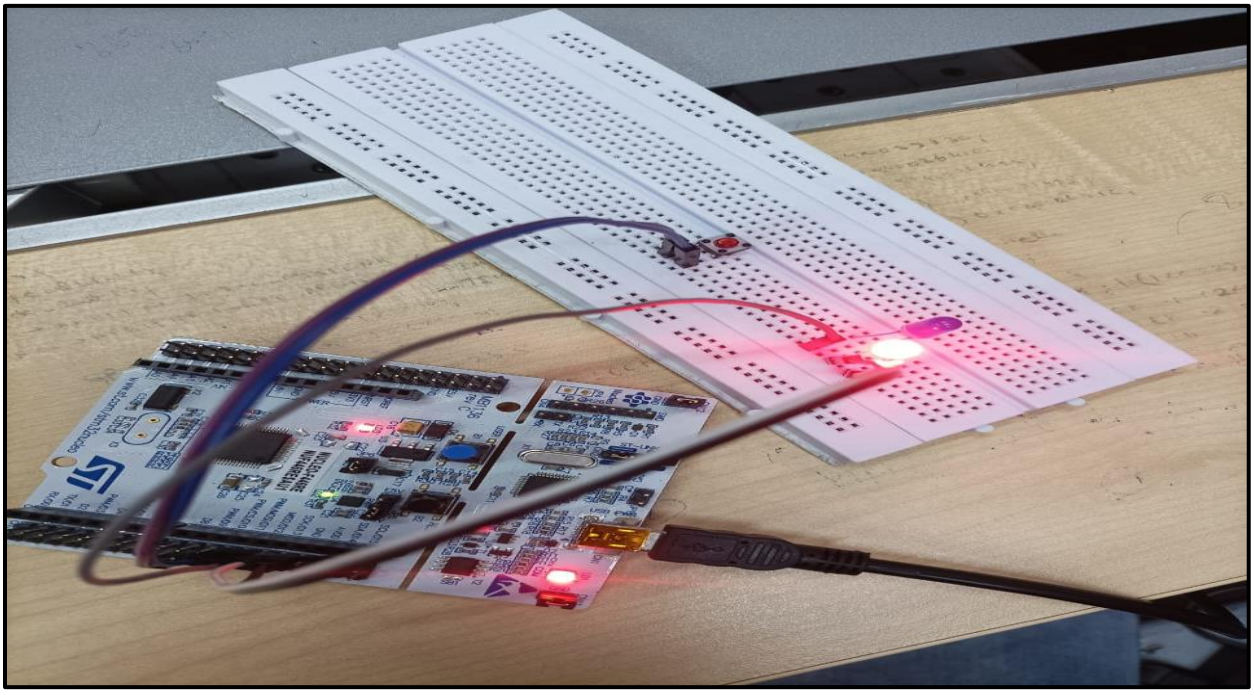
```c
/**
 * @file sysTick.c
 * @brief SysTick timer initialization and delay functions.
 *
 * This file contains functions to initialize the SysTick timer and provide a
 * delay function. The delay function uses the SysTick timer to create precise
 * delays based on the system core clock.
 */
#include "stm32f446xx.h"
#include "pll.h"
#include "sysTick.h"
volatile uint32_t ms_counter = 0;
volatile uint32_t millis = 0;

void SysTick_Init(){
    SysTick->VAL = 0;
    SysTick->LOAD = (HCLK_FREQ / 1000)- 1;
    SysTick->CTRL = SysTick_CTRL_CLKSOURCE_Msk |
                    SysTick_CTRL_TICKINT_Msk |
                    SysTick_CTRL_ENABLE_Msk;
}

void SysTick_Handler(){
    if (ms_counter) ms_counter--;
    millis++;
}

void delay_ms(uint32_t ms){
    ms_counter = ms;
    while (ms_counter);
}

uint32_t getMillis(){
    return millis;
}
void Delay(uint32_t delay) {
    SysTick->LOAD = delay - 1;    // Set reload register
    SysTick->VAL = 0;             // Reset the SysTick counter value
    SysTick->CTRL |= 1;           // Start SysTick

    while (!(SysTick->CTRL & (1 << 16))); // Wait for the COUNTFLAG

    SysTick->CTRL &= ~1;          // Stop SysTick
}
```

OutPut :

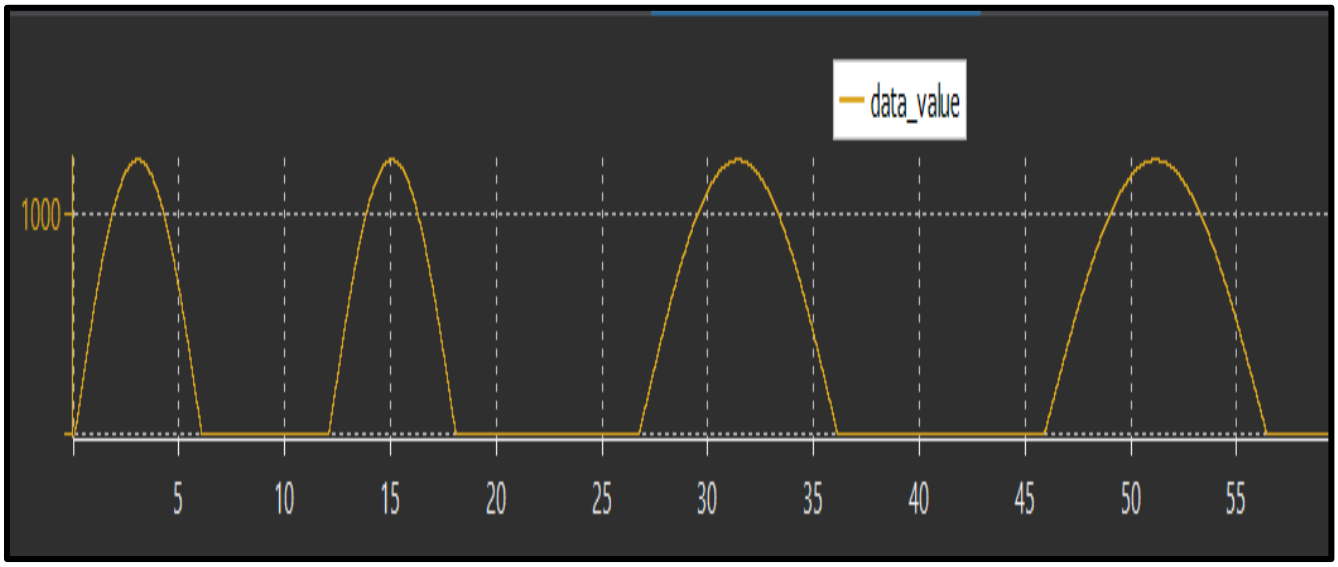
➢ Wave shape change on pressing PC13 Push button.



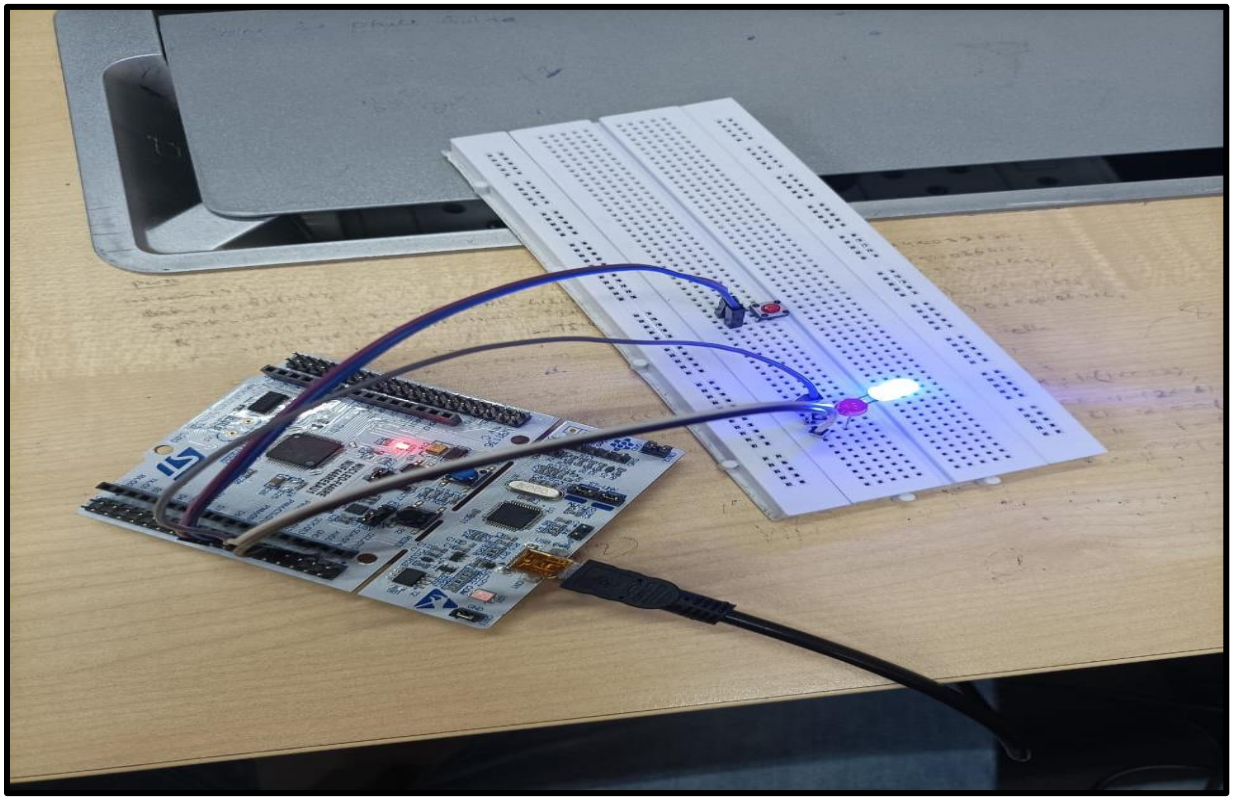
➢ Hardware Connection :



➢ Wave-frequency change on pressing PA7 external Push button.



➢ Hardware Connection



Future scope:

A. Enhanced User Interface:

- LCD Display: Integrate an LCD to display the current waveform type, frequency, and other relevant parameters.
- Rotary Encoders: Use rotary encoders for more precise control over frequency and waveform selection.

B. Additional Waveforms:

- Arbitrary Waveforms: Implement the ability to generate user-defined arbitrary waveforms.
- Noise Generation: Include white noise or other types of noise generation for testing and analysis purposes**.**

C. Data Logging and Analysis:

- SD Card Logging: Implement data logging to an SD card for later analysis of generated waveforms.
- Real-time Analysis: Add features for real-time analysis and display of waveform characteristics like frequency, amplitude, and harmonics.

CONCLUSION:

This project successfully shows how to generate different waveforms (sine, square, and triangle) with the STM32F446xx microcontroller. We created a flexible waveform generator by combining the microcontroller's DAC, GPIO, EXTI, and SysTick timer. The solution allows users to interact with the waveform type and frequency via buttons, with LED indications giving visible feedback. This project demonstrates the STM32F446xx's capacity to handle analog signal production and real-time user inputs, making it an invaluable tool for signal processing, testing, and teaching applications.