Project Name:-

Electric Vehicle Workshop Assignments

By

Ankit Kumar

1st March Batch

PROJECT DESCRIPTION

Battery Management System (BMS) Implementation on STM32.

Project Objective :-

This project introduces students to Battery Management Systems (BMS), essential for managing the performance, safety, and lifespan of batteries in electric vehicles and other applications. Students will learn how to implement a BMS using STM32, focusing on firmware and hardware integration.

Introduction to BMS (Battery Management System)

A Battery Management System (BMS) is an electronic control system that monitors and regulates the operation of a rechargeable battery pack. Its primary role is to ensure safety, longevity, and optimal performance of the battery.

A modern BMS typically performs the following key functions:

1. Voltage Regulation

Monitors the voltage of individual cells and the entire pack.

Prevents:

- Overvoltage, which can lead to thermal runaway or permanent cell damage.
- Undervoltage, which can reduce cell life or make the pack unstable.

Balances cell voltages using:

- Passive balancing (dissipating excess energy as heat).
- Active balancing (redistributing charge between cells).

2. Charging/Discharging Management

- Controls the flow of current during charging and discharging.
- Ensures the battery operates within safe current limits.

Interfaces with:

- Charger (to regulate charging current/voltage).
- Motor controller or load (to limit discharge current).

Prevents:

- Overcurrent events.
- Short circuits.
- Unsafe operation due to abnormal load conditions.

3. State-of-Charge (SOC) Estimation

SOC is essentially the battery's "fuel gauge."

The BMS estimates SOC using methods such as:

- Coulomb counting (tracking current in/out).
- Open-circuit voltage (OCV) correlation with SOC.
- Kalman filters or model-based estimation for higher accuracy.

Accurate SOC estimation helps:

- Predict remaining runtime.
- Protect the battery from deep discharge or over-charge.
- Optimize energy usage in applications like EVs and energy storage.

4. Thermal Management

Temperature has a huge impact on battery performance and safety.

A BMS:

- Continuously monitors cell and pack temperature.
- Activates cooling (fans, liquid cooling) or heating systems if needed.

Prevents operation outside safe temperature range to avoid:

- Thermal runaway.
- Reduced charge acceptance.
- Accelerated aging.

Thermal control is critical in high-power applications such as electric vehicles.

BMS Firmware & Hardware Interaction

A BMS is composed of hardware (sensors, protection circuits, communication ICs) and firmware (the logic and algorithms that control battery operation). Together, they form an integrated safety and management system.

1. BMS Hardware Components

Key hardware elements include :-

a. Voltage & Current Sensors- Cell voltage monitors, Pack voltage measurement circuits, Current sensors (Shunt resistor, Hall-effect sensor, etc.).

b. Temperature Sensors- Typically NTC thermistors located throughout the pack.

c. Protection Circuits- FETs or relays for switching the pack on/off, Over current, short circuit, and over temperature protections.

d. Balancing Circuits- Passive balancing resistors, Active balancing ICs and charge transfer components.

e. Communication Interfaces- CAN bus, UART, SPI, I2C—for reporting data and receiving commands.

2. BMS Firmware Functions

Firmware is the software logic programmed into the BMS microcontroller. It performs:

a. Data Acquisition- Continuously reads sensor data: voltage, current, temperature.

b. State Estimation- SOC, SOH (State of Health), SOF (State of Function/Power).

c. Decision Making- Determine when to- Disconnect the battery, Actively balance cells, Start cooling/heating, Slow down charge/discharge rates.

d. Safety Logic- Trigger alarms and protective actions on fault detection.

e. Communication- Sends battery status to external systems (e.g., a vehicle's control unit).

How Firmware and Hardware Work Together

1. Monitoring Loop- Hardware sensors feed raw data → Firmware processes and filters this data.

2. Control Loop- Firmware analyzes conditions → commands hardware switches, balancers, or thermal systems.

3. Protection Loop- Hardware protection may act independently (hardware-level cut-off), Firmware supervises long-term safety and system-level fault handling.

4. Diagnostics- Firmware logs errors and communicates battery health to host systems.

For Coding in Master microcontroller Side –

Firstly we have created new STM32 PROJECT with MCU/MPU Selector STM32F103C8.

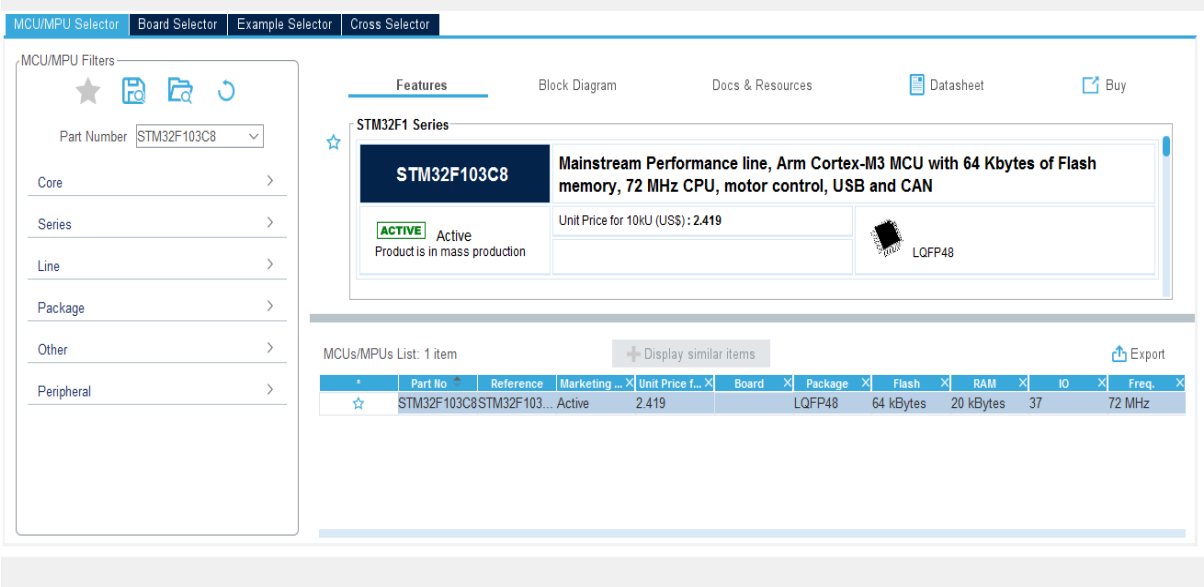


Fig. MCU/MPU Selection

In System Core will we do the following setting.

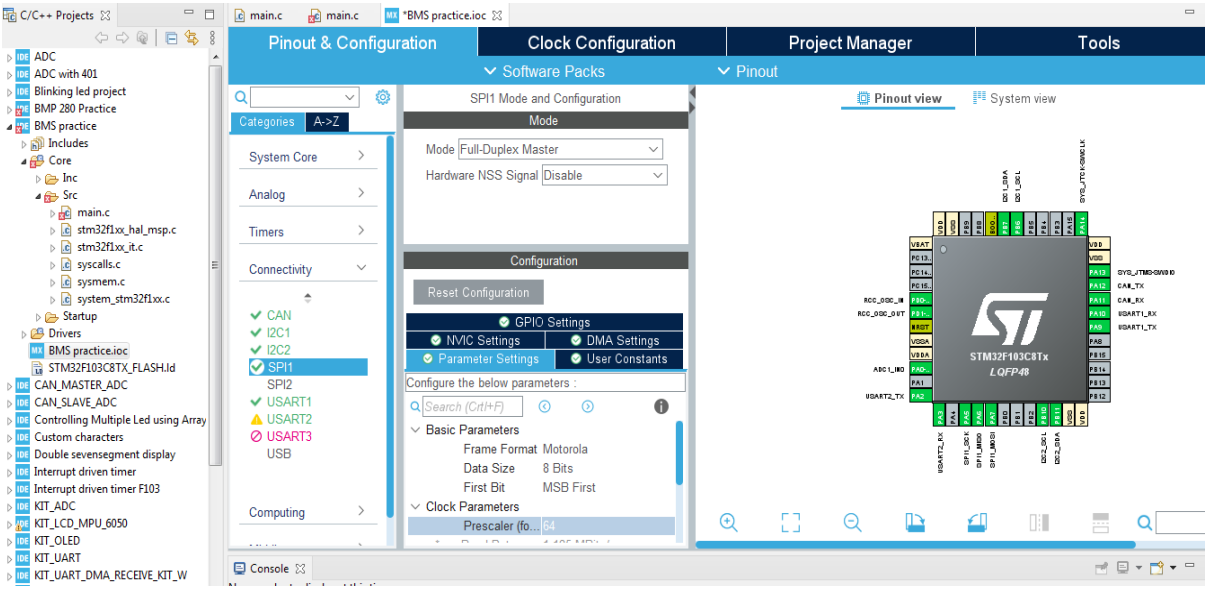


Fig. Connectivity Settings

Coding to be done in main.c file

```
/* USER CODE BEGIN Header */
/**
  ******************************************************************************
  * @file           : main.c
  * @brief          : Main program body
  ******************************************************************************
  * @attention
  *
  * <h2><center>&copy; Copyright (c) 2025 STMicroelectronics.
  * All rights reserved.</center></h2>
  *
  * This software component is licensed by ST under BSD 3-Clause license,
  * the "License"; You may not use this file except in compliance with the
  * License. You may obtain a copy of the License at:
  *                        opensource.org/licenses/BSD-3-Clause
  *
  ******************************************************************************
  */
/* USER CODE END Header */
/* Includes ------------------------------------------------------------------*/
#include "main.h"

/* Private includes ----------------------------------------------------------*/
/* USER CODE BEGIN Includes */
#include "BMSKitAppl.h"
#include "command.h"
#include "can.h"
#include "custom.h"
#include "rtc.h"
#include "ssd1306.h"
#include <stdio.h>

extern void bmp280_init();
extern void measure_bmp_para();

/* USER CODE END Includes */

/* Private typedef -----------------------------------------------------------*/
/* USER CODE BEGIN PTD */
#define tx 1
#define rx 1
/* USER CODE END PTD */

/* Private define ------------------------------------------------------------*/
/* USER CODE BEGIN PD */
/* USER CODE END PD */

/* Private macro -------------------------------------------------------------*/
/* USER CODE BEGIN PM */

/* USER CODE END PM */

/* Private variables ---------------------------------------------------------*/
ADC_HandleTypeDef hadc1;

CAN_HandleTypeDef hcan;
```

```c
I2C_HandleTypeDef hi2c1;
I2C_HandleTypeDef hi2c2;

RTC_HandleTypeDef hrtc;

SPI_HandleTypeDef hspi1;

UART_HandleTypeDef huart1;
UART_HandleTypeDef huart2;

/* USER CODE BEGIN PV */

/* USER CODE END PV */

/* Private function prototypes -----------------------------------------------*/
void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_CAN_Init(void);
static void MX_I2C1_Init(void);
static void MX_SPI1_Init(void);
static void MX_USART1_UART_Init(void);
static void MX_ADC1_Init(void);
static void MX_I2C2_Init(void);
static void MX_USART2_UART_Init(void);
static void MX_RTC_Init(void);
/* USER CODE BEGIN PFP */

/* USER CODE END PFP */

/* Private user code ---------------------------------------------------------*/
/* USER CODE BEGIN 0 */
void i2c_eeprom_init()
{
      MX_I2C1_Init();
}
/* USER CODE END 0 */

/**
  * @brief  The application entry point.
  * @retval int
  */
extern fontDef_t Font_7x10;

extern float bmp_temperature;
extern float bmp_pressure;
extern font bmp_altitude;
extern int can_datacheck;
int main(void)
{
  /* USER CODE BEGIN 1 */

  /* USER CODE END 1 */

  /* MCU Configuration--------------------------------------------------------*/

  /* Reset of all peripherals, Initializes the Flash interface and the Systick. */
  HAL_Init();

  /* USER CODE BEGIN Init */
```

```
  /* USER CODE END Init */

  /* Configure the system clock */
  SystemClock_Config();

  /* USER CODE BEGIN SysInit */

  /* USER CODE END SysInit */

  /* Initialize all configured peripherals */
  MX_GPIO_Init();
  MX_CAN_Init();
  MX_I2C1_Init();
  MX_SPI1_Init();
  MX_USART1_UART_Init();
  MX_ADC1_Init();
  MX_I2C2_Init();
  MX_USART2_UART_Init();
  MX_RTC_Init();
  /* USER CODE BEGIN 2 */
HAL_I2C_DeInit(&hi2c1)
HAL_ADCEx_Calibration_Start(&hadc1);
//if(HAL_RTCEx_BKUPRead(&hrtc, RTC_BKP_DR1)!=0x2345)
//{
//    set_time(15, 40, 00);
//    set_date(24, 9,11,7);
//}
//bmp280_init();
//can_init();
//print_oled();

char buffer[32];
  /* USER CODE END 2 */

  /* Infinite loop */
  /* USER CODE BEGIN WHILE */
  while (1)
  {
        HAL_Delay(1000);

        uint16_t adc_value_battery_voltage=get_adc_value(ADC_INPUT_VOLTAGE);
        float battery_voltage= (ADC_To_Voltage(adc_value_battery_voltage)*2);

        uint16_t adc_value_load_current=get_adc_value(ADC_LOAD_CURRENT);
        float load_current= (ADC_To_Voltage(adc_value_load_current)/0.1)*100;

        uint16_t adc_value_input_current = get_adc_value(ADC_INPUT_CURRENT);
        float input_current= (ADC_To_Voltage(adc_value_input_current)/0.1)*100;

        ssd1306_I2C_Write(SSD1306_I2C_ADDR, 0x00, 0xAB);
        SSD1306_ON();

        SSD1306_Clear();

        SSD1306_GotoXY(10,10);
        sprintf(buffer, "B V : %0.4f",Battery_Voltage);
        SSD1306_Puts(buffer,&font_7x10,SSD1306_COLOR_WHITE);
```

```
        SSD1306_GotoXY(10,25);
        sprintf(buffer, "L C : %0.4f",load_current);
        SSD1306_Puts(buffer,&font_7x10,SSD1306_COLOR_WHITE);

        SSD1306_GotoXY(10,40);
        sprintf(buffer, "I C : %0.4f",input_current);
        SSD1306_Puts(buffer,&font_7x10,SSD1306_COLOR_WHITE);

        SSD1306_UpdateScreen();
        HAL_Delay(1000);

    /* USER CODE END WHILE */

    /* USER CODE BEGIN 3 */
  }
  /* USER CODE END 3 */
}

/**
  * @brief System Clock Configuration
  * @retval None
  */
void SystemClock_Config(void)
{
  RCC_OscInitTypeDef RCC_OscInitStruct = {0};
  RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};
  RCC_PeriphCLKInitTypeDef PeriphClkInit = {0};

  /** Initializes the RCC Oscillators according to the specified parameters
  * in the RCC_OscInitTypeDef structure.
  */
  RCC_OscInitStruct.OscillatorType =
RCC_OSCILLATORTYPE_HSI|RCC_OSCILLATORTYPE_LSI;
  RCC_OscInitStruct.HSIState = RCC_HSI_ON;
  RCC_OscInitStruct.HSICalibrationValue = RCC_HSICALIBRATION_DEFAULT;
  RCC_OscInitStruct.LSIState = RCC_LSI_ON;
  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_HSI;
  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();
  }
  PeriphClkInit.PeriphClockSelection = RCC_PERIPHCLK_RTC|RCC_PERIPHCLK_ADC;
  PeriphClkInit.RTCClockSelection = RCC_RTCCLKSOURCE_LSI;
  PeriphClkInit.AdcClockSelection = RCC_ADCPCLK2_DIV2;
  if (HAL_RCCEx_PeriphCLKConfig(&PeriphClkInit) != HAL_OK)
  {
```

```
    Error_Handler();
  }
}

/**
  * @brief ADC1 Initialization Function
  * @param None
  * @retval None
  */
static void MX_ADC1_Init(void)
{

  /* USER CODE BEGIN ADC1_Init 0 */

  /* USER CODE END ADC1_Init 0 */

  ADC_ChannelConfTypeDef sConfig = {0};

  /* USER CODE BEGIN ADC1_Init 1 */

  /* USER CODE END ADC1_Init 1 */
  /** Common config
  */
  hadc1.Instance = ADC1;
  hadc1.Init.ScanConvMode = ADC_SCAN_DISABLE;
  hadc1.Init.ContinuousConvMode = DISABLE;
  hadc1.Init.DiscontinuousConvMode = DISABLE;
  hadc1.Init.ExternalTrigConv = ADC_SOFTWARE_START;
  hadc1.Init.DataAlign = ADC_DATAALIGN_RIGHT;
  hadc1.Init.NbrOfConversion = 1;
  if (HAL_ADC_Init(&hadc1) != HAL_OK)
  {
    Error_Handler();
  }
  /** Configure Regular Channel
  */
  sConfig.Channel = ADC_CHANNEL_0;
  sConfig.Rank = ADC_REGULAR_RANK_1;
  sConfig.SamplingTime = ADC_SAMPLETIME_1CYCLE_5;
  if (HAL_ADC_ConfigChannel(&hadc1, &sConfig) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN ADC1_Init 2 */

  /* USER CODE END ADC1_Init 2 */

}

/**
  * @brief CAN Initialization Function
  * @param None
  * @retval None
  */
static void MX_CAN_Init(void)
{

  /* USER CODE BEGIN CAN_Init 0 */
```

```
  /* USER CODE END CAN_Init 0 */

  /* USER CODE BEGIN CAN_Init 1 */

  /* USER CODE END CAN_Init 1 */
  hcan.Instance = CAN1;
  hcan.Init.Prescaler = 16;
  hcan.Init.Mode = CAN_MODE_NORMAL;
  hcan.Init.SyncJumpWidth = CAN_SJW_1TQ;
  hcan.Init.TimeSeg1 = CAN_BS1_1TQ;
  hcan.Init.TimeSeg2 = CAN_BS2_1TQ;
  hcan.Init.TimeTriggeredMode = DISABLE;
  hcan.Init.AutoBusOff = DISABLE;
  hcan.Init.AutoWakeUp = DISABLE;
  hcan.Init.AutoRetransmission = DISABLE;
  hcan.Init.ReceiveFifoLocked = DISABLE;
  hcan.Init.TransmitFifoPriority = DISABLE;
  if (HAL_CAN_Init(&hcan) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN CAN_Init 2 */

  /* USER CODE END CAN_Init 2 */

}

/**
  * @brief I2C1 Initialization Function
  * @param None
  * @retval None
  */
static void MX_I2C1_Init(void)
{

  /* USER CODE BEGIN I2C1_Init 0 */

  /* USER CODE END I2C1_Init 0 */

  /* USER CODE BEGIN I2C1_Init 1 */

  /* USER CODE END I2C1_Init 1 */
  hi2c1.Instance = I2C1;
  hi2c1.Init.ClockSpeed = 100000;
  hi2c1.Init.DutyCycle = I2C_DUTYCYCLE_2;
  hi2c1.Init.OwnAddress1 = 0;
  hi2c1.Init.AddressingMode = I2C_ADDRESSINGMODE_7BIT;
  hi2c1.Init.DualAddressMode = I2C_DUALADDRESS_DISABLE;
  hi2c1.Init.OwnAddress2 = 0;
  hi2c1.Init.GeneralCallMode = I2C_GENERALCALL_DISABLE;
  hi2c1.Init.NoStretchMode = I2C_NOSTRETCH_DISABLE;
  if (HAL_I2C_Init(&hi2c1) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN I2C1_Init 2 */

  /* USER CODE END I2C1_Init 2 */
```

```c
}

/**
  * @brief I2C2 Initialization Function
  * @param None
  * @retval None
  */
static void MX_I2C2_Init(void)
{

  /* USER CODE BEGIN I2C2_Init 0 */

  /* USER CODE END I2C2_Init 0 */

  /* USER CODE BEGIN I2C2_Init 1 */

  /* USER CODE END I2C2_Init 1 */
  hi2c2.Instance = I2C2;
  hi2c2.Init.ClockSpeed = 100000;
  hi2c2.Init.DutyCycle = I2C_DUTYCYCLE_2;
  hi2c2.Init.OwnAddress1 = 0;
  hi2c2.Init.AddressingMode = I2C_ADDRESSINGMODE_7BIT;
  hi2c2.Init.DualAddressMode = I2C_DUALADDRESS_DISABLE;
  hi2c2.Init.OwnAddress2 = 0;
  hi2c2.Init.GeneralCallMode = I2C_GENERALCALL_DISABLE;
  hi2c2.Init.NoStretchMode = I2C_NOSTRETCH_DISABLE;
  if (HAL_I2C_Init(&hi2c2) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN I2C2_Init 2 */

  /* USER CODE END I2C2_Init 2 */

}

/**
  * @brief RTC Initialization Function
  * @param None
  * @retval None
  */
static void MX_RTC_Init(void)
{

  /* USER CODE BEGIN RTC_Init 0 */

  /* USER CODE END RTC_Init 0 */

  /* USER CODE BEGIN RTC_Init 1 */

  /* USER CODE END RTC_Init 1 */
  /** Initialize RTC Only
  */
  hrtc.Instance = RTC;
  hrtc.Init.AsynchPrediv = RTC_AUTO_1_SECOND;
  hrtc.Init.OutPut = RTC_OUTPUTSOURCE_ALARM;
  if (HAL_RTC_Init(&hrtc) != HAL_OK)
  {
    Error_Handler();
```

```
  }
  /* USER CODE BEGIN RTC_Init 2 */

  /* USER CODE END RTC_Init 2 */

}

/**
  * @brief SPI1 Initialization Function
  * @param None
  * @retval None
  */
static void MX_SPI1_Init(void)
{

  /* USER CODE BEGIN SPI1_Init 0 */

  /* USER CODE END SPI1_Init 0 */

  /* USER CODE BEGIN SPI1_Init 1 */

  /* USER CODE END SPI1_Init 1 */
  /* SPI1 parameter configuration*/
  hspi1.Instance = SPI1;
  hspi1.Init.Mode = SPI_MODE_MASTER;
  hspi1.Init.Direction = SPI_DIRECTION_2LINES;
  hspi1.Init.DataSize = SPI_DATASIZE_8BIT;
  hspi1.Init.CLKPolarity = SPI_POLARITY_LOW;
  hspi1.Init.CLKPhase = SPI_PHASE_1EDGE;
  hspi1.Init.NSS = SPI_NSS_SOFT;
  hspi1.Init.BaudRatePrescaler = SPI_BAUDRATEPRESCALER_2;
  hspi1.Init.FirstBit = SPI_FIRSTBIT_MSB;
  hspi1.Init.TIMode = SPI_TIMODE_DISABLE;
  hspi1.Init.CRCCalculation = SPI_CRCCALCULATION_DISABLE;
  hspi1.Init.CRCPolynomial = 10;
  if (HAL_SPI_Init(&hspi1) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN SPI1_Init 2 */

  /* USER CODE END SPI1_Init 2 */

}

/**
  * @brief USART1 Initialization Function
  * @param None
  * @retval None
  */
static void MX_USART1_UART_Init(void)
{

  /* USER CODE BEGIN USART1_Init 0 */

  /* USER CODE END USART1_Init 0 */

  /* USER CODE BEGIN USART1_Init 1 */
```

```c
  /* USER CODE END USART1_Init 1 */
  huart1.Instance = USART1;
  huart1.Init.BaudRate = 115200;
  huart1.Init.WordLength = UART_WORDLENGTH_8B;
  huart1.Init.StopBits = UART_STOPBITS_1;
  huart1.Init.Parity = UART_PARITY_NONE;
  huart1.Init.Mode = UART_MODE_TX_RX;
  huart1.Init.HwFlowCtl = UART_HWCONTROL_NONE;
  huart1.Init.OverSampling = UART_OVERSAMPLING_16;
  if (HAL_UART_Init(&huart1) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN USART1_Init 2 */

  /* USER CODE END USART1_Init 2 */

}

/**
  * @brief USART2 Initialization Function
  * @param None
  * @retval None
  */
static void MX_USART2_UART_Init(void)
{

  /* USER CODE BEGIN USART2_Init 0 */

  /* USER CODE END USART2_Init 0 */

  /* USER CODE BEGIN USART2_Init 1 */

  /* USER CODE END USART2_Init 1 */
  huart2.Instance = USART2;
  huart2.Init.BaudRate = 115200;
  huart2.Init.WordLength = UART_WORDLENGTH_8B;
  huart2.Init.StopBits = UART_STOPBITS_1;
  huart2.Init.Parity = UART_PARITY_NONE;
  huart2.Init.Mode = UART_MODE_TX_RX;
  huart2.Init.HwFlowCtl = UART_HWCONTROL_NONE;
  huart2.Init.OverSampling = UART_OVERSAMPLING_16;
  if (HAL_UART_Init(&huart2) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN USART2_Init 2 */

  /* USER CODE END USART2_Init 2 */

}

/**
  * @brief GPIO Initialization Function
  * @param None
  * @retval None
  */
static void MX_GPIO_Init(void)
{
```

```
  /* GPIO Ports Clock Enable */
  __HAL_RCC_GPIOD_CLK_ENABLE();
  __HAL_RCC_GPIOA_CLK_ENABLE();
  __HAL_RCC_GPIOB_CLK_ENABLE();

}

/* 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****/
```

Explanation

```
uint16_t adc_value_battery_voltage=get_adc_value(ADC_INPUT_VOLTAGE);
```

- It reads raw ADC count corresponding to input voltage signal
- Hardware perspective-
  1. Input voltage (battery/adapter/Supply)
  2. Passed through a voltage divider
  3. Reduced voltage applied to MCU ADC pin
  4. ADC converts analog voltage to digtal number

```
float battery_voltage= (ADC_To_Voltage(adc_value_battery_voltage)*2);
```

- Converts ADC count to voltage.
- This voltage is only what ADC pin sees not real input voltage.

```
uint16_t adc_value_load_current=get_adc_value(ADC_LOAD_CURRENT);
```

- It reads raw ADC count corresponding to load current
- Hardware perspective-
  1. Load current
  2. Passed through a voltage divider
  3. Reduced voltage applied to MCU ADC pin
  4. ADC converts analog load current to digtal number

```
float load_current= (ADC_To_Voltage(adc_value_load_current)/0.1)*100;
```

- Converts ADC count to voltage.
- This voltage is only what ADC pin sees not real load current.

```
uint16_t adc_value_input_current = get_adc_value(ADC_INPUT_CURRENT);
```

- It reads raw ADC count corresponding to input current
- Hardware perspective-

1. input current
2. Passed through a voltage divider
3. Reduced voltage applied to MCU ADC pin
4. ADC converts analog input current to digtal number

```
float input_current= (ADC_To_Voltage(adc_value_input_current)/0.1)*100;
```

- Converts ADC count to voltage.
- This voltage is only what ADC pin sees not real load current.

Here resistor are connected which are acting as load for battery when we increase the load we can see that load current increases.

When Battery is operating on single load (only single resistor is turned ON), we can see Battery voltage as 4.2V, Load current as 33.85Amp, Input current as 4.8352 Amp.

Fig. OLED showing values when one load resistance is turned ON

When Battery is operating on three load (All three resistor is turned ON), we can see Battery voltage as 4.2V, Load current as 63.66Amp, Input current as 6.44 Amp.

Fig. OLED showing values when all 3 load resistance are Turned ON

Conclusion

Through this project, we gain a practical understanding of Battery Management Systems (BMS) and their critical role in ensuring battery safety, efficiency, and longevity. By implementing a BMS using an STM32 microcontroller, we bridge theoretical concepts with real-world application, developing hands-on skills in firmware development, sensor interfacing, data acquisition, and hardware–software integration.

PROJECT DESCRIPTION

CAN Protocol Implementation with STM32 for Automotive Communication.

Project Objective :-

This project aims to provide students with practical experience in implementing the CAN protocol on STM32, which is essential for communication between various electronic control units (ECUs) in automotive applications.

Introduction to CAN (Controller Area Network)

The Controller Area Network (CAN) is a serial communication protocol developed by Bosch to enable efficient data exchange between Electronic Control Units (ECUs) in a vehicle. It allows multiple ECUs to communicate over a single shared bus, eliminating the need for complex point-to-point wiring.

Significance in the Automotive Industry -

- Modern vehicles contain dozens of ECUs that manage systems such as engine control, braking (ABS), airbags, transmission, lighting, and infotainment. CAN plays a critical role in the automotive industry because it:
- Reduces wiring complexity, vehicle weight, and manufacturing cost
- Improves reliability through robust error detection and fault confinement
- Supports scalability, allowing new ECUs to be added without major redesign
- Meets strict automotive safety and performance requirements

Due to these advantages, CAN has become a standard communication protocol in almost all modern vehicles and is also widely used in industrial and embedded systems.

Role in Real-Time Communication between ECUs -

CAN is designed for real-time communication, which is essential for safety-critical vehicle functions. It uses a priority-based arbitration mechanism, where messages with higher priority (lower identifier value) gain immediate access to the bus. This ensures that time-critical message such as braking or airbag signals are transmitted with minimal delay.

Additionally, CAN provides:

- Deterministic communication, ensuring predictable message delivery times
- Automatic error handling, including error detection and retransmission
- High reliability, even in noisy automotive environments

As a result, CAN enables fast, reliable, and coordinated operation of ECUs, making it a backbone of modern automotive electronic systems.

CAN Data frame

A CAN (Controller Area Network) data frame is the message based protocol used to send data between nodes (ECUs, sensors, controllers) on a CAN bus.

Each frame carries:

- An identifier (who the message is for / its priority)
- Up to 8 bytes of data (classic CAN)
- Control and error-checking information

Basic Structure of a CAN Data Frame -

A CAN data frame is divided into several fields, sent from left to right on the bus:

1. Start of Frame (SOF) - 1 dominant bit, Signals the beginning of a message

2. Arbitration Field - Contains the Identifier (ID), Determines message priority (lower ID = higher priority)

3. Control Field - Indicates frame type, Specifies the data length (DLC)

4. Data Field - 0 to 8 bytes (64 bits max in classic CAN)

5. CRC Field - Cyclic Redundancy Check, Used for error detection

6. ACK Field - Receiving nodes acknowledge correct reception

7. End of Frame (EOF) - Marks the end of the message

Types of CAN Data Frames

1. Standard CAN Frame (11-bit Identifier)

This is the most commonly used CAN format.

Key features:

- Identifier length: 11 bits,
- Max IDs: 2,048 (2^{11}),
- Shorter frame → faster transmission,
- Widely used in automotive systems,

Typical use cases:

- Engine control units
- Sensors and actuators

- Body electronics

2. Extended CAN Frame (29-bit Identifier)

Introduced to allow many more message IDs.

Key features:

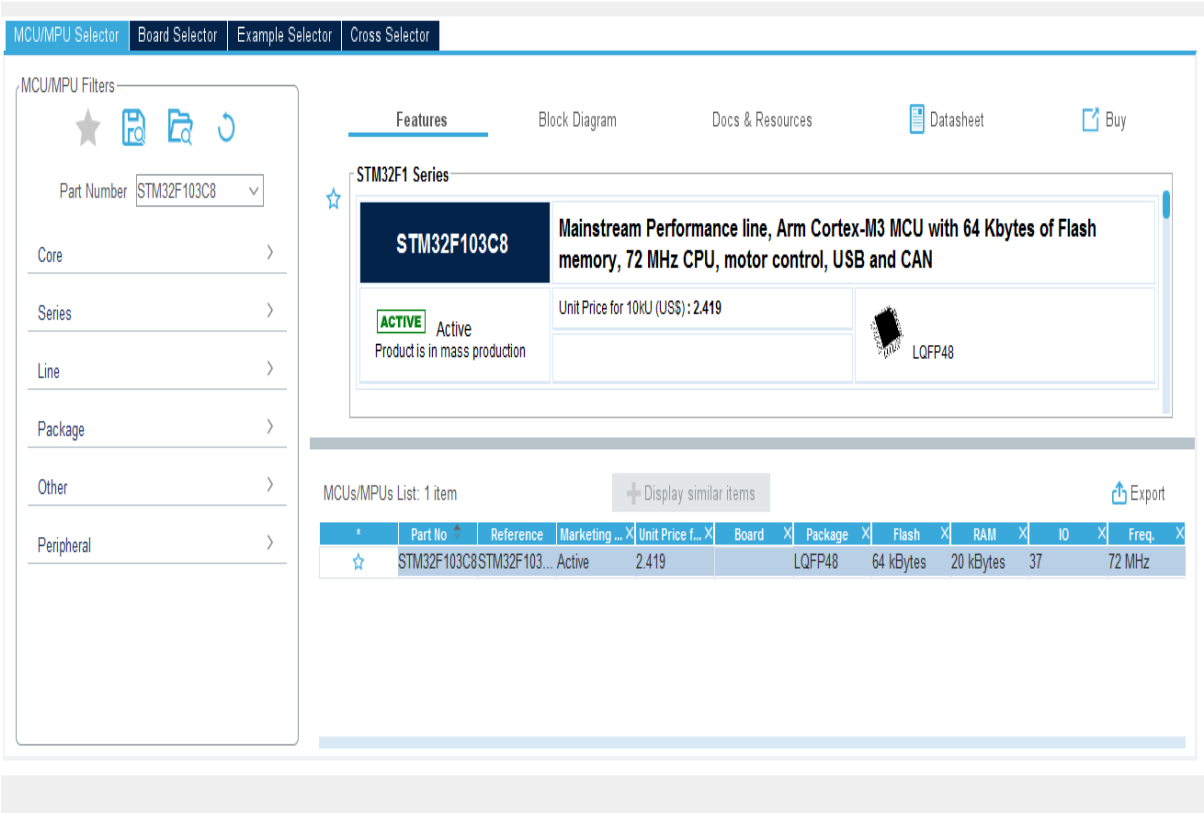
- Identifier length:29 bits
- Max IDs: ~536 million
- Longer frame → slightly slower
- Used in complex or large networks

Typical use cases:

- Heavy vehicles (trucks, buses)
- Industrial automation
- Systems needing many unique message IDs

<u>For Coding in Master microcontroller Side –</u>

Firstly we have created new STM32 PROJECT with MCU/MPU Selector STM32F103C8.



<u>Fig. MCU/MPU Selection</u>

In System Core will we do the following settings.

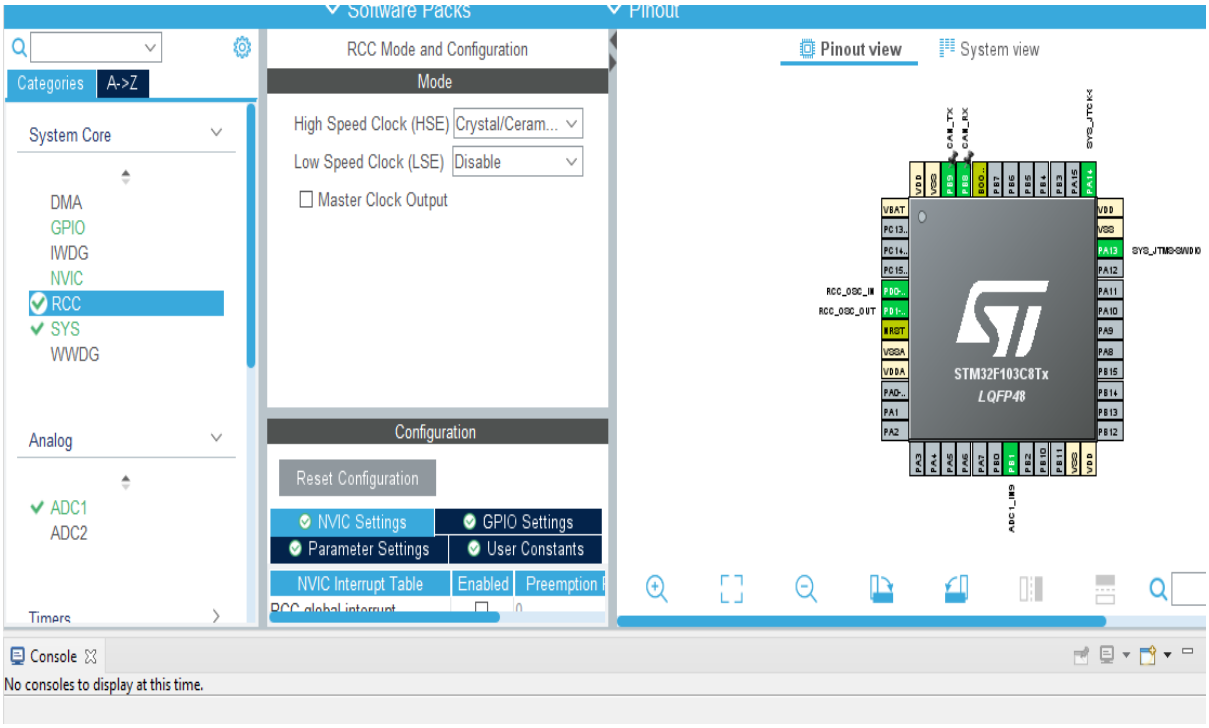


Fig. Settings in RCC

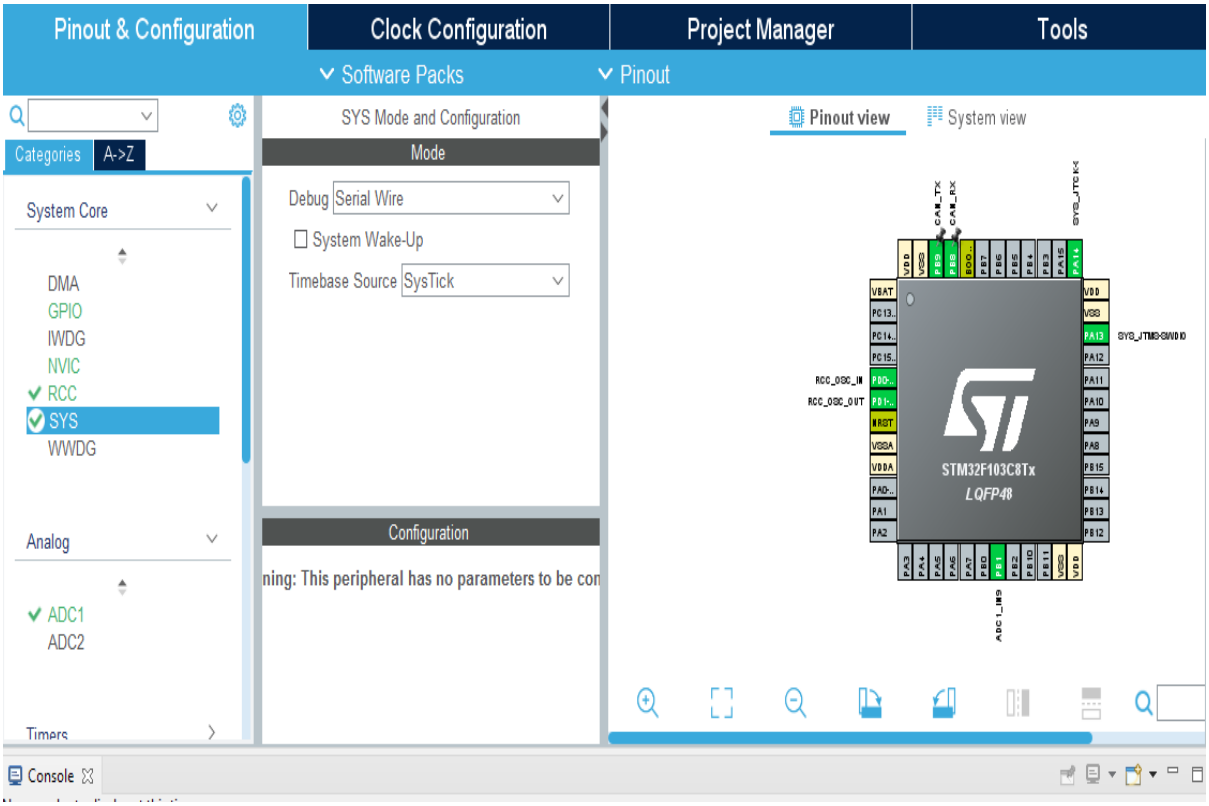


Fig. Settings in SYS

In Analog we will select the IN9

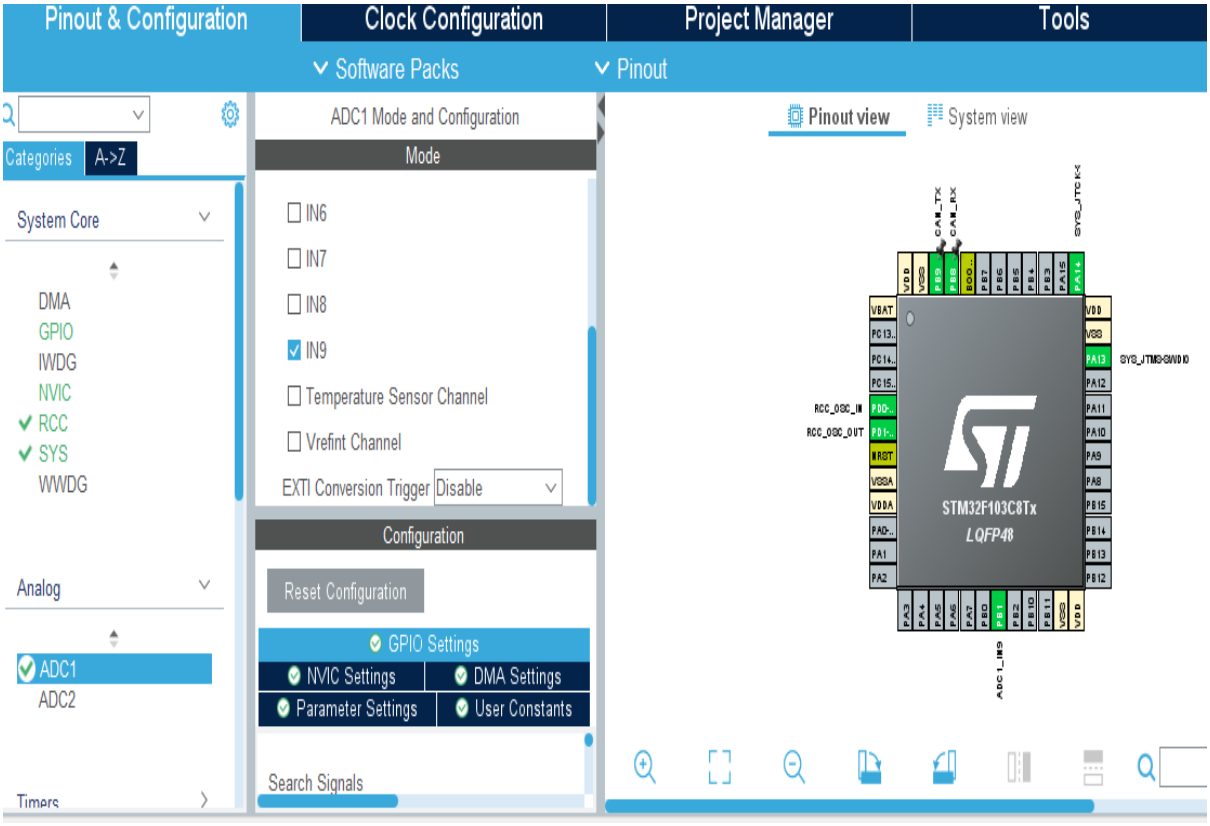


Fig. Selection of ADC

In connectivity we will select the CAN and changed the Tx and Rx pin to PB8, PB9.

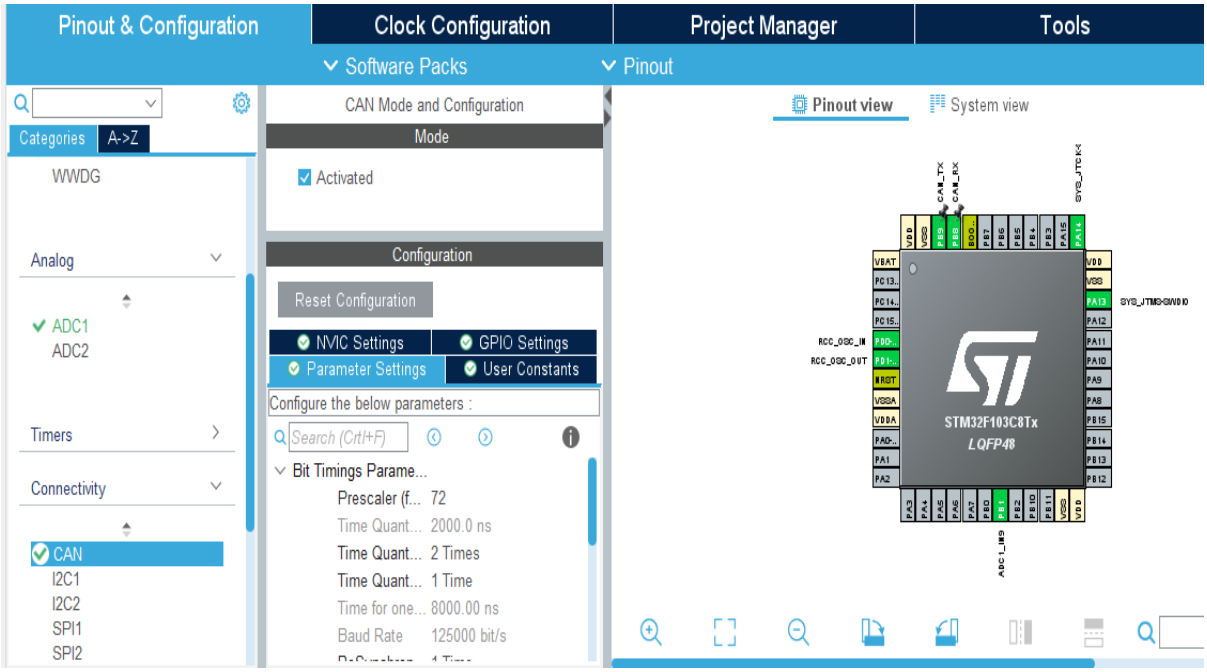


Fig. Assigning of CAN -Tx, Rx Pin to PB8, PB9

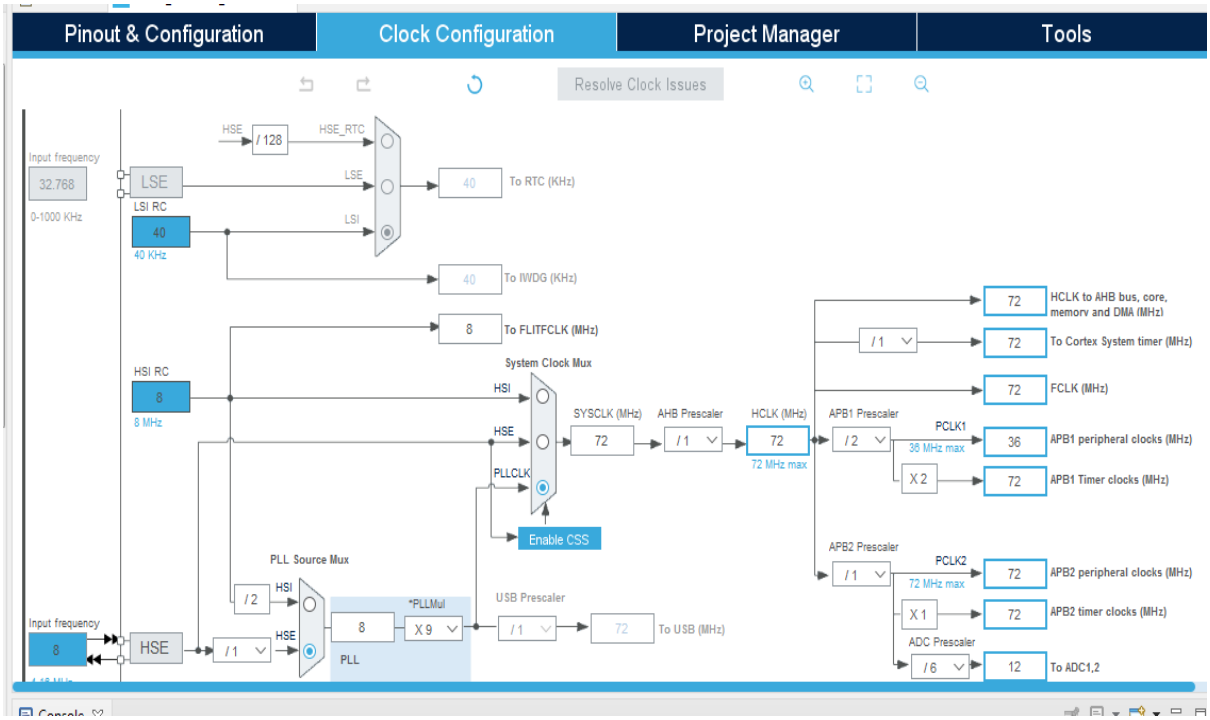


Fig. Clock Configuration Settings.

Code for Master Side –

```
/* USER CODE BEGIN Header */
/**
  ******************************************************************************
  * @file           : main.c
  * @brief          : Main program body
  ******************************************************************************
  * @attention
  *
  * <h2><center>&copy; Copyright (c) 2025 STMicroelectronics.
  * All rights reserved.</center></h2>
  *
  * This software component is licensed by ST under BSD 3-Clause license,
  * the "License"; You may not use this file except in compliance with the
  * License. You may obtain a copy of the License at:
  *                        opensource.org/licenses/BSD-3-Clause
  *
  ******************************************************************************
  */
/* USER CODE END Header */
/* Includes ------------------------------------------------------------------*/
#include "main.h"

/* Private includes ----------------------------------------------------------*/
/* USER CODE BEGIN Includes */

/* USER CODE END Includes */
```

```c
/* Private typedef -----------------------------------------------------------*/
/* USER CODE BEGIN PTD */

/* USER CODE END PTD */

/* Private define ------------------------------------------------------------*/
/* USER CODE BEGIN PD */
/* USER CODE END PD */

/* Private macro -------------------------------------------------------------*/
/* USER CODE BEGIN PM */

/* USER CODE END PM */

/* Private variables ---------------------------------------------------------*/
ADC_HandleTypeDef hadc1;

CAN_HandleTypeDef hcan;

/* USER CODE BEGIN PV */

/* USER CODE END PV */

/* Private function prototypes -----------------------------------------------*/
void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_ADC1_Init(void);
static void MX_CAN_Init(void);
/* USER CODE BEGIN PFP */

/* USER CODE END PFP */

/* Private user code ---------------------------------------------------------*/
/* USER CODE BEGIN 0 */
uint16_t readValue;
CAN_TxHeaderTypeDef TxHeader;
uint32_t TxMailbox;
uint8_t TxData[8]="hello---";

/* USER CODE END 0 */

/**
  * @brief  The application entry point.
  * @retval int
  */
int main(void)
{
  /* USER CODE BEGIN 1 */

  /* USER CODE END 1 */

  /* MCU Configuration--------------------------------------------------------*/

  /* Reset of all peripherals, Initializes the Flash interface and the Systick. */
  HAL_Init();

  /* USER CODE BEGIN Init */

  /* USER CODE END Init */
```

```c
  /* Configure the system clock */
  SystemClock_Config();

  /* USER CODE BEGIN SysInit */

  /* USER CODE END SysInit */

  /* Initialize all configured peripherals */
  MX_GPIO_Init();
  MX_ADC1_Init();
  MX_CAN_Init();
  /* USER CODE BEGIN 2 */
  HAL_ADC_Start(&hadc1);
  HAL_CAN_Start(&hcan);
  TxHeader.DLC = 8;  // data length
  TxHeader.IDE = CAN_ID_STD;
  TxHeader.RTR = CAN_RTR_DATA;
  TxHeader.StdId = 0x6A5;  // ID can be between Hex1 and Hex7FF (1-2047 decimal)

  /* USER CODE END 2 */

  /* Infinite loop */
  /* USER CODE BEGIN WHILE */
  while (1)
  {

            HAL_ADC_PollForConversion(&hadc1,1000);
                  readValue = HAL_ADC_GetValue(&hadc1);
                  // readValue is between 0-4095 but we need to transmit maximum
255 (one byte)
                  // We divide readValue by 16 to get 0-255. Better way to divide
is by shifting bits
                  // (readValue >> 4) is same as (readValue / 16)
                  TxData[7] = readValue >> 4;
                  HAL_CAN_AddTxMessage(&hcan, &TxHeader, TxData, &TxMailbox);
                  HAL_Delay(1000);
                  /* USER CODE END WHILE */

    /* USER CODE BEGIN 3 */
  }
  /* USER CODE END 3 */
}

/**
  * @brief System Clock Configuration
  * @retval None
  */
void SystemClock_Config(void)
{
  RCC_OscInitTypeDef RCC_OscInitStruct = {0};
  RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};
  RCC_PeriphCLKInitTypeDef PeriphClkInit = {0};

  /** Initializes the RCC Oscillators according to the specified parameters
  * in the RCC_OscInitTypeDef structure.
  */
  RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSE;
  RCC_OscInitStruct.HSEState = RCC_HSE_ON;
```

```
  RCC_OscInitStruct.HSEPredivValue = RCC_HSE_PREDIV_DIV1;
  RCC_OscInitStruct.HSIState = RCC_HSI_ON;
  RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
  RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSE;
  RCC_OscInitStruct.PLL.PLLMUL = RCC_PLL_MUL9;
  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_PLLCLK;
  RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
  RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV2;
  RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV1;

  if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_2) != HAL_OK)
  {
    Error_Handler();
  }
  PeriphClkInit.PeriphClockSelection = RCC_PERIPHCLK_ADC;
  PeriphClkInit.AdcClockSelection = RCC_ADCPCLK2_DIV6;
  if (HAL_RCCEx_PeriphCLKConfig(&PeriphClkInit) != HAL_OK)
  {
    Error_Handler();
  }
}

/**
  * @brief ADC1 Initialization Function
  * @param None
  * @retval None
  */
static void MX_ADC1_Init(void)
{

  /* USER CODE BEGIN ADC1_Init 0 */

  /* USER CODE END ADC1_Init 0 */

  ADC_ChannelConfTypeDef sConfig = {0};

  /* USER CODE BEGIN ADC1_Init 1 */

  /* USER CODE END ADC1_Init 1 */
  /** Common config
  */
  hadc1.Instance = ADC1;
  hadc1.Init.ScanConvMode = ADC_SCAN_DISABLE;
  hadc1.Init.ContinuousConvMode = DISABLE;
  hadc1.Init.DiscontinuousConvMode = DISABLE;
  hadc1.Init.ExternalTrigConv = ADC_SOFTWARE_START;
  hadc1.Init.DataAlign = ADC_DATAALIGN_RIGHT;
  hadc1.Init.NbrOfConversion = 1;
  if (HAL_ADC_Init(&hadc1) != HAL_OK)
  {
    Error_Handler();
```

```
  }
  /** Configure Regular Channel
  */
  sConfig.Channel = ADC_CHANNEL_9;
  sConfig.Rank = ADC_REGULAR_RANK_1;
  sConfig.SamplingTime = ADC_SAMPLETIME_1CYCLE_5;
  if (HAL_ADC_ConfigChannel(&hadc1, &sConfig) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN ADC1_Init 2 */

  /* USER CODE END ADC1_Init 2 */

}

/**
  * @brief CAN Initialization Function
  * @param None
  * @retval None
  */
static void MX_CAN_Init(void)
{

  /* USER CODE BEGIN CAN_Init 0 */

  /* USER CODE END CAN_Init 0 */

  /* USER CODE BEGIN CAN_Init 1 */

  /* USER CODE END CAN_Init 1 */
  hcan.Instance = CAN1;
  hcan.Init.Prescaler = 72;
  hcan.Init.Mode = CAN_MODE_NORMAL;
  hcan.Init.SyncJumpWidth = CAN_SJW_1TQ;
  hcan.Init.TimeSeg1 = CAN_BS1_2TQ;
  hcan.Init.TimeSeg2 = CAN_BS2_1TQ;
  hcan.Init.TimeTriggeredMode = DISABLE;
  hcan.Init.AutoBusOff = DISABLE;
  hcan.Init.AutoWakeUp = DISABLE;
  hcan.Init.AutoRetransmission = DISABLE;
  hcan.Init.ReceiveFifoLocked = DISABLE;
  hcan.Init.TransmitFifoPriority = DISABLE;
  if (HAL_CAN_Init(&hcan) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN CAN_Init 2 */

  /* USER CODE END CAN_Init 2 */

}

/**
  * @brief GPIO Initialization Function
  * @param None
  * @retval None
  */
static void MX_GPIO_Init(void)
```

```
{

  /* GPIO Ports Clock Enable */
  __HAL_RCC_GPIOD_CLK_ENABLE();
  __HAL_RCC_GPIOB_CLK_ENABLE();
  __HAL_RCC_GPIOA_CLK_ENABLE();

}

/* 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****/
```

After flashing the code in master side and connecting the CAN wire we can see the live expression in the debugger.

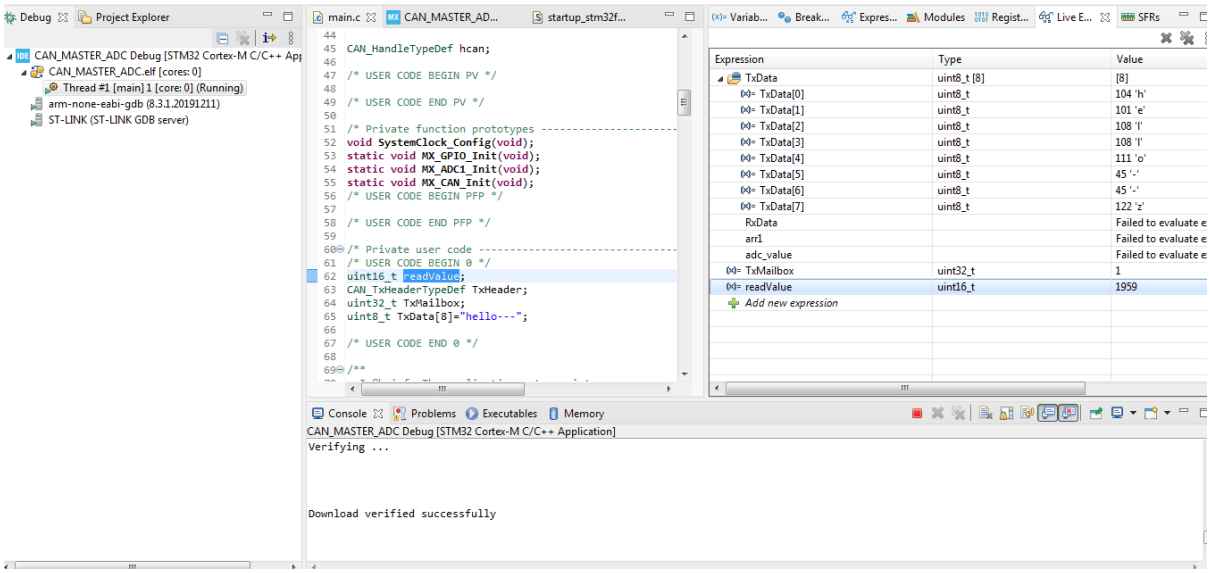


Fig. Debugger showing value on Tx side

For Coding in Slave microcontroller Side –

Firstly we have created new STM32 PROJECT with MCU/MPU Selector STM32F103C8.

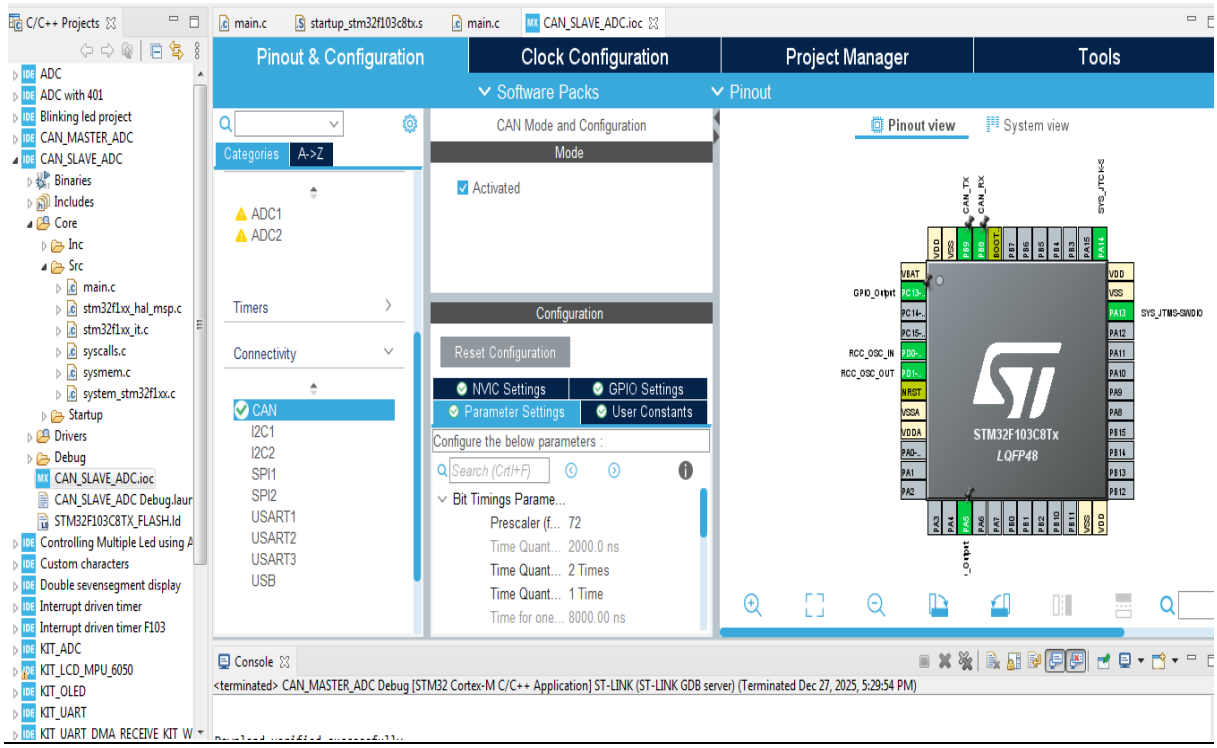


Fig. Settings to be done on slave side microcontroller

Code for Slave Side –

```
/* USER CODE BEGIN Header */
/**
  ******************************************************************************
  * @file           : main.c
  * @brief          : Main program body
  ******************************************************************************
  * @attention
  *
  * <h2><center>&copy; Copyright (c) 2025 STMicroelectronics.
  * All rights reserved.</center></h2>
  *
  * This software component is licensed by ST under BSD 3-Clause license,
  * the "License"; You may not use this file except in compliance with the
  * License. You may obtain a copy of the License at:
  *                        opensource.org/licenses/BSD-3-Clause
  *
  ******************************************************************************
  */
/* USER CODE END Header */
/* Includes ------------------------------------------------------------------*/
#include "main.h"

/* Private includes ----------------------------------------------------------*/
/* USER CODE BEGIN Includes */

/* USER CODE END Includes */

/* Private typedef -----------------------------------------------------------*/
/* USER CODE BEGIN PTD */

/* USER CODE END PTD */

/* Private define ------------------------------------------------------------*/
/* USER CODE BEGIN PD */
/* USER CODE END PD */

/* Private macro -------------------------------------------------------------*/
/* USER CODE BEGIN PM */

/* USER CODE END PM */

/* Private variables ---------------------------------------------------------*/
CAN_HandleTypeDef hcan;

/* USER CODE BEGIN PV */

/* USER CODE END PV */

/* Private function prototypes -----------------------------------------------*/
void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_CAN_Init(void);
/* USER CODE BEGIN PFP */

/* USER CODE END PFP */
```

```c
/* Private user code ---------------------------------------------------------*/
/* USER CODE BEGIN 0 */
CAN_FilterTypeDef sFilterConfig;
CAN_RxHeaderTypeDef RxHeader;
uint8_t RxData[8];
uint8_t delayLED;
void HAL_CAN_RxFifo1MsgPendingCallback(CAN_HandleTypeDef *hcan)
{
  HAL_CAN_GetRxMessage(hcan, CAN_RX_FIFO1, &RxHeader, RxData);
  delayLED = RxData[7];
}

/* USER CODE END 0 */

/**
  * @brief  The application entry point.
  * @retval int
  */
int main(void)
{
  /* USER CODE BEGIN 1 */

  /* USER CODE END 1 */

  /* MCU Configuration--------------------------------------------------------*/

  /* Reset of all peripherals, Initializes the Flash interface and the Systick. */
  HAL_Init();

  /* USER CODE BEGIN Init */

  /* USER CODE END Init */

  /* Configure the system clock */
  SystemClock_Config();

  /* USER CODE BEGIN SysInit */

  /* USER CODE END SysInit */

  /* Initialize all configured peripherals */
  MX_GPIO_Init();
  MX_CAN_Init();
  /* USER CODE BEGIN 2 */
  HAL_CAN_Start(&hcan);
  // Configure the filter
  sFilterConfig.FilterActivation = CAN_FILTER_ENABLE;
  sFilterConfig.FilterFIFOAssignment = CAN_FILTER_FIFO1;
  sFilterConfig.FilterMode = CAN_FILTERMODE_IDMASK;
  sFilterConfig.FilterIdHigh = 0x6A5<<5;
  sFilterConfig.FilterIdLow = 0;
  sFilterConfig.FilterMaskIdHigh = 0x7FF<<5; // SET 0 to unfilter
  sFilterConfig.FilterMaskIdLow = 0;
  sFilterConfig.FilterScale = CAN_FILTERSCALE_32BIT;
  HAL_CAN_ConfigFilter(&hcan, &sFilterConfig);
  // Activate the notification
  HAL_CAN_ActivateNotification(&hcan, CAN_IT_RX_FIFO1_MSG_PENDING);
```

```
  /* USER CODE END 2 */

  /* Infinite loop */
  /* USER CODE BEGIN WHILE */
  while (1)
  {
        HAL_GPIO_TogglePin(GPIOA, GPIO_PIN_5);
              HAL_Delay(delayLED);

    /* USER CODE END WHILE */

    /* USER CODE BEGIN 3 */
  }
  /* USER CODE END 3 */
}

/**
  * @brief System Clock Configuration
  * @retval None
  */
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_HSE;
  RCC_OscInitStruct.HSEState = RCC_HSE_ON;
  RCC_OscInitStruct.HSEPredivValue = RCC_HSE_PREDIV_DIV1;
  RCC_OscInitStruct.HSIState = RCC_HSI_ON;
  RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
  RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSE;
  RCC_OscInitStruct.PLL.PLLMUL = RCC_PLL_MUL9;
  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_PLLCLK;
  RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
  RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV2;
  RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV1;

  if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_2) != HAL_OK)
  {
    Error_Handler();
  }
}

/**
  * @brief CAN Initialization Function
  * @param None
  * @retval None
  */
```

```c
static void MX_CAN_Init(void)
{

  /* USER CODE BEGIN CAN_Init 0 */

  /* USER CODE END CAN_Init 0 */

  /* USER CODE BEGIN CAN_Init 1 */

  /* USER CODE END CAN_Init 1 */
  hcan.Instance = CAN1;
  hcan.Init.Prescaler = 72;
  hcan.Init.Mode = CAN_MODE_NORMAL;
  hcan.Init.SyncJumpWidth = CAN_SJW_1TQ;
  hcan.Init.TimeSeg1 = CAN_BS1_2TQ;
  hcan.Init.TimeSeg2 = CAN_BS2_1TQ;
  hcan.Init.TimeTriggeredMode = DISABLE;
  hcan.Init.AutoBusOff = DISABLE;
  hcan.Init.AutoWakeUp = DISABLE;
  hcan.Init.AutoRetransmission = DISABLE;
  hcan.Init.ReceiveFifoLocked = DISABLE;
  hcan.Init.TransmitFifoPriority = DISABLE;
  if (HAL_CAN_Init(&hcan) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN CAN_Init 2 */

  /* USER CODE END CAN_Init 2 */

}

/**
  * @brief GPIO Initialization Function
  * @param None
  * @retval None
  */
static void MX_GPIO_Init(void)
{
  GPIO_InitTypeDef GPIO_InitStruct = {0};

  /* GPIO Ports Clock Enable */
  __HAL_RCC_GPIOC_CLK_ENABLE();
  __HAL_RCC_GPIOD_CLK_ENABLE();
  __HAL_RCC_GPIOA_CLK_ENABLE();
  __HAL_RCC_GPIOB_CLK_ENABLE();

  /*Configure GPIO pin Output Level */
  HAL_GPIO_WritePin(GPIOC, GPIO_PIN_13, GPIO_PIN_RESET);

  /*Configure GPIO pin Output Level */
  HAL_GPIO_WritePin(GPIOA, GPIO_PIN_5, GPIO_PIN_RESET);

  /*Configure GPIO pin : PC13 */
  GPIO_InitStruct.Pin = GPIO_PIN_13;
  GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
  HAL_GPIO_Init(GPIOC, &GPIO_InitStruct);
```

```
  /*Configure GPIO pin : PA5 */
  GPIO_InitStruct.Pin = GPIO_PIN_5;
  GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
  HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);

}

/* 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****/
```

After flashing the code in slave side we can see the live expression in debugger which was sent from Master side.

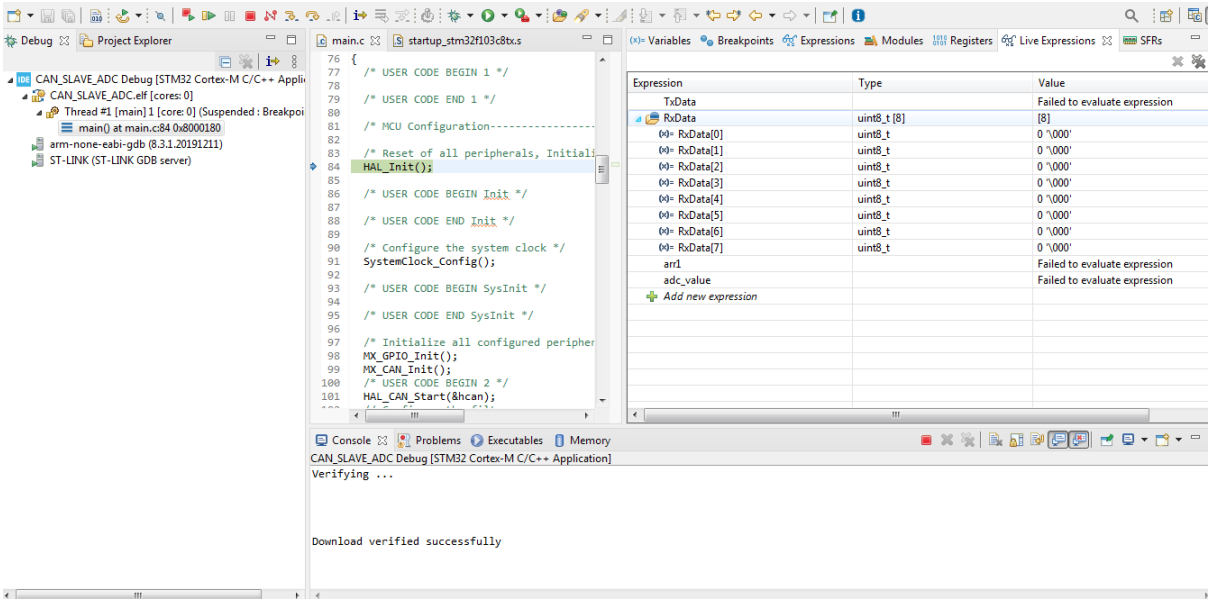


Fig. Debugger showing value on Rx side

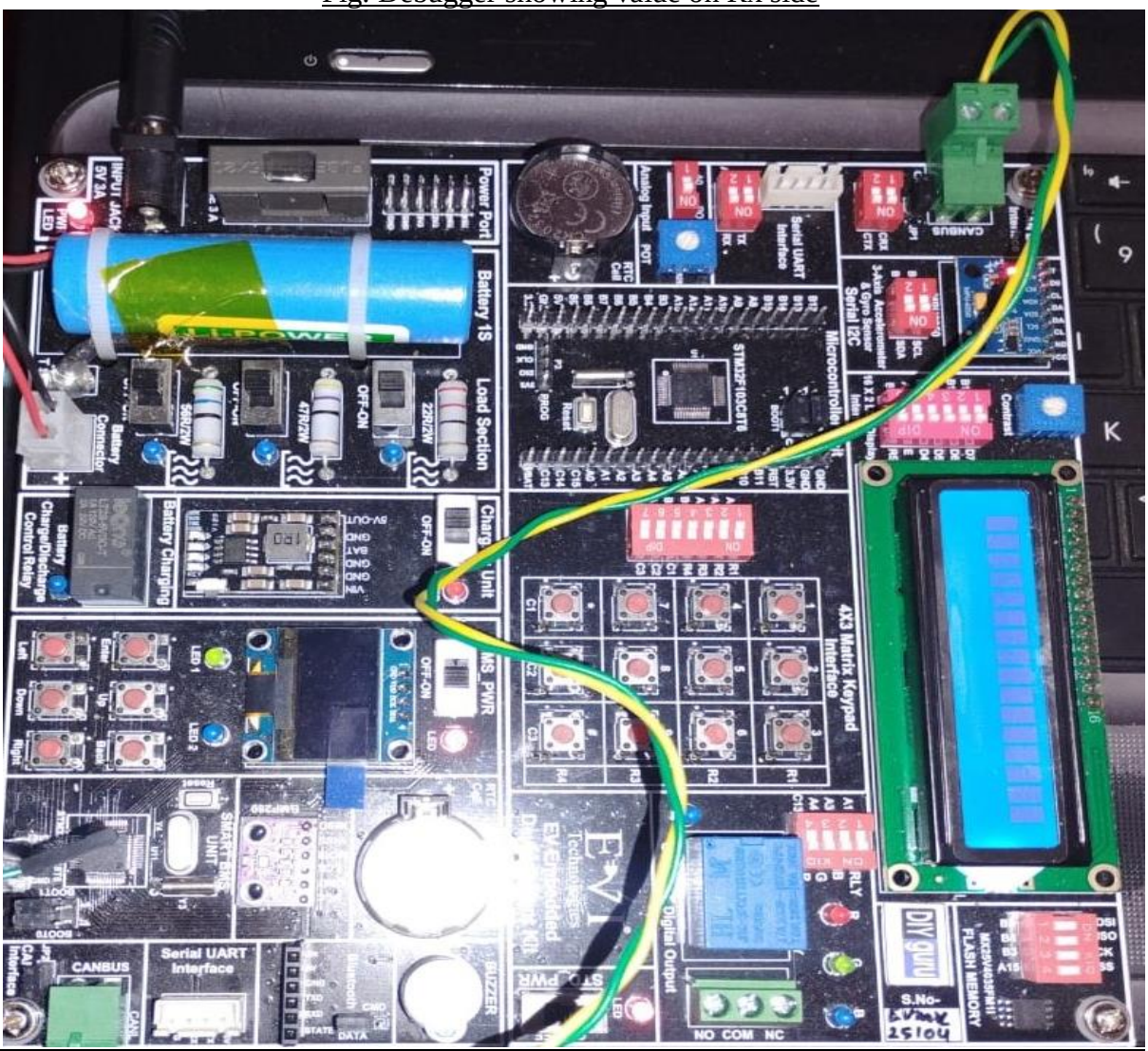


Fig.CAN H & CAN L wire connection between microcontrollers

Conclusion-

This project successfully provide us with hands-on experience in implementing the CAN (Controller Area Network) protocol on an STM32 microcontroller. Through practical configuration, message transmission, and reception, students gained a clear understanding of how reliable, real-time communication is achieved between multiple microcontroller nodes.

PROJECT DESCRIPTION

Combined I2C & SPI Protocol Implementation for Sensor Integration on STM32.

Project Objective :-

This project combines both I2C and SPI protocols for sensor integration, teaching students how to interface with multiple sensors using both communication protocols on the STM32 platform. Students will work with the MPU6050 (I2C) and BMP280 (SPI) sensors.

I2C Protocol Theory -

I2C is a serial communication protocol developed by Philips (now NXP) that allows multiple digital devices to communicate using only two wires.

The two lines-

1. SDA (Serial Data Line): Carries the data
2. SCL (Serial Clock Line): Carries the clock signal (generated by the master)

Both lines are open- drain, meaning devices only pull the line low. Pull-up resistors bring the lines high when no device is driving them. Role of I2C in Connecting Low-Speed Devices (e.g. Sensors) . I2C is especially suited for low-speed, short-distance communication inside electronic systems. It is ideal for sensors because (i) Simple wiring – only two wires regardless of the number of devices (ii) Multiple devices on one bus – each device has a unique address (iii) Low power consumption - Enough speed for sensors (temperature, pressure, humidity, accelerometers, RTCs, EEPROMs).

Typical applications-

- Temperature and humidity sensors
- Light and proximity sensors
- EEPROM memory chips
- Real-time clocks (RTC)
- LCD/OLED displays
- Speed modes- (i) Standard mode:100 kbps (ii)Fast mode: 400 kbps (iii)Fast mode Plus:1 Mbps (iv)High-speed mode: up to 3.4 Mbps (For most sensors, 100–400 kbps is more than sufficient.)
- Key Benefit: Provides real time motion and orientation data, essential for vehicle safety and control systems.

I2C Master–Slave Communication

- Master: Controls the bus and generates the clock (usually a microcontroller)
- Slave: Responds to the master (sensor, memory, etc.)

  Only the master can start or stop a communication.

I2C Data Frame Structure

An I2C communication happens in a well-defined sequence called a data frame.

1. **Start Condition (S)** - Occurs when SDA goes low while SCL is high, Signals the beginning of communication
2. **Address Frame** - 7-bit (or 10-bit) slave address, 1 R/W bit,
   `0` → Write (master sends data), `1` → Read (master receives data Example :- [Slave Address (7 bits)] + [R/W bit]
3. **Acknowledge Bit (ACK/NACK)** - After every byte (8 bits), the receiver sends:
   - ACK (0): Data received successfully
   - NACK (1): Data not accepted or end of transfer
4. **Data Bytes** - Each data byte is 8 bits, Can be one or many bytes, Each byte is followed by an ACK/NACK
5. **Stop Condition (P)** - Occurs when SDA goes high while SCL is high, Signals the end of communication

Key Advantages and Limitations

Advantages –

- Simple hardware
- Supports multiple devices
- Low cost and low power

Limitations –

- Slower than SPI
- Short communication distance
- Address conflicts if devices share the same address

SPI Protocol Theory -

SPI (Serial Peripheral Interface) is a synchronous serial communication protocol used to exchange data between a master (usually a microcontroller) and one or more slave/peripheral devices (sensors, displays, memory chips, etc.).

It was designed for short-distance, high-speed communication on a circuit board.

SPI Signals (Lines)

SPI typically uses 4 wires:

1. SCLK (Serial Clock)- Clock generated by the master to synchronize data transfer.
2. MOSI (Master Out, Slave In)- Data sent from the master to the slave.

3. MISO (Master In, Slave Out)- Data sent from the slave to the master.
4. SS / CS (Slave Select / Chip Select)-Active-low signal used to select which slave is communicating, Each slave usually has its own CS line.

How SPI Works (Basic Operation)

1. Master pulls CS low to select a slave.
2. Master generates the clock (SCLK).
3. Data is shifted: Master sends data on MOSI, Slave sends data on MISO.
4. Data is sampled on clock edges.
5. CS goes high to end the communication.

- SPI supports full-duplex communication, meaning sending and receiving happen at the same time.

- **SPI Is Faster than I2C because**
  (i) More Data Lines - Separate lines for sending (MOSI) and receiving (MISO), I2C: Single bidirectional data line (SDA), SPI can transmit and receive simultaneously
  (ii) No Address Overhead - I²C sends device addresses before data, SPI uses CS lines instead of addresses, No extra bits = faster data transfer.
  (iii) Higher Clock Speeds - Typical speeds:

  I2C- Standard mode: 100 kHz, Fast mode: 400 kHz, Fast+ / HS: up to ~3.4 MHz

  SPI- Commonly 10–50 MHz, Some devices support even higher

  (iv) Simpler Protocol-

  SPI has: No start/stop conditions, No acknowledgments (ACK/NACK),

  I2C has both → Less protocol overhead → faster communication.

SPI Data Frame Structure

SPI does not have a fixed frame format like UART or I²C. The data frame is defined by the device.

<u>Key Advantages and Limitations</u>

Advantages -

- Very fast

- Simple hardware
- Full-duplex
- Low protocol overhead

Limitations-

- Uses more pins than I²C
- No built-in error checking
- Short-distance communication only

Common Use

- SD cards
- Flash memory
- ADCs and DACs
- Displays (TFT, OLED)
- High-speed sensors

MPU6050 Sensor:

The MPU6050 is a 6-axis MEMS (micro electronics mechanical system) motion sensor that combines

- 3-axis accelerometer (X, Y, Z linear acceleration)
- 3-axis gyroscope (angular velocity / rotation)

It communicates with a microcontroller using the I2C protocol.

Accelerometer- Measures acceleration caused by:

- Vehicle movement
- Braking and acceleration
- Road inclination (gravity component)

Gyroscope-Measures angular velocity helping detect

- Vehicle rotation
- Turning and yaw motion
- Sudden directional changes

The sensor outputs digital data over I2C, making it easy to interface with automotive controllers.

Automotive Applications of MPU6050

- Motion Sensing & Vehicle Dynamics
- Electronic Stability Control (ESC)
- Vehicle tilt and rollover detection
- Inertial Navigation Systems (INS)
- Advanced Driver Assistance Systems (ADAS)
- Suspension monitoring
- Crash and impact detection

BMP280 Barometric Pressure Sensor

The BMP280 is a high-precision barometric pressure and temperature sensor developed for:

- Atmospheric pressure measurement
- Altitude estimation
- Environmental monitoring

It supports both SPI and I2C, but SPI is preferred in high-speed, noise-prone automotive environments.

Working-

- Measures air pressure using a MEMS pressure diaphragm.
- Converts pressure changes into digital data.
- Can estimate altitude changes using pressure variation.
- Includes an internal temperature sensor for compensation

Automotive Applications of BMP280

- Environmental Monitoring
- Cabin pressure monitoring
- Altitude detection (mountain driving, tunnels)
- Climate control optimization (HVAC systems)
- Engine air intake and efficiency analysis
- Weather-aware vehicle systems
- Key Benefit: Accurate pressure and temperature data helps improve comfort, safety, and fuel efficiency.

Implementation of I2C communication protocol with MPU6050 to capture accelerometer & gyroscope data

First we will select the MCU/MPU STM32F103C8 microcontroller.

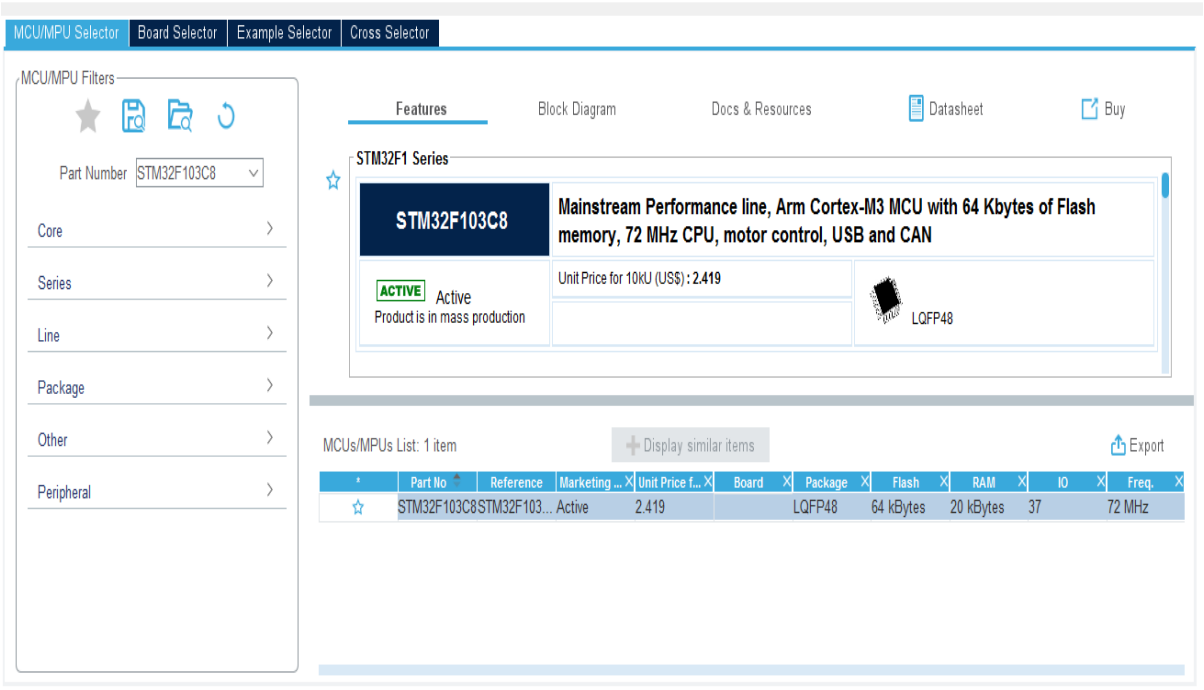


Fig. MPU/MCU Selection in STM32 cube IDE

We will be doing settings in .ioc file like system core, connectivity we will select I2C and to display on LCD we will be selecting few GPIO pins.

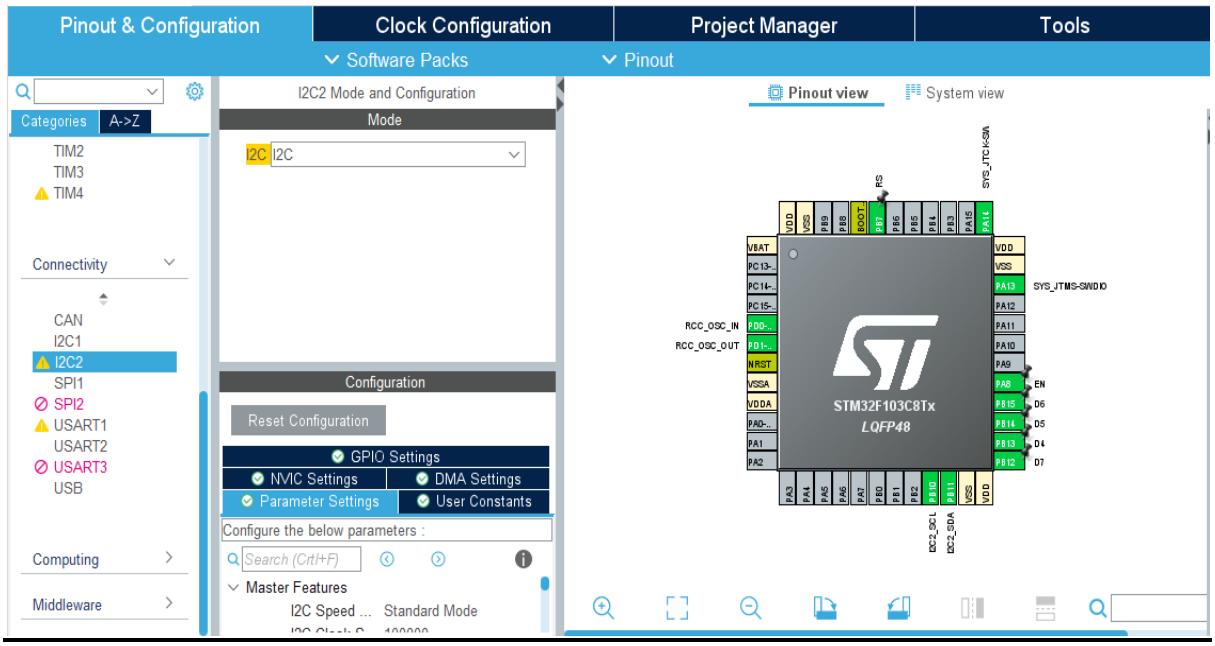


Fig. Setting to done on .ioc file

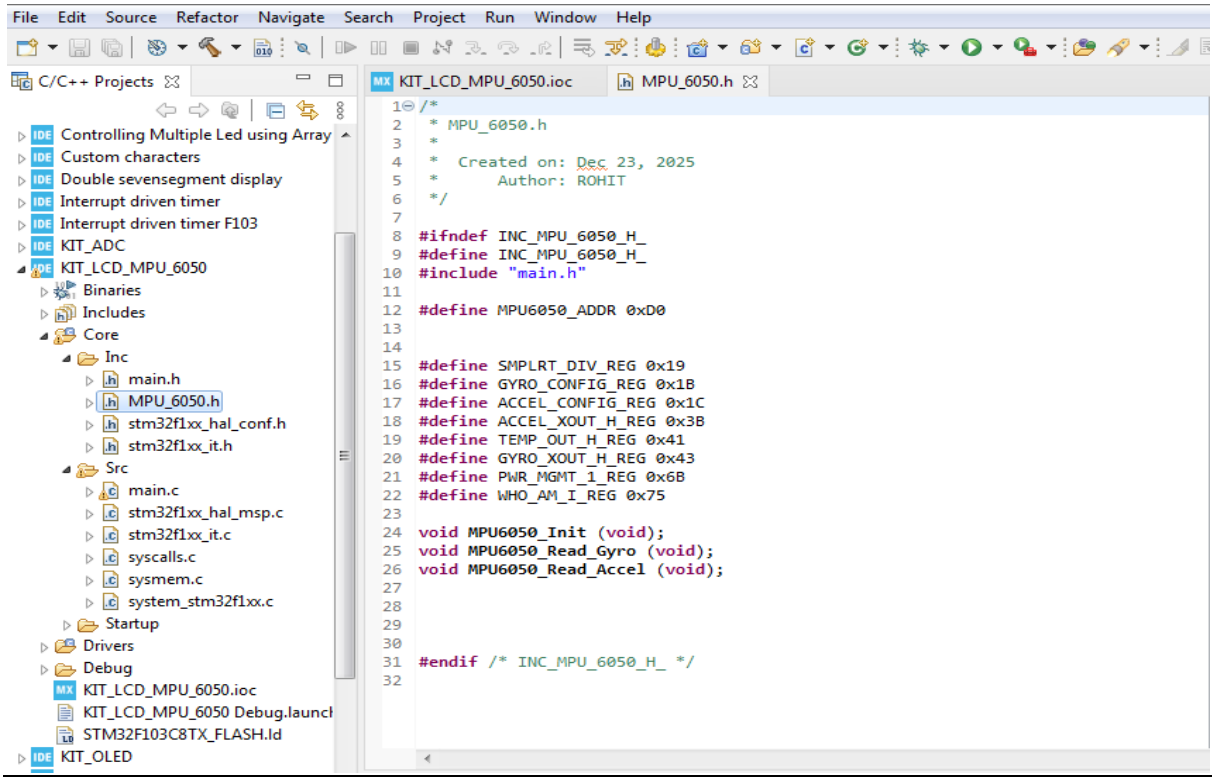


Fig. In Inc file to be added

Coding to be done in main.c file

```
/* USER CODE BEGIN Header */
/**
  ******************************************************************************
  * @file           : main.c
  * @brief          : Main program body
  ******************************************************************************
  * @attention
  *
  * <h2><center>&copy; Copyright (c) 2025 STMicroelectronics.
  * All rights reserved.</center></h2>
  *
  * This software component is licensed by ST under BSD 3-Clause license,
  * the "License"; You may not use this file except in compliance with the
  * License. You may obtain a copy of the License at:
  *                        opensource.org/licenses/BSD-3-Clause
  *
  ******************************************************************************
  */
/* USER CODE END Header */
/* Includes ------------------------------------------------------------------*/
#include "main.h"

/* Private includes ----------------------------------------------------------*/
/* USER CODE BEGIN Includes */
#include "MPU_6050.h"
#include  "stdio.h"

/* USER CODE END Includes */
```

```c
/* Private typedef -----------------------------------------------------------*/
/* USER CODE BEGIN PTD */

/* USER CODE END PTD */

/* Private define ------------------------------------------------------------*/
/* USER CODE BEGIN PD */
/* USER CODE END PD */

/* Private macro -------------------------------------------------------------*/
/* USER CODE BEGIN PM */
#define MPU6050_ADDR 0xD0


#define SMPLRT_DIV_REG 0x19
#define GYRO_CONFIG_REG 0x1B
#define ACCEL_CONFIG_REG 0x1C
#define ACCEL_XOUT_H_REG 0x3B
#define TEMP_OUT_H_REG 0x41
#define GYRO_XOUT_H_REG 0x43
#define PWR_MGMT_1_REG 0x6B
#define WHO_AM_I_REG 0x75




/* USER CODE END PM */

/* Private variables ---------------------------------------------------------*/
I2C_HandleTypeDef hi2c2;

TIM_HandleTypeDef htim1;

/* USER CODE BEGIN PV */
int16_t Accel_X_RAW = 0;
int16_t Accel_Y_RAW = 0;
int16_t Accel_Z_RAW = 0;

int16_t Gyro_X_RAW = 0;
int16_t Gyro_Y_RAW = 0;
int16_t Gyro_Z_RAW = 0;

float Ax, Ay, Az, Gx, Gy, Gz;
char arr1[10];

#define timer htim1
/* USER CODE END PV */

/* Private function prototypes -----------------------------------------------*/
void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_I2C2_Init(void);
static void MX_TIM1_Init(void);
/* USER CODE BEGIN PFP */
void lcd_init (void);
void lcd_send_cmd(char cmd);
void lcd_send_data(char data);
void lcd_send_string(char *str);
```

```c
void lcd_put_cur(int row, int col);
void lcd_clear (void);
/* USER CODE END PFP */

/* Private user code ---------------------------------------------------------*/
/* USER CODE BEGIN 0 */

/* USER CODE END 0 */

/**
  * @brief  The application entry point.
  * @retval int
  */
int main(void)
{
  /* USER CODE BEGIN 1 */

  /* USER CODE END 1 */

  /* MCU Configuration--------------------------------------------------------*/

  /* Reset of all peripherals, Initializes the Flash interface and the Systick. */
  HAL_Init();

  /* USER CODE BEGIN Init */

  /* USER CODE END Init */

  /* Configure the system clock */
  SystemClock_Config();

  /* USER CODE BEGIN SysInit */

  /* USER CODE END SysInit */

  /* Initialize all configured peripherals */
  MX_GPIO_Init();
  MX_I2C2_Init();
  MX_TIM1_Init();
  /* USER CODE BEGIN 2 */
  HAL_TIM_Base_Start(&htim1);
    lcd_init();
    lcd_clear();
  MPU6050_Init();
  HAL_Delay(100);


  // Test WHO_AM_I on LCD first
//  uint8_t whoami;
//  HAL_I2C_Mem_Read(&hi2c2, MPU6050_ADDR, 0x75, 1, &whoami, 1, 1000);
//  lcd_put_cur(0,0);
//  sprintf(arr1, "WHO=0x%02X", whoami);  // Should show 0x70
//  lcd_send_string(arr1);


  /* USER CODE END 2 */

  /* Infinite loop */
  /* USER CODE BEGIN WHILE */
```

```c
  while (1)
  {
          MPU6050_Read_Accel();
             MPU6050_Read_Gyro();
//           lcd_put_cur(0,0);
//           sprintf(arr1, "Ax=%.2f Ay=%.2f", Ax, Ay);
//           lcd_send_string(arr1);
//
//           lcd_put_cur(1,0);
//           sprintf(arr1, "Az=%.2f", Az);
//           lcd_send_string(arr1);
//           HAL_Delay(1000);

             lcd_put_cur(0,0);
             sprintf(arr1, "Gx=%.2f Gy=%.2f", Gx, Gy);
             lcd_send_string(arr1);

             lcd_put_cur(1,8);
             sprintf(arr1, "Gz=%.2f", Gz);
             lcd_send_string(arr1);
             HAL_Delay(1000);

//       lcd_put_cur(0, 0);
//       lcd_send_string("ANKIT KUMAR");
//       lcd_put_cur(1, 0);
//       lcd_send_string("EMBEDDED ENG.");
    /* USER CODE END WHILE */

    /* USER CODE BEGIN 3 */
  }
  /* USER CODE END 3 */
}

/**
  * @brief System Clock Configuration
  * @retval None
  */
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_HSE;
  RCC_OscInitStruct.HSEState = RCC_HSE_ON;
  RCC_OscInitStruct.HSEPredivValue = RCC_HSE_PREDIV_DIV1;
  RCC_OscInitStruct.HSIState = RCC_HSI_ON;
  RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
  RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSE;
  RCC_OscInitStruct.PLL.PLLMUL = RCC_PLL_MUL9;
  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_PLLCLK;
  RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
  RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV2;
  RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV1;

  if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_2) != HAL_OK)
  {
    Error_Handler();
  }
}

/**
  * @brief I2C2 Initialization Function
  * @param None
  * @retval None
  */
static void MX_I2C2_Init(void)
{

  /* USER CODE BEGIN I2C2_Init 0 */

  /* USER CODE END I2C2_Init 0 */

  /* USER CODE BEGIN I2C2_Init 1 */

  /* USER CODE END I2C2_Init 1 */
  hi2c2.Instance = I2C2;
  hi2c2.Init.ClockSpeed = 100000;
  hi2c2.Init.DutyCycle = I2C_DUTYCYCLE_2;
  hi2c2.Init.OwnAddress1 = 0;
  hi2c2.Init.AddressingMode = I2C_ADDRESSINGMODE_7BIT;
  hi2c2.Init.DualAddressMode = I2C_DUALADDRESS_DISABLE;
  hi2c2.Init.OwnAddress2 = 0;
  hi2c2.Init.GeneralCallMode = I2C_GENERALCALL_DISABLE;
  hi2c2.Init.NoStretchMode = I2C_NOSTRETCH_DISABLE;
  if (HAL_I2C_Init(&hi2c2) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN I2C2_Init 2 */

  /* USER CODE END I2C2_Init 2 */

}

/**
  * @brief TIM1 Initialization Function
  * @param None
  * @retval None
  */
static void MX_TIM1_Init(void)
{

  /* USER CODE BEGIN TIM1_Init 0 */

  /* USER CODE END TIM1_Init 0 */

  TIM_ClockConfigTypeDef sClockSourceConfig = {0};
```

```
  TIM_MasterConfigTypeDef sMasterConfig = {0};

  /* USER CODE BEGIN TIM1_Init 1 */

  /* USER CODE END TIM1_Init 1 */
  htim1.Instance = TIM1;
  htim1.Init.Prescaler = 72-1;
  htim1.Init.CounterMode = TIM_COUNTERMODE_UP;
  htim1.Init.Period = 65535;
  htim1.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
  htim1.Init.RepetitionCounter = 0;
  htim1.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_DISABLE;
  if (HAL_TIM_Base_Init(&htim1) != HAL_OK)
  {
    Error_Handler();
  }
  sClockSourceConfig.ClockSource = TIM_CLOCKSOURCE_INTERNAL;
  if (HAL_TIM_ConfigClockSource(&htim1, &sClockSourceConfig) != HAL_OK)
  {
    Error_Handler();
  }
  sMasterConfig.MasterOutputTrigger = TIM_TRGO_RESET;
  sMasterConfig.MasterSlaveMode = TIM_MASTERSLAVEMODE_DISABLE;
  if (HAL_TIMEx_MasterConfigSynchronization(&htim1, &sMasterConfig) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN TIM1_Init 2 */

  /* USER CODE END TIM1_Init 2 */

}

/**
  * @brief GPIO Initialization Function
  * @param None
  * @retval None
  */
static void MX_GPIO_Init(void)
{
  GPIO_InitTypeDef GPIO_InitStruct = {0};

  /* GPIO Ports Clock Enable */
  __HAL_RCC_GPIOD_CLK_ENABLE();
  __HAL_RCC_GPIOB_CLK_ENABLE();
  __HAL_RCC_GPIOA_CLK_ENABLE();

  /*Configure GPIO pin Output Level */
  HAL_GPIO_WritePin(GPIOB, D7_Pin|D4_Pin|D5_Pin|D6_Pin
                          |RS_Pin, GPIO_PIN_RESET);

  /*Configure GPIO pin Output Level */
  HAL_GPIO_WritePin(EN_GPIO_Port, EN_Pin, GPIO_PIN_RESET);

  /*Configure GPIO pins : D7_Pin D4_Pin D5_Pin D6_Pin
                           RS_Pin */
  GPIO_InitStruct.Pin = D7_Pin|D4_Pin|D5_Pin|D6_Pin
                          |RS_Pin;
  GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
```

```
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
  HAL_GPIO_Init(GPIOB, &GPIO_InitStruct);

  /*Configure GPIO pin : EN_Pin */
  GPIO_InitStruct.Pin = EN_Pin;
  GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
  HAL_GPIO_Init(EN_GPIO_Port, &GPIO_InitStruct);

}

/* USER CODE BEGIN 4 */
void MPU6050_Init (void)
{
	uint8_t check;
	uint8_t Data;

	// check device ID WHO_AM_I
	HAL_I2C_Mem_Read (&hi2c2, MPU6050_ADDR, 0x75, 1, &check, 1, 1000);
	if (check == 0x70)  // 0x70 will be returned by the sensor if everything
goes well
	{
		// power management register 0X6B we should write all 0's to wake
the sensor up
		Data = 0;
		HAL_I2C_Mem_Write(&hi2c2, MPU6050_ADDR, 0x6B, 1,&Data, 1, 1000);

		// Set DATA RATE of 1KHz by writing SMPLRT_DIV register
		Data = 0x07;
		HAL_I2C_Mem_Write(&hi2c2, MPU6050_ADDR, 0x19, 1, &Data, 1, 1000);

		// Set Gyroscopic configuration in GYRO_CONFIG Register
		Data = 0x00;  // XG_ST=0,YG_ST=0,ZG_ST=0, FS_SEL=0 -> ± 250 ?/s
		HAL_I2C_Mem_Write(&hi2c2, MPU6050_ADDR, 0x1B, 1, &Data, 1, 1000);

		// Set accelerometer configuration in ACCEL_CONFIG Register
		Data = 0x00;  // XA_ST=0,YA_ST=0,ZA_ST=0, FS_SEL=0 -> ± 2g
		HAL_I2C_Mem_Write(&hi2c2, MPU6050_ADDR, 0x1C, 1, &Data, 1, 1000);
	}

}



void MPU6050_Read_Accel (void)
{
	uint8_t Rec_Data[6];

	// Read 6 BYTES of data starting from ACCEL_XOUT_H (0x3B) register
	HAL_I2C_Mem_Read (&hi2c2, MPU6050_ADDR, 0x3B, 1, Rec_Data, 6, 1000);

	Accel_X_RAW = (int16_t)(Rec_Data[0] << 8 | Rec_Data [1]);
	Accel_Y_RAW = (int16_t)(Rec_Data[2] << 8 | Rec_Data [3]);
	Accel_Z_RAW = (int16_t)(Rec_Data[4] << 8 | Rec_Data [5]);

	/*** convert the RAW values into acceleration in 'g'
	     we have to divide according to the Full scale value set in FS_SEL
```

```
            I have configured FS_SEL = 0. So I am dividing by 16384.0
            for more details check ACCEL_CONFIG Register              ****/

        Ax = (float)Accel_X_RAW/16384.0;
        Ay = (float)Accel_Y_RAW/16384.0;
        Az = (float)Accel_Z_RAW/16384.0;
}









void MPU6050_Read_Gyro (void)
{
        uint8_t Rec_Data[6];

        // Read 6 BYTES of data starting from GYRO_XOUT_H register
        if (HAL_I2C_Mem_Read (&hi2c2, MPU6050_ADDR, 0x43, 1, Rec_Data, 6, 1000) ==
HAL_OK)
        {
                Gyro_X_RAW = (int16_t)(Rec_Data[0] << 8 | Rec_Data [1]);
                Gyro_Y_RAW = (int16_t)(Rec_Data[2] << 8 | Rec_Data [3]);
                Gyro_Z_RAW = (int16_t)(Rec_Data[4] << 8 | Rec_Data [5]);

                /*** convert the RAW values into dps (°/s)
                        we have to divide according to the Full scale value set in
FS_SEL
                        I have configured FS_SEL = 0. So I am dividing by 131.0
                        for more details check GYRO_CONFIG Register
****/

                Gx = (float)Gyro_X_RAW/131.0;
                Gy = (float)Gyro_Y_RAW/131.0;
                Gz = (float)Gyro_Z_RAW/131.0;
        }
}

void delay(uint16_t us)
{

__HAL_TIM_SET_COUNTER(&timer, 0);

while (__HAL_TIM_GET_COUNTER(&timer) < us);

}
void send_to_lcd(char data, int rs)
{
        HAL_GPIO_WritePin(RS_GPIO_Port,RS_Pin, rs);
        HAL_GPIO_WritePin(D7_GPIO_Port, D7_Pin, ((data>>3)&0x01));
        HAL_GPIO_WritePin(D6_GPIO_Port, D6_Pin, ((data>>2)&0x01));
        HAL_GPIO_WritePin(D5_GPIO_Port, D5_Pin, ((data>>1)&0x01));
        HAL_GPIO_WritePin(D4_GPIO_Port, D4_Pin, ((data>>0)&0x01));
```

```
	HAL_GPIO_WritePin(EN_GPIO_Port, EN_Pin,1);
	delay(20);
	HAL_GPIO_WritePin(EN_GPIO_Port, EN_Pin,0);
		delay(20);

}
void lcd_send_cmd(char cmd)
{
	char datatosend;
	datatosend=((cmd>>4)&0x0f);
	send_to_lcd(datatosend,0);
	datatosend=((cmd)&0x0f);
	send_to_lcd(datatosend,0);
}
void lcd_send_data(char data)
{
	char datatosend;
	datatosend=((data>>4)&0x0f);
	send_to_lcd(datatosend,1);
	datatosend=((data)&0x0f);
	send_to_lcd(datatosend,1);
}
void lcd_clear(void)
{
	lcd_send_cmd(0x01);
	HAL_Delay(2);
}
void lcd_put_cur(int row,int col)
{
	switch(row)
	{
	case 0:
	col |=0x80;
	break;
	case 1:
	col |=0xc0;
	break;
	}
	lcd_send_cmd(col);
}
void lcd_init(void)
{
	HAL_Delay(50);
	lcd_send_cmd(0x30);
	HAL_Delay(5);
	lcd_send_cmd (0x30);
	HAL_Delay(1);
	lcd_send_cmd (0x30);
	HAL_Delay(10);
	lcd_send_cmd (0x20);
	HAL_Delay(10);

	lcd_send_cmd(0x20);
	HAL_Delay(1);
	lcd_send_cmd (0x08);
	HAL_Delay(1);
	lcd_send_cmd (0x01);
	HAL_Delay(1);
```

```
	HAL_Delay(1);
	lcd_send_cmd (0x06);
	HAL_Delay(1);
	lcd_send_cmd (0x0c);

}
void lcd_send_string (char *str)
{
	while (*str) lcd_send_data (*str++);
}
/* 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****/
```

After flashing the code in the microcontroller STM32F103C8T6 we can see the reading of Angular velocity- Gx, Gy, Gz And Accleration Ax, Ay, Az.

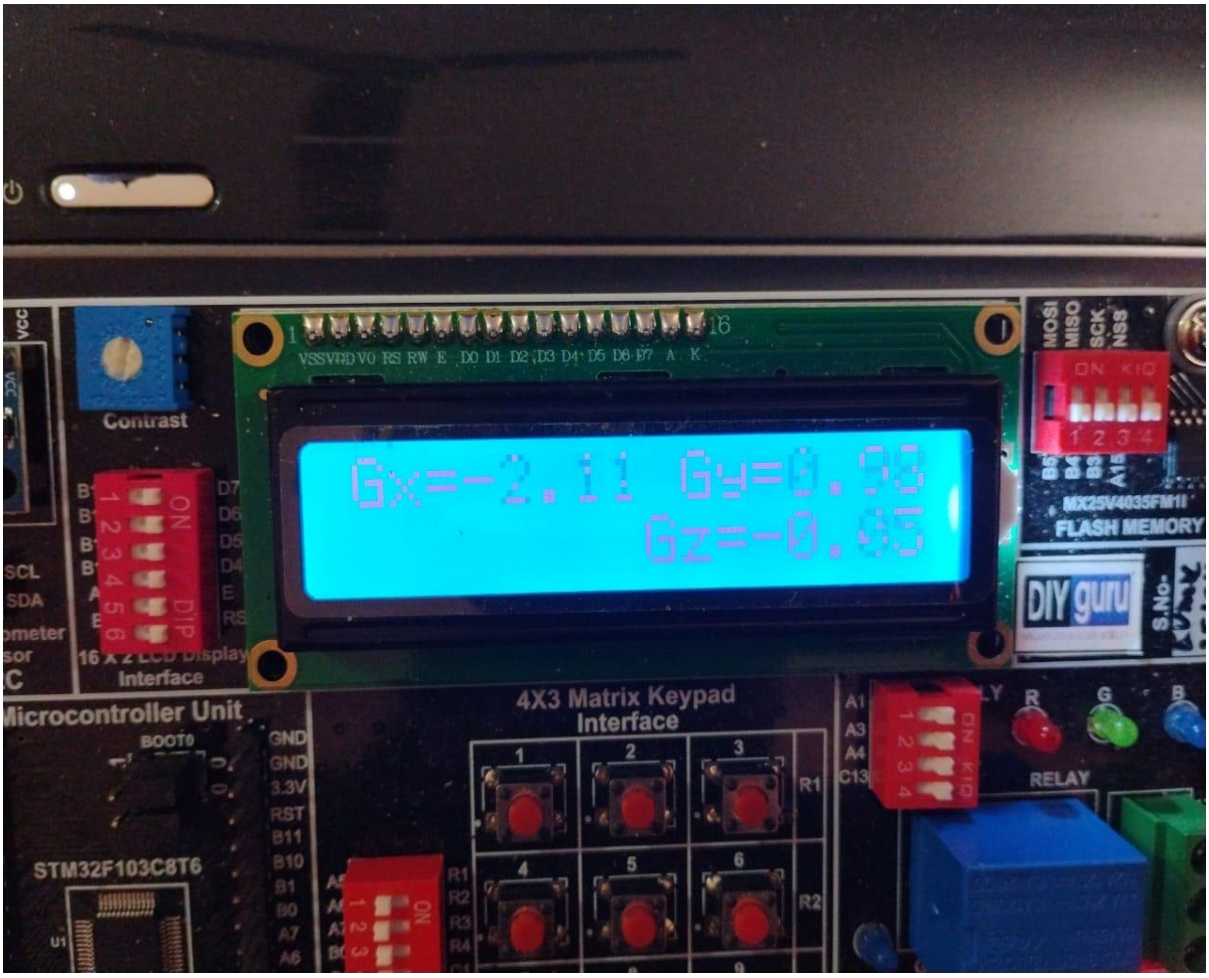


Fig. Angular velocity values printed on LCD

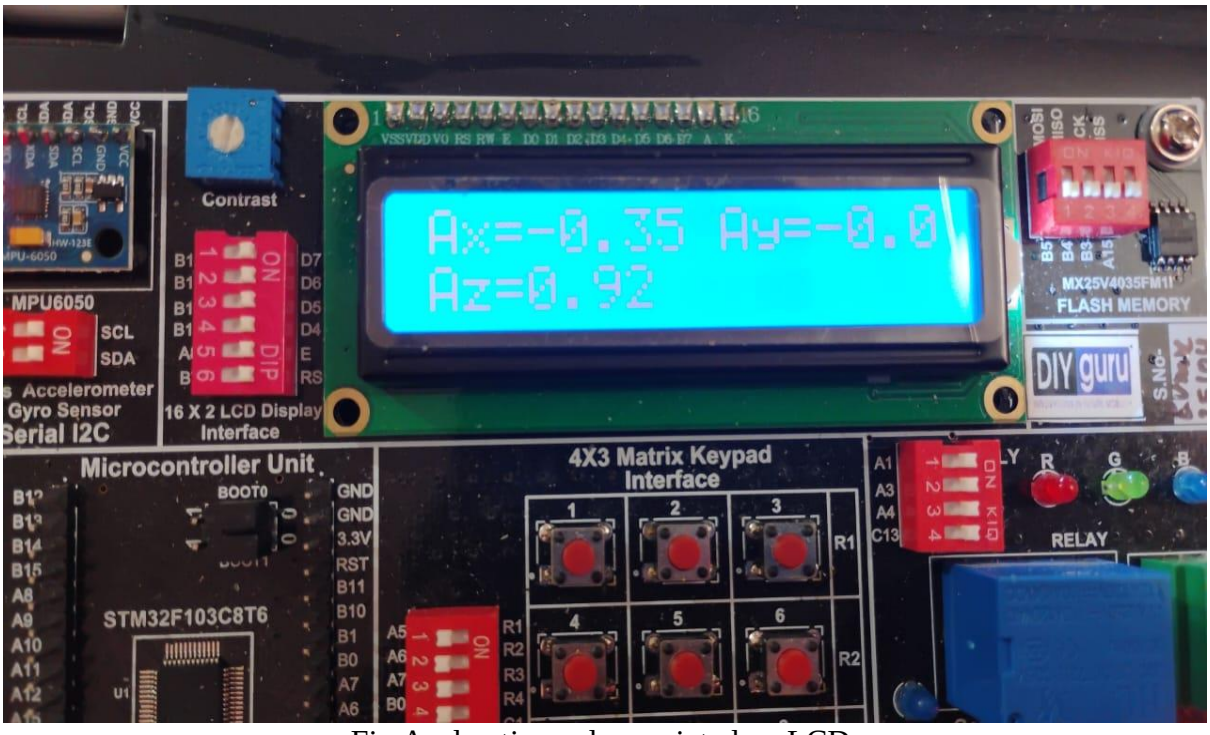


Fig.Accleration values printed on LCD

Implementation of SPI communication protocol with BMP280 to read temperature

First we will select the MCU/MPU STM32F103C8 microcontroller.

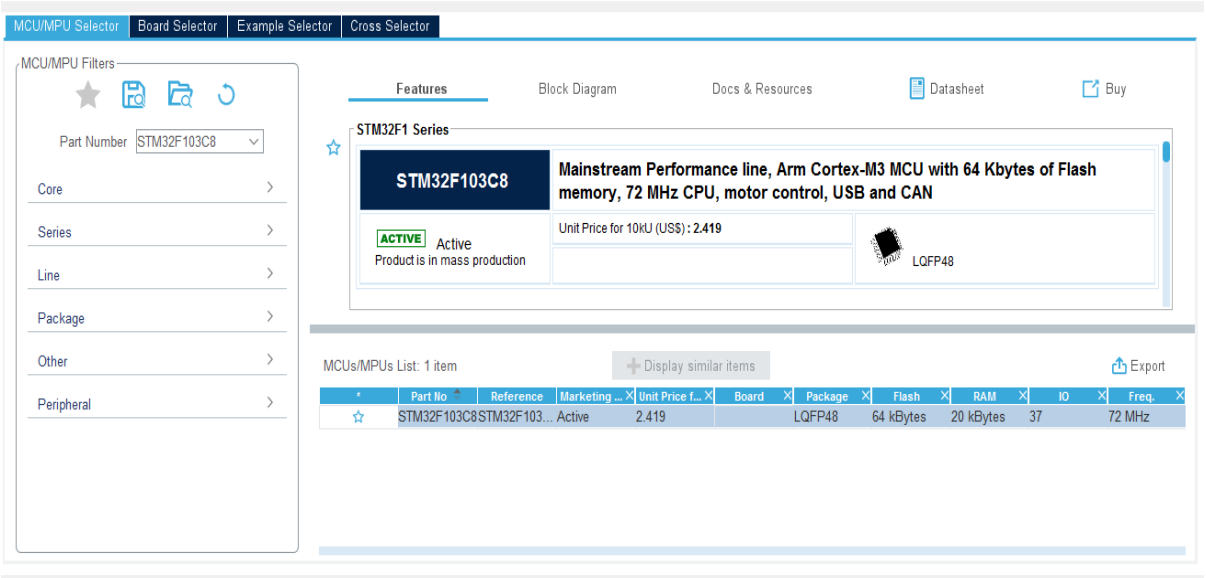


Fig. MPU/MCU Selection in STM32 cube IDE

We will be doing settings in .ioc file like system core, connectivity we will select I2C and SPI.

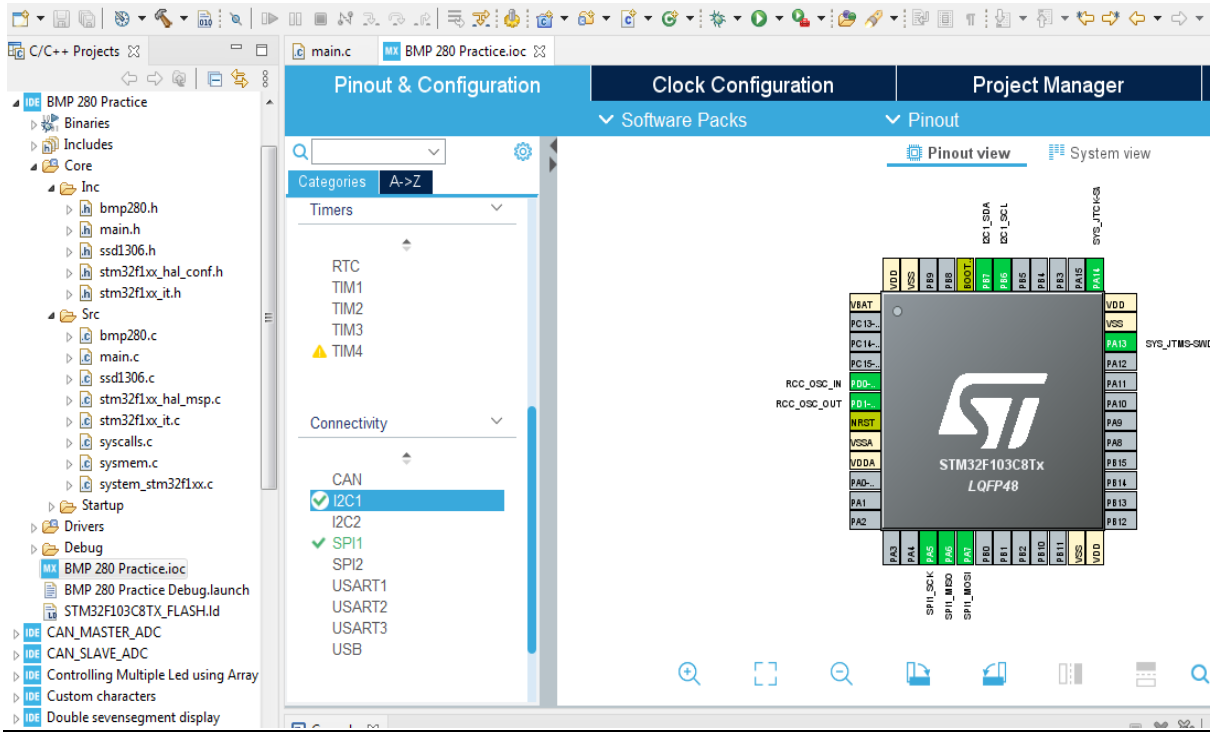


Fig. MPU/MCU Selection in STM32 cube IDE

Coding to be done in main.c file

```
/* USER CODE BEGIN Header */
/**
  ******************************************************************************
  * @file           : main.c
  * @brief          : Main program body
  ******************************************************************************
  * @attention
  *
  * <h2><center>&copy; Copyright (c) 2025 STMicroelectronics.
  * All rights reserved.</center></h2>
  *
  * This software component is licensed by ST under BSD 3-Clause license,
  * the "License"; You may not use this file except in compliance with the
  * License. You may obtain a copy of the License at:
  *                        opensource.org/licenses/BSD-3-Clause
  *
  ******************************************************************************
  */
/* USER CODE END Header */
/* Includes ------------------------------------------------------------------*/
#include "main.h"
extern SPI_HandleTypeDef hspi1;
extern I2C_HandleTypeDef hi2c1;
/* Private includes ----------------------------------------------------------*/
/* USER CODE BEGIN Includes */
#include "bmp280.h"
#include "ssd1306.h"
#include <stdio.h>

/* USER CODE END Includes */

/* Private typedef -----------------------------------------------------------*/
/* USER CODE BEGIN PTD */

/* USER CODE END PTD */

/* Private define ------------------------------------------------------------*/
/* USER CODE BEGIN PD */
/* USER CODE END PD */

/* Private macro -------------------------------------------------------------*/
/* USER CODE BEGIN PM */

/* USER CODE END PM */

/* Private variables ---------------------------------------------------------*/
I2C_HandleTypeDef hi2c1;

SPI_HandleTypeDef hspi1;

/* USER CODE BEGIN PV */

/* USER CODE END PV */

/* Private function prototypes -----------------------------------------------*/
void SystemClock_Config(void);
static void MX_GPIO_Init(void);
```

```c
static void MX_I2C1_Init(void);
static void MX_SPI1_Init(void);
/* USER CODE BEGIN PFP */

/* USER CODE END PFP */

/* Private user code ---------------------------------------------------------*/
/* USER CODE BEGIN 0 */

/* USER CODE END 0 */

/**
  * @brief  The application entry point.
  * @retval int
  */
int main(void)
{
  /* USER CODE BEGIN 1 */
      HAL_Init();
          SystemClock_Config();
          MX_GPIO_Init();
          MX_SPI1_Init();
          MX_I2C1_Init();

          BMP280_Init(&hspi1);
          SSD1306_Init(&hi2c1);

          char text[32];
          float temp;
  /* USER CODE END 1 */

  /* MCU Configuration--------------------------------------------------------*/

  /* Reset of all peripherals, Initializes the Flash interface and the Systick. */
  HAL_Init();

  /* USER CODE BEGIN Init */

  /* USER CODE END Init */

  /* Configure the system clock */
  SystemClock_Config();

  /* USER CODE BEGIN SysInit */

  /* USER CODE END SysInit */

  /* Initialize all configured peripherals */
  MX_GPIO_Init();
  MX_I2C1_Init();
  MX_SPI1_Init();
  /* USER CODE BEGIN 2 */

  /* USER CODE END 2 */

  /* Infinite loop */
  /* USER CODE BEGIN WHILE */
  while (1)
  {
```

```
        temp = BMP280_ReadTemperature(&hspi1);

                SSD1306_Clear();
                sprintf(text, "Battey Temp:");
                SSD1306_Print(text);

                sprintf(text, "%.2f C", temp);
                SSD1306_Print(text);

                SSD1306_UpdateScreen(&hi2c1);
                HAL_Delay(1000);
    /* USER CODE END WHILE */

    /* USER CODE BEGIN 3 */
  }
  /* USER CODE END 3 */
}

/**
  * @brief System Clock Configuration
  * @retval None
  */
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_HSE;
  RCC_OscInitStruct.HSEState = RCC_HSE_ON;
  RCC_OscInitStruct.HSEPredivValue = RCC_HSE_PREDIV_DIV1;
  RCC_OscInitStruct.HSIState = RCC_HSI_ON;
  RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
  RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSE;
  RCC_OscInitStruct.PLL.PLLMUL = RCC_PLL_MUL9;
  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_PLLCLK;
  RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
  RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV2;
  RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV1;

  if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_2) != HAL_OK)
  {
    Error_Handler();
  }
}

/**
  * @brief I2C1 Initialization Function
  * @param None
```

```c
  * @retval None
  */
static void MX_I2C1_Init(void)
{

  /* USER CODE BEGIN I2C1_Init 0 */

  /* USER CODE END I2C1_Init 0 */

  /* USER CODE BEGIN I2C1_Init 1 */

  /* USER CODE END I2C1_Init 1 */
  hi2c1.Instance = I2C1;
  hi2c1.Init.ClockSpeed = 100000;
  hi2c1.Init.DutyCycle = I2C_DUTYCYCLE_2;
  hi2c1.Init.OwnAddress1 = 0;
  hi2c1.Init.AddressingMode = I2C_ADDRESSINGMODE_7BIT;
  hi2c1.Init.DualAddressMode = I2C_DUALADDRESS_DISABLE;
  hi2c1.Init.OwnAddress2 = 0;
  hi2c1.Init.GeneralCallMode = I2C_GENERALCALL_DISABLE;
  hi2c1.Init.NoStretchMode = I2C_NOSTRETCH_DISABLE;
  if (HAL_I2C_Init(&hi2c1) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN I2C1_Init 2 */

  /* USER CODE END I2C1_Init 2 */

}

/**
  * @brief SPI1 Initialization Function
  * @param None
  * @retval None
  */
static void MX_SPI1_Init(void)
{

  /* USER CODE BEGIN SPI1_Init 0 */

  /* USER CODE END SPI1_Init 0 */

  /* USER CODE BEGIN SPI1_Init 1 */

  /* USER CODE END SPI1_Init 1 */
  /* SPI1 parameter configuration*/
  hspi1.Instance = SPI1;
  hspi1.Init.Mode = SPI_MODE_MASTER;
  hspi1.Init.Direction = SPI_DIRECTION_2LINES;
  hspi1.Init.DataSize = SPI_DATASIZE_8BIT;
  hspi1.Init.CLKPolarity = SPI_POLARITY_LOW;
  hspi1.Init.CLKPhase = SPI_PHASE_1EDGE;
  hspi1.Init.NSS = SPI_NSS_SOFT;
  hspi1.Init.BaudRatePrescaler = SPI_BAUDRATEPRESCALER_128;
  hspi1.Init.FirstBit = SPI_FIRSTBIT_MSB;
  hspi1.Init.TIMode = SPI_TIMODE_DISABLE;
  hspi1.Init.CRCCalculation = SPI_CRCCALCULATION_DISABLE;
  hspi1.Init.CRCPolynomial = 10;
```

```c
  if (HAL_SPI_Init(&hspi1) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN SPI1_Init 2 */

  /* USER CODE END SPI1_Init 2 */

}

/**
  * @brief GPIO Initialization Function
  * @param None
  * @retval None
  */
static void MX_GPIO_Init(void)
{

  /* GPIO Ports Clock Enable */
  __HAL_RCC_GPIOD_CLK_ENABLE();
  __HAL_RCC_GPIOA_CLK_ENABLE();
  __HAL_RCC_GPIOB_CLK_ENABLE();

}

/* 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****/
```

After flashing the code in the microcontroller STM32F103C8T6 we can see the reading of Battery temp

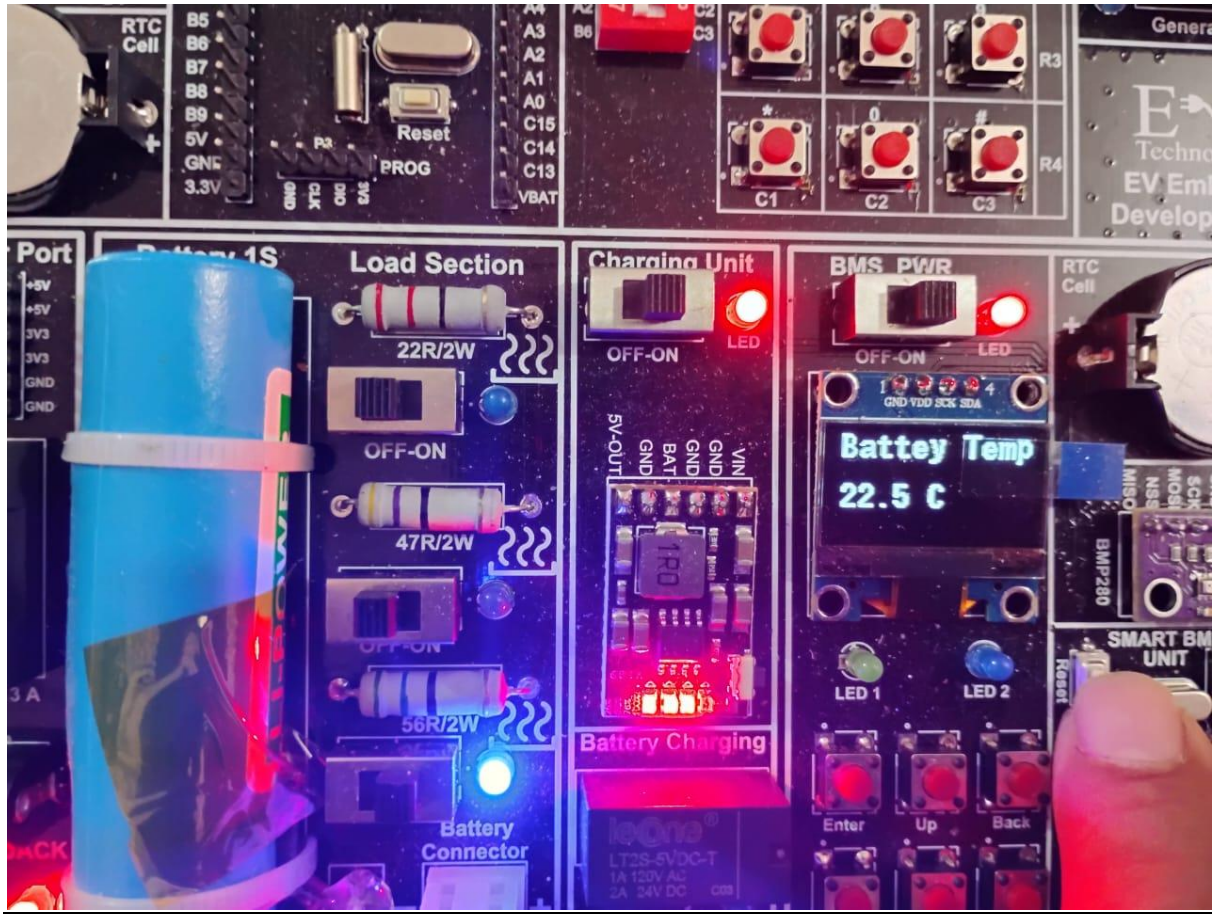


Fig. Battery Temp values printed on LCD

Conclusion-

This project successfully demonstrates the integration of multiple sensors using both I2C and SPI communication protocols on the STM32 platform. By interfacing the MPU6050 through I2C and the BMP280 through SPI.