Board-to-Board Communication using I2C between STM32 and Rugged Board

Introduction:-

This project focuses on establishing board-to-board communication between two embedded systems: the STM32 microcontroller and the Rugged Board, using the I2C (Inter-Integrated Circuit) protocol. The STM32 will be the master device, and the Rugged Board will act as the slave. I2C is a widely used communication protocol for connecting multiple devices over short distances. This project will specifically use STM32's I2C pins (D14 for SDA and D15 for SCL) to communicate with the Rugged Board's I2C pins through the MicroBus interface.

System Overview:-

STM32 Setup:

- SCL Pin (PB6): The data line for I2C communication.
- SDA Pin (PB7): The clock line for I2C communication.
- I2C Protocol: The STM32 acts as the master and initiates communication, sending data to the Rugged Board, which acts as the slave.

Rugged Board Setup:

- Expansion header I2C Pins: Rugged Board provides I2C pins (SDA, SCL) that are used for communication with the STM32.
- SCL is a GPIO_31 and SDA is a GPIO_32.
- Communication Mode: Slave mode, where the Rugged Board listens for data from the STM32 and responds accordingly.

Communication Protocol: I2C

I2C (Inter-Integrated Circuit) is a simple, two-wire communication protocol that allows multiple devices to communicate using a master-slave configuration. It uses two lines:

- **SDA (Serial Data Line)**: Carries data to/from the devices.

- **SCL (Serial Clock Line)**: Synchronizes the data transfer by providing a clock signal.

The STM32 will communicate with the Rugged Board using these two lines, where:

- The **STM32** will initiate data transfer (Master).
- The **Rugged Board** will respond to the data transfer (Slave).

Hardware Requirements:-

STM32:

- Microcontroller: STM32 (e.g., STM32F4 series or STM32L4)
- Pins Used:
  - PB7 (SDA): Data line for I2C.
  - PB6 (SCL): Clock line for I2C.

Rugged Board:

- Microcontroller: Rugged Board microcontroller (typically based on ARM architecture).
- MicroBus Interface: Use MicroBus pins for I2C communication (SDA, SCL).
- Pins Used:
  - SDA (MicroBus I2C): Data line for communication with STM32.
  - SCL (MicroBus I2C): Clock line for communication with STM32.

Wiring:

- STM32 PB7 → Rugged Board SDA (MicroBus)
- STM32 PB6 → Rugged Board SCL (MicroBus)
- Common Ground: Connect ground pins of both boards.

Software Requirements:-

- STM32 Software:
  - IDE: STM32CubeIDE or KEIL uVision.
  - Libraries: STM32 HAL (Hardware Abstraction Layer) for I2C communication.
  - Firmware: Write firmware to configure STM32 as the I2C master.
- Rugged Board Software:
  - Operating System: Linux-based OS (e.g., Debian, Ubuntu).

- Programming Language: Python (using `smbus2 as smbus` library for I2C communication.
- Libraries: `smbus` for interacting with I2C on the Rugged Board.

Communication Flow

Master (STM32) Side:

- Initialization:
  The STM32 configures I2C in master mode and sets the SDA and SCL pins (PB7 and PB6). Ground Connection Both Board sorted.
- Start Condition:
  The STM32 generates a start condition, indicating the beginning of communication.
- Addressing:
  The STM32 sends the 7-bit address of the Rugged Board as the slave address.
- Data Transmission:
  The STM32 sends data (e.g., a byte) to the Rugged Board via I2C.
- Stop Condition:
  After completing the data transfer, STM32 generates a stop condition to terminate communication.

Slave (Rugged Board) Side:

- Initialization:
  The Rugged Board Configure I2c in Slave Mode and set the SDA and SCL pins Mikro Bus I2C pins or Expansion Header pin31 is SCL and pin32 SDA .

- **Listening for Data:**
  The Rugged Board, acting as the slave, continuously monitors the I2C bus for communication from the STM32.
- **Receiving Data:**
  When the Rugged Board receives data from the STM32, it processes the data (e.g., stores it or responds).
- **Sending Response:**
  The Rugged Board can send a response back to the STM32, depending on the communication protocol.

STM32 source Code :-

```
/* USER CODE BEGIN Header */
```

```c
/**

******************************************************************************
***
 * @file           : main.c
 * @brief          : Main program body

******************************************************************************
***
 * @attention
 *
 * Copyright (c) 2024 STMicroelectronics.
 * All rights reserved.
 *
 * This software is licensed under terms that can be found in the LICENSE
file
 * in the root directory of this software component.
 * If no LICENSE file comes with this software, it is provided AS-IS.
 *

******************************************************************************
***
 */
/* USER CODE END Header */
/* Includes
------------------------------------------------------------------*/
#include "main.h"
#include "string.h"
/* Private includes
----------------------------------------------------------*/
/* USER CODE BEGIN Includes */
/* USER CODE END Includes */
/* Private typedef
-----------------------------------------------------------*/
/* USER CODE BEGIN PTD */
uint8_t buffer[256];
//uint8_t i2c_buffer[256]; // Buffer for receiving I2C data
uint8_t i2c_address = 0x68;
//#define STM32_SLAVE_ADDRESS 0x69  // Define the slave address for STM32
#define RUGGED_SLAVE_ADDRESS 0x55
/* USER CODE END PTD */
/* Private define
------------------------------------------------------------*/
/* USER CODE BEGIN PD */
extern I2C_HandleTypeDef hi2c1;
extern UART_HandleTypeDef huart1;
extern UART_HandleTypeDef huart2;
/* USER CODE END PD */
/* Private macro
-------------------------------------------------------------*/
/* USER CODE BEGIN PM */
/* USER CODE END PM */
```

```c
/* Private variables
---------------------------------------------------------*/
I2C_HandleTypeDef hi2c1;
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_USART2_UART_Init(void);
static void MX_I2C1_Init(void);
static void MX_USART1_UART_Init(void);
/* USER CODE BEGIN PFP */
/* USER CODE END PFP */
/* Private user code
---------------------------------------------------------*/
/* USER CODE BEGIN 0 */
void I2C_Init(void)
{
   // Initialize I2C peripheral (HAL_I2C_Init will be called by CubeMX or
manually in code)
   // Make sure to set the correct I2C address and configuration
   HAL_I2C_Init(&hi2c1);
}
// Function to handle I2C transmission
void I2C_Transmit(uint8_t* data, uint16_t length)
{
   // Send data to the Rugged Board via I2C
   if (HAL_I2C_Master_Transmit(&hi2c1, RUGGED_SLAVE_ADDRESS << 1, data,
length, 1000) != HAL_OK)
   {
       // If transmission fails, handle the error
       Error_Handler();
   }
}
// Function to handle I2C reception
void I2C_Receive(uint8_t* buffer, uint16_t length)
{
   // Receive data from the Rugged Board via I2C
   if (HAL_I2C_Master_Receive(&hi2c1, RUGGED_SLAVE_ADDRESS << 1, buffer,
length, 1000) != HAL_OK)
   {
       // If reception fails, handle the error
       Error_Handler();
   }
}
/* USER CODE END 0 */
/**
 * @brief  The application entry point.
 * @retval int
 */
```

```c
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_USART2_UART_Init();
 MX_I2C1_Init();
 MX_USART1_UART_Init();
 /* USER CODE BEGIN 2 */
 /* USER CODE END 2 */
 /* Infinite loop */
 /* USER CODE BEGIN WHILE */
while (1)
     {
         // Step 1: Send the message via USART1 and USART2
         sprintf((char*)buffer, "HELLO from STM32 DATA\r\n");
         HAL_UART_Transmit(&huart1, buffer, strlen((char*)buffer), 1000);
// Send message via USART1
         HAL_UART_Transmit(&huart2, buffer, strlen((char*)buffer), 1000);
// Send message via USART2
         // Step 2: Send data via I2C to Rugged Board every 5 seconds
         I2C_Transmit(buffer, strlen((char*)buffer));  // Send I2C data to
Rugged Board
         // Step 3: Wait for 5 seconds before next loop iteration
         HAL_Delay(5000);  // Delay for 5 seconds
         // Step 4: Continuously receive data from Rugged Board via I2C
         if (HAL_I2C_Master_Receive(&hi2c1, RUGGED_SLAVE_ADDRESS << 1,
buffer, sizeof(buffer), 1000) == HAL_OK)
         {
             // If data is received from I2C, process and send it via
USART2
             HAL_UART_Transmit(&huart2, buffer, strlen((char*)buffer),
1000);
         } else {
             // Handle I2C reception failure
             sprintf((char*)buffer, "No data received via I2C\r\n");
             HAL_UART_Transmit(&huart2, buffer, strlen((char*)buffer),
1000); // Send error message via USART2
         }
     }
}
```

```c
/**
 * @brief System Clock Configuration
 * @retval None
 */
void SystemClock_Config(void)
{
 RCC_OscInitTypeDef RCC_OscInitStruct = {0};
 RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};
 /** Configure the main internal regulator output voltage
 */
 __HAL_RCC_PWR_CLK_ENABLE();
 __HAL_PWR_VOLTAGESCALING_CONFIG(PWR_REGULATOR_VOLTAGE_SCALE3);
 /** Initializes the RCC Oscillators according to the specified parameters
 * in the RCC_OscInitTypeDef structure.
 */
 RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSI;
 RCC_OscInitStruct.HSIState = RCC_HSI_ON;
 RCC_OscInitStruct.HSICalibrationValue = RCC_HSICALIBRATION_DEFAULT;
 RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
 RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSI;
 RCC_OscInitStruct.PLL.PLLM = 16;
 RCC_OscInitStruct.PLL.PLLN = 336;
 RCC_OscInitStruct.PLL.PLLP = RCC_PLLP_DIV4;
 RCC_OscInitStruct.PLL.PLLQ = 2;
 RCC_OscInitStruct.PLL.PLLR = 2;
 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
 * @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 */
```

```c
  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 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 */
  /* 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 */
```

```c
 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_InitTypeDef GPIO_InitStruct = {0};
/* USER CODE BEGIN MX_GPIO_Init_1 */
/* USER CODE END MX_GPIO_Init_1 */
 /* GPIO Ports Clock Enable */
 __HAL_RCC_GPIOC_CLK_ENABLE();
 __HAL_RCC_GPIOH_CLK_ENABLE();
 __HAL_RCC_GPIOA_CLK_ENABLE();
 __HAL_RCC_GPIOB_CLK_ENABLE();
 /*Configure GPIO pin Output Level */
 HAL_GPIO_WritePin(LD2_GPIO_Port, LD2_Pin, GPIO_PIN_RESET);
 /*Configure GPIO pin : B1_Pin */
 GPIO_InitStruct.Pin = B1_Pin;
 GPIO_InitStruct.Mode = GPIO_MODE_IT_FALLING;
 GPIO_InitStruct.Pull = GPIO_NOPULL;
 HAL_GPIO_Init(B1_GPIO_Port, &GPIO_InitStruct);
 /*Configure GPIO pin : LD2_Pin */
 GPIO_InitStruct.Pin = LD2_Pin;
 GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
 GPIO_InitStruct.Pull = GPIO_NOPULL;
 GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
 HAL_GPIO_Init(LD2_GPIO_Port, &GPIO_InitStruct);
/* USER CODE BEGIN MX_GPIO_Init_2 */
/* USER CODE END MX_GPIO_Init_2 */
}
/* 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 */
```

Rugged Board source Code:-

```
import smbus2 as smbus

import time

import sys

import os

import re



# I2C Configuration

I2C_DEVICE = 0  # Adjust this to match the RuggedBoard's I2C_2 bus index

SLAVE_ADDRESS_STM32 = 0x55  # STM32 I2C address

SLAVE_ADDRESS_OTHER = 0x56  # Another device I2C address
```

```
BUFFER_SIZE = 32  # Buffer size for I2C read

SEND_STRING = "hi"  # Data to send to STM32

SEND_INTERVAL = 15  # Interval for sending data to STM32 in seconds


# Initialize I2C bus

def I2C_Init():

    try:

    # Check if the I2C device exists

    i2c_device_path = '/dev/i2c-{}'.format(I2C_DEVICE)

    if not os.path.exists(i2c_device_path):

    print("Error: {} not found!".format(i2c_device_path))

    sys.exit(1)


    # Initialize I2C bus

    bus = smbus.SMBus(I2C_DEVICE)

    print("I2C bus {} initialized.".format(I2C_DEVICE))

    return bus

    except IOError as e:

    print("I2C initialization failed: {}".format(e))

    sys.exit(1)

    except Exception as e:

    print("Unexpected error during I2C initialization: {}".format(e))

    sys.exit(1)


# Function to sanitize received data
```

```
def sanitize_data(data):
    # Convert bytes to string and handle null-terminated strings
    raw_string = ''.join(chr(byte) for byte in data if 0x20 <= byte <= 0x7E)
    sanitized_string = raw_string.split('\x00')[0]
    return sanitized_string


# Function to receive data from a device
def I2C_ReceiveData(bus, address):
    try:
    print("Waiting for data from device at address 0x{:02x}...".format(address))
    # Read data from the device
    received_data = bus.read_i2c_block_data(address, 0, BUFFER_SIZE)

    if received_data:
    received_string = sanitize_data(received_data)
    print("Received from 0x{:02x}: {}".format(address, received_string))
    return received_string
    else:
    print("No data received from 0x{:02x}.".format(address))
    return None
    except IOError as e:
    print("Failed to read from I2C address 0x{:02x}: {}".format(address, e))
    except Exception as e:
    print("Unexpected error while receiving data from 0x{:02x}: {}".format(address,
e))
```

```
        return None

# Function to send data to a device
def I2C_SendData(bus, address, data):
        try:
        byte_data = [ord(c) for c in data]
        bus.write_i2c_block_data(address, 0, byte_data)
        print("Sent '{}' to 0x{:02x}".format(data, address))
        except IOError as e:
        print("Failed to write to I2C address 0x{:02x}: {}".format(address, e))
        except Exception as e:
        print("Unexpected error while sending data to 0x{:02x}: {}".format(address, e))

# Main function
def main():
        bus = I2C_Init()
        last_sent_time = 0  # Track last time data was sent to STM32

        while True:
        current_time = time.time()

        # Periodically send data to STM32 every 15 seconds
        if current_time - last_sent_time >= SEND_INTERVAL:
        I2C_SendData(bus, SLAVE_ADDRESS_STM32, SEND_STRING)
        last_sent_time = current_time
```

```
# Continuously check for messages from another board
other_response = I2C_ReceiveData(bus, SLAVE_ADDRESS_OTHER)
if other_response:
print("Other Device Response: {}".format(other_response))

# Check for responses from STM32
stm32_response = I2C_ReceiveData(bus, SLAVE_ADDRESS_STM32)
if stm32_response:
print("STM32 Response: {}".format(stm32_response))

# Small delay to reduce CPU usage
time.sleep(1)

if __name__ == "__main__":
    main()
```

Expecting output :-

Sent data: 'Hello from Rugged Board via I2C'

Received: `HELLO from STM32 I2C Data`