UART Project Steps:-

To create a **full-duplex communication** system between an **STM32** microcontroller and a **Rugged Board** over UART using **UART1** on the STM32 and **UART3** on the Rugged Board, you will need to follow several steps. This guide will provide a comprehensive approach, including hardware connections, STM32 firmware, and Python code for the Rugged Board. The communication will involve sending and receiving data simultaneously from both sides (STM32 and Rugged Board).

Hardware Setup:-

Connections:-

- **STM32 UART1** will communicate with **Rugged Board UART3**.
- You'll connect the **TX** (Transmit) pin of **STM32 UART1** to the RX (Receive) pin of **Rugged Board UART3** and vice versa.
- Additionally, connect the **GND** (Ground) between both devices to ensure a common reference.

PIn Connections :-

To set up communication between the **STM32** and the **Rugged Board** over **UART**:

- **STM32** will communicate using **USART1**, with the following pin configuration:
    - **USART1_TX** (Transmit) is connected to **PA9**.
    - **USART1_RX** (Receive) is connected to **PA10**.
- **Rugged Board** will use **UART3**, with the following pin configuration:
    - **UART3_TX** (Transmit) is connected to the corresponding **RX** pin on the STM32 (**PA10**).
    - **UART3_RX** (Receive) is connected to the corresponding **TX** pin on the STM32 (**PA9**).

This setup allows full-duplex communication where the **STM32** sends data via **USART1_TX** (PA9) and receives data via **USART1_RX** (PA10), while the **Rugged Board** uses **UART3_TX** and **UART3_RX** for data transmission and reception, respectively.

STM32 Configuration and Firmware:-

- On the STM32 side, you'll use the **HAL (Hardware Abstraction Layer)** to configure **UART1** for both transmission and reception. Below is the code to set up UART on the STM32 and send/receive data.

STM32 Code:-

1. **Initialize UART1**: Set up the **UART1** pins and configure the UART peripheral with the correct baud rate, parity, stop bits, etc.

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

*****************************************************************************
*
 * @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 <stdio.h>
#include <string.h>
/* Private includes
------------------------------------------------------------*/
/* USER CODE BEGIN Includes */
/* USER CODE END Includes */
/* Private typedef
-------------------------------------------------------------*/
/* USER CODE BEGIN PTD */
uint8_t buffer[100];
/* 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
---------------------------------------------------------*/
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_USART1_UART_Init(void);
static void MX_USART2_UART_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 */
 /* 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_USART1_UART_Init();
 MX_USART2_UART_Init();
 /* USER CODE BEGIN 2 */
 /* USER CODE END 2 */
 /* Infinite loop */
 /* USER CODE BEGIN WHILE */
```

```
 while (1)
 {
   /* USER CODE END WHILE */
   /* USER CODE BEGIN 3 */
        // 1. Transmit data to USART1 (TX)
                sprintf((char*)buffer, "HELLO from STM32\r\n");  // Example
message to send
                HAL_UART_Transmit(&huart1, buffer, strlen((char*)buffer),
1000);  // Send the message via USART1
                HAL_UART_Transmit(&huart2, buffer, strlen((char*)buffer),
1000);  // Also send the same message via USART2
                // 2. Wait for 5 seconds before checking for received data
                HAL_Delay(5000);  // Delay for 5 seconds (5000 ms)
                // 3. Receive data from USART1 (RX) and store it in the
buffer
                if (HAL_UART_Receive(&huart1, buffer, sizeof(buffer), 5000)
== HAL_OK) {
                    // If data is received within the timeout, transmit it
to USART2
                    HAL_UART_Transmit(&huart2, buffer,
strlen((char*)buffer), 1000);  // Transmit received data to USART2
                } else {
                    // No data received in the 5-second timeout
                    sprintf((char*)buffer, "No data received");
                    HAL_UART_Transmit(&huart2, buffer,
strlen((char*)buffer), 1000);  // Send the "No data received" message
                }
                // 4. Wait for another 5 seconds before repeating the cycle
                HAL_Delay(5000);  // Delay for 5 seconds before the next
loop iteration
 }
 /* USER CODE END 3 */
}
/**
 * @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_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();
  }
}
/**
  * @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)
{
/* USER CODE BEGIN MX_GPIO_Init_1 */
/* USER CODE END MX_GPIO_Init_1 */
  /* GPIO Ports Clock Enable */
  __HAL_RCC_GPIOA_CLK_ENABLE();
/* 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 */
```

Explanation of STM32 Code:

- **UART_Init**: This function initializes the UART1 peripheral with a baud rate of 115200, 8 data bits, no parity, and 1 stop bit.
- **UART_Transmit**: This function sends data to the Rugged Board via UART1.
- **UART_Receive**: This function receives data from the Rugged Board over UART1.
- **Main loop**: The STM32 continuously sends "Hello from STM32!" and waits for a response from the Rugged Board.

Rugged Board Configuration and Code:-

On the Rugged Board you'll use Python with the **`pyserial`** library to communicate over UART. The Python code will handle both sending and receiving data via **UART3**.

Rugged Board Code:

- **Install pyserial**: To interact with the UART on the Rugged Board, you'll need to install the `pyserial` library.

  pip3 install pyserial

Python Code for the Rugged Board (Receiver and Sender):

import serial

import time

SERIAL_PORT = '/dev/ttyS3' # Update this if necessary

BAUD_RATE = 115200

```
try:
    # Initialize serial connection
    ser = serial.Serial(SERIAL_PORT, BAUD_RATE, timeout=1)
    print("Connected to {} at {} baud.".format(SERIAL_PORT, BAUD_RATE))
except serial.SerialException as e:
    print("Failed to connect to {}: {}".format(SERIAL_PORT, e))
    exit(1)

def main():
    print("Starting UART communication...")
    while True:
    try:
    # Send data to STM32
    message = "Hello from Rugged Board via UART3\r\n"
    ser.write(message.encode('utf-8'))
    print("Sent: {}".format(message.strip()))

    # Receive data from STM32
    if ser.in_waiting > 0:
        received = ser.readline().decode('utf-8').strip()
        print("Received: {}".format(received))

    # Wait for a short period to manage communication timing
    time.sleep(1)
```

```
except serial.SerialException as e:

print("Serial communication error: {}".format(e))

break


if __name__ == '__main__':

try:

main()

except KeyboardInterrupt:

print("Exiting UART communication...")

finally:

if ser.is_open:

ser.close()

print("Serial port closed.")
```

Explanation of Rugged Board Code:

- **send_data**: Sends data over UART3 to STM32.
- **receive_data**: Checks if there is incoming data from STM32 and prints it.
- **Main loop**: The Rugged Board continuously sends "Hello from Rugged Board!" and waits for a response.

Testing and Troubleshooting:-

- Once both the STM32 and Rugged Board are set up, connect them via UART as described earlier.
- Run the STM32 program and the Rugged Board Python script.
- If both sides are working correctly, you should see that both devices are sending and receiving data.

output :-

Sent: Hello from Rugged Board via UART3

Received: HELLO from STM32

Sent: Hello from Rugged Board via UART3

Sent: Hello from Rugged Board via UART3

Sent: Hello from Rugged Board via UART3

Received: HELLO from STM32

Sent: Hello from Rugged Board via UART

https://github.com/DODDIMOHANKUMAR/Phytec-OJT-Mohankumar/tree/Project