NYMBLE Task: UART Communication and Baud Rate Calculation on STM32 Nucleo Board.

Document by :
Rajaram Balakrishnan
Firmware Developer.
rajarambk12@gmail.com

Device and Software Setup:-

Devices Used:
1. Nucleo G474RE Board: Microcontroller development board based on STM32G474RE, featuring ARM Cortex-M4 core.
2. TTL Converter: For UART transmission between the Nucleo Board and PC.

Software:
1. STM32Cube IDE: Integrated development environment for STM32 microcontrollers, used for firmware development and debugging.
2. PySerial (Python): Python library for serial communication, used for sending and receiving strings between the PC and Nucleo board.

Approach:

1. PC-Side Communication:
- **Python Code:** A Python script using the `pyserial` library is used to send and receive strings through UART. The communication is handled in a non-blocking manner via parallel processing (using threads) to allow continuous data transfer and reception.

2. Nucleo Board Configuration:
- **UART Setup:** UART 4 is configured for communication on the Nucleo G474RE board.
- Timer Configuration: Timer 2 is configured to calculate the baud rate based on the received bits within a specified time window.
- **Baud Rate Calculation:**
  - After a fixed amount of time, the number of bits received is calculated. The baud rate is then approximated by dividing the total number of bits by the time elapsed.

3. SWV (Serial Wire Viewer):
- The SWV functionality is used for debugging, specifically to print the calculated baud rate in the console.

Limitations: (EEPROM)

The Nucleo G474RE board lacks an onboard EEPROM and no external EEPROM is available with me at present, the following algorithm outlines how EEPROM can be integrated for storing and retrieving data.

1. Initialize EEPROM Communication:
- Enable I2C or SPI communication with the external EEPROM chip, depending on the chip's interface.

2. Define EEPROM Memory Address:
- Store the base address of the EEPROM memory (where data will be written) as macros in the firmware.

3. Write Data Byte-by-Byte:
- Since the task requires writing data byte-by-byte to the EEPROM, you can use an interrupt service routine (ISR) for UART data reception. In this case, the call `WRITE_EEPROM` function after receiving each byte of data.
- In my code it is inside the UART Interrupt Service Routine (UART_ISR) in `main.c`, at line 401 (depending on the actual code structure).

4. READ_EEPROM Function:
- Implement a function to retrieve stored data from EEPROM when needed.

Alternate method: Flash Memory Usage

Given the limitations of not having an external EEPROM, an alternative solution is to use Flash memory to store the data. However, there are certain constraints when using flash:

- Flash Memory Constraints:
- Flash access is highly secure, and thus flash writes require disabling interrupts, which may disrupt UART communication.
- Writing to Flash is slower compared to writing to EEPROM.So in the meantime we may loose UART data received.
- Hence instead of writing byte by byte it is advised to write the entire block of data at once.

Algorithm for Flash Memory Usage:

1. Erase Flash Memory:
- Before writing to Flash, ensure that the memory region is erased, otherwise, a flash write error will occur.

2. Disable Interrupts:
- Disabling interrupts (including UART) is necessary while writing to the flash memory to ensure data integrity.

3. Write & erase Data in Flash:
- Data is written to Flash memory by the above method, but flash erasure can be done only per **sector** or **page** and not **byte** by **byte** which is another limitation. In G474RE MCU(which is used) one page is **2KB** long, so erasing memory less than **2KB** is not possible.

4. Example Code for Writing to Flash:

```
void WriteToFlash(uint32_t* data, uint32_t len)
{
   HAL_FLASH_Unlock(); // Unlock Flash for writing

   // Write data to flash byte-by-byte
   for (uint32_t i = 0; i < len; i++)
   {
      // Write to flash at the specified address
      HAL_FLASH_Program(FLASH_TYPEPROGRAM_DOUBLEWORD,
FLASH_MEMORY_ADDRESS + i, data[i]);
   }

   HAL_FLASH_Lock(); // Lock Flash after writing
}
```