Implementing Protocol Buffers on the NUCLEO-STM32H7A3ZI-Q Board

Program Overview

A Python script running on a PC can send fixed-length messages to the STM32H7A3ZI-Q board using Protocol Buffers. The user can issue 'on' and 'off' commands via the terminal to control LED1 on the board.

How to Use the Program

1. The C program must be uploaded to the microcontroller. This can be done by importing the project into STM32CubeIDE. If this is not possible, the .bin file located in the protobuffer/Debug folder can be uploaded using another programming tool.
2. Next, the Python script (user.py) should be run, and commands can be sent to the microcontroller via the terminal.

Example

Terminal:

Enter a command ('on' / 'off'): on
Command sent: ON
Enter a command ('on' / 'off'): off
Command sent: OFF

Detailed Description and Operation

The program uses the libary (https://github.com/nanopb/nanopb) to generate the necessary Protocol Buffer files from led_blink.proto in the proto_generator/messages folder. The generated led_blink.c file is included in the C program, while led_blink.py is used in the Python script. Additionally, the files pb_common.c/.h, pb_encode.c/.h, and pb.h are required for the microcontroller program.

C Program

Peripherals Used:

- **USART3**: Used for communication between the STM32 and the PC, configured in asynchronous interrupt mode at 115200 Baudrate.
- **LED1**: The LED controlled by the commands.

main.c

Global Variables:

- volatile uint8_t buffer[BUFFER_LEN] - Stores incoming data.

- static volatile uint8_t data[32] - Temporarily holds incoming characters.
- static volatile bool LedStat - Indicates whether message decoding was successful.
- static uint32_t DataCounter - Counts the number of unprocessed data elements.
- static volatile uint8_t UartSign - Indicates valid data.

Definitions:

```
#define MESSAGE_LENGTH 2    // Predefined message length
#define MESSAGE_ID 0x08     // Unique identifier for the message

#define LED_ON 1
#define LED_OFF 0
```

Functions:

- **void HAL_UART_RxCpltCallback(UART_HandleTypeDef *huart)**
  Handles UART interrupts. It checks if the interrupt came from USART3 and reads valid messages into the circular buffer. Invalid data is ignored. If the buffer is full, the program processes one data item before continuing.
- **LedStatus decode()**
  Decodes the Protocol Buffer message and returns the extracted information.
- **void execute(LedStatus message)**
  Sets the LED state according to the received message. If an invalid character is received, it is ignored.
- **int __io_putchar(int ch)**
  Overrides the standard putchar function to enable printf() debugging.

circularbuffer.c/.h

Global Variables:

- volatile uint32_t writePtr
- volatile uint32_t readPtr

Definitions:

```
#define BUFFER_LEN (500)  // Minimum size required for MESSAGE_LENGTH
```

Functions:

- **void getdata_frombuffer(uint8_t data[], uint8_t len)**
  Extracts len bytes of data from the buffer at the position pointed to by readPtr. This function is only called if there is unprocessed data, ensuring that a message is always returned.

- **bool writedata_tobuffer(uint8_t data[], uint8_t len)**
  Writes len bytes of data into the buffer. If writePtr catches up with readPtr, new data is not written to avoid overwriting unprocessed messages. In such cases, the function returns 1.

Python Script (user.py)

The Python script allows the user to control the LED by sending 'on' or 'off' commands through the terminal.

Functionality:

- Reads user input from the terminal.
- Sets message.status accordingly.
- Encodes and sends the message to the microcontroller via the serial port.