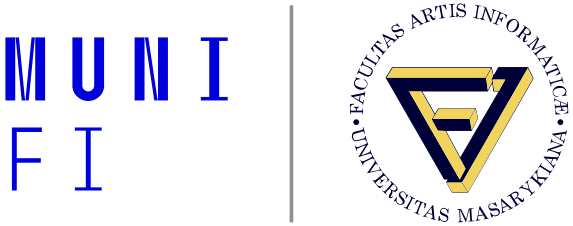

PV198 - UART II

One-chip Controllers

Jan Koniarik

Faculty of Informatics, Masaryk University

Context

- Until now, most of the projects were simple, this time we will increase the complexity.
- The data will get more complex:
  - Instead of working with simple data, we will use layered protocol
- We will use more advanced API:
  - Until now, most of the API you used is **blocking**
  - Today, we will try **non-blocking** API

Goal

Implement an app that waits for an input on UART to setup RGB LED. In case the input is corrupted, the app returns an error message over the same UART.

Protocol

The app shall use two-layered protocol for communication:

1. Outter layer is binary protocol:
   - | **SIZE_HIGH** | **SIZE_LOW** | <inner message> | **CHECKSUM** |
   - where **SIZE_HIGH**, **SIZE_LOW**, and **CHECKSUM** are 8bit values
   - **SIZE** is the number of bytes the <inner message> is made of
   - **CHECKSUM** is calculated by XORing all bytes of <inner message>
2. Inner layer is ASCII-based protocol:
   - Input: string consisting of three RGB values delimited by comma
     - "255,255,255" turns on all three LEDs fully
   - Output: Error messages

Guideline

When implementing the app, focus on decomposition, there should be clearly separate functionalities for:

- Loading data from UART
- Code for outter layer of the message
- Code for inner layer of the message
- Work with RGB LED

Template

The provided template contains parts of the functionality already implemented in libraries:

- circular_buffer.h - queue of bytes with fixed size - good for output data
- matrix_buffer.h - queue of byte arrays with fixed size - good for input messages
- protocol.h - utilities to work with the outter layer of the protocol

Task I

- Get yourself familiar with the matrix buffer and protocol libraries.
- Use *UART_ReadBlocking* to store input data into the matrix buffer.
- Hint: MatrixBuffer should be a global

- Once we are able to load input data into buffer, we can start processing the message
- Note that the matrix buffer gives as ability to handle a burst of messages in case the processing would be slow

Task II

- Extract the inner message from currently processed message
- Use *sscanf* to extract RGB values
- Verify the RGB range of received values
- Set the RGB values to the LEDS

- The API of libraries has one property: most functionality returns *bool* indicating whenever operation succeeded.
- It is crucial in embedded to have proper error reporting capability.

Task III

- Study the circular_buffer.h.
- Write a function that can take any message, wrap it in outter protocol and insert into circular buffer.
- Use the function to error report potential problems in processing of the input message.
- Hint: CircularBuffer should be a global

- We have capability to process messages now.
- But you should see that the process is blocking and there is chance to miss input messages.
- Note: There is also python script in the project to test it.

Interrupts

- We can use interrupts to managed sending and receiving of the data
- For the sake of simplicity, we will:
  - Step A: send output data with interrupts and read with blocking API
  - Step B: send output data with interrupts and read with interrupts

Interrupts

- Start by going into peripheral and switching it into interrupt mode
- Prepare the interrupt handler function in main.c

Interrupts

To send data over UART with simple usage of interrupts, we should:

- Enable the specific interrupt for sending byte if we have bytes to sent:
  - UART_EnableInterrupts(UART3_PERIPHERAL, kUART_TxDataRegEmptyInterruptEnable);
- Check in the interrupt that the source is free tx:
  - if (kUART_TxDataRegEmptyFlag & status)
- Write one byte at a time:
  - UART_WriteByte
- Disable the interrupt once the buffer is empty

Task IV

Make all the necessary steps for sending the data with interrupts.

Interrupts

To receive data over UART with simple usage of interrupts, we should:

- Enable the specific interrupt for receiving bytes forever:
  - UART_EnableInterrupts(UART3_PERIPHERAL, kUART_RxDataRegFullInterruptEnable);
- Check in the interrupt that the source is rx byte:
  - kUART_RxDataRegFullFlag or kUART_RxOverrunFlag
- Read one byte at a time:
  - UART_ReadByte

Task V

Make all the necessary steps for receiving the data with interrupts. Do not forget to implement proper usage of MatrixBuffer in the interrupt, the rows have to be switched properly!

MUNI
FACULTY
OF INFORMATICS