Colorado School of Mines
Department of Mechanical Engineering
MEGN 540: Mechatronics

Laboratory 1: USB Based Communication

Objective: Initiate USB communication, become familiar with Python3 based Serial Monitor GUI and, utilize the RingBuffer to enable an event based programming paradigm.

Equipment:

· Lab Kit Mouse Keyboard Monitor Internet Connection

Background:

The atmega32u4 is has an USB interface that we will be programming. USB is a serial communication protocol that has an added layer of communication between the host machine and the device that allows the host machine to recognize devices, use standard messaging standards, and synchronize the communication speed. This extra layer of complexity requires significant programming capabilities to implement, fortunately an open source library exists that we will use https://github.com/abcminiuser/lufa, which has already been incorporated as a submodule in our git repo and cloned for use. Your task will be to take a slightly simplified example program from lufa and make the necessary modifications to read and write serial.

Functions to add: (note the file can be found in MEGN540/c_lib):

void Message_Handling_Init();

MEGN540_MessageHandeling.h/c

Function Message_Handling_Init initializes the message handling and all associated state flags and data to their default conditions.

void Message_Handling_Task();

MEGN540_MessageHandeling.h/c

Function Message_Handler processes USB messages as necessary and sets status flags to control the flow of the program. It returns true unless the program receives a reset message.

void usb_read_next_byte();

SerialIO.h/c

(non-blocking) Function usb_read_next_byte takes the next USB byte and reads it into a ring buffer for latter processing.

void usb_write_next_byte();

SerialIO.h/c

(non-blocking) Function usb_write_next_byte takes the next byte from the output ringbuffer and writes it to the USB port (if free).

void usb_send_byte(uint8_t byte);

SerialIO.h/c

(non-blocking) Function usb_send_byte Adds a character to the output buffer.

void usb_send_data(void* p_data, uint8_t data_len);

SerialIO.h/c

(non-blocking) Function usb_send_data adds the data buffer to the output ring buffer.

void usb_send_str(char* p_str);

SerialIO.h/c

(non-blocking) Function usb_send_str adds a c-style (null terminated) string to the output buffer.

void usb_send_msg(char* format, char cmd, void* p_data, uint8_t data_len);

SerialIO.h/c

(non-blocking) Function usb_send_msg sends a message according to the MEGN540 USB message format.

uint8_t usb_msg_length();

SerialIO.h/c

(non-blocking) Funtion usb_msg_length returns the number of bytes in the receive buffer awaiting processing.

uint8_t usb_msg_peek();

SerialIO.h/c

(non-blocking) Function usb_msg_peek returns (without removal) the next byte in teh receive buffer (null if empty).

uint8_t usb_msg_get();

SerialIO.h/c

(non-blocking) Function usb_msg_get removes and returns the next byte in the receive buffer (null if empty)

bool usb_msg_read_into(void* p_obj, uint8_t data_len);

SerialIO.h/c

(non-blocking) Function usb_msg_read_into populates the object with the data in the recieve buffer and removes the bytes as they are read. Returns false if receive buffer does not contain enough bytes to fill the container and terminates without reading or removing.

void usb_flush_input_buffer();

SerialIO.h/c

(non-blocking) Function usb_flush_input_buffer sets the length of the recieve buffer to zero and disreguards

---

Assignment:

You will turn your zumo car into a calculator for this assignment. From the provided python gui, you'll send the car a mathematical operator (+, -, *, /) and two floating point values. The car will process the data and return the result in the format specified for the class. If the message received does not start with an operation (+, -, *, /) the car will respond as defined by the class.

1. Review the provided functions, understand what they are doing, and how their called. The current script acts as an echo chamber, every byte received is sent back to the host.
2. Make and program the current script onto the car. Launch the Python3 gui from the SerialMonitor folder
    a. Open a terminal and navigate to the MEGN540 folder
    b. Launch the script
       >> python3 SerialMonitor/serial_monitor.py
       OR
       >> ./open_serial_monitor
    c. Connect to the device
    d. Send some messages and see how it works
    e. For Python bit formatting checkout the format character meaning here: https://docs.python.org/3/library/struct.html
3. Using the USB_Echo_Task() as inspiration and your ring buffer code from homework. Write the above mentioned functions. The USB interface only sends 8 bits in a frame (see the static CDC_LineEncoding_t LineEncoding1 in SerialIO.c), so don't block while sending messages larger than 8bits, just come back to it. That means you are going to have to dig into just what the function Endpoint_WaitUntilReady() is doing, so you can check to see if the endpoint is ready instead and only send another byte if it is.
    a. VS code lets you trace a function. Right click on it, click GoTo and Definition. OR search for the function name in the lufa directory.
4. Implement in the Lab1.c main loop a message handling capability that can return the desired results.

Deliverables:

1. Demo: Make a 1-2-Minute Video (<50MB) that talks through your code and how you completed the lab. Make sure to show me how you handled the functions added to SerialIO
2. Demo: Upload a video of you interacting with the car demoing each operation as well as error states (not sending a math operation).
3. Lab Report: Prepare a summary of what you have completed for this assignment. Follow a typical lab report format. Make sure that you:
    a. Include the information that you found important and critical while completing this lab, and would be useful if you wanted to replicate it in the future.
    b. Indicate which pins on the electrical schematic were used in this project, and how you can tell which registers you needed to initialize and set.
    c. Discuss how you might expand this interaction to enable other messaging in the future (m for move or a for acceleration values etc).