

## Laboratory 1: USB Based Communication

**Objective:** Initiate USB communication, become familiar with Python3 based Serial Monitor GUI and, utilize the RingBuffer and Task Management 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):

MEGN540\_MessageHandling.h/c

```
void Message_Handling_Init();
```

Function Message\_Handling\_Init initializes the message handling and all associated state flags and data to their default conditions.

```
void Message_Handling_Task();
```

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.

Task\_Management.h/c

```
void Task_Activate( Task_t* task );
```

Function Task\_Activate sets the state of is\_active to true

```
void Task_Cancel( Task_t* task );
```

Function Task\_Cancel sets the state of is\_active to false

```
bool Task_Is_Ready( Task_t* task );
```

Function Task\_Is\_Ready checks to see if the given task is ready for execution. For Lab 1, this just returns the value of the is\_active flag. For Lab 2 on, it also checks timing.

bool Task\_Run\_If\_Ready( Task\_t\* task );

Function Task\_Run\_If\_Ready checks to see if the given task is ready for execution, for Lab 1 this is equivalent to the is active flag status. If the task function pointer is NULL then it just returns if the task is ready and resets the time\_last\_ran.

#### SerialIO.h/c

void usb\_read\_next\_byte();

(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();

(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);

(non-blocking) Function usb\_send\_byte Adds a character to the output buffer.

void usb\_send\_data(void\* p\_data, uint8\_t data\_len);

(non-blocking) Function usb\_send\_data adds the data buffer to the output ring buffer.

void usb\_send\_str(char\* p\_str);

(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 );

(non-blocking) Function usb\_send\_msg sends a message according to the MEGN540 USB message format.

uint8\_t usb\_msg\_length();

(non-blocking) Funtion usb\_msg\_length returns the number of bytes in the receive buffer awaiting processing.

uint8\_t usb\_msg\_peek();

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

uint8\_t usb\_msg\_get();

(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);

(non-blocking) Function usb\_msg\_read\_into populates the object with the data in the receive 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();

(non-blocking) Function usb\_flush\_input\_buffer sets the length of the receive buffer to zero and disregards

#### Lab1\_Tasks.h/c

void Multiply\_And\_Send(float value\_left, float value\_right );

```
void Divide_And_Send(float value_left, float value_right );  
void Add_And_Send(float value_left, float value_right );  
void Subtract_And_Send(float value_left, float value_right );
```

---

### 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).