USB Device Project Tutorial

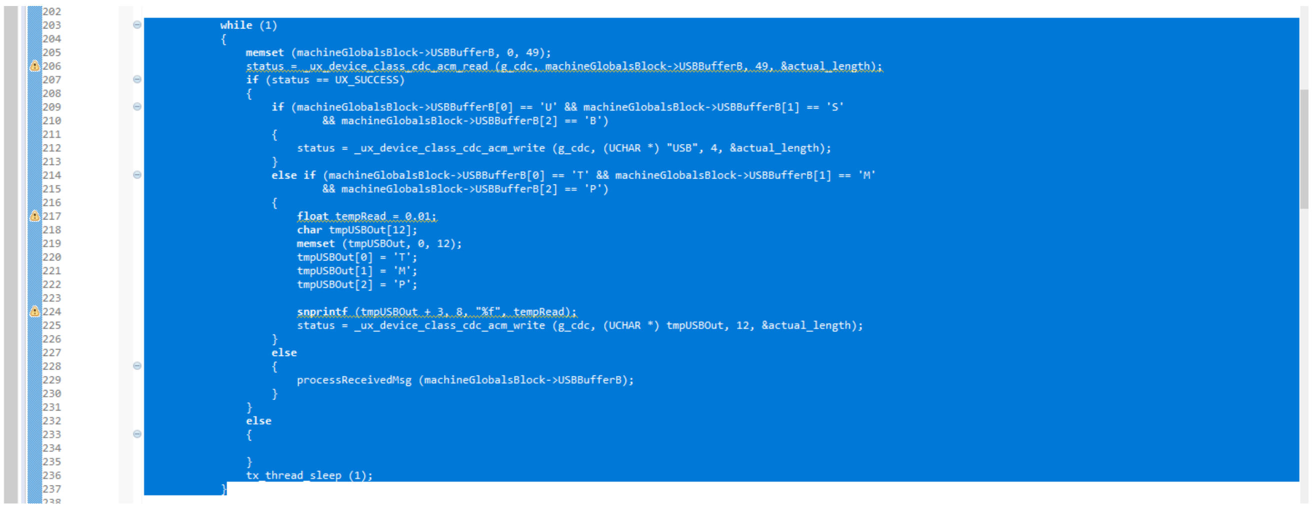
Introduction

This document is meant to be a quick start guide on working with the USB Device interface and Ganymede libraries. For in-depth coverage of the inner workings and settings for the USB Device interface and API, please refer to the appropriate Renesas documentation, which can be found by right-clicking on the module in the configuration view, and selecting Module Resources.

This document assumes the reader has already cloned the Tutorials repository and ExampleProject project into their e2 Studio environment, and is familiar with the process of creating their own unique project based on this. For more information on that, please refer to the Project Creation Tutorial located in the Documents folder of the Tutorials Repo. To reach the Tutorials repo, please go to "https://github.com/NYUAD-LabOps/Tutorials".

USB Thread

The example project contains a USB thread. This thread contains example code both for the Device and Host interfaces. Open USB.c and scroll down until you see this section of code:



```c
    while (1)
    {
        memset (machineGlobalsBlock->USBBufferB, 0, 49);
        status = _ux_device_class_cdc_acm_read (g_cdc, machineGlobalsBlock->USBBufferB, 49, &actual_length);
        if (status == UX_SUCCESS)
        {
            if (machineGlobalsBlock->USBBufferB[0] == 'U' && machineGlobalsBlock->USBBufferB[1] == 'S'
                    && machineGlobalsBlock->USBBufferB[2] == 'B')
            {
                status = _ux_device_class_cdc_acm_write (g_cdc, (UCHAR *) "USB", 4, &actual_length);
            }
            else if (machineGlobalsBlock->USBBufferB[0] == 'T' && machineGlobalsBlock->USBBufferB[1] == 'M'
                    && machineGlobalsBlock->USBBufferB[2] == 'P')
            {
                float tempRead = 0.01;
                char tmpUSBOut[12];
                memset (tmpUSBOut, 0, 12);
                tmpUSBOut[0] = 'T';
                tmpUSBOut[1] = 'M';
                tmpUSBOut[2] = 'P';

                snprintf (tmpUSBOut + 3, 8, "%f", tempRead);
                status = _ux_device_class_cdc_acm_write (g_cdc, (UCHAR *) tmpUSBOut, 12, &actual_length);
            }
            else
            {
                processReceivedMsg (machineGlobalsBlock->USBBufferB);
            }
        }
        else
        {

        }
        tx_thread_sleep (1);
    }
```

Everything outside of this loop is related to the USB Host interface and can be ignored, or deleted.

Getting inside the loop, we see that initially there is a memset operation which clears the USB buffer of any previous contents which could remain from a prior receive operation.

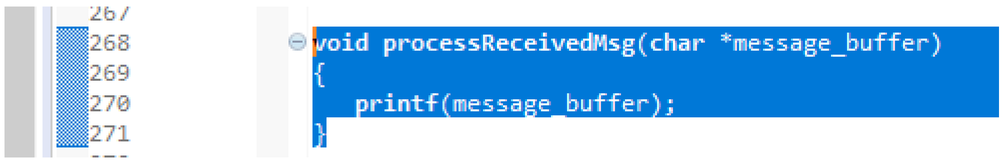
Next, we see a call to "_ux_device_class_cdc_acm_read". This function will wait until the Device interface reports that data has been received, at which point it will attempt to read 49 bytes. The received data will then be placed in the buffer found at "machineGlobalsBlock→USBBufferB", and the actual number of bytes received will be placed in the variable "actual_length".

Next, the status of the read operation is returned to the "status" variable, and on the next line, we see an if statement regarding this variable. If data was received successfully, then it continues into the enclosed lines. Next, the software moves into a series of statements that react to the received data.

First, it checks if the first three received characters are “USB”. This a control code that is sent by the GUI during the COM port identification process, and allows the controller to identify itself. In this case, the GUI sends the message “USB” to every serial port available, and if it receives a “USB” in response, it registers the sending COM port as the controller. You can see that the operation of this sending process is similar to receiving.

Next, if the message was not a “USB”, it checks for a “TMP”. This is another control code, but in this case it is shorthand for “Temperature”, which means the GUI is requesting a temperature update. This is an example case where the GUI would regularly request the temperature of a 3D printer extruder. This section of code shows how to identify a request from a connected device and return a floating point number as an ASCII string.

Finally, if the two previous checks have failed, we see “processReceivedMsg (machineGlobalsBlock→USBBufferB)” is called. This passes the starting pointer of the USB Buffer to the processReceivedMsg() function. Scrolling down, we see that this function simply prints the contents of the buffer to the console:



```
void processReceivedMsg(char *message_buffer)
{
    printf(message_buffer);
}
```

This function is here to provide an example and placeholder which can be modified for the intended application.

This concludes the USB Device Tutorial.