E/19/129

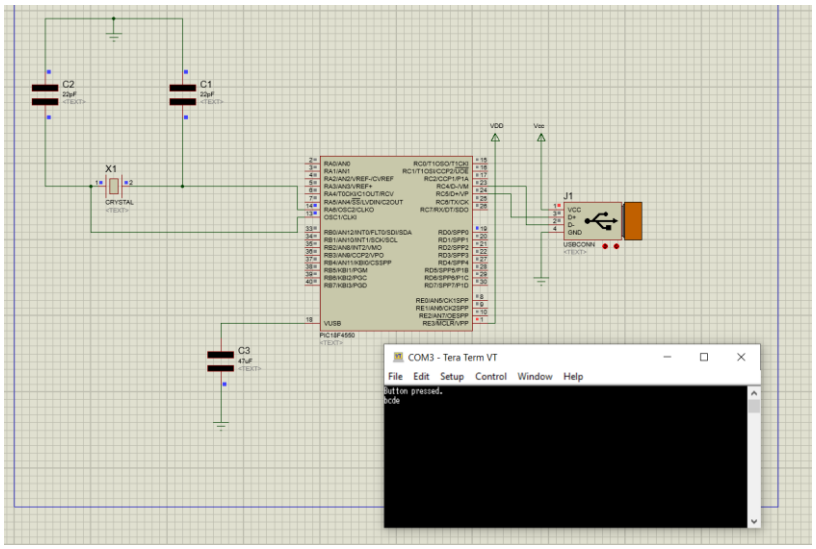
K.H. Gunawardana

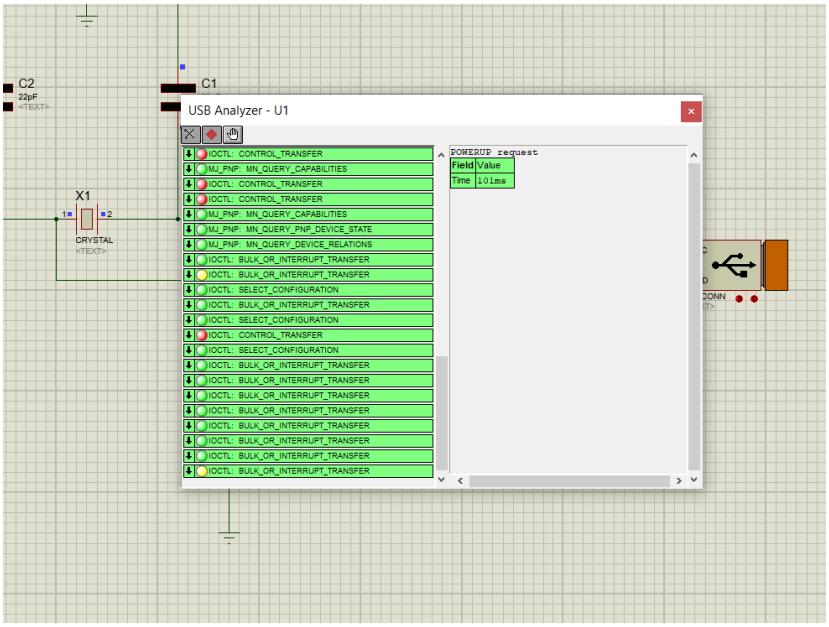
CO326 – Lab03

USB Port I/O

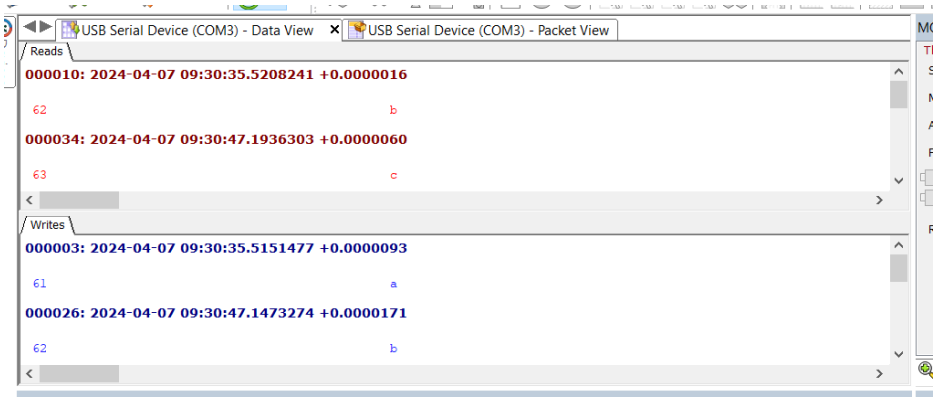
Example

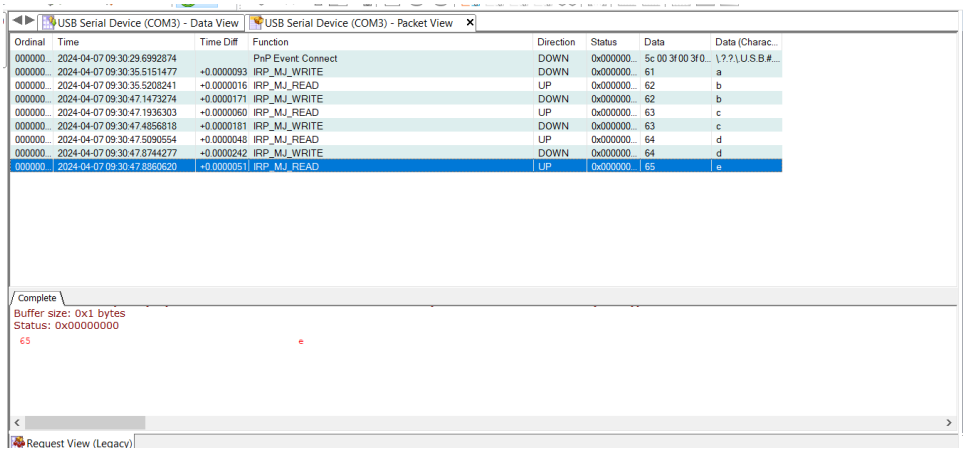
Design:





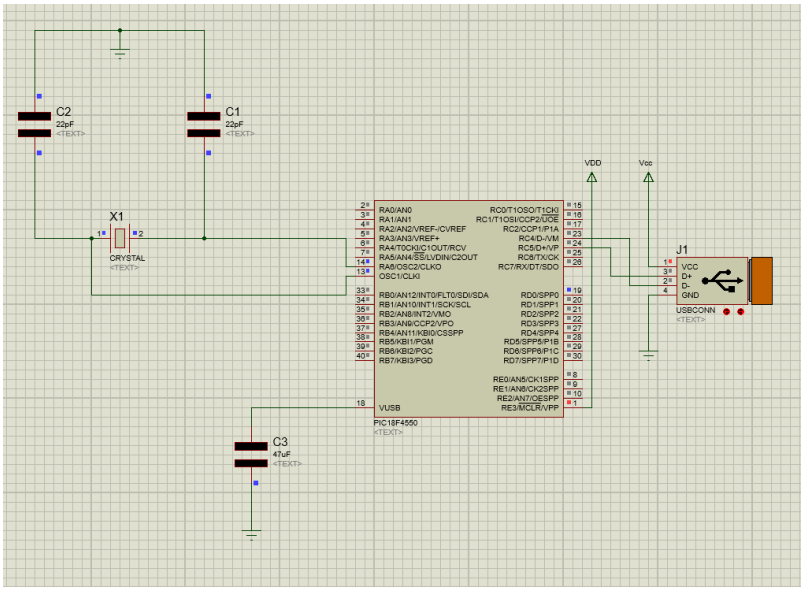
HDD Monitor:

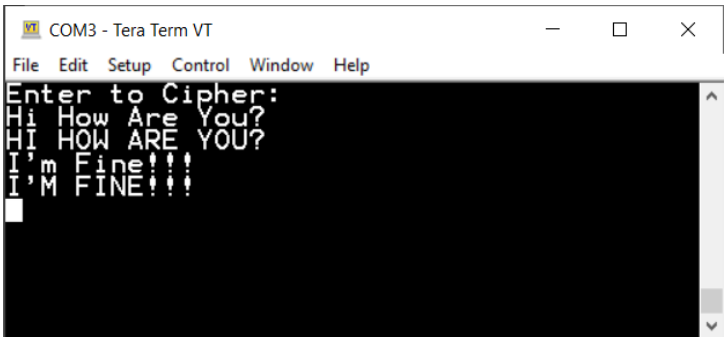

USB Serial Device (COM3) - Data View
USB Serial Device (COM3) - Packet View
Reads
000010: 2024-04-07 09:30:35.5208241 +0.0000016
62 b
000034: 2024-04-07 09:30:47.1936303 +0.0000060
63 c
Writes
000003: 2024-04-07 09:30:35.5151477 +0.0000093
61 a
000026: 2024-04-07 09:30:47.1473274 +0.0000171
62 b

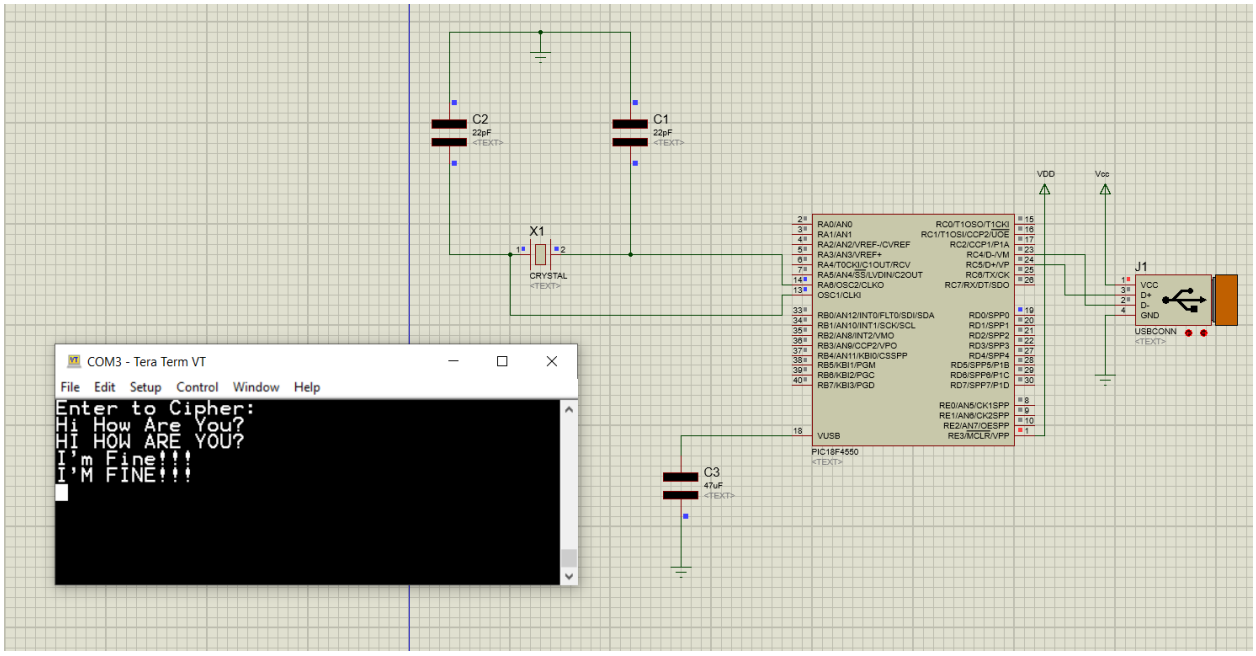

USB Serial Device (COM3) - Data View
USB Serial Device (COM3) - Packet View
Ordinal Time Time Diff Function Direction Status Data Data (Charac...
000000... 2024-04-07 09:30:29.6992874 PnP Event Connect DOWN 0x000000... 5c 00 3f 00 3f 0... \.?.?.\.U.S.B.#...
000000... 2024-04-07 09:30:35.5151477 +0.0000093 IRP_MJ_WRITE DOWN 0x000000... 61 a
000000... 2024-04-07 09:30:35.5208241 +0.0000016 IRP_MJ_READ UP 0x000000... 62 b
000000... 2024-04-07 09:30:47.1473274 +0.0000171 IRP_MJ_WRITE DOWN 0x000000... 62 b
000000... 2024-04-07 09:30:47.1936303 +0.0000060 IRP_MJ_READ UP 0x000000... 63 c
000000... 2024-04-07 09:30:47.4856818 +0.0000181 IRP_MJ_WRITE DOWN 0x000000... 63 c
000000... 2024-04-07 09:30:47.5090554 +0.0000048 IRP_MJ_READ UP 0x000000... 64 d
000000... 2024-04-07 09:30:47.8744277 +0.0000242 IRP_MJ_WRITE DOWN 0x000000... 64 d
000000... 2024-04-07 09:30:47.8860620 +0.0000051 IRP_MJ_READ UP 0x000000... 65 e
Complete
Buffer size: 0x1 bytes
Status: 0x00000000
65 e
Request View (Legacy)

Lab Task

Screenshots:







Code: app_device_cdc_bacis.c

```
/*******************************************************************************
Copyright 2016 Microchip Technology Inc. (www.microchip.com)
Licensed under the Apache License, Version 2.0 (the "License");
you may not use this file except in compliance with the License.
You may obtain a copy of the License at
   http://www.apache.org/licenses/LICENSE-2.0
Unless required by applicable law or agreed to in writing, software
distributed under the License is distributed on an "AS IS" BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
See the License for the specific language governing permissions and
limitations under the License.

To request to license the code under the MLA license (www.microchip.com/mla_license),
please contact mla_licensing@microchip.com
*******************************************************************************/

/** INCLUDES *******************************************************/
#include "system.h"
#include <stdint.h>
#include <string.h>
#include <stddef.h>
#include "usb.h"
#include "app_led_usb_status.h"
#include "app_device_cdc_basic.h"
#include "usb_config.h"

/** VARIABLES ******************************************************/
static bool buttonPressed;
static bool end = false;
static char buttonMessage[] = "Enter to Cipher:\r\n";
static uint8_t readBuffer[CDC_DATA_OUT_EP_SIZE];
static uint8_t writeBuffer[CDC_DATA_IN_EP_SIZE];
static uint8_t myWriteBuffer[CDC_DATA_IN_EP_SIZE];
static uint8_t count = 0;



// Function to convert lowercase English letters to uppercase
void convertToUpperCase(char *str) {
```

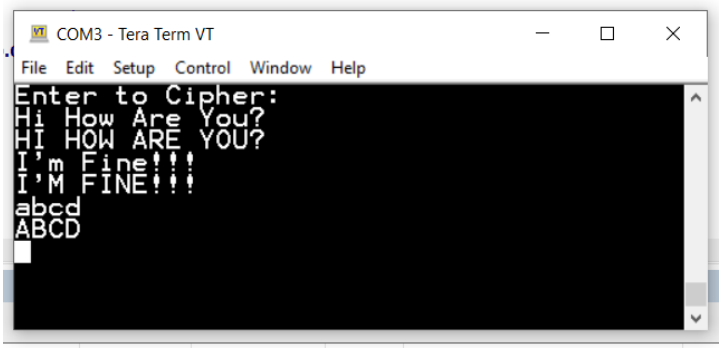
```
    while (*str) {
        if (*str >= 'a' && *str <= 'z') {
            *str = *str - 'a' + 'A';
        }
        str++;
    }
}

/********************************************************************
* Function: void APP_DeviceCDCBasicDemoInitialize(void);
* Overview: Initializes the demo code
* PreCondition: None
* Input: None
* Output: None
********************************************************************/
void APP_DeviceCDCBasicDemoInitialize()
{
    line_coding.bCharFormat = 0;
    line_coding.bDataBits = 8;
    line_coding.bParityType = 0;
    line_coding.dwDTERate = 9600;
    buttonPressed = false;
}

/********************************************************************
* Function: void APP_DeviceCDCBasicDemoTasks(void);
* Overview: Keeps the demo running.
* PreCondition: The demo should have been initialized and started via
*   the APP_DeviceCDCBasicDemoInitialize() and APP_DeviceCDCBasicDemoStart() demos
*   respectively.
* Input: None
* Output: None
********************************************************************/
void APP_DeviceCDCBasicDemoTasks()
{
    /* If the USB device isn't configured yet, we can't really do anything
     * else since we don't have a host to talk to.  So jump back to the
     * top of the while loop. */
    if( USBGetDeviceState() < CONFIGURED_STATE )
```

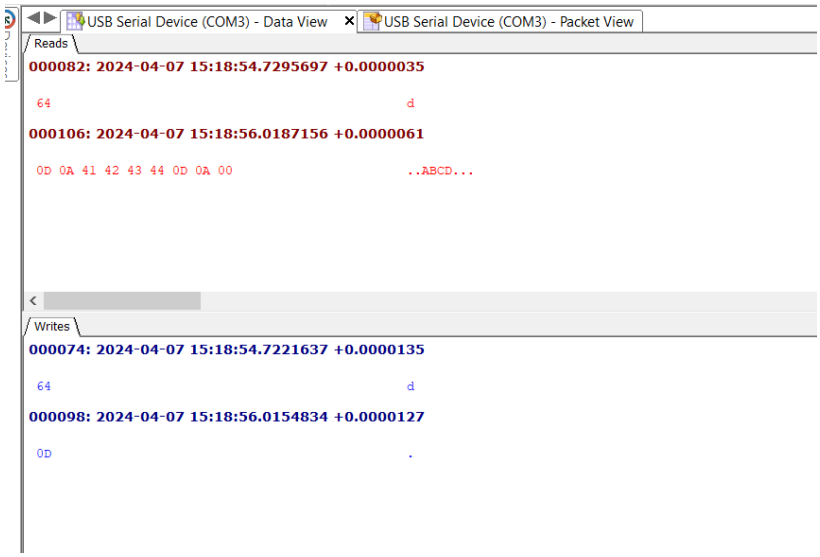
```
{
    return;
}

/* If we are currently suspended, then we need to see if we need to
 * issue a remote wakeup.  In either case, we shouldn't process any
 * keyboard commands since we aren't currently communicating to the host
 * thus just continue back to the start of the while loop. */
if( USBIsDeviceSuspended()== true )
{
    return;
}

/* If the user has pressed the button associated with this demo, then we
 * are going to send a "Button Pressed" message to the terminal.
 */
if(BUTTON_IsPressed(BUTTON_DEVICE_CDC_BASIC_DEMO) == true)
{
    /* Make sure that we only send the message once per button press and
     * not continuously as the button is held.
     */
    if(buttonPressed == false)
    {
        /* Make sure that the CDC driver is ready for a transmission.
         */
        if(mUSBUSARTIsTxTrfReady() == true)
        {
            putrsUSBUSART(buttonMessage);
            buttonPressed = true;
        }
    }
}
else
{
    /* If the button is released, we can then allow a new message to be
     * sent the next time the button is pressed.
     */
    buttonPressed = false;
}
```

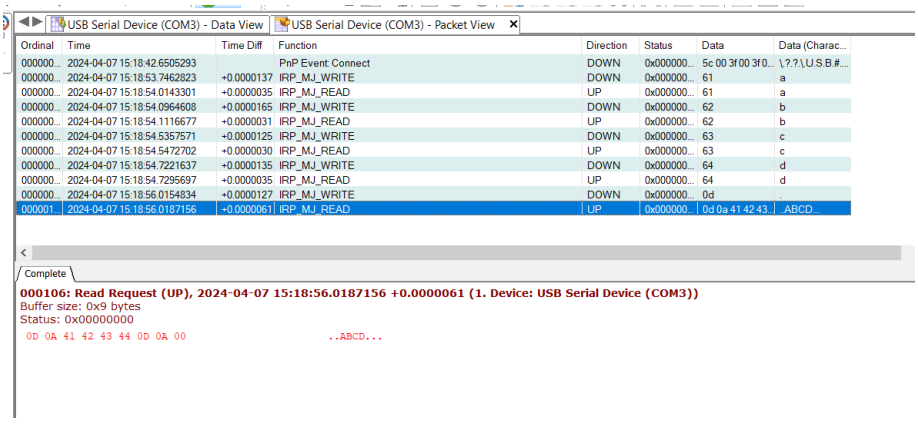
```
if( USBUSARTIsTxTrfReady() == true)
{
   uint8_t i;
   uint8_t numBytesRead;
   numBytesRead = getsUSBUSART(readBuffer, sizeof(readBuffer));

   /* For every byte that was read... */
   for(i=0; i<numBytesRead; i++)
   {

      switch(readBuffer[i])
      {
         case 0x0D:
            end = true;
            break;

         default:
            writeBuffer[i] = readBuffer[i];
            // store in a buffer
            myWriteBuffer[count] = readBuffer[i];
            count++;
            break;
      }

   }

   if(numBytesRead > 0 && end != true)
   {
      /* After processing all of the received data, we need to send out
       * the "echo" data now.
       */
      putUSBUSART(writeBuffer, numBytesRead);
   }
}
// print the cipher
if (end && mUSBUSARTIsTxTrfReady() == true){
   if(count > 0)
   {
```

```
            // Add newline and carriage return characters at the end of myWriteBuffer
            myWriteBuffer[count] = '\r';
            count++;
            myWriteBuffer[count] = '\n';
            count++;
            myWriteBuffer[count] = '\0';
            count++;

            // Convert received bytes to uppercase
            convertToUpperCase(myWriteBuffer);

            // Shift the characters in myWriteBuffer to right by 2 positions
            for (int j = count - 1; j >= 0; j--) {
                myWriteBuffer[j+2] = myWriteBuffer[j];
            }

            // Add newline and carriage return characters at the beginning of myWriteBuffer
            myWriteBuffer[0] = '\r';
            myWriteBuffer[1] = '\n';

            // display the cipher text
            putrsUSBUSART(myWriteBuffer);
        }

        // initiate
        end = false;
        count = 0;
    }
    CDCTxService();
}
```

HDD Monitor:







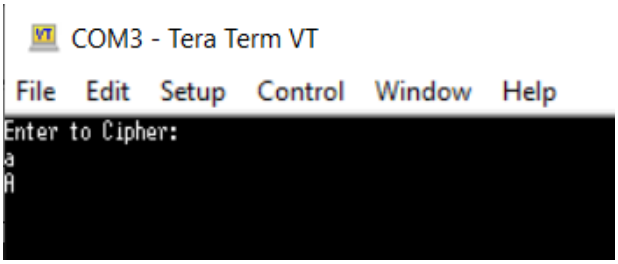
Problems and Solutions:

- **Terminal Connection Issue**: Initially, there was an issue with connecting the Tera Terminal to the application. This problem was resolved by restarting the computer, which likely refreshed the USB connections and allowed the terminal to establish a connection with the application.
- **Printing Characters Overlapping**: Characters were being printed on top of each other, causing readability issues. This was addressed by appending carriage return (`\r`) and newline (`\n`) characters at the end of each message and also at the beginning. This ensured that each message started on a new line, maintaining readability.

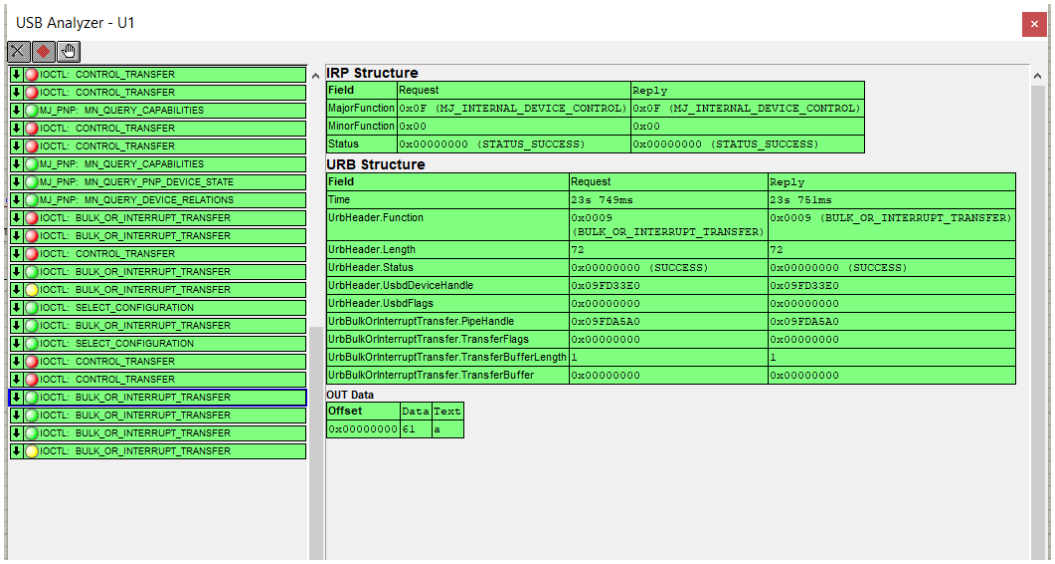
- **Echoing First Character Repeatedly**: Another issue encountered was the echoing of the first character repeatedly. To resolve this, a temporary array was introduced to store the full line of input before processing. This allowed for the complete line to be processed and echoed back without repetition of the first character.

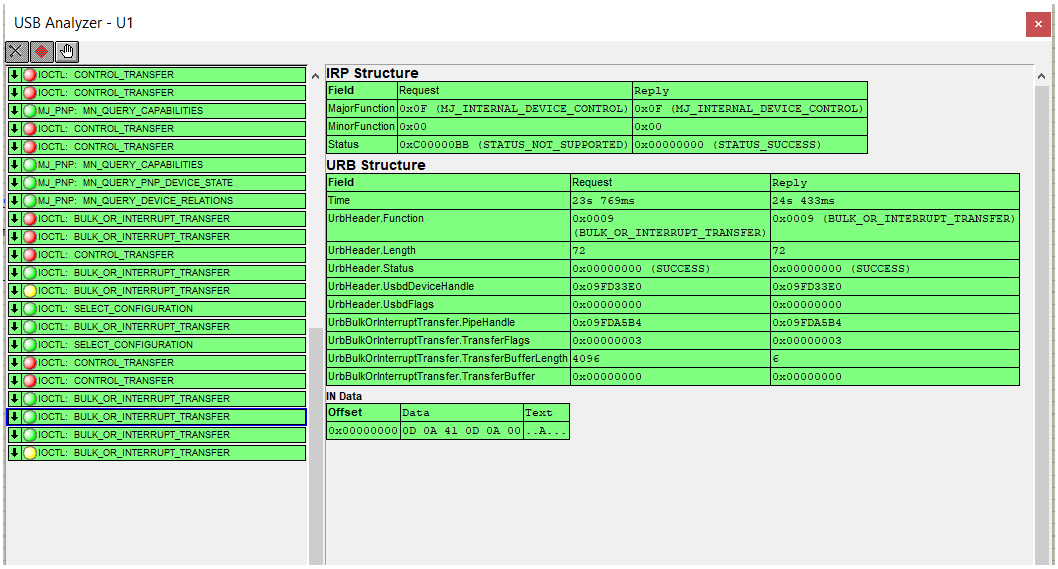
Explanations:

Consider ‘a’ as the input, thus using the Tera Terminal we can see, that first ‘a’ is echo and after pressing the enter ‘A’ gives back as the output.

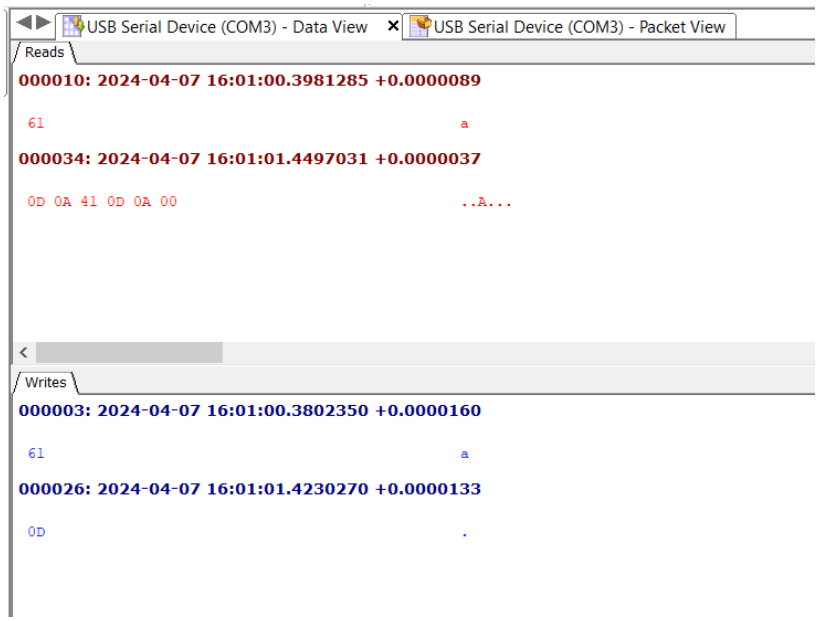


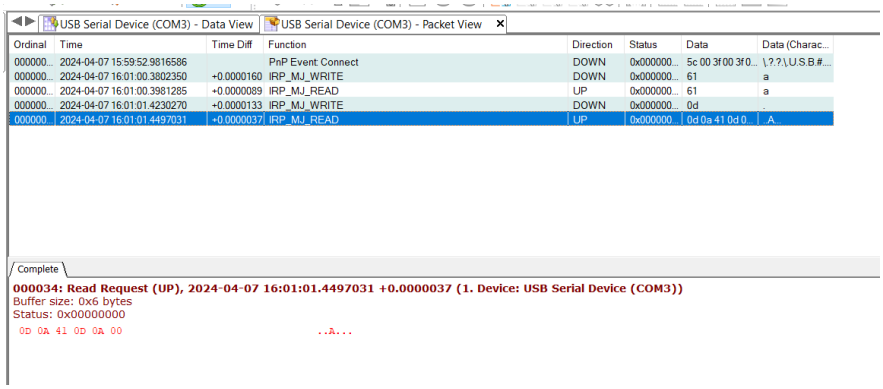
USB Analyzer:





HDD Monitor:





When observing the USB analyser, two types of packets can be identified: IN and OUT.

- IN Packets: These packets are sent into the USB port from the PIC18F4550 microcontroller. They represent data read by the USB port from the device. In the application context, IN packets contain data sent from the device to the host, such as characters typed on the keyboard or sensor readings.

- OUT Packets: Conversely, OUT packets are sent out through the USB port to the microcontroller. These packets contain commands or data from the host computer to be processed by the device. In the application, OUT packets could include commands or configuration settings sent from Tera Term to the USB device, such as baud rate adjustments or other control commands.