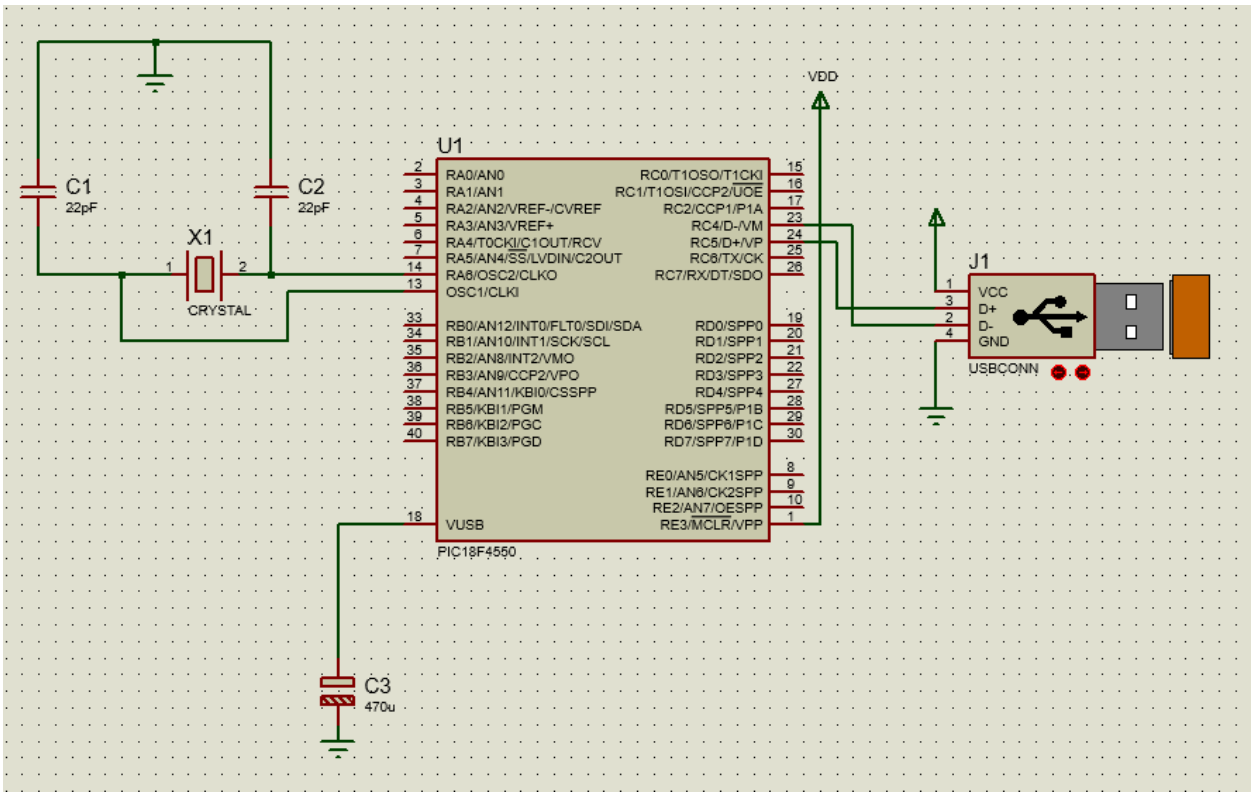
CO326 - Lab 3

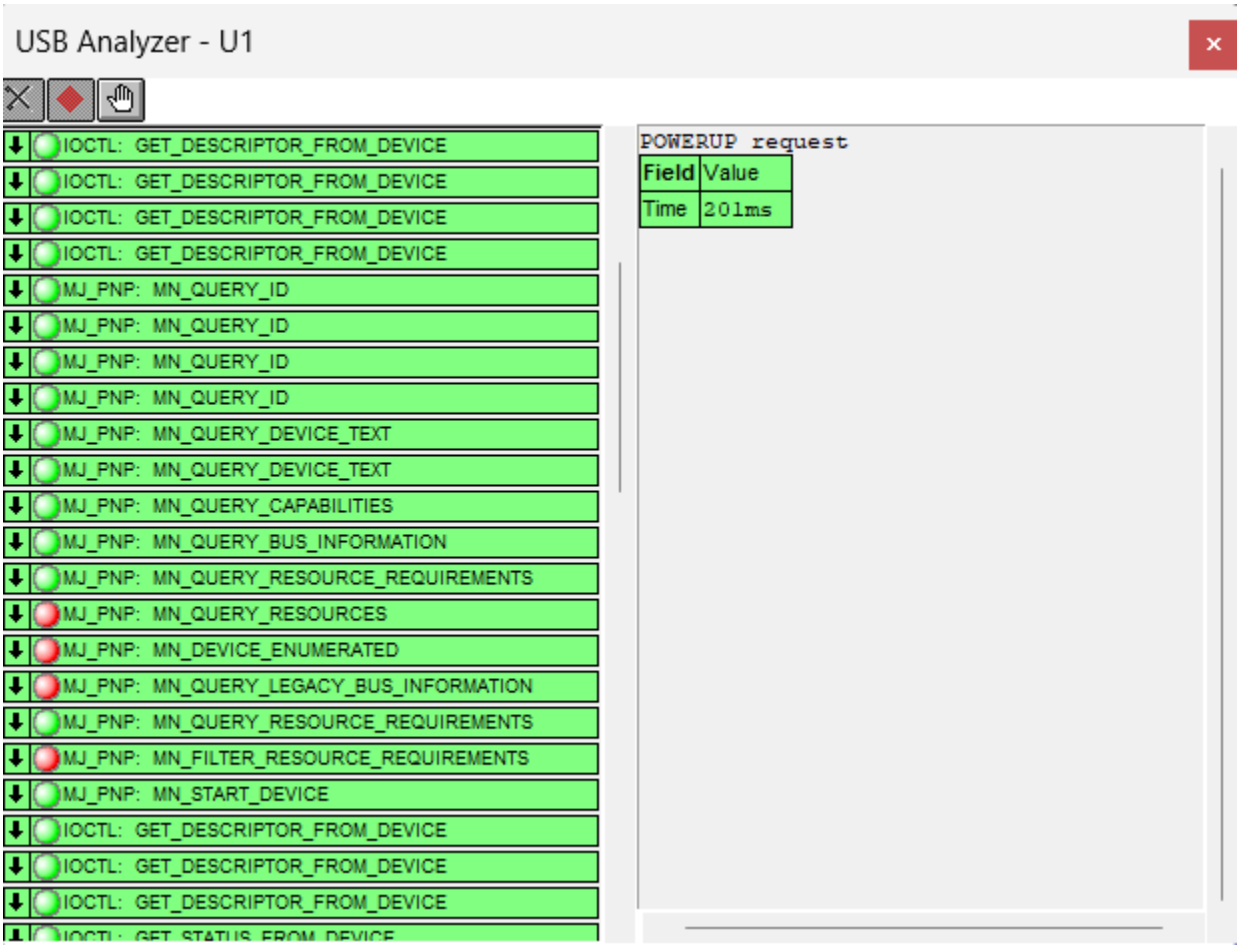
Jayathunga W.W.K.

E/19/166

Proteus Setup

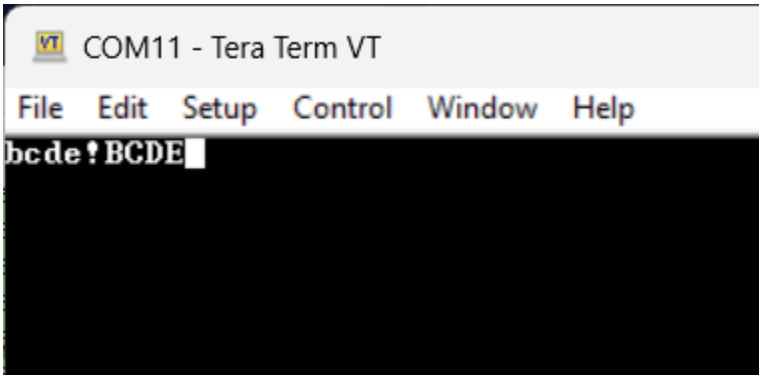


USB Analyzer

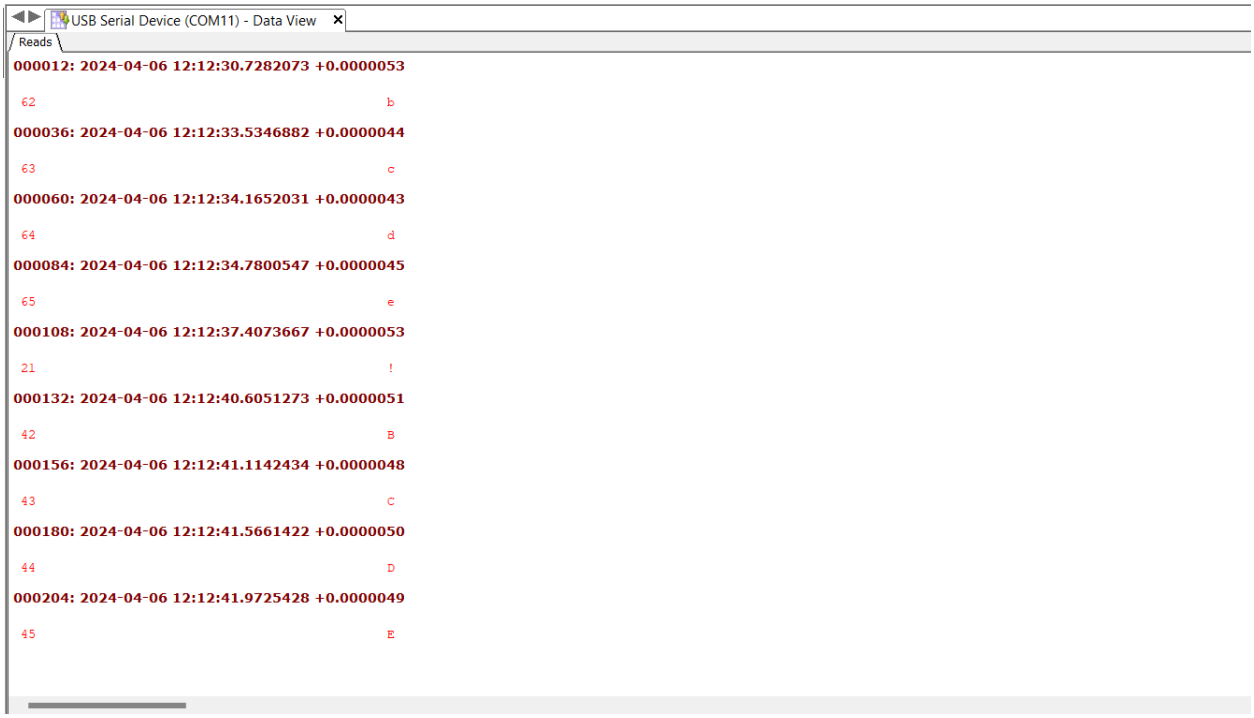


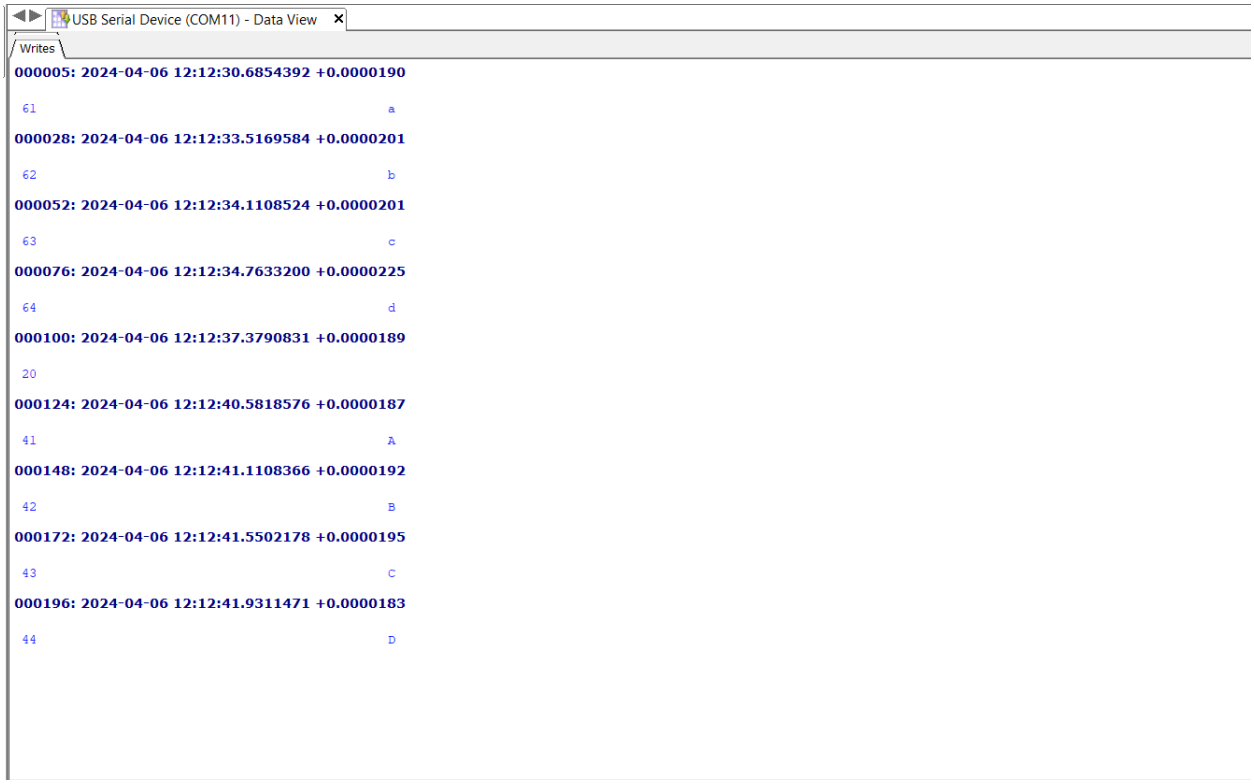
Lab Example (When the input is abcd ABCD)

Tera Term Terminal Window



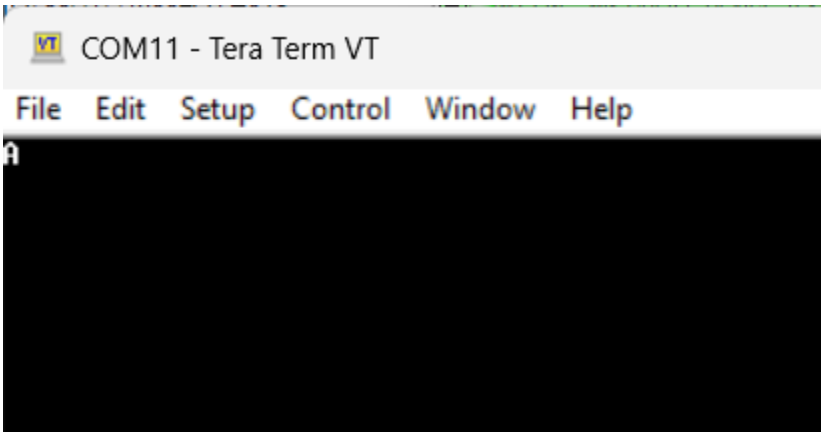
Device Monitoring Studio Screen



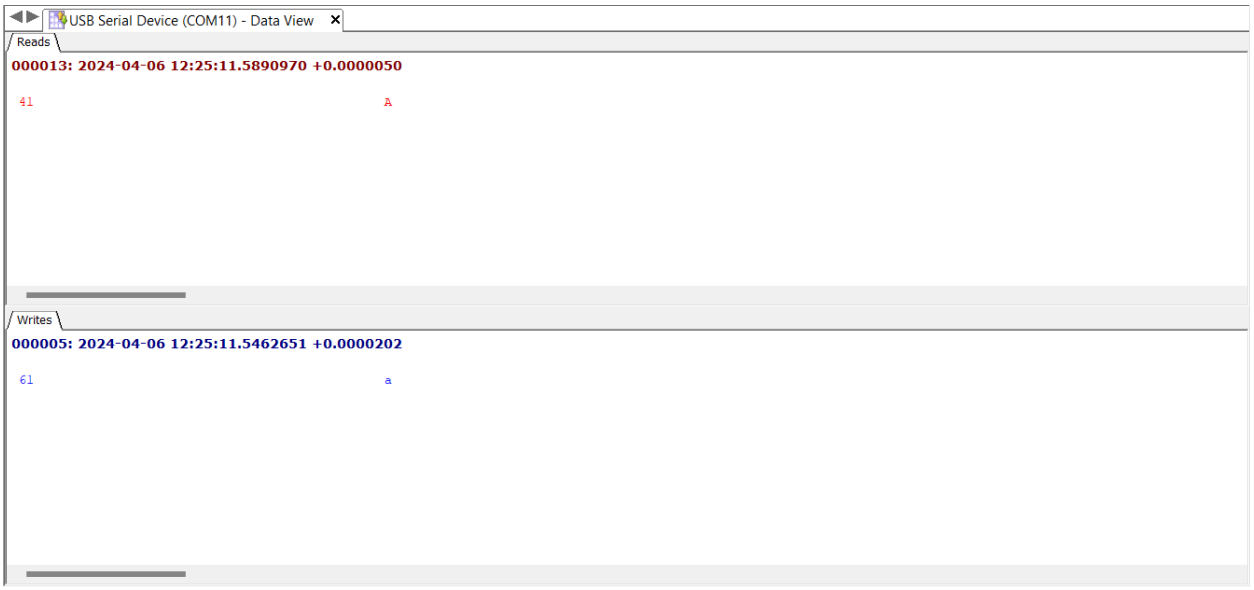


Lab Exercise (When the letter ‘a’ is typed)

Tera Term Terminal Screen

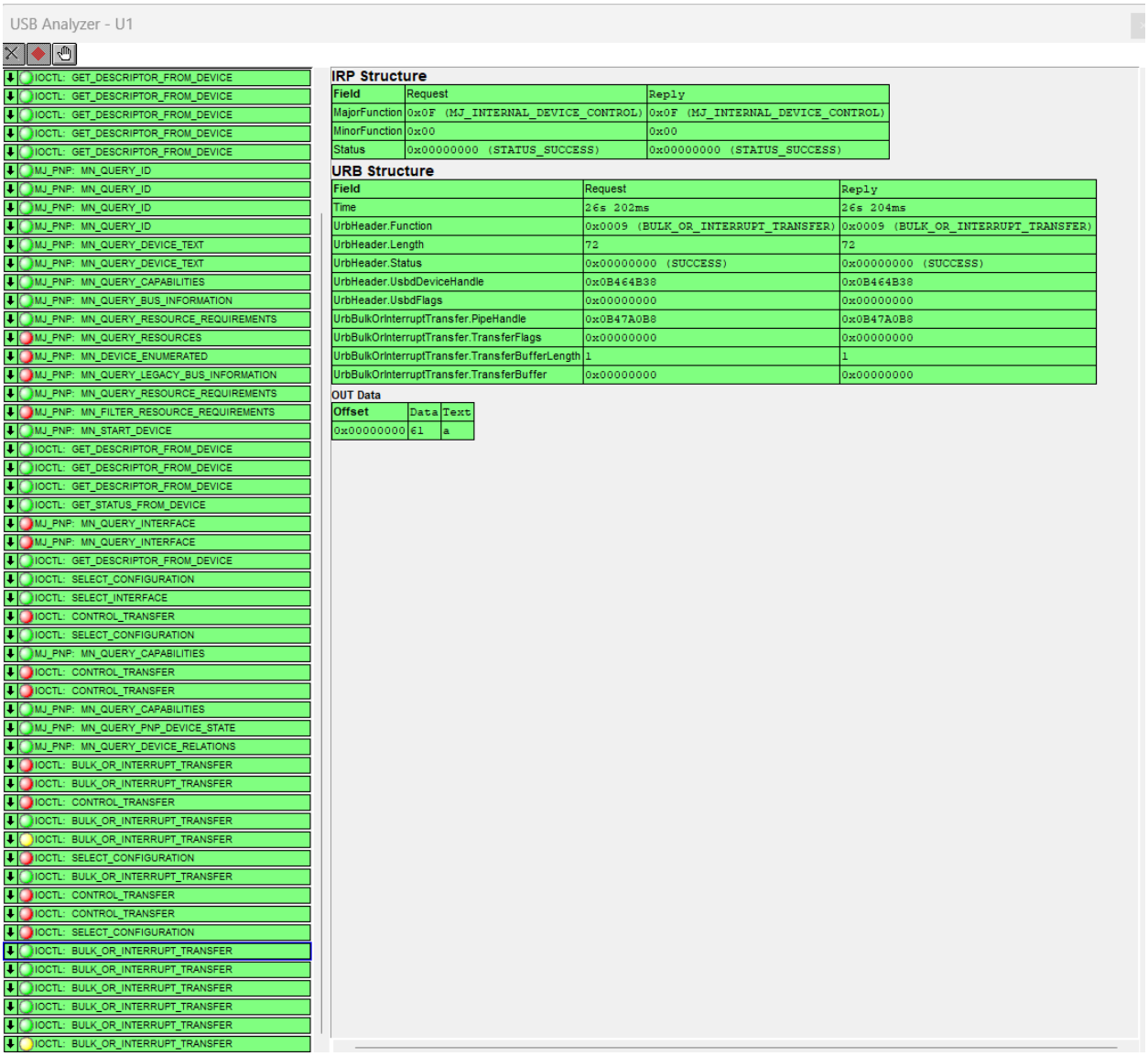


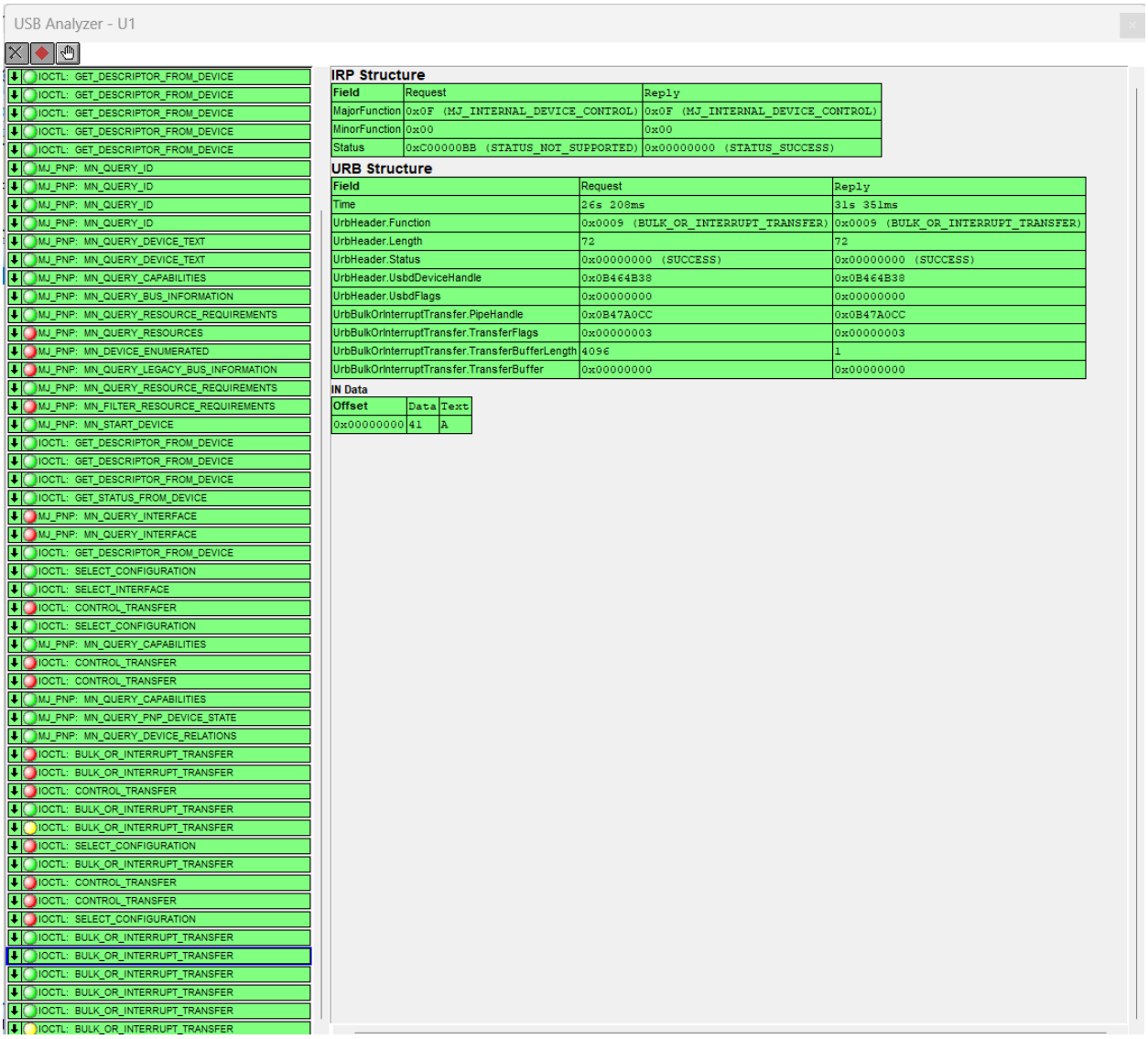
USB Monitoring Studio Screen



‘a’ is sent to the microcontroller through the USB port and the microcontroller converts it to ‘A’ and sends it back through the USB port.

USB Analyzer





There are two different kinds of packets that can be observed when using the USB analyzer: IN packets and OUT packets. In USB communication, IN and OUT packets refer to the direction of data transfer between a USB device (like your PIC microcontroller) and the USB host (usually your computer)

- The packet that the PIC18F4550 microcontroller sends into the USB port is referred to as IN, (Data that the USB port reads). The device is considered to be "initiating" the data transfer by pushing information towards the host. The host expects this data and has issued a request for it beforehand (often through control transfers).

- OUT denotes the packet that is transmitted to the microcontroller via the USB port.(data that the USB port wrote). The host is considered to be "sending out" instructions or data for the device to process. This could be commands to control the device, data for the device to display, or files to be transferred to the device.

Code from MPLAB for the Lab Task

```
/*******************************************************************************
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 char buttonMessage[] = "Button pressed.\r\n";
static uint8_t readBuffer[CDC_DATA_OUT_EP_SIZE];
static uint8_t writeBuffer[CDC_DATA_IN_EP_SIZE];

/*********************************************************************
* 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;
    }

    /* Check to see if there is a transmission in progress, if there isn't, then
     * we can see about performing an echo response to data received.
     */
    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++)
        {
            /* If we receive line feed command, the program stops
            scanning */
            char c = readBuffer[i];
            if(c == 0x0D)
            {
                writeBuffer[i] = c;
                break;
            }
            else if (readBuffer[i]>96 && readBuffer[i]<123)
            {
                /* if a lowecase character is read, convert it to
                uppecase */
                writeBuffer[i] = readBuffer[i]-32;
            }
            else
            {
                writeBuffer[i] = readBuffer[i];
            }
          }

        if(numBytesRead > 0)
        {
            /* After processing all of the received data, we need to send out
             * the "echo" data now.
             */
            putUSBUSART(writeBuffer,numBytesRead);
        }
    }

    CDCTxService();
}
```

Problems Encountered

1. *Compilation Errors:*

   Problem: I have encountered errors related to function redefinition, where the same function is defined multiple times.

   Solution: Checked for duplicate inclusions of header files or functions defined in both a header and a source file. Ensured each function is declared (prototyped) only once in a header and defined (with function body) only in the corresponding source file.

2. *No USB Recognition:*

   Problem: The computer has not recognized the simulated USB device.

   Solution: Verified the virtual COM port selection in Tera Term matches the one created by the simulation. Ensured the virtual USB driver for Proteus is installed and functioning correctly. I had to reinstall it.

3. *Incorrect Data Transmission:*

   Problem: The received data on the terminal was not be what I expected (e.g., missing characters, incorrect capitalization).

   Solution: Carefully reviewed my code for logic errors in character conversion and loop implementation. Ensured the conversion only applies to lowercase English alphabets (a-z) and the loop continues reading characters until the carriage return. Utilized debugging tools within MPLAB X IDE to step through my code line by line. This helped identify where the conversion might be failing.

4. *Difficulty Understanding Code Modification:*

   Problem: I struggled to understand how to modify the code to achieve the desired functionality.

   Solution: Referred to online resources and tutorials on C programming for conditional statements (if/else) and lookup tables for character conversion.
   Searched for examples of similar code modifications related to USB communication and character manipulation.

5. *Crashing of Proteus*

Problem: After the simulation mode was left on for a few minutes, Proteus continued to crash. After that, each time Proteus crashed, the.hex file needed to be uploaded.

Solution: To some extent, the problem was resolved by turning on the simulation mode only when necessary, which removed the requirement to repeatedly upload the.hex file.