Massachusetts Institute of Technology
Department of Electrical Engineering and Computer Science
6.115 Microprocessor Project Laboratory Spring 2020

Final Project Tip

Compu-Trifecta

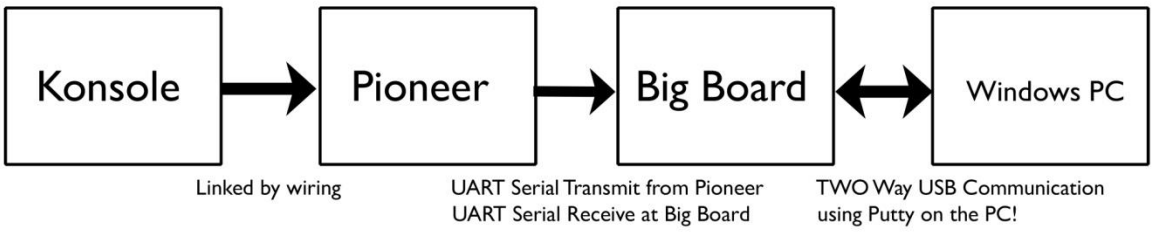
OR

Communicating with USB and Serial!

Ok, so this one is a little tricky, but quite rewarding...

Suppose we want to connect two PSoC boards so that they can talk to each other. Better still, what if we have visions of world domination, and wish to connect even more computers together? For example, let's imagine how we would connect our Pioneer board so that it could talk to our Big Board, and also connect our Big Board at the same time so that it could talk to a Windows 10 PC. There are of course many ways to create this "triple communication," but I'll just pick one here for illustration. Let's have the Pioneer board TRANSMIT a byte to the Big Board every half second using 9600 Baud Serial. Pioneer sends, Big Board receives. ALSO, Big Board will simultaneously conduct bi-directional communication over a USB cable with a PUTTY window on the PC! Here's an overview diagram and a peek at the final hardware:

Overview:



NOTE! AFTER Big Board has been programmed using the "normal" USB programming port.....

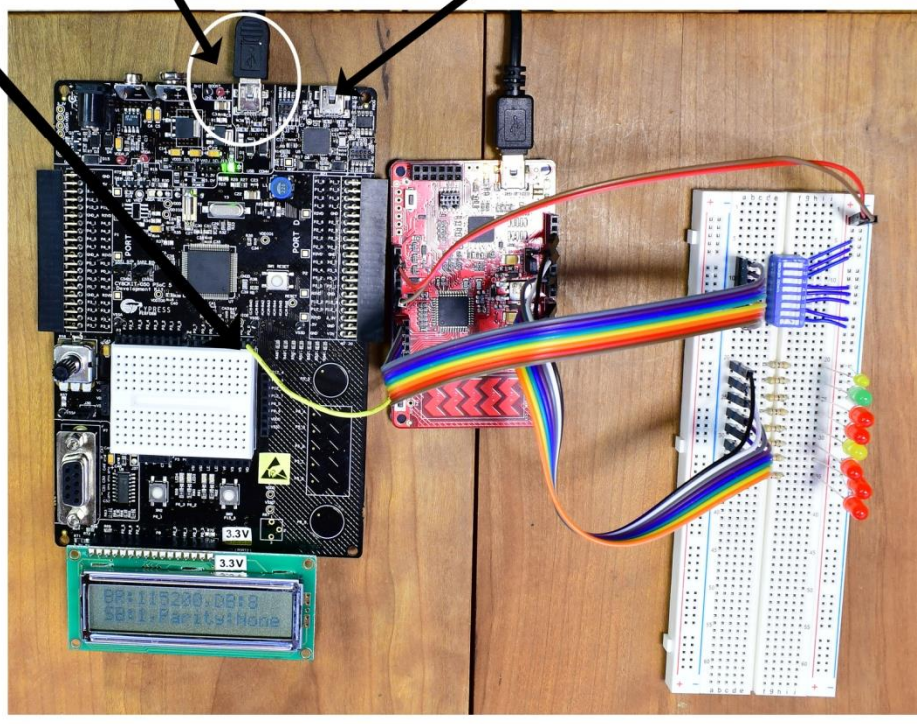
THEN we MOVE the cable so that it connects the "communication" USB port in the white ellipse to the Windows 10 PC!

I have added one wire that connects Big Board P0_0 to Pioneer P1_0 for serial communication, direct digital, NO line driver voltages.

NEVER, EVER connect two electronic systems with one wire, as I did here! Why? Systems MUST share a common ground, or the voltage on the wire could be meaningless or damaging to one or both systems.

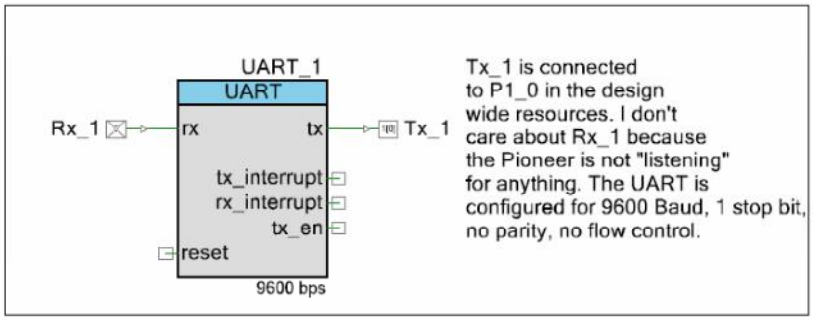
Why am I getting away with this? Because BOTH Big Board AND Pioneer are powered by USB from the SAME Windows PC. That creates a (sloppy) common ground!

Big Board is running a modified Cypress USB-UART example project...



1. **My Pioneer board** is programmed with a "simple" project that reads the Konsole switches, looks at the first four of them (I'm just working with the first nibble), and sends that as a byte to the Big Board every half second. The transmission is automatic, every half second. If I change the Konsole switches, it will get "picked up" and transmitted in the next half second message. The nibble I care about is in the lower half of the transmitted byte.

Here's the **PIONEER TopDesign Sheet:**



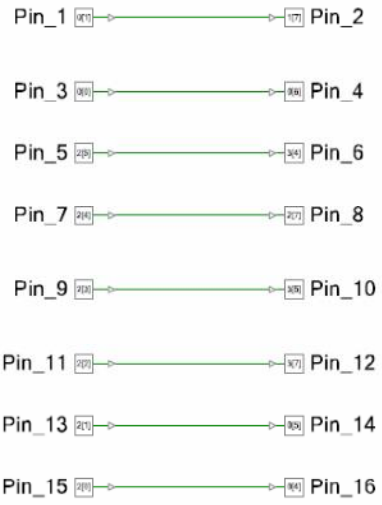
<<< This Box is the "big addition" to my Pioneer work to create the Compu-Trifecta....

These 8 pins are "inputs" connected to the 8 switches of the DIP package on the staff Kovid Konsole.

Each pin here is configured as a "resistive pull down" digital INPUT.

The 8 pins are connected to
P2_0 = J2 pin 1 through
P0_1 = J2 pin 15

Here, Pin_1 is connected to P2_0 and so on....

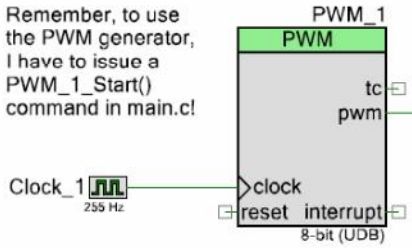


These 8 pins are "outputs" connected to the 8 LED's on the staff Kovid Konsole.

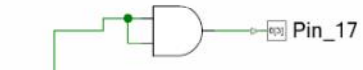
Pin_2 is connected to P1_7
Pin_4 is connected to P0_6 (J3 pin 6)
etc.

Pressing an INPUT switch to give a high INPUT lights up the associated OUTPUT LED!

Remember, to use the PWM generator, I have to issue a PWM_1_Start() command in main.c!

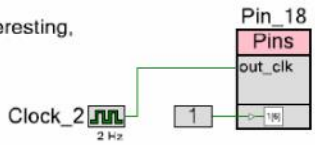


I have created a 1 Hz clock here, coming out of this PWM block. To do this, I set the PWM block up as 8 bit, and drove it with a 255 Hz clock, and set it for 50% duty cycle. This gives me a tasty 1 Hz 50% duty cycle clock at the PWM output.



In my design wide resources, pin assignments, my "Pin_17" is connected to Port 1_6, i.e., the pin that will flash the red part of the RGB LED on the Pioneer board.

This silly AND gate is just functioning as a "buffer" here. It is a place holder for something interesting, like the JK flip-flops in VE Lab 21.



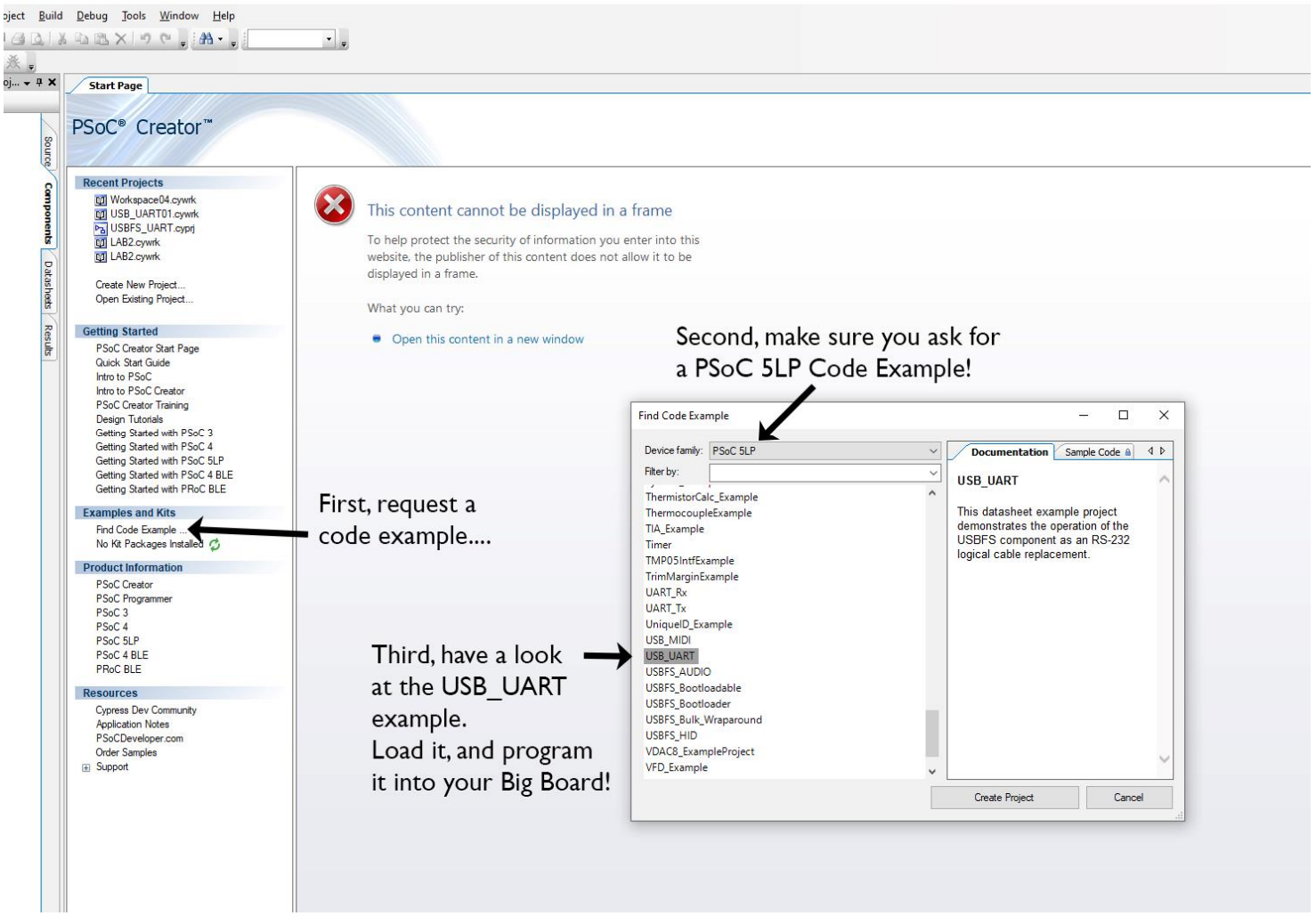
2. And here's **my Pioneer board** code:

```
#include <project.h>

int main()
{
      uint8 numb;
      uint8 p1, p2, p3, p4;

    CyGlobalIntEnable; /* Enable global interrupts. */
    PWM_1_Start();
    UART_1_Start();                         // initialize UART
    UART_1_ClearTxBuffer();

    /* Place your initialization/startup code here (e.g. MyInst_Start()) */

    for(;;)
    {
       p1 = Pin_1_Read();  // I'll use a nibble to adjust the offset on my
                           // ASCII.  p1 is LSB.
       p2 = Pin_3_Read();
       p3 = Pin_5_Read();
       p4 = Pin_7_Read(); // p4 is MSB
       numb = 0 + p1 + 2*p2 + 4*p3 + 8*p4;  // Put the nibble together, 0 if
                                            // no Kovid Keys are pressed

       UART_1_WriteTxData(numb); // Transmit the number
       CyDelay(255);  // Wait half a second
       CyDelay(255);


    }
}
```

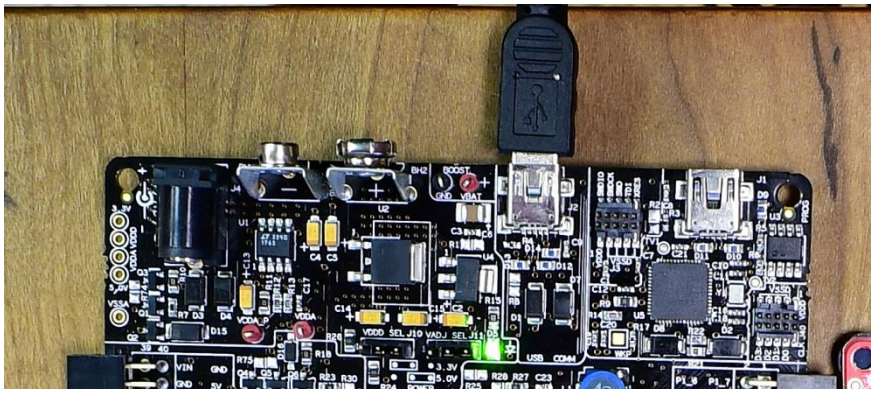
You can see the plan above. I define four integers, p1, p2, p3, p4. Each one will be loaded with either a "zero" or a "one" by Pin_X_Read() for the first four switch inputs on my Konsole. I assemble a "final answer" in the uint8 variable "numb", and then I transmit "numb" out the UART. I do this every half second, like it or not, over and over. If you change any of the first four dip switches on my Konsole, a new number will be computed for "numb" and transmitted in the "next" half second.

We haven't set it up yet, but the Big Board is supposed to be "receiving" this byte every half second. Let's see how the Big Board is set up, and what it will do with the number.....

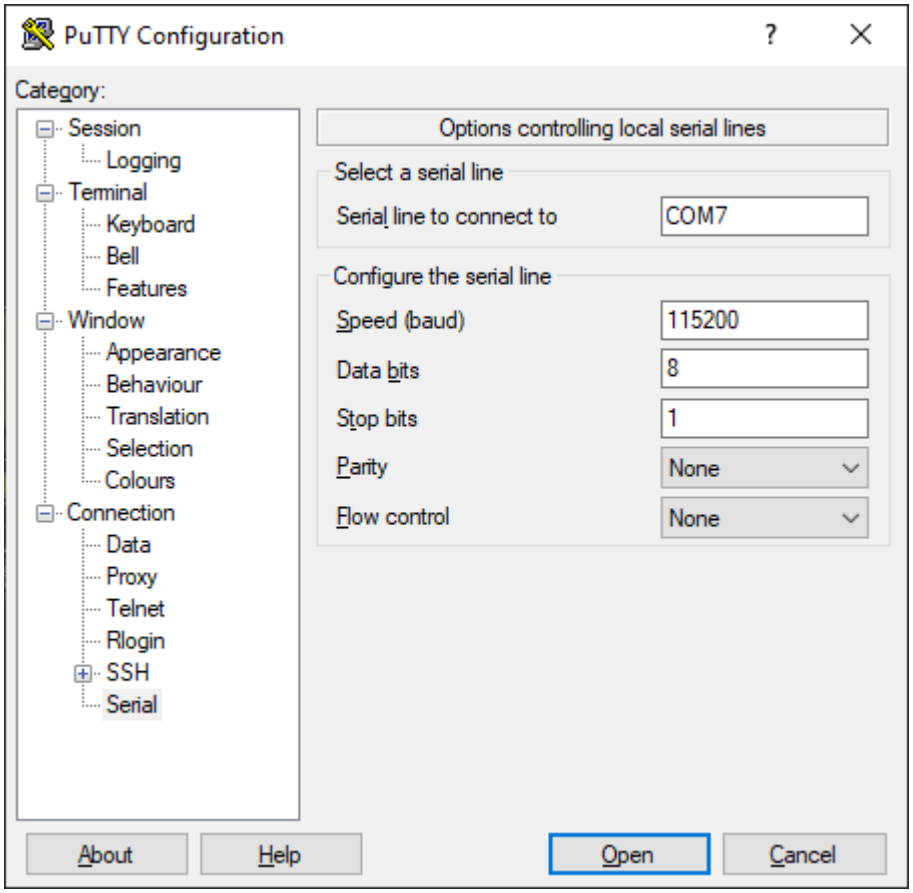
3. FIRST, try this, just to see how USB-UART (NOT regular UART!) communications work. It's a little bit of work. But, good news, Cypress took care of us by including a demo project. It's built into Creator! Here's how to see it: Unplug your Pioneer and put it somewhere safe. Close Creator. Connect your Big Board to your PC using the normal USB programming cable and connector. Open Creator. Try the Cypress Demo Project first:



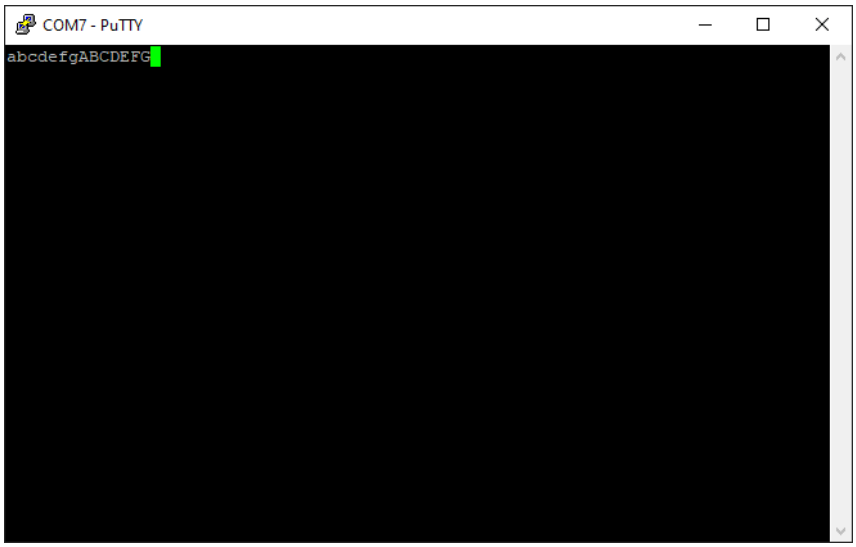
When you have programmed your Big Board with this project, you can close Creator temporarily if you want. Unplug the USB cable from the "programming" USB port, and move it to the "communication" USB port as shown in this picture:



4. As we describe in Take Home Lab 2, use "Device Manager" on your Windows PC to find out where the "Big Board" showed up as a COM port (yes, even though it's a "USB" connection, it will show up on your PC as a COM port!). On my PC, it showed up as COM7. So I set up to talk to the Big Board from my PC by opening a PUTTY window:



Note how fast the USB-UART connection is! I can use 115200 Baud rate, which rocks. Now I click open, and I get this PUTTY terminal:



This remarkable gift project from Cypress does a relatively simple demo. When I type "abcdefgABCDEFG" into the terminal window, PUTTY sends the characters to the Big Board over the USB cable. The Big Board sends (or "echos") them back so they show up correctly after each key press, just as I typed them, in the PUTTY terminal window.

Neat!

5. Now let's have some fun with the Big Board project that Cypress gave us. I've now **modified** the TopDesign Sheet as shown below:

USB UART Datasheet Example Project

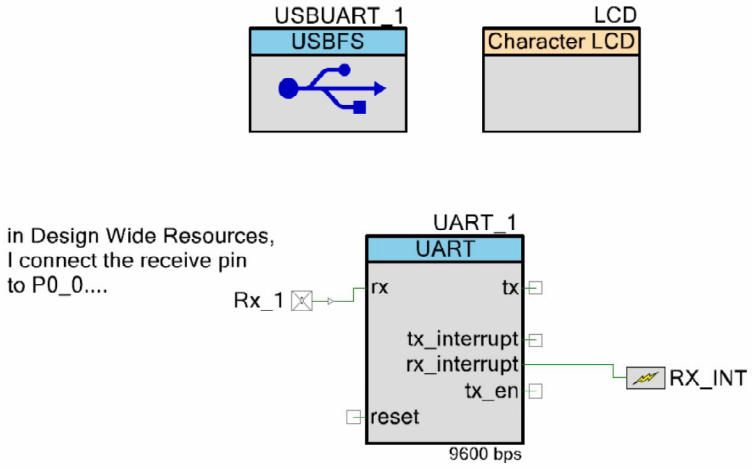
This project enumarates on the computer as a Virtual Com port. Receives data from hyper terminal then sends received data backward. LCD shows the Line settings.

UNCLE STEVE HERE: I'VE MODIFIED THIS SHEET FROM THE INCLUDED CYPRESS EXAMPLE. I'LL USE CYPRESS' USB_UART TO TALK TO A PC/PUTTY INTERACTION. AND I'VE ADDED A REGULAR
UART TO LISTEN FOR BYTES COMING FROM MY PIONEER BOARD...

Development Kit Configuration

1. Build the project and program the hex file into the target device.
2. Select 3.3V in SW3 and plug-in power to the CY8CKIT-001.
3. Connect USB cable from the computer to the CY8CKIT-001.
4. UNCLE STEVE HERE: I'M LEAVING THESE INSTRUCTIONS AS THEY COME UP IN THE ORIGINAL CYPRESS PROJECT. HOWEVER, I DON'T THINK YOU NEED TO DO THIS DRIVER INSTALLATION THAT IS DISCUSSED FOR WINDOWS 7. IT COMES UP AUTOMATICALLY ON MY WINDOWS 10 INSTALLATION, HOPEFULLY ON YOURS TOO: Install drivers for the device manually. In Windows 7, open "Device Manager", find the "USBUART" device in the "Other devices" branch of the tree. Open the context menu and select "Update Driver Software". Browse to USBUART_1_cdc.inf file from the project generated sources directory as a

---



Specifically, my "big addition" is UART_1. I'm just using it to "listen" for a byte in this little demo Trifecta project. It's configured as 9600 Baud, 8 bits, 1 Stop bit, no parity, no flow control in order to receive from the Pioneer board.

The next TWO pages show the code, MODIFIED from the Cypress demo project. The modification allows the UART to receive a byte transmitted by Pioneer. When the Big Board receives a key press letter from the PUTTY window, it now ADDS the Pioneer byte (stored in the variable "SILLY") to the key press ASCII code before returning it to the PC window.

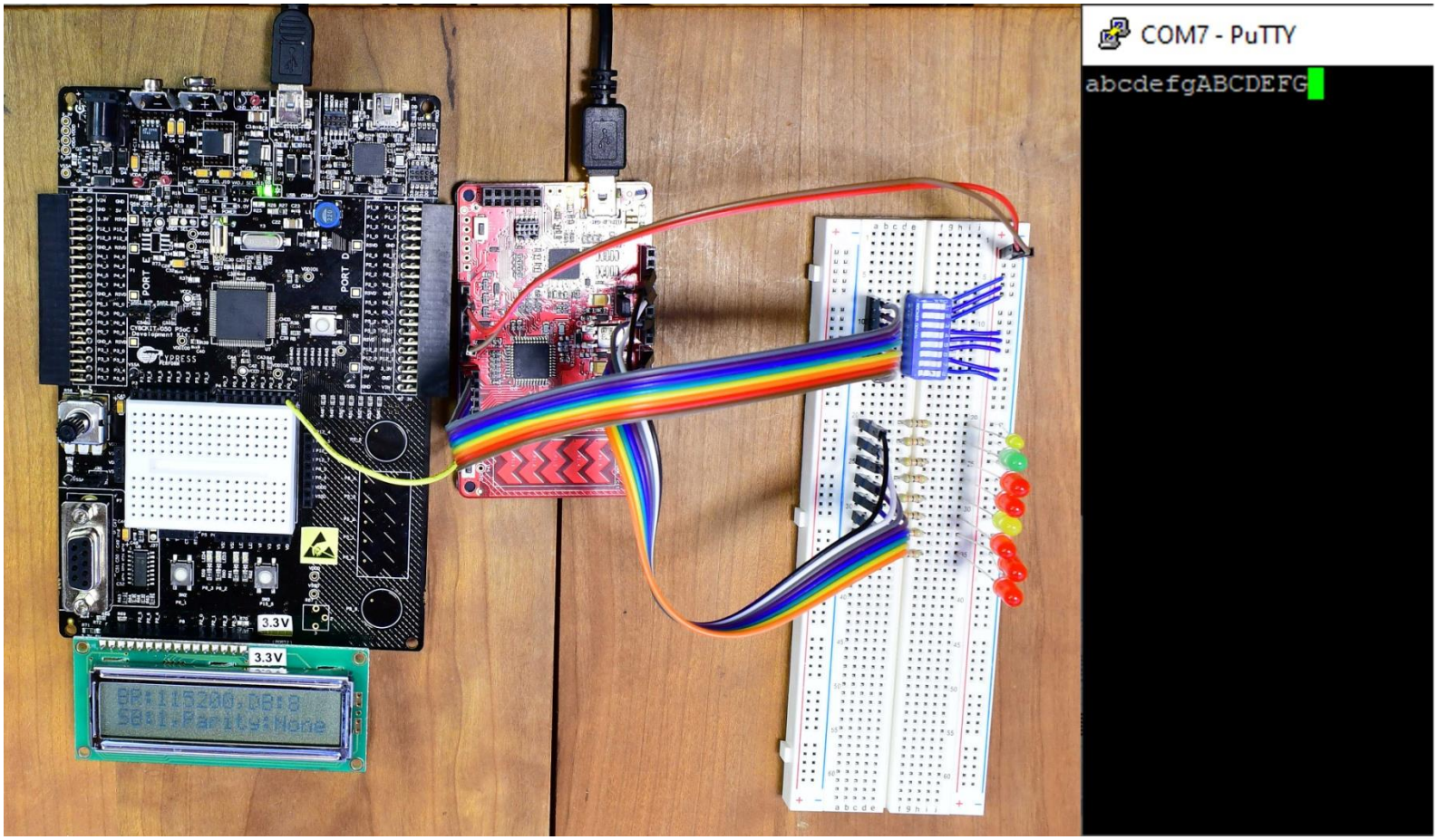
My comments showing my changes are in CAPITAL letters.

Check it out:

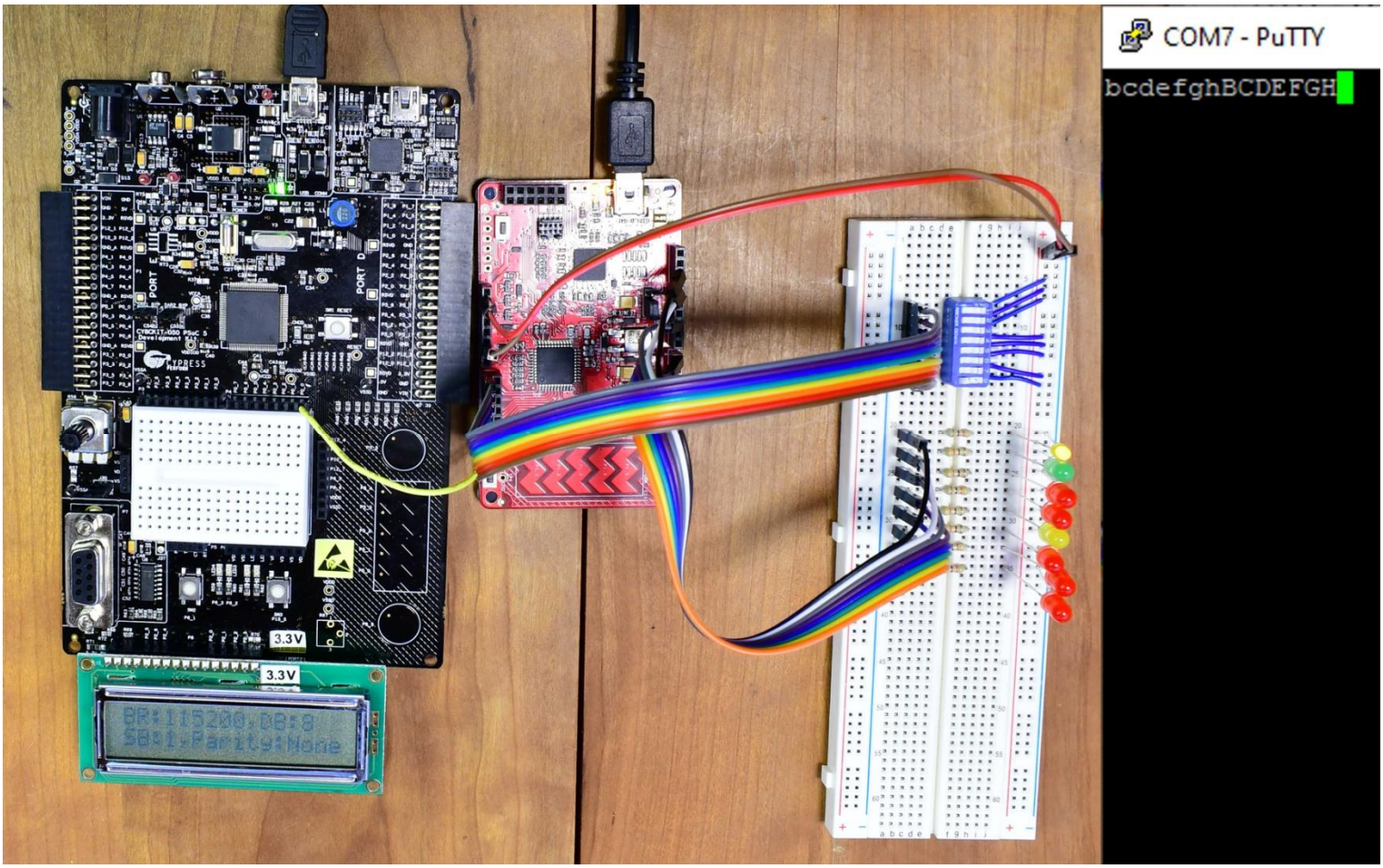
```
#include <device.h>
#include "stdio.h"

#if defined (__GNUC__)
    asm (".global _printf_float");
#endif
#define BUFFER_LEN  64u
char8 *parity[] = { "None", "Odd", "Even", "Mark", "Space" };
char8 *stop[] = { "1", "1.5", "2" };
uint8 silly[] = {1};  // MY VARIABLE "SILLY" LET'S ME RECEIVE A BYTE
                      // FROM PIONEER AND ADD IT TO THE ECHO BYTE.

CY_ISR(RX_INT)  // I'VE MODIFIED THE CY USB_UART PROJECT TO INCLUDE UART COMMUNICATION
{               // MY PLAN IS TO GET A BYTE WHENEVER MY PSOC PIONEER BOARD SENDS ONE.
    uint8 numb;
    numb = UART_1_ReadRxData();  // I STORE THE BYTE IN NUMB
    silly[0] = numb;             // AND PUT IT IN THE GLOBAL ARRAY "SILLY"
}
int main()
{
    uint16 count;
    uint8 buffer[BUFFER_LEN];
    char8 lineStr[20];
    uint8 state;

    CyGlobalIntEnable;    /* Enable Global Interrupts */
    USBUART_1_Start(0u, USBUART_1_3V_OPERATION);  /*Start USBFS with 3V operation */
    LCD_Start();/* Start LCD */
    LCD_PrintString("USB-UART example");

    RX_INT_StartEx(RX_INT);  //I'VE MODIFIED USB_UART TO
                             // INCLUDE STARTUPS FOR A REGULAR UART INTERRUPT
    UART_1_Start();          // AND A START FOR THE UART ITSELF
    UART_1_ClearRxBuffer();

    for(;;)      /* Main Loop: */

    {
        if(USBUART_1_IsConfigurationChanged() != 0u)
                          /* Host could send double SET_INTERFACE request */
        {
            if(USBUART_1_GetConfiguration() != 0u)
                          /* Init IN endpoints when device configured */
            {
            /* Enumeration is done, enable OUT endpoint for receive data from Host */
                USBUART_1_CDC_Init();
            }
        }
        if(USBUART_1_GetConfiguration() != 0u)
                                    /* Service USB CDC when device configured */
        {
            if(USBUART_1_DataIsReady() != 0u)    /* Check for input data from PC */
            {
                count = USBUART_1_GetAll(buffer); /*Read data and re-enable OUT endpoint */
                if(count != 0u)
                {
                    while(USBUART_1_CDCIsReady() == 0u);
                          /* Wait till component is ready to send more data to the PC
*/

                    buffer[0] += silly[0]; // NORMALLY, WE WOULD JUST SEND BACK
                                           // "BUFFER". BUT, HERE, I ADD
                                           // "SILLY" TO THE NUMBER IN BUFFER.
```

```
                    USBUART_1_PutData(buffer, count);       /* Send data back to PC */

                    /* If the last sent packet is exactly maximum packet size,
                    *  it shall be followed by a zero-length packet to assure the
                    *  end of segment is properly identified by the terminal.
                    */
                    if(count == BUFFER_LEN)
                    {
                        while(USBUART_1_CDCIsReady() == 0u);
                                /*Wait till component is ready to send data to PC */
                        USBUART_1_PutData(NULL, 0u);
                                /* Send zero-length packet to PC */
                    }
                }
            }

            state = USBUART_1_IsLineChanged(); /* Check for Line settings changed */
            if(state != 0u)
            {
                if(state & USBUART_1_LINE_CODING_CHANGED)   /* Show new settings */
                {

sprintf(lineStr,"BR:%4ld,DB:%d",USBUART_1_GetDTERate(),(uint16)USBUART_1_GetDataBits()
);
                    LCD_Position(0u, 0u);
                    LCD_PrintString("                    ");
                    LCD_Position(0u, 0u);
                    LCD_PrintString(lineStr);
                    sprintf(lineStr,"SB:%s,Parity:%s",
stop[(uint16)USBUART_1_GetCharFormat()], \

parity[(uint16)USBUART_1_GetParityType()]);
                    LCD_Position(1u, 0u);
                    LCD_PrintString("                    ");
                    LCD_Position(1u, 0u);
                    LCD_PrintString(lineStr);
                }
                if(state & USBUART_1_LINE_CONTROL_CHANGED)  /* Show new settings */
                {
                    state = USBUART_1_GetLineControl();
                    sprintf(lineStr,"DTR:%s,RTS:%s",  (state &
USBUART_1_LINE_CONTROL_DTR) ? "ON" : "OFF", \
                                                    (state &
USBUART_1_LINE_CONTROL_RTS) ? "ON" : "OFF");
                    LCD_Position(1u, 0u);
                    LCD_PrintString("                    ");
                    LCD_Position(1u, 0u);
                    LCD_PrintString(lineStr);

                }
            }
        }
    }
}
```

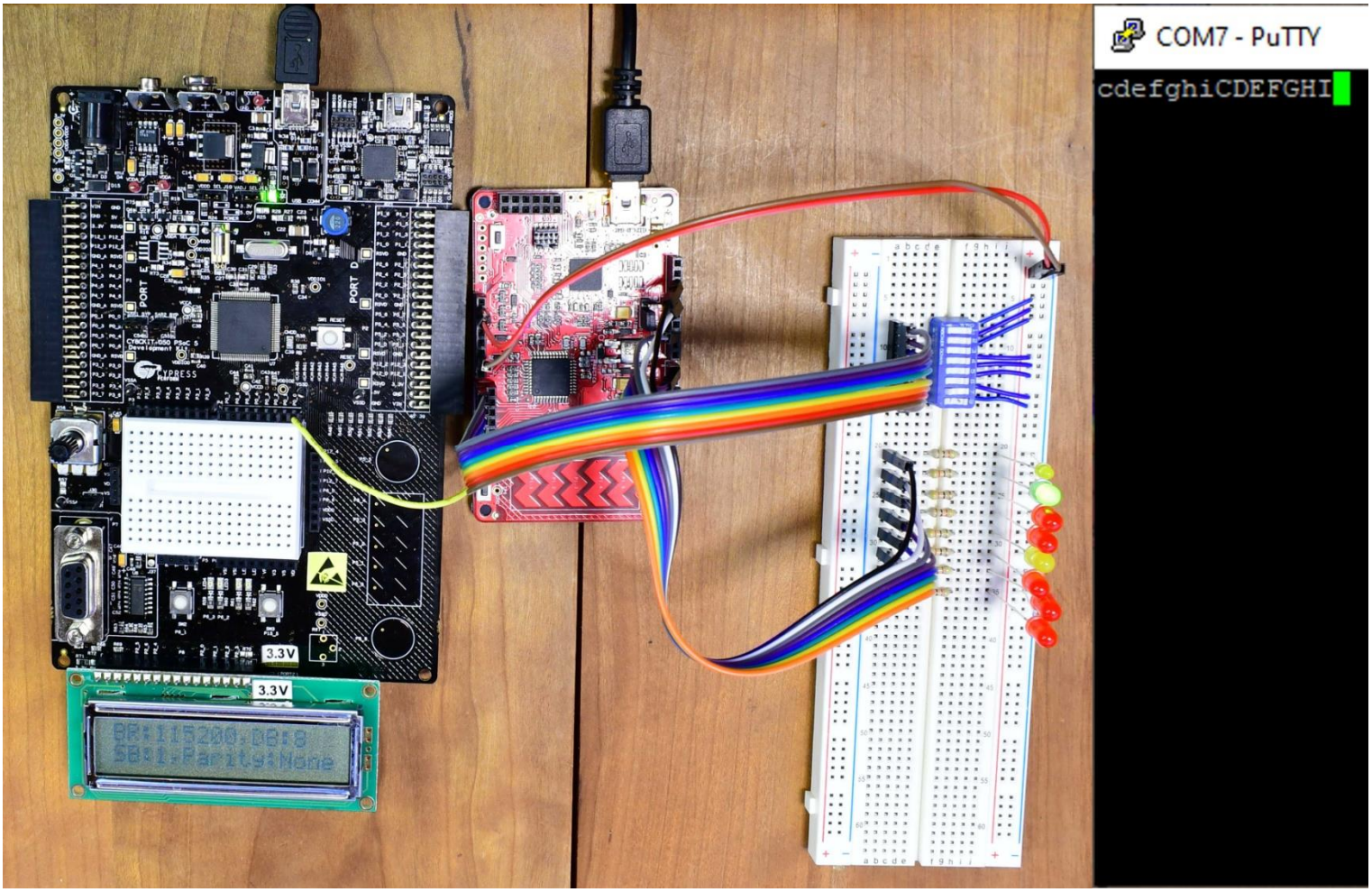
And here are the results for three different Konsole settings. First, I start with all Konsole switches in the off position. The PIONEER Board is sending "0" to the Big Board every half second. The "Big Board" adds the zero to the echo return, which of course causes no change. So I'll see exactly what I type. Here's the PUTTY terminal window as I type "abcdefgABCDEFG":



Now, I'll hit the FIRST switch on my Konsole to "ON". Notice that the first LED is glowing this time in the picture below of the Konsole. That should send a "1" every half second to the Big Board. So, when the Big Board "echos" the PUTTY terminal typing, it will add a "1" to every character. The letter "a" will echo as a "b" and so forth. Here's the PUTTY terminal window again when I type "abcdefgABCDEFG":



Finally, I'll set just the SECOND switch on my Konsole to "ON". Notice that the second LED is "on" this time in the picture below of the Konsole. That should send a "2" every half second to the Big Board. So, when the Big Board "echos" the PUTTY terminal typing, it will add a "2" to every character. The letter "a" will echo as a "c" and so forth. Here's the PUTTY terminal window again when I type "abcdefgABCDEFG":



Better than a winning day at the races....