Michael Tangy

ECE 381 – Spring 2015

Lab 8

Bit-Banging PS/2 Keyboard Interface

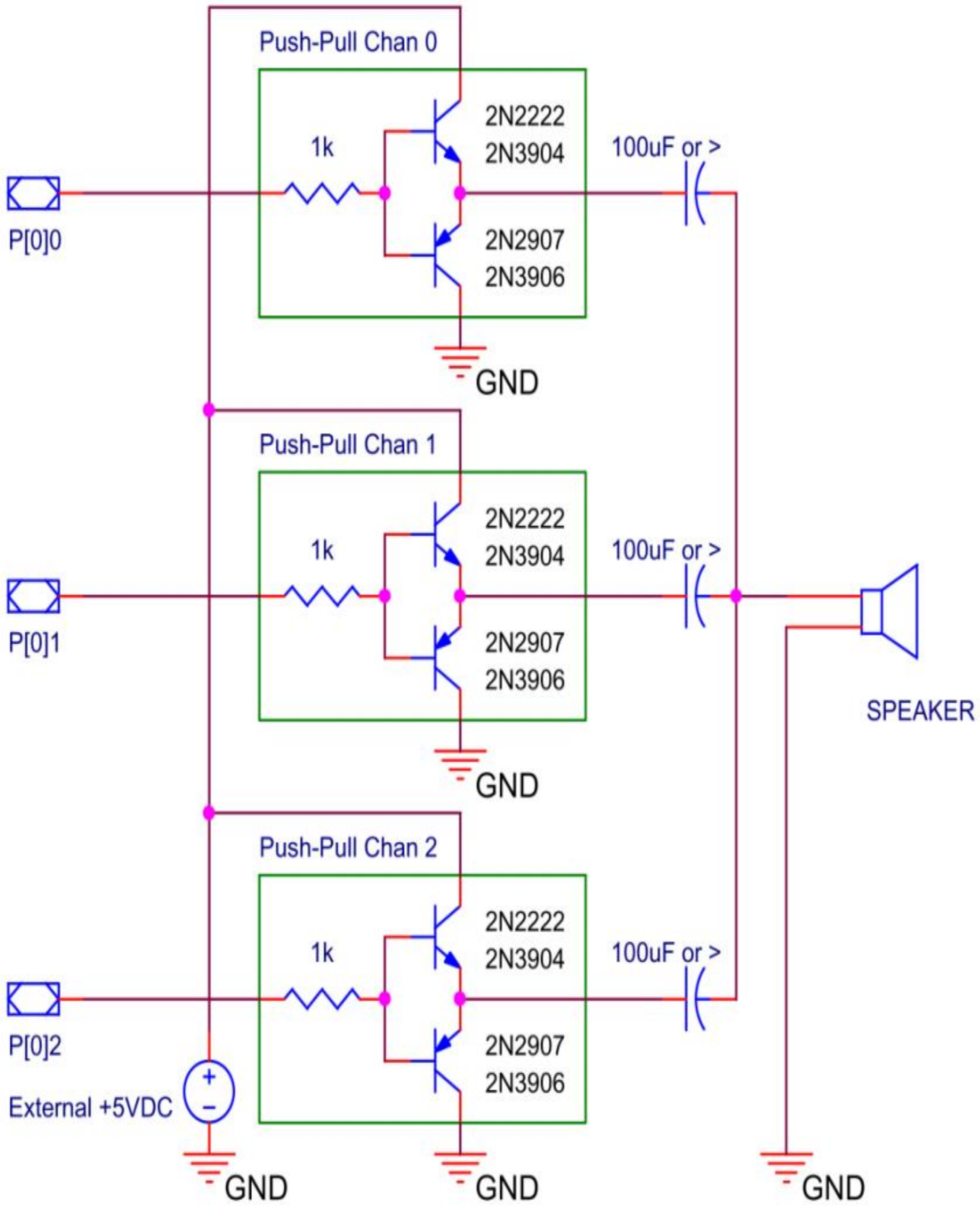
The purpose of this lab was for us to gain an understanding of what Bit banging is and how it can be implemented in microcontrollers to emulate hardware used to interact with external components you're trying to interface with your system. This lab also allowed us to gain an understanding of how PS/2 hardware works. In this lab we had to comb through the prewritten code that implements the bit banging for PS/2 keyboards and add snippets of code that add certain functionality.

Parts Needed:

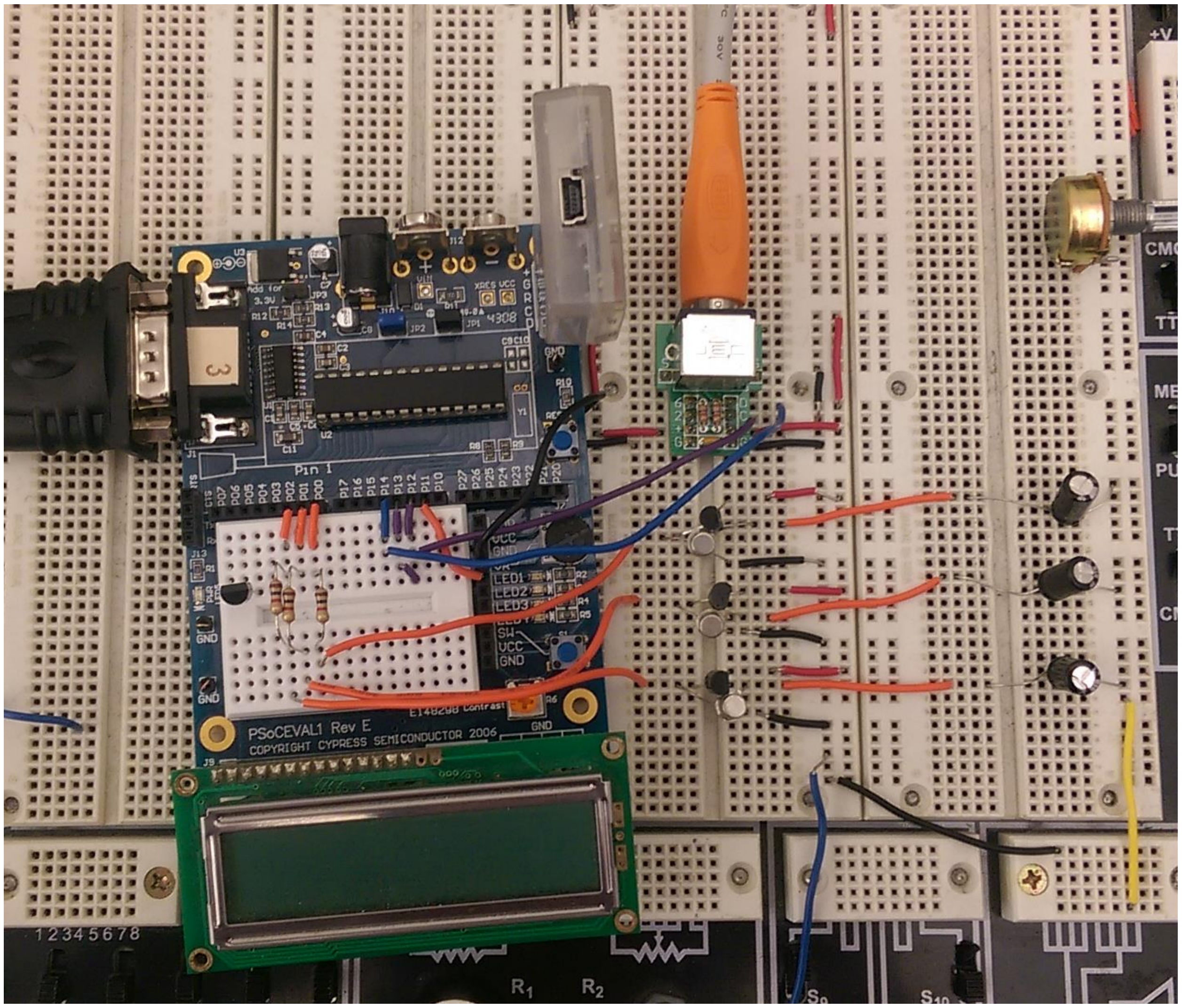
- PSoC 1 Microcontroller with 2x16 LCD
- DB-9 to DB-9 cable or DB-9 to USB serial cable
- PS/2 Keyboard
- 3- 1KΩ resistors
- 3- 2N2222 transistors
- 3- 2N2907 transistors
- 3- 100µF capacitors
- Speaker
- PS/2 6-pin mini DIN adapter

Hardware/Pin-Out Configuration:

After hooking your PSoC up with your computer you need to wire your speaker and keyboard. This involves making your three channel transistor-capacitor network refer to wiring diagram and picture below to verify your circuit is correct. Next wire your PS/2 6-pin mini DIN adapter so you can hook up your PS/2 keyboard, this involves tying the adapters "+" and "G" to power and ground and clock to port 1 pins 2 and 3 and data to port 1 pin 4. Next wire the LED by connecting port 1 pin 0 to LED1 on the PSoC evaluation board.



Push-Pull Chan 0
1k
2N2222
2N3904
2N2907
2N3906
100uF or >
P[0]0
GND
Push-Pull Chan 1
1k
2N2222
2N3904
2N2907
2N3906
100uF or >
P[0]1
SPEAKER
GND
Push-Pull Chan 2
1k
2N2222
2N3904
2N2907
2N3906
100uF or >
P[0]2
External +5VDC
+
–
GND
GND
GND



Next confirm your wiring is correct by downloading and opening the PS/2 bit banging project and downloading it to the PSoC. Next check and see the speaker works by pressing the pause buttun to put the keyboard in piano mode. From there press all the printable keys to see if they make sounds. Next you want to see if realterm is out putting the correct information, set its baud rate to 57600 and press pause again to put it back in keyboard mode then type printable keys and see if they apear on the realterm prompt.

Software Description:

The first requirement of this lab was to change the brightness levels of LED1 from 256 to 8. This requires changing the last two cases of the nested switch statement in the "`KeyboardAction`" function in the Keyboard.c file. The increment and decrement assignment needs to be changed from 1 to 32 since the LED's 254 brightness indexes and you want to make it have 8 levels.

The next requirement was adding a Rubout sequence to the backspace key to delete a character. This required adding an extra if-statement to the first if-else of the KeyboardToASCII function. The if-statement returns the rubout sequence if the backspace scan code gets entered.

The next requirement asks us to print out constant strings when alt is held down and a certain sequence of keys are pressed. This was accomplished by adding nested if's in the if (altdwn) statement in the KeyboardToASCII function. These nested if's raise flags if the given Key is pressed and then print out the string's if all the correct keys are pressed in the correct sequence.

The next requirement asks us to add two more pulse width modulator modules routed to pins P[0]1 and P[0]2. This allows us to use each modulator for each of the three rows of keys in order to harmonize the tones since each row produces its own signals and if you press two at the same time they combine before leaving the speaker.

The last requirement ask us to turn ctrl+alt+del into a restart trigger. This is done by adding a if(ctrldown&&deletedwn) statement in the if(altdwn) statement that calls the `M8C_Reset` subroutine. In order to do this you need to declare and define "deletedwn". You can define it by adding cases for it in the keyboard action function the same way altdwn is defined.

CODE:

-Added code to change the brightness levels

```
case 0x05:  // F1 - Dim the LED
if (brightindex > 1) {
if(brightindex>32)
{
brightindex-=32;
}
else {brightindex=1;}
PWM8LED_WritePulseWidth(brightindex);
}
else if (brightindex == 1){
brightindex--;
PWM8LED_Stop();
}
break;
case 0x06:  // F2 - Brighten the LED
if (brightindex == 0) {
brightindex+=32;
PWM8LED_WritePulseWidth(brightindex);
PWM8LED_Start();
}
else if (brightindex <= 223) {
brightindex+=32;
PWM8LED_WritePulseWidth(brightindex);
}
else
{brightindex=255;}

break;
```

-Added code to add rubout when backspace is pressed

```
if (scancode == 0x66) return(0x08); // Backspace returns rubout
```

-Added code to print constant strings when pressing ALT

```
///Print strings with ALT+123, ALT+314, ALT+000

if(AltDown){

if(scancode == 0x16 || scancode == 0x69 && !twoPressed && !onePressed){
onePressed = TRUE;

}else
if (scancode == 0x1E || scancode == 0x72 && onePressed && !twoPressed){
twoPressed = TRUE;
}else
if (scancode == 0x26 || scancode == 0x7A && twoPressed && onePressed){

UART_CPutString("\r\n");
UART_CPutString("\r\n");
UART_CPutString("The quick brown fox jumps over the lazy dog");
UART_CPutString("\r\n");
UART_CPutString("\r\n");
onePressed = FALSE;
twoPressed = FALSE;

}// end final if

if(scancode == 0x26 || scancode == 0x7A && !twoPressed && !onePressed ){
onePressed = TRUE;

}else
if (scancode == 0x16 || scancode == 0x69 &&  onePressed && !twoPressed){
twoPressed = TRUE;

}else
if (scancode == 0x25 || scancode == 0x6B && twoPressed && onePressed){

UART_CPutString("\r\n");
UART_CPutString("\r\n");
UART_CPutString("3.1415926535897932386");
UART_CPutString("\r\n");
UART_CPutString("\r\n");
onePressed = FALSE;
twoPressed = FALSE;

}// end final if

if(scancode == 0x45 || scancode == 0x70){   //&& !onePressed && !twoPressed){
zeroButtun++;

if(zeroButtun == 3 && scancode == 0x45){

UART_CPutString("\r\n");
UART_CPutString("\r\n");
UART_CPutString("I love ECE 381!");
UART_CPutString("\r\n");
UART_CPutString("\r\n");
zeroButtun = 0;

}

if(!AltDown){zeroButtun = 0;}

}// end final if
```

-Added code for CTRL+ALT+DELETE restart

```
if (CtrlDown && DeleteDown) {

M8C_Reset;
}
}//end if ALT-Down


case 0x71:  // E0 71 - DELETE                  ///added this
                        deletestatus |= 0x01;
                        DeleteDown = 1;
                        break;
                  //// Any scan codes not handled specially in the above case
statements are
                  //// handled here:
                  default:

case 0x71:  // E0 F0 71 - DELETE                     ///added
                        deletestatus &= ~0x01;
                        if (!deletestatus) DeleteDown = 0;
                        break;
case 0x71:  // DELETE               //added
                        deletestatus |= 0x02;
                        DeleteDown = 1;
                        break;
```

-Added code to harmonize tones

```
if (PianoMode) {
i = 0;
while((PianoScancode[i][0] != scancode) && (PianoScancode[i][0] != 0)) {
i++;
}
if (PianoScancode[i][0] > 0) {

if((PianoScancode[i][0]==0x1A)||(PianoScancode[i][0]==0x22)||(PianoScancode[i
][0]==0x21)||
(PianoScancode[i][0]==0x2A)||(PianoScancode[i][0]==0x32)||(PianoScancode[i][0
]==0x31)||
(PianoScancode[i][0]==0x3A)||(PianoScancode[i][0]==0x41))//If z row
{
PWMPiano1_WritePeriod(PianoScancode[i][1]);
PWMPiano1_WritePulseWidth(PianoScancode[i][2]);
PWMPiano1_Start();
}
else if
((PianoScancode[i][0]==0x1C)||(PianoScancode[i][0]==0x1B)||(PianoScancode[i][
0]==0x23)||(PianoScancode[i][0]==0x2B)||
(PianoScancode[i][0]==0x34)||(PianoScancode[i][0]==0x33)||(PianoScancode[i][0
]==0x3b)||(PianoScancode[i][0]==0x42))//If A row
{
PWMPiano2_WritePeriod(PianoScancode[i][1]);
PWMPiano2_WritePulseWidth(PianoScancode[i][2]);
PWMPiano2_Start();
}
else if
((PianoScancode[i][0]==0x15)||(PianoScancode[i][0]==0x1D)||(PianoScancode[i][
0]==0x24)||(PianoScancode[i][0]==0x2D)||
(PianoScancode[i][0]==0x2C)||(PianoScancode[i][0]==0x35)||(PianoScancode[i][0
]==0x3C)||(PianoScancode[i][0]==0x43))
//If Q row
{
PWMPiano3_WritePeriod(PianoScancode[i][1]);
PWMPiano3_WritePulseWidth(PianoScancode[i][2]);
PWMPiano3_Start();
}

}
}
```

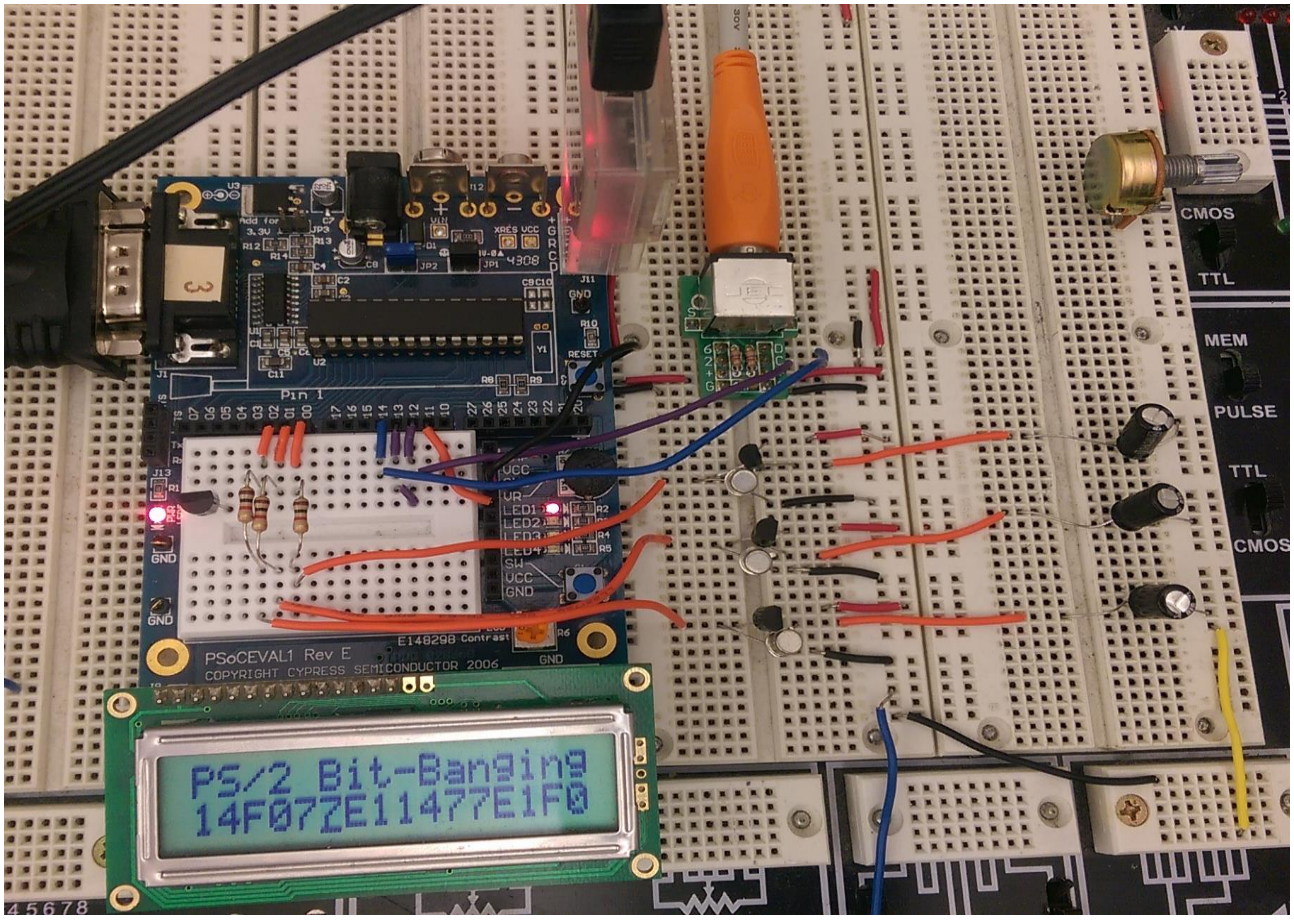
//to stop tone's

```
if (PianoMode && (!keyspressed)) {
PWMPiano1_Stop();
PWMPiano2_Stop();
PWMPiano3_Stop();
}
```

Testing:

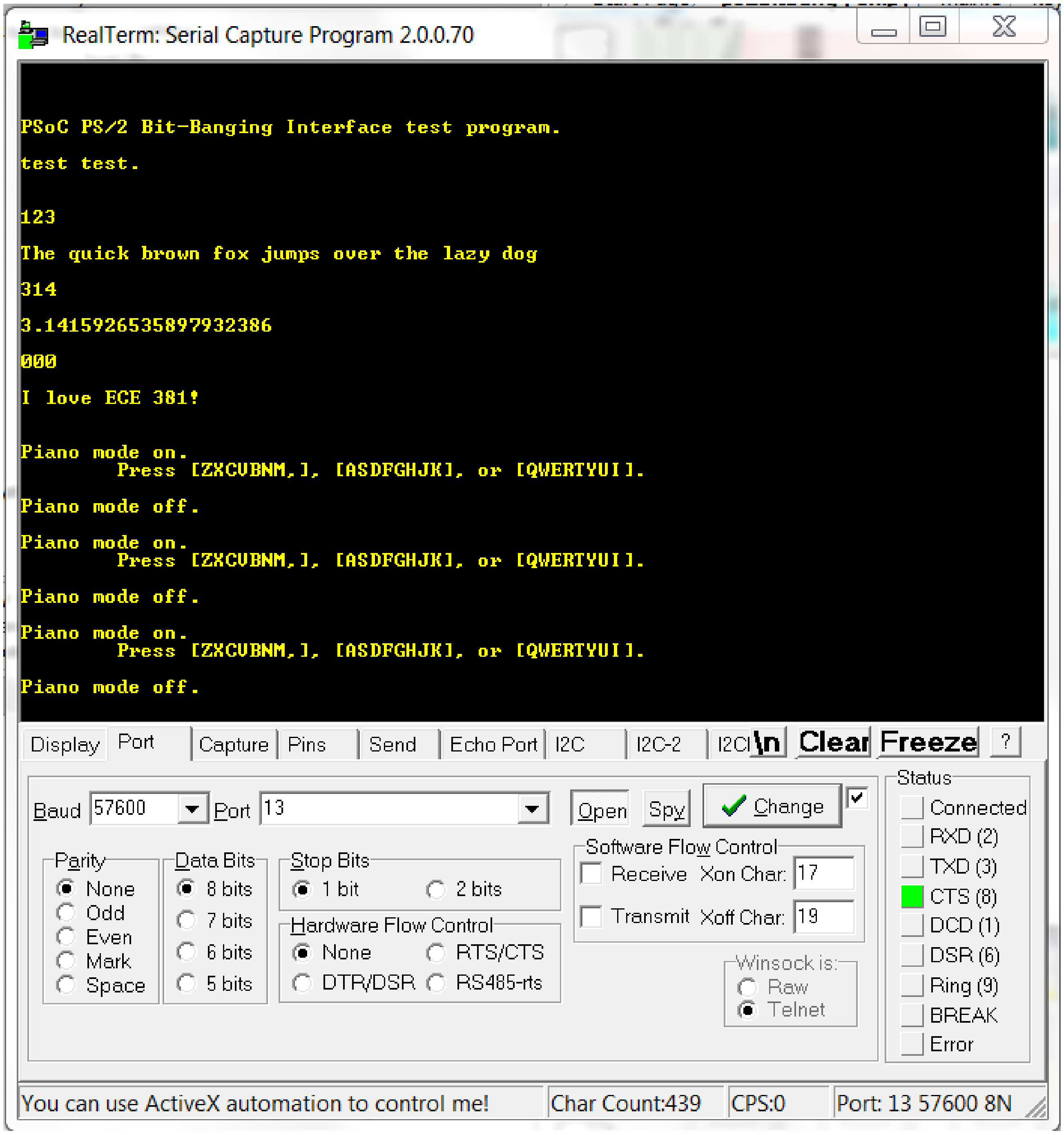
After the program is written is should be built and uploaded to the PSoC. Turn the PSoC on and go to the Real Term console and look for the initial message. Next test the LED by pressing the print screen button to turn LED1 on, then press F2 four times to bring the LED to its maximum brightness then press F1 eight times to turn it off. Next test the backspace button by pressing printable keys and then pressing backspace to delete them. Next test the ALT down constant strings by holding down the ALT key and pressing 123, the string should then print to real term saying `"The quick brown fox jumps over the lazy dog"` next hold down ALT and press 314 then `"3.141592653589793238"` should print on real term. Next hold down ALT and press 000 and `"I love ECE 381"` should print on real term. After that press CTRL+ALT+DELETE and see if the PSoC restarts. The last feature to check is the harmonized tones, to do this press the pause button to enable piano mode and press and hold a key in the top row like "T" then press hold a key in the middle row like "J" and see if those combine tones make a different tone then "J" by its self, if it does then the rows are properly harmonized.

Initial LCD message when PSoC turns on:



```
PS/2 Bit-Banging
14F07ZE11477E1F0
```

Real term prompt after testing:



RealTerm: Serial Capture Program 2.0.0.70

```
PSoC PS/2 Bit-Banging Interface test program.
test test.

123
The quick brown fox jumps over the lazy dog
314
3.14159265358979323846
000
I love ECE 381!

Piano mode on.
        Press [ZXCVBNM,], [ASDFGHJK], or [QWERTYUI].
Piano mode off.
Piano mode on.
        Press [ZXCVBNM,], [ASDFGHJK], or [QWERTYUI].
Piano mode off.
Piano mode on.
        Press [ZXCVBNM,], [ASDFGHJK], or [QWERTYUI].
Piano mode off.
```

Baud 57600 Port 13

Parity: [✓ None] Odd Even Mark Space

Data Bits: [✓ 8 bits] 7 bits 6 bits 5 bits

Stop Bits: [✓ 1 bit] 2 bits

Hardware Flow Control: [✓ None] RTS/CTS DTR/DSR RS485-rts

Software Flow Control: Receive Xon Char: 17 Transmit Xoff Char: 19

Winsock is: Raw [✓ Telnet]

Status: Connected, RXD (2), TXD (3), 🟩 CTS (8), DCD (1), DSR (6), Ring (9), BREAK, Error

You can use ActiveX automation to control me! | Char Count:439 | CPS:0 | Port: 13 57600 8N