Lab 4 Report
Dr. Ryan Gerdes
ECE 3710
By: Jonathan Tousley
Partner: Aaron Kunz

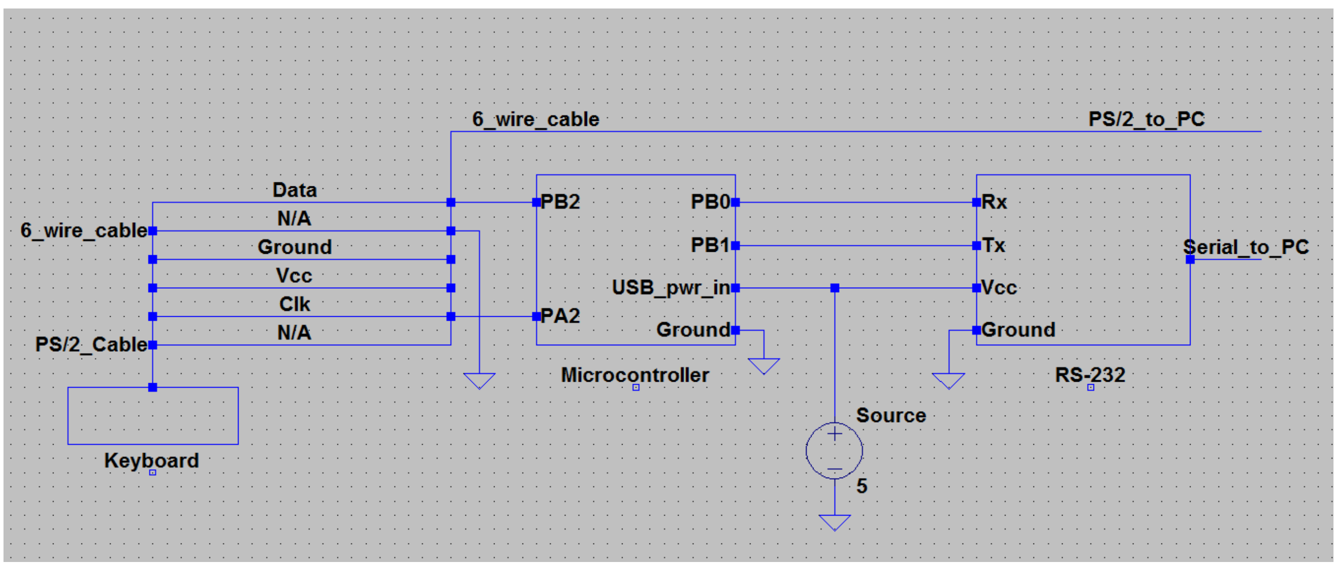
Overview:

The microcontroller was configured as a PS/2 keylogger. Upon enable from a button, the microcontroller would record keystrokes sent between the keyboard and the PC. Upon disable, the microcontroller would send the recorded data to the PC using UART via RS-232. For operation, this project requires:

- A 5V voltage source
- A UART to RS-232 module
- A serial cable
- A PS/2 keyboard

Hardware details:

Using UART1 and the RS-232 module, the microcontroller was connected to the PC. To connect the keyboard to the computer, an extension cable was cut in half and the spliced wires were inserted into the breadboard. Then, the clock and data wires were connected to the microcontroller as well as their respective pins on the other half of the extension cable. The clock was connected to PA2 and the data to PB2. See schematic below:



Software details:

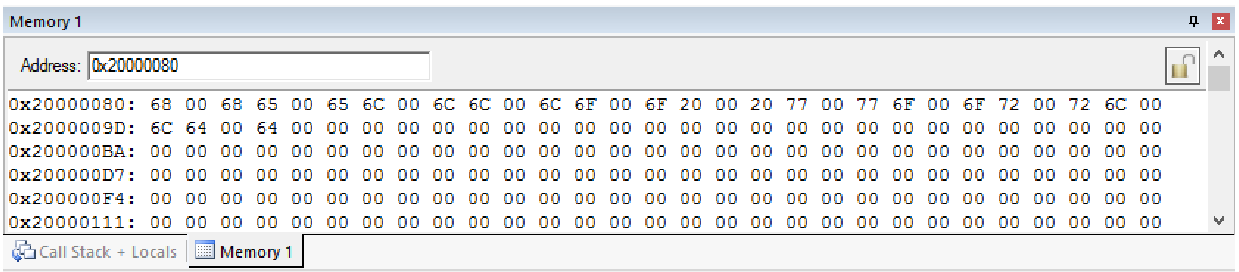
The microcontroller performed no computations (but should have been computing the value of pi) while it waited for the keyboard to send data to the PC. On a falling edge from PF0 (a button press), the microcontroller alters state. When enabled, the microcontroller enables interrupts for the keyboard clock. Upon a falling edge, the microcontroller is interrupted. Using knowledge of the consistent performance of a PS/2 keyboard, the data is then preserved from one interrupt to another in an intelligent way. Upon the last interrupt, the key code is saved into an array in memory. When the microcontroller is disabled via the button, the saved data is sent to the PC.

UART:

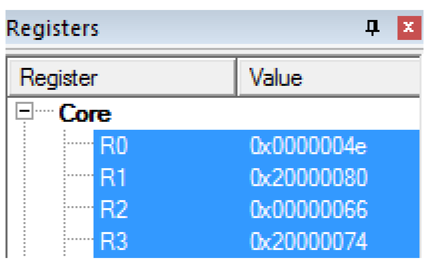
Using essentially the same design as lab 3 for UART, the only modifications necessary were to translate the code from assembly into C. This was straightforward enough, except when setting the BRDI, because a type cast was necessary to write all four bytes. Also, the clock was not set to 16MHz by default, so a manual configuration was necessary for the clock to get the baud rate correct. It took us a long time to figure that out.

Requirements:

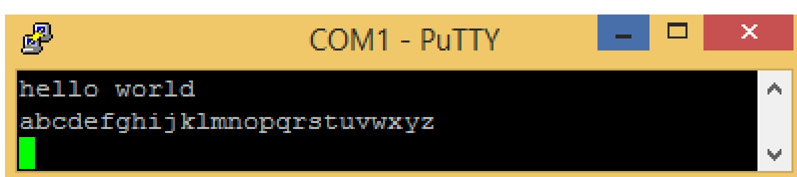
1. This project required the ISR to be triggered for each clock tick, which is verified by the accuracy of the recorded data, as well as the code itself.
2. The keystrokes must be stored in memory. Here are two screenshots:



Register 2 holds the ASCII value 'f':

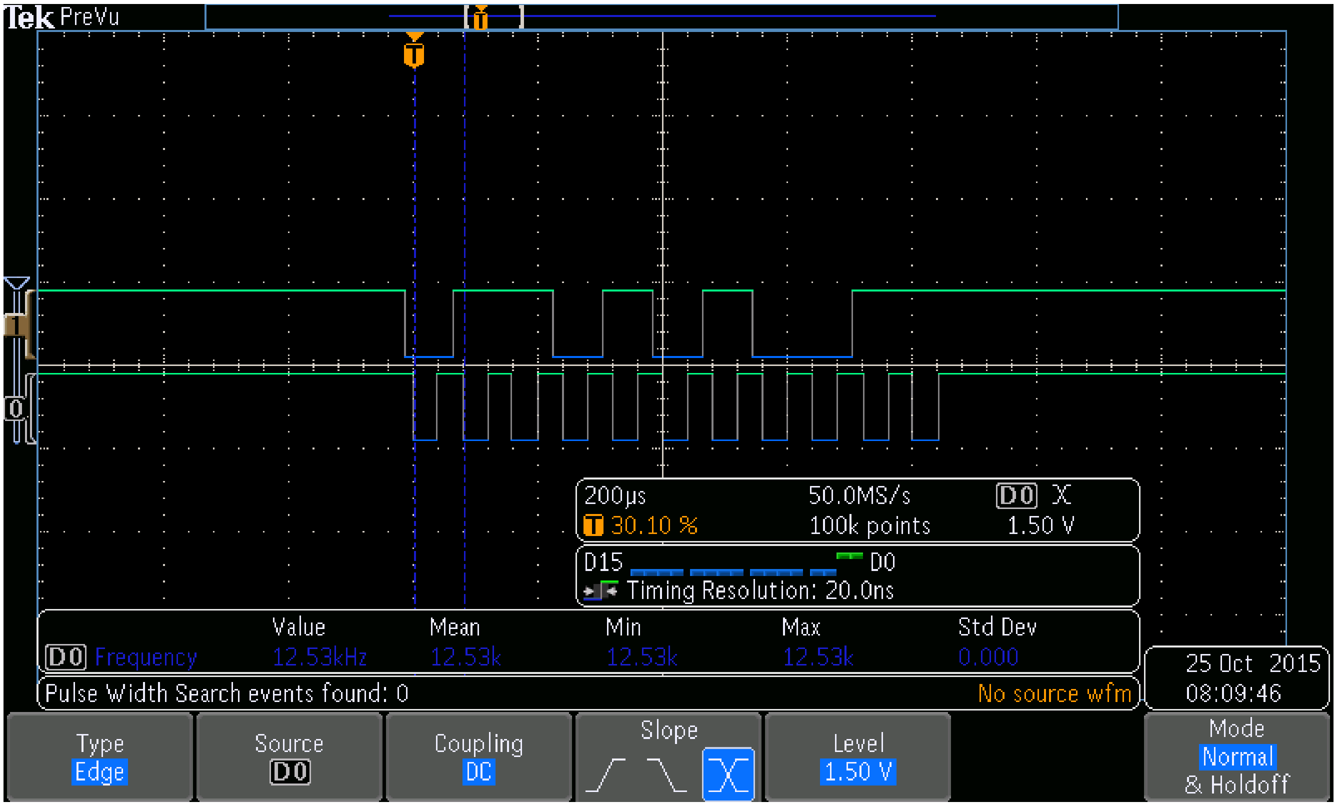


3. A button must start/stop the keylogger. This is done using SW2.
4. Interfacing with the button must be handled via interrupts. The priority of the button must be higher than the priority of the clock. The priority of the button is set to be zero, and the priority of the clock is set as 1. Also, the code shows the button being handled via interrupts.
5. The program only needs to capture lowercase alphanumeric characters and space. Here is a screenshot:

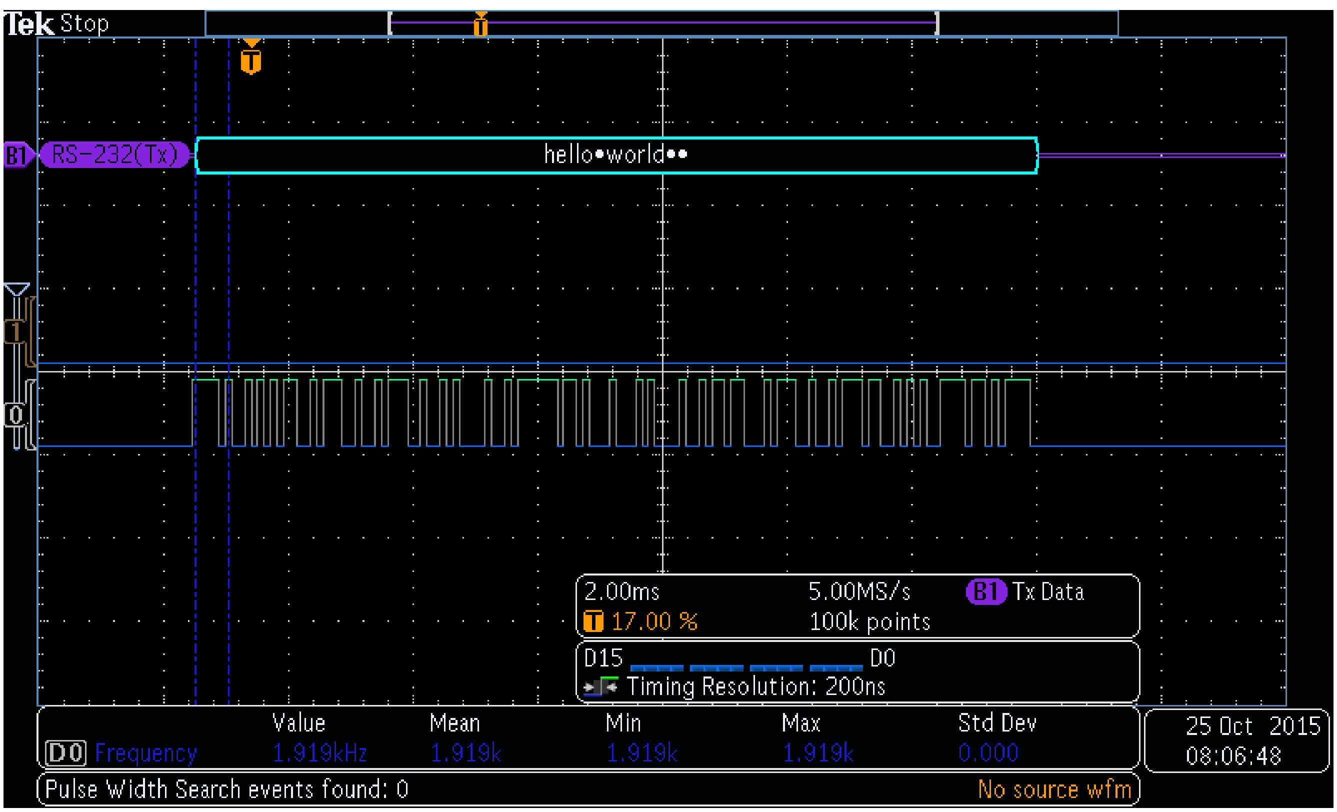


Some additional things:

1. The Tiva C pins which are not 5V tolerant are: PD[4:5], PB[0:1].
2. Here is a screenshot of data being sent from the keyboard to the PC:



3. Here is a screenshot of a string being sent to the PC:



The rest of this document contains only code, here it is:

```
#include "stdbool.h"

// This is a look up table for basic key strokes.
// This translates from ps/2 to ascii.
// The table starts at 0x15, so you should shift
// the table by 0x15 when you reference it.
//
unsigned char ps2_to_ascii[] = {
	'q','1',0x00,0x00,0x00,'z','s','a','w','2',0x00,

	0x00,'c','x','d','e','4','3',0x00,0x00,' ','v','f','t','r','5',0x00,0x00,

	'n','b','h','g','y','6',0x00,0x00,0x00,'m','j','u','7','8',0x00,

	0x00,',','k','i','o','0','9',0x00,0x00,'.',0x00,'l',0x00, 'p'};

unsigned char *SYSCTL = (unsigned char *) 0x400FE000;

unsigned char *PA = (unsigned char *) 0x40004000;
unsigned char *PB = (unsigned char *) 0x40005000;
unsigned char *PC = (unsigned char *) 0x40006000;
unsigned char *PD = (unsigned char *) 0x40007000;
unsigned char *PE = (unsigned char *) 0x40024000;
unsigned char *PF = (unsigned char *) 0x40025000;

unsigned char *UART1 = (unsigned char *) 0x4000D000;

unsigned char *SysTick = (unsigned char *) 0xE000E000;
unsigned char *TM0 = (unsigned char *) 0x40030000;
unsigned char *TM1 = (unsigned char *) 0x40031000;
unsigned char *TM2 = (unsigned char *) 0x40032000;

unsigned char *NVIC = (unsigned char *) 0xE000E000;

unsigned volatile char CNT = 0;
unsigned volatile char Data[1500];
unsigned volatile long int OFFSET = 0x0;
					//number of key codes recorded so far
unsigned volatile char BYTE = 0x0;
volatile bool enable = false;

void enableClock(){
	*(int*)&SYSCTL[0x060] = 0x8E3D40;		//sysclk
	SYSCTL[0x618] = 0x2;			//enable UART1
	SYSCTL[0x608] = 0x23;			//enable PA PB PF

	__asm__ {nop};
```

```
        __asm__ {nop};
        __asm__ {nop};
}

void enablePortA(void)
{
        //enable PA2
        PA[0x420] = 0x00;                       //AFSEL
        PA[0x51C] = 0x04;                       //DEN
        PA[0x400] = 0x00;                       //DIR
        PA[0x404] = 0x00;                       //IS 0 = edge
        PA[0x408] = 0x00;                       //IBE 0 = rising or falling edge
        PA[0x40C] = 0x00;                       //IEV 0 = falling edge
        PA[0x410] = 0x04;                       //IM 1 = enable PA2

        NVIC[0x400] = 0x20;                     //Interrupt priority port A = 1
}

void enablePortB()
{
        PB[0x420] |= 0xFF;                      //AFSEL
        PB[0x51c] |= 0xFF;                      //DEN
        PB[0x514] |= 0xFF;                      //PDR
        PB[0x52C] = 0x11;                               //PCTL
}

void enablePortF()
{
        //GPIO unlock
        *(int*)&PF[0x520] = 0x4C4F434B;

        PF[0x524] = 0x01;                       //Commit register

        PF[0x400] = 0x0E;                       //DIR
        PF[0x510] = 0x01;                       //PUR
        PF[0x51C] = 0x1F;                       //DEN

        PF[0x404] = 0x00;                       //IS 0 = edge
        PF[0x408] = 0x00;                       //IBE 0 = rising or falling edge
        PF[0x40C] = 0x00;                       //IEV 0 = falling edge
        PF[0x410] = 0x01;                       //IM 1 = enable PF0
        PF[0x38] = 0x02;                        //LED Red

        *(int*)&NVIC[0x100] = 0x40000001;               //GPIOF and GPIOA interrupts
}

void enableUART1()
{
        UART1[0x30] = 0x00;                             //CTL
```

```
	*(int*)&UART1[0x24] = 0x0068;	//BRDI
	UART1[0x28] = 0xB;							//BRDF
	UART1[0x2C] = 0x70;								//Serial params
	*(int*)&UART1[0x30] = 0x101;	//CTL - sets Tx UART
	__asm__ {nop};
	__asm__ {nop};
}

//as per startup.s
void GPIOA_Handler(void)
{
	PA[0x41C] |= 0x04;		//acknowledge interrupt
	if(CNT > 0 && CNT < 9){
			//ignore start bit
			BYTE |= (((PB[0x3FC] & 0x4) >> 2) << (CNT - 1));
	}
	CNT++;

	if(CNT == 11){
		//finished
		CNT = 0;
		Data[OFFSET++] = ps2_to_ascii[BYTE - 0x15];
		BYTE = 0;
	}

	return;
}

void writeDataUart(void)
{
	unsigned int i;
	for(i = 0; i < OFFSET; i++){
		if((UART1[0x18] & 0x20) == 0x0){					//UART TXFF- If
UART fifo has space
			if(Data[i] == 0x00){
				i++;
				continue;
			}
			UART1[0x0] = Data[i];
		}
		else{
			i--;
		}
	}

	//yeah, I used gotos. Assembly doesn't care.
	//this code just makes sure
	//a carriage return ("0D")
	//and a new line ("0A")
```

```
	//are sent after every string
	sendCarriage:
	if((UART1[0x18] & 0x20) == 0x0){				//UART TXFF-  If UART fifo
has space
		UART1[0x0] = 0x0D;
		//carriage return
	}
	else{
		goto sendCarriage;
	}

	sendNewLine:
	if((UART1[0x18] & 0x20) == 0x0){				//UART TXFF-  If UART fifo
has space
		UART1[0x0] = 0x0A;
		//carriage return
	}
	else{
		goto sendNewLine;
	}

}

void GPIOF_Handler(void)
{
	PF[0x41C] |= 0x01;
	//acknowledge (clear) interrupt
	if(!enable){
		//enable
		PA[0x41C] |= 0x04;
	//acknowledge interrupt PA
		NVIC[0x280] = 0x01;
	//UNPEND disable pending register
		NVIC[0x100] = 0x01;
	//set enable interrupt register
		PF[0x38] = 0x08;
	//change light to green
		enable = true;
	}
	else{
		PF[0x38] = 0x02;
	//LED Red
		NVIC[0x180] = 0x01;
	//DISn interrupt disable register
		writeDataUart();
		enable = false;
	}

	return;
```

```
}

int main(void)
{
	enableClock();
	enableUART1();
	enablePortF();
	enablePortA();
	enablePortB();

	while(1);

}
```