Short Report on activities undertaken in EC0504 Lab Or How i had fun and learning in Microcontroller lab, an autodidactic adventure

Rohit Shetty
4NI13EC042
Department of Electronics and communication
National Institute of Engineering
Mysuru

Acknowledgement

I would like to thank Shri. S. PARAMESHWARA Asst. prof ,Dept of EnC NIE Mysuru. For his support and guidance throughout the course and for providing access to labs and equipments for my experiments.He has been very helpful and supportive.

I would like to acknowledge the support of Department of EnC, NIE Mysuru for opening the access to equipments and labs.

Also i would acknowledge the efforts of Opensource community and bloggers who have shared their knowledge with the world ,from which i have learnt a lot.

Content

Preface

This lab has been in true sense a lab,as i was able to kickstart my journey into new exciting fields. Following document discusses the work i have tried with things interesting me. There were lot more ideas i would have loved to work on but lack of time didn't allow it. Also there are many things i have not completed due to the same reason, however they have been included in true sense. There many things that touched just basics. This course i have tried my hands on MSP430EZ430 low power microcontroller,Atmega16 ,Arduino , 8051, computer computer communication using serial.

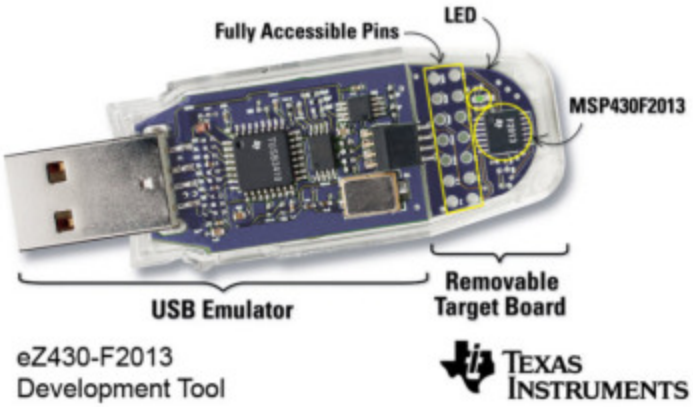
Linux has been the platform of choice for the reason its opensource, a platform built for learning and sharing. Also due to large community of hackers and developers available. All the code has been tried on GNU/Linux Ubuntu 12.04 on Lenovo G580.

MSP430-EZ430 F2013

The **MSP430** is a mixed-signal microcontroller family from Texas Instruments. Built around a 16-bit CPU, the MSP430 is designed for low cost and, specifically, low power consumption embedded applications.

The MSP430 can be used for low powered embedded devices. The current drawn in idle mode can be less than 1 μA. The top CPU speed is 25 MHz. It can be throttled back for lower power consumption. The MSP430 also uses six different low-power modes, which can disable unneeded clocks and CPU. Additionally, the MSP430 is capable of wake-up times below 1 microsecond, allowing the microcontroller to stay in sleep mode longer, minimizing its average current consumption. The device comes in a variety of configurations featuring the usual peripherals: internal oscillator, timer including PWM, watchdog, USART, SPI, I²C, 10/12/14/16/24-bit ADCs, and brownout reset circuitry. Some less usual peripheral options include comparators (that can be used with the timers to do simple ADC), on-chip op-amps for signal conditioning, 12-bit DAC, LCD driver, hardware multiplier,USB, and DMA for ADC results. Apart from some older EPROM (MSP430E3xx) and high volume mask ROM (MSP430Cxxx) versions, all of the devices are in-system programmablevia JTAG (full four-wire or Spy-Bi-Wire) or a built in bootstrap loader (BSL) using UART such as RS232, or USB on devices with USB support.

There are, however, limitations that preclude its use in more complex embedded systems. The MSP430 does not have an external memory bus, so it is limited to on-chip memory (up to 512 KB flash memory and 66 KB RAM) which may be too small for applications that require large buffers or data tables. Also, although it has a DMA controller, it is very difficult to use it to move data off the chip due to a lack of a DMA output strobe.[1]



The particular model i have been using is MSP430F2013. This particular model is complete development tool including hardware and software emulator.This comes in the form factor of USB stick,This being a low powered device is being able to run on usb port power,is only 3.6 V tolerable. This is suitable for low powered embedded devices and wearables. My venture into this device has been very basic, i am still trying to learn its advanced architecture.There seems to be mildly good community around it. Here is the link to data sheet http://goo.gl/5Eoiqf .This comes with a CD packages packed with datasheets, and development software suite like IAR Embedded Workbench and Code Composer Studio for windows. But for the linux user you need to scout for more information.Next part deals with setting up the development environment for Linux.

The basic work cycle for Developing a MSP microcontrollers program in linux would be as same as other microcontrollers

1) Write code
2) Compile
3) Debug out level 1, syntactical errors
4) re compile until no errors.
5) convert the executable to machine executable format, here .elf
6) burn the code into MSP using uploader/debugger
7) debug the code using msp debugger
8) debug the level two errors, logical error

Before starting the development you will need a compiler to compile the C code to Machine executable code. A debugger that helps in debugging by putting appropriate break points that can be controlled from laptop and registers can be reviewed.This uses Spy Biwire debugging interface,instead of JTAG.

For compiling we will use MSPGCC a port of GCC c compiler for MSP architectures , MSPGCC being opensource is free for commercial and noncommercial use and is quite stable.

Debugger that we will use is MspDebug. As from its manual quotes

“MSPDebug is a command-line tool designed for debugging and programming the MSP430 family of MCUs. It supports the eZ430-F2013, eZ430-RF2500,Launchpad, Chronos, FET430UIF, GoodFET, Olimex MSP430-JTAG-TINY and MSP430-JTAG-ISO programming tools, as well as a simulation mode.“

Following are the steps to install MSPGcc and MSPdebug for linux based systems.(debian to be specific)

Install following packages

- binutils-msp430
- gcc-msp430
- msp430-libc
- mspdebug

To install them use this bash command for debian/ubuntu users

$$sudo\ apt-get\ install\ binutils-msp430\ gcc-msp430\ msp430-libc\ mspdebug$$

This requires root access. This installs the required packages.

Following are the example commands required to run the compiler

$$msp430-gcc\ -Os\ -Wall\ -g\ -mmcu=msp430x2013\ -c\ filename.c$$

Here msp430-gcc is the compiler name we are invoking, with flags , “mmcu =” referring to the type of microcontroller to be compiled for, and filename.c being the file to be compiled. This produces a compiled executable filename.o.
To convert this to Machine understandable elf format we run this

$$msp430-gcc\ -g\ -mmcu=msp430x2013\ -o\ filename.elf\ filename.o$$

This produces filename.elf which is then to be loaded to msp430.
This is done using mspdebug , there are two ways to go about

1) Direct burn the code

$$sudo\ mspdebug\ uif\ 'erase'\ 'load\ filename.elf'\ 'exit'$$

This burns the code to msp430 MCU.

2) Debug mode

here you get into mspdebug mode
d

Here this launches a command terminal. where you can execute certain commands like erase, load , run , break etc
erase- This erases the whole Flash memort
load filename.elf - this loads the filename.elf into memory
run - this runs the loaded code

step- runs one lineof instruction at a time and dumps the memory hex
Stop,break - breaks the execution and displays the memory dump.
I got these details from

1) http://www.mycontraption.com/programming-the-msp430-launchpad-on-ubuntu/
2) http://mspgcc.sourceforge.net/manual/c19.html
3) http://mspdebug.sourceforge.net/faq.html

Now to make the job easier
I combined first two lines of commands to compile and convert C code to machine executable into one compile.sh bash file
and uploading is put in another bash file, upload.sh.
and give executable permission
So now all you have to do is write code and do
./compile filename
./upload filename
[Code is put up in
https://github.com/rohitshetty/EC0504LabsReport/tree/master/msp/CompileUpload]

Hello World in MSP430EZ430 F2013

*A*ny programming venture starts with hello,world. Not because it is the tradition but because that helps one to know the development cycle and ensure everything is set right.
If you have successfully set up the working environment then this step tells you how exactly to go about.
Open Any text editor of your choice(say geany, gedit,Nano,pico,VI emac etc).
Save in the code , i will get back on dissecting in short

```
#include<msp430f2013.h>
/*
This header file to be included for all msp430f2013 versions
*/
volatile int i;
void configtime(void){
/*
This Function sets the clocksource, 8MHz for precision requiring jobs and 3 to 12Khz for sleep
mode and other non precise mode
refer datasheet
*/

BCSCTL1 = CALBC1_8MHZ; //Configure Basic clock source control 1 to calibrated Constant1 8 MHz;
DCOCTL = CALDCO_8MHZ; // Configure Digital clock oscillator to 8 MHz
BCSCTL3 = LFXT1S_2;  //  Set Basic clock source control 3 to 12 Khz
}

void main(){
```

```
WDTCTL = WDTPW+WDTHOLD;         // MSP430 has internal watch dog timer inbuilt. You have to
disable it, or send it alive pulse in short duration
configtime();
P1DIR=0xff;                     // This sets the P1 Port to Output
P1OUT=0xff;                    // Output is set to one
while(1){
P1OUT=~P1OUT;                   //Blink
for(i=0;i<32700;i++);          //Delay
}
}
```

Now save this as hello.c and execute it by
./compile hello
./upload hello
this should have your code burnt and led blinking.
So what does this code do?
Code is well commented. Lets just have a glance.
MSP430 microcontrollers have watchdog timers on by default.Hence They need to be disabled for prototyping else they will restart the code everytime they dont get a heart beat pulse, this is done by `WDTCTL = WDTPW+WDTHOLD`
WDTCTL the register responsible for watchDogtimer requires password and that is passed as upper byte with WDTPW and WDTHOLD puts watchdog on hold.
Next we need to specify the clock, although this is not necessary but is helpful.
Here we select 8MHz clock for fast precise needs and 3 Khz for slow not so precise needs like waking up from sleep. One of the major advantage of MSP430 is that it has ranges of power sleep mode,to save the power and battery life. Also there are various clock source that can be used and switched dynamically to save battery. example switching to high precision clock only when communication demands and staying at low clock speed most of the time etc.
Next is

```
P1DIR=0xff;
P1OUT=0xff;
```

Here P1DIR sets the port 1 on output mode, and P1OUT puts it in high state.
Remaining should be self explanatory.

Usage of mspdebug
open mspdebug by issuing the following command
sudo mspdebug uif
This will put you in debugger mode
you can issue
erase
to erase the flash
load filename.elf
to load the hex
run
to run

Code is put up here
https://github.com/rohitshetty/EC0504LabsReport/tree/master/msp/HelloWorld

This PDF has more details on basics
http://www.essp.utdallas.edu/uploads/CE4370/Fall10/Tutorialv0_2.pdf
One can always refer this.

reference

[1] - wikipedia http://en.wikipedia.org/wiki/TI_MSP430
http://www.mycontraption.com/programming-the-msp430-launchpad-on-ubuntu/
http://mspgcc.sourceforge.net/manual/c19.html
http://mspdebug.sourceforge.net/faq.html
http://www.essp.utdallas.edu/uploads/CE4370/Fall10/Tutorialv0_2.pdf

playing with ATmega16

Atmega16 is an entry level microcontroller with quite interesting features like 16bit,8bit timers, 8 channel 10bit ADCs, interrupts , inbuilt TWI,SPI,UART hardwares.
To start developing in ATmega you should have basic knowledge of C and register manipulation.
Like with any other MCU, Atmega needs a cross compiler and an utility to upload/maniplate the code.
Here we use AvrGCC, a port of GNU cross compiller for AVR architecture and avrdude and utility to upload the code. You need to install this, which is very simple.
Then follow the steps at
https://github.com/rohitshetty/bashAtmega
Which explains how to setup the working environment in linux.

Lets start with a basic helloworld code.

```
#include<avr/io.h>
#include<util/delay.h>

void main(){
DDRB=0xff;        h  //Turn the port into output mode
PORTB=0xff;          //Turn the port high
while(1){
PORTB=~PORTB;        //Toggle
_delay_ms(500);      //inbuilt delay routine in util/delay.
}
}
```

Lets disect this code.
avr/io.h is required as it has all the port details and various routines required. util/delay.h contains inbuilt delay functions which is useful until one learns about timers.
DDRx where x can be A,B,C,D is useful to set whether the PORTx acts as input or output port.
PORTx=value sets the port to high or low ,if DDRx is set high and enables pull up resistors if PORTx is high and DDRx is low.
_delay_ms(time); this inserts a delay in ms range.

Code can be found at
https://github.com/rohitshetty/EC0504LabsReport/tree/master/atmega/helloWorld

Timer, pooling method

Atmega16 has 2 8bit timer and one 16bit timer. They are very versatile as they can be used for Clock generation, Timers, Counters , PWM mode etc.Here i have just scratched the surface to just get the taste of it.
Following is the code and its dissection

```
#define F_CPU 12000000UL
#include<avr/io.h>
#include<util/delay.h>

void main(){
int i;
DDRA = 0xff;
PORTA = 0xff;
while(1){


PORTA =  ~PORTA ;

for(i=0;i<251;i++){      //251*1ms
TCCR1B |= (1<<CS10);    //This sets prescaler to 1
while(TCNT1<12001);     //wait till timer reaches 12001 1ms
TCNT1=0;                //Reset
}
}
}
```

Here F_CPU is an constant variable thats sets the clock variable to 12000000 hz this is necessary for well working of software with hardware.
TCCR1B according to datasheet is the lower byte configuration for timers, CS10 bit enables the timer in no prescaling mode.
TCNT1 is the counter for counter in mode one.
The formula to calculate the up value is

$$count = (required\ delay / clock\ period) - 1$$

This is an excellant resource to learn about atmega microcontrollers
http://maxembedded.com/2011/06/avr-timers-timer0/

16x2 LCD interfaced with Atmega16A

LCD is a cheap way to provide display functionality to your microcontrollers.Also the interfacing is quite easy.

LCD can be interfaced in two ways.

1) 8bit mode
2) 4bit mode

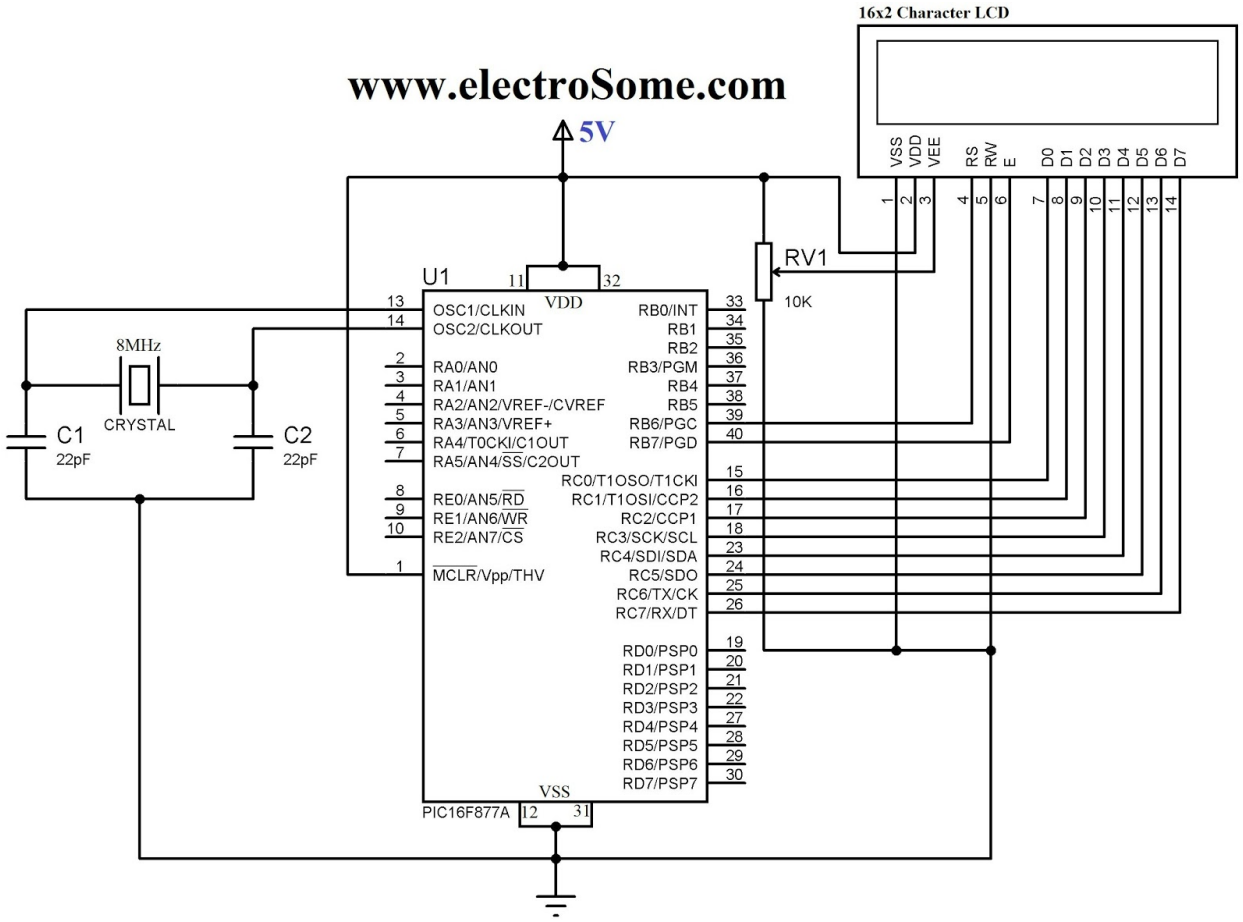
in 4Bit mode only 4 data bus and 3 command bus are needed but the programming would be a bit complicated

in 8Bit mode it has 8data bus and 3 command bus.

more details

http://www.engineersgarage.com/embedded/avr-microcontroller-projects/interface-lcd-4bit-mode-circuit

Interfacing



taken from electrosome.com

In 8bit mode we connect one port to data bus and 3 pins to control bus, as shown.
The algorithm to send the code would be

1) initialize the LCD by sending the sequence of bytes in command mode
2) send the data in data mode
3) repeat.

LCD has 3 control pin RS,R/w- and EN.
R/w- tells the LCD if the command/data is for reading or writing. most of the time its just writing.
RS tells whether the data is command or display data.
Command is RS=0 and data would be RS=1
And En pin is enable pin is used to provide pulse for latching data.
After putting data and latching LCD expects a delay of around 250 microseconds.
Code will not be put up here for space constraints but the code can be refered from
https://github.com/rohitshetty/EC0504LabsReport/tree/master/atmega/lcd

Here lcd.c is the library containing four subroutines. lcdinit() initializes the LCD by sending required commands , more about them can be found at
http://www.8051projects.net/lcd-interfacing/commands.php
lcdcmd() this function enables command mode and sends command
lcdData() enables data mode,sends display data
lcdMsg() sends the string data , one character at a time to LCD

To use the library you would have to include the library and then add the init function and just send the code on the go.

Arduino

“**Arduino** is an open-source computer hardware and software company, project and user community that designs and manufactures kits for building digital devices and interactive objects that can sense and control the physical world.[1] Arduino boards may be purchased preassembled, or as do-it-yourself kits; at the same time, the hardware design information is available for those who would like to assemble an Arduino from scratch.” This is how wikipedia describes arduino.
Arduino is built over atmega328 microcontroller. it has 13Digital IO pin, 6 analog pins.
A USART port, SPI port etc.
Setting up arduino IDE on any platform is very easy hence i am not including it.

Interfacing of HC05 bluetooth module

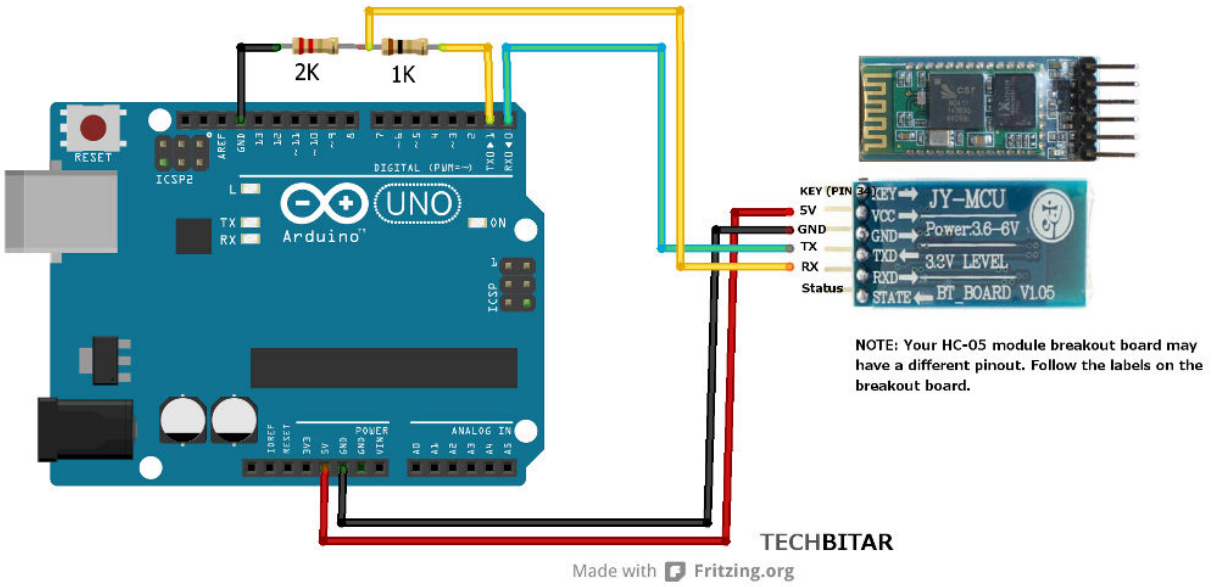


image credits: instructable.com

HC05 is a lowcost serial to bluetooth module using which one can send data serially into the module which it sends through bluetooth to connected devices.
This is by default configured in slave mode. Bluetooth technology/ communication demands on slave and one master least. Master is responsible for initiating connections. it is quite easy to make connection between arduino and phone over bluetooth, but difficult to pair two hc05 (refer https://alselectro.wordpress.com/2014/10/21/bluetooth-hc05-how-to-pair-two-modules/)
Here we go with basics.
Connecting Bluetooth module with arduino is nothing more than just setting up the serial communication line.

We connected the RX of arduino to HC05 TX and vice versa.
This is the simple code to send data.

```
void setup(){

Serial.begin(9600);

}


void loop(){

Serial.println("Hello World");

}
```

Here Serial.begin(baudrate) sets the baudrate for communication. and Serial.println(“”) prints the data inside the parenthesis.
Once this is done, we need to set our mobile ready.
Install BlueTerm app which is open source.
And pair both of them. and connect.
Voila!

This bluetooth module can be used with atmega or msp in serial mode without much ado.

Arduino interfaced with PS2 mouse

PS2 protocol have 2 data/clock pins. One defines the clock and other is data pin.
Data is read whenever clock is ground. each data frame has a start bit, 8 data bits, one parity and one stop bit.

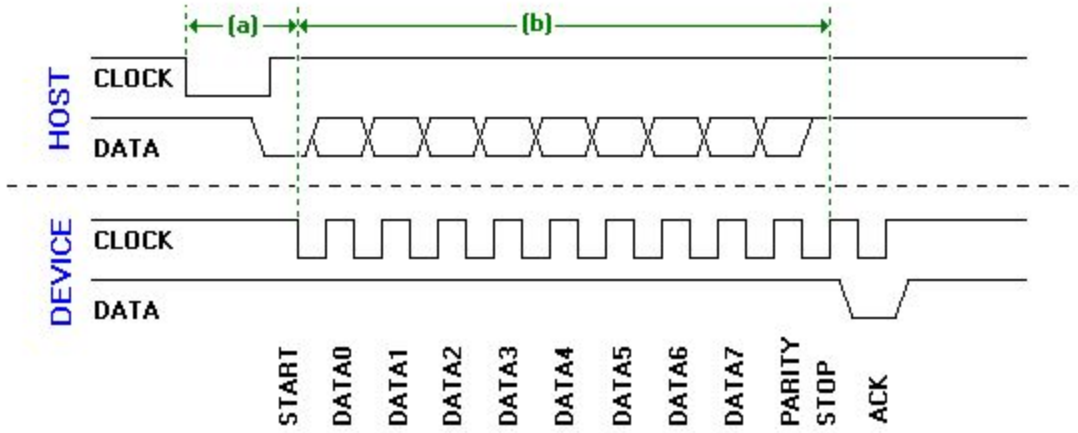


image from computer-engineering.org

Following is the pinout

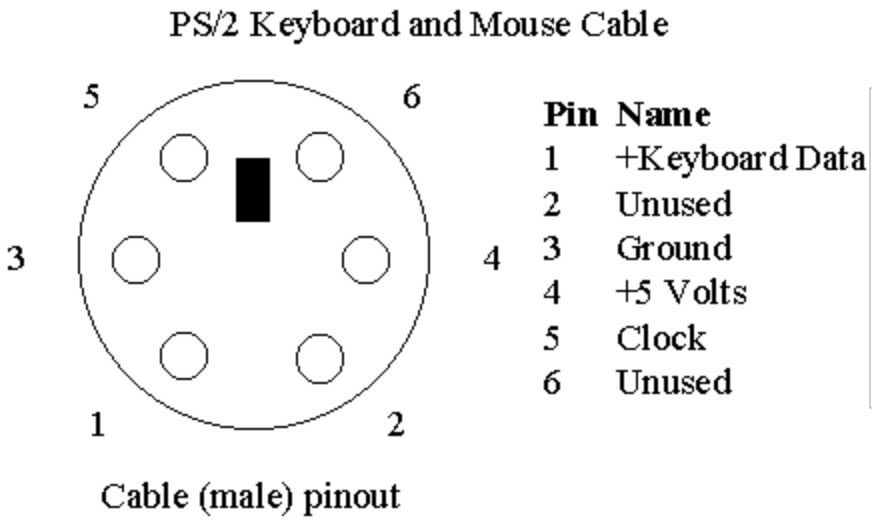


image from sparkfun.com

Interfacing this to arduino is simple.
Connect clock to any one of the interrupts, in my case pin 2, and other to digital io.
Basic algorithm would be

1) wait till clock is low
2) once low catch next 11 bits
3) extract data bit
4) parity check, if correct go else repeat

5) decode the data

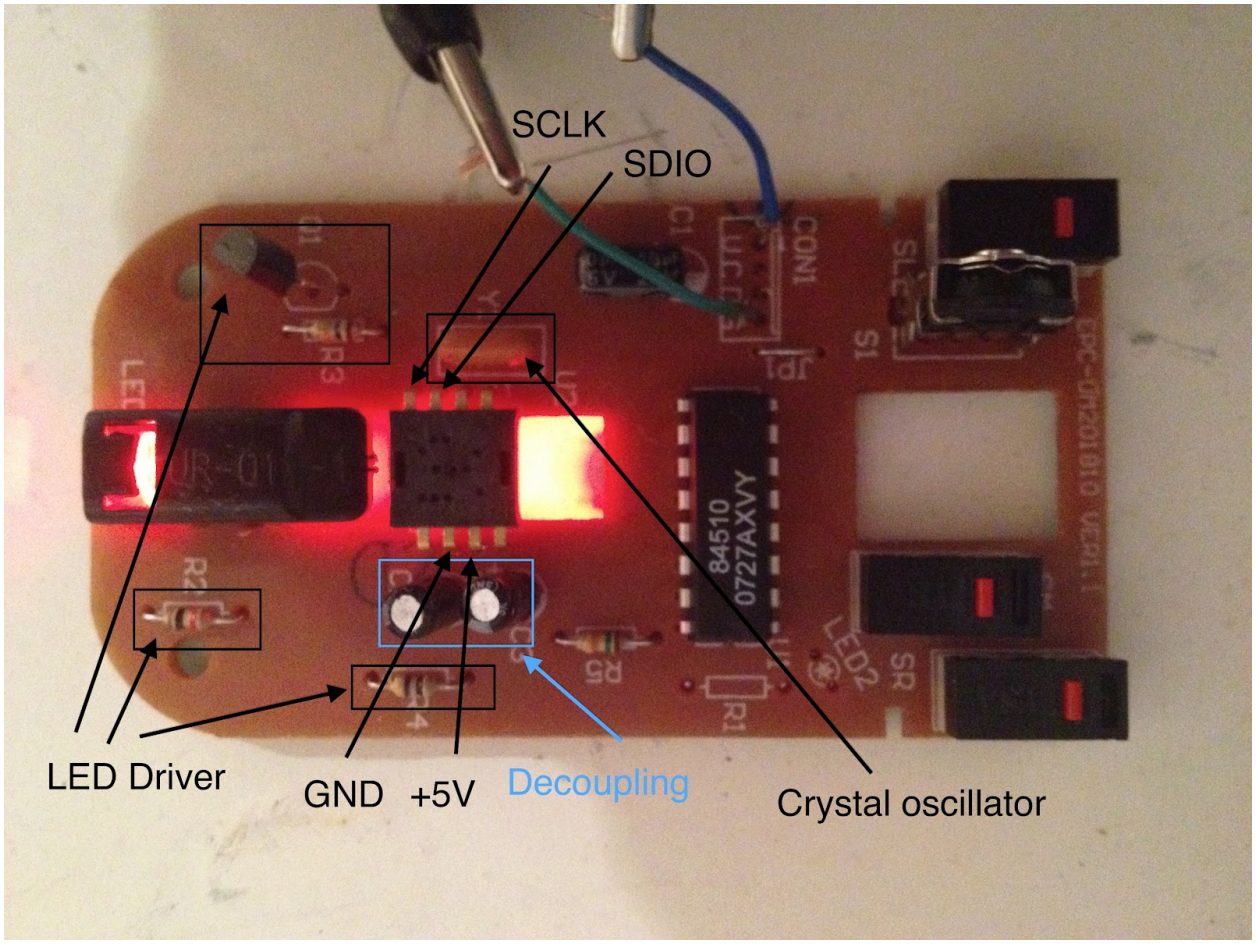
The PS2 mouse has the following data structure that is sent

	D7	D6	D5	D4	D3	D2	D1	D0
1st	YV	XV	YS	XS	1	0	R	L
2nd	X7	X6	X5	X4	X3	X2	X1	X0
3rd	Y7	Y6	Y5	Y4	Y3	Y2	Y1	Y0

image from http://hades.mech.northwestern.edu

The first byte sent is about details, like
first bit stands for left button
second is for right
third is always low
fourth us high
fifth stands for x axis sign
sixth stands for y axis sign
seventh and eighth stands for overflow

second and third contains x and y direction.
I have been successful to extract data in 3 bytes but they seem to be skewed and are not always accurate. This is also based on my previous work with interfacing keyboard to atmega16.



taken from http://2.bp.blogspot.com/

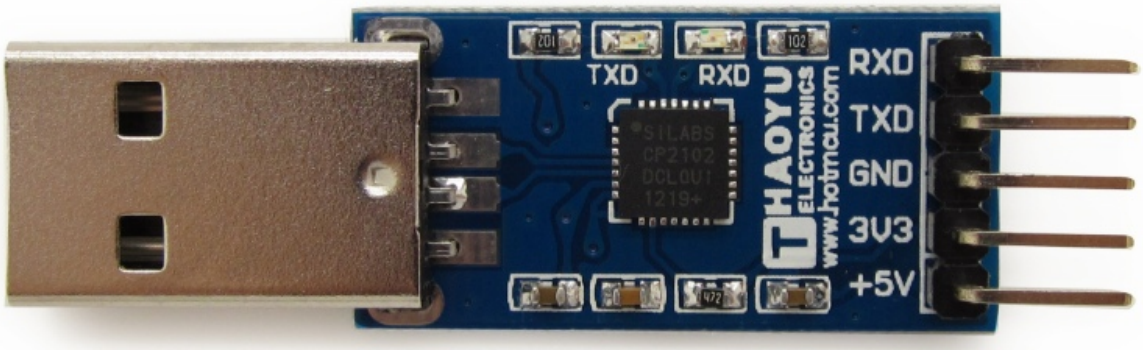
Hence the code is not yet online, will be made online soon.

Currently i am using a library to write data to mouse from arduino playground to write into the mouse. This once i reverse engineer the protocol will be replaced.

This has lots of interesting applications like low cost gesture recognition devices etc.

Computer to Computer Communication using serial port by example of a Chat application

Asynchronous Serial communication is most easiest of all type of communication in electronics.
Data is sent serially one bit at a time on agreed speed. The agreed data format is one start bit,8 data bit, one end bit and parity.
This can be used to transfer data and files in between two computers.
The rate at which data will be sent is previously agreed upon and is called as baudrate. here we used an baudrate of 115200.
Old computers had inbuilt serial port so that data could be sent serially directly. but newer system has chucked it for the reasons of usefulness, and also those were not hot pluggable like USB.
However there exists a work around.



This is called as USB to serial modules. These convert the serial communication protocols to USB protocols.
In Linux every device is a file, meant that they can be opened , written and read from.This makes the development over linux very easy.
whenever this device is plugged in it can be found in
ls /dev/
This prints all the connected device. this will be in format of /dev/ttyUSB* or /dev/ttyACM0 or so on.

The RX of this is connected to Tx of the other module. which are then interconnected with common ground.
This acts as our communication path.

Python is used to create the chat app.It is used for the easy of accessing serial port, and for GUI.
You will need to install pyserial and tkinter to run this.

Following is the example of just reading and writing with pyserial

```
import serial
connection = serial.Serial("Device name",115200)
connection.write("Hello World \n")
while True:
  if(connection.inWaiting()>0):
    print connection.readline()
```

This sends hello world over serial line and waits till there is some data. if there is some data it just prints it out.

Here is the copy of the chat app
https://github.com/rohitshetty/EC0504LabsReport/tree/master/serial
This needs to be installed in both the systems connected.

here is how it works.
Using Tkinter i have created a interface and simple GUI.This runs in its threadm at interval of 10 ms it checks if there is some incoming data, if yes then that is recived and updated.While if there is some data being typed that i caught and once pressed enter or send the data is sent over to other chatee.
Explaining of code is out of scope as it has lots of tkinter codes.