Interfacing the BS2 to an IDE hard drive.

On researching the IDE drive, I came across several interesting projects where the basic research was already completed. In short, I will not attempt to rewrite the research done, but give credit where credit is due. In all, the site www.pjrc.com/tech/8051/ide is the primary source of the information contained and used for this project.

The IDE drive is in all actually a group of I/O ports. Basic Input Output Systems (BIOS) have the ability to read and write at this level without drivers or special hard or soft ware, making the IDE drive the selection in any project needing mass storage. IDE can be use in either 8 or 16 bit mode, however the loss of 1/2 the drive capability is usually the deciding factor. In my interface, I have a BS2P40, so I was able to use the 16-bit mode without resorting to serial shift registers to move data between the drive and stamp. I didn't try to write to the hard drive, however, the mechanics are the same, just reverse of reading. Maybe, if some show interest I'll further examine that aspect of this.

The code used to access the drive was derived from the assembler software written by Paul Stoffregen for the 8051 with an 8255 controller. His software is in the public domain, but I guess if he did the work, we should know and give him credit for it, so here his is plug.

The IDE hard drive consists of two connectors. One connector supplying both the +5v and +12v required for the hard drive. Note the power connector has both grounds tied to one another. This is important. A large number of pins are connected to ground on the DRIVE side of the connector, as well as on the STAMP side. And the STAMP is on the same ground. For my project, I use an old external hard drive box with a power supply. It provided the +5 for the STAMP 2 DEV board and the +5v & +12v for the hard drive. All grounds are common between both sub systems. This is very important; without the grounds being common, flections and differences in the grounds made for some very unexpected results.

On the IDE side of the cable, on older drives, there is a pin missing. This pin's ID is 20. It's not used, and should be missing. Its location is easily noticed on the ribbon cable, it's the pin right under the notch. Pin 20 also marks which side the even numbers are on. The cable is numbered in columns, not rows. There are 2 columns of 20 pins, not 20 rows of 2 pins. (IE: NUMBERING IS NOT LIKE A CHIP!) Pin #1 is located on the other side of the connecter from pin #20. If you hold the cable so the wholes are facing down, and the connector is towards you, pin #1 is located in the upper right corner. In Figure 1, pin #1 is labeled with a 1. Table 1 shows the pin out of the IDE connector.

The Guts of the project: Wiring it all up

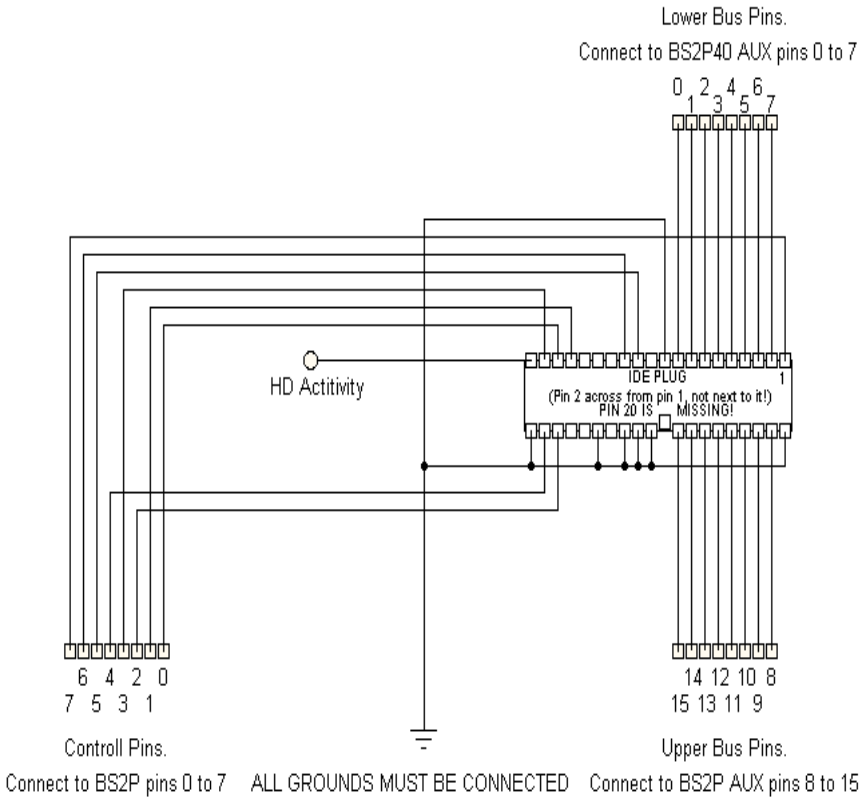
The drawing is a very basic schematic of all the connections to VCC, GND, BS2 and IDE, and one LED. The led pictured doesn't have it's current limiting resister, but its there. I should share with you, the hardest part of doing this project was wiring. I kept

on getting the bit orders messed up when I was connecting them to the STAMP. Rewriting the assembler code was simpler then keeping the pins right.

Table 1:The IDE Connector

1	/Reset
2	Ground
3	D7
4	D8
5	D6
6	D9
7	D5
8	D10
9	D4
10	D11
11	D3
12	D12
13	D2
14	D13
15	D1
16	D14
17	D0
18	D15
19	Ground
20	KEY
21	Reserved
22	Ground
23	/IOW
24	Ground
25	/IOR
26	Ground
27	/IOCHRDY
28	SPSYNC/ALE
29	Reserved
30	Ground
31	INTRQ
32	/IOCS16
33	ADDR1
34	/PDIAG
35	ADDR0
36	ADDR2
37	/CS1FX
38	/CS3FX
39	/DASP
40	Ground

Figure 1: Basic Connection from IDE hard drive cable to BS2. (NOTE: All current limiting and protection resistors not shown!)



As you can see from the pin out diagram, 7 lines are used for ground, 16 lines are data. Of the remaining pins, only 8 are used in the project. I did not connect the unused pins to anything. You should include the protection resistors between the stamp and the interface, as well as the current limiting resistor for the activity led. The digital emulation software I use doesn't support analog devices, so they aren't included in figure 1.

In the future, I plan on interfacing external RAM for the required 512-byte buffer required for all data transfers between the STAMP and hard drive, as well as drive info and maybe the beginnings of a simple file system. For now, I'm using the internal EEPROM for the buffer, and NO file system. The entire drive is ONE BIG FILE. Another enhancement is to modify the circuit so it uses 8-port expanders. Then I would use I2C to set or get the data. This would reduce the pin count down to 9 pins to a BS2 stamp, leaving pins to data logging.

On the subject of File Systems: A file system is simply a method of organizing information on a hard drive for easier retrieval of its contents, or the management of a number of files. That's it. Nothing special. In this projects current state, you can view it as a random access (that means you can pick a point and start reading) read / write FILE. It's one big file. To implement even the simplest file systems, you would use bytes to inform the file system program of what's there, and where it's located at, and how big it was. From the simplest systems, like that of the Commodore days, to the super complex, the bottom line is the information is stored in one way or another in a table of bytes. (For the Commodore it was a BAM, the PC, it's the FAT, for Linux it's ROMFS). The bytes in the table included a "human name" of the file; it's starting location on the media (hard drive, floppy drive, ram drive, etc), the number of sectors (blocks, etc), and the number of bytes in the last sector. No magic to it, just a simple table. Think of it like an index, with comments.

The Glory of the project: The Code

Being a programmer for more years then I care to admit, I've learned several things. One thing is NOT to reinvent the wheel. If someone has working code, study it, understand it, and use it if you can. Paul Stoffregen's code works. It was written in assembler for the 8051 microcontroller, with an attached 8255. (Ok, I hope I get the numbers right. I'm not familiar with the controller and it's PIA chips, but I tried. There's a link to Paul's work at the end of this document.) After studying his code, and looking for things that are part of his environment, I've evolved the following code. It's not pretty, I'm sure the great gurus at Parallax would be able to create better code, it does work.

The code is a collection of routines: Higher-level sub routines that can be called from your program, and Lower level routines called by the higher level ones. For now, there is no file system; so all that overhead is not needed. Basically, you call the Set_IDE routine with a starting sector. Next you set the IDE_POINTER to where in EEPROM you want the read or write from/ then you call either the Read_IDE or Write_IDE routine.

The buffer you will be using is 512 bytes long, so make sure you don't set IDE_POINTER to the end of EEROM… errors will happen.

There are some setup issues you must first do with the hard drive, especially because we aren't including any sort of management or control issues. In short, you should use a regular PC to setup the hard drive. I used a Segate 1.2 gig hard drive. I plugged it into the PC as a secondary drive, set the BIOS, set LBA as the access mode. I booted off a 95 floppy, ran FDISK, made a single partition, and then formatted the drive. When it was all said and done, I copied the autoexec.bat from the floppy to the hard drive, so I had something to look for. Lastly, I used a drive editor to discover where the starting sector was, which is located in the MBR (sector 0, starting at byte 192). I did this to confirm Paul's work. Lastly, and incorrectly I hard coded the starting sector into the code.

Some enhancements could easily be added: Such as the automatic discovery of the starting sector, the number of sectors used, and the number of bytes currently being used by the file. But, all of that falls under the heading of "File System", and thusly, I didn't get into it. As it sits right now, it reads and writes 512 byte sectors to any sector you want. Time and interest in the program / hardware will tell where it goes.

Links

http://www.pjrc.com/tech/8051/ide/ -> This is Paul Stoffregen page, just loaded with information on the subject, and one of my primary sources for the project.

As always, comments, both good and bad, are always welcome. And, just for the record, I've made this as an attachment so it can be downloaded and printed, should you decide to do so.

Fred Kerber
AKA Kaos Kidd

NOTE: Due to the bad combination of beer, misfortunately placed laptop and some overzealous football fans, my laptop, the home of all my code and other rather important information, is no longer among the digitally working collection of computers in my house. It is now the newest member in my digital graveyard of broken parts. Dell and I are in communications at the moment, regarding the warranties, expressed and implied, about the accidental damage insurance I have with the device. Once resolved, and any attempts at recovering the data on the hard drive are successful and the code in question is part of what is recovered, I will forward and share said code for this project.

KK