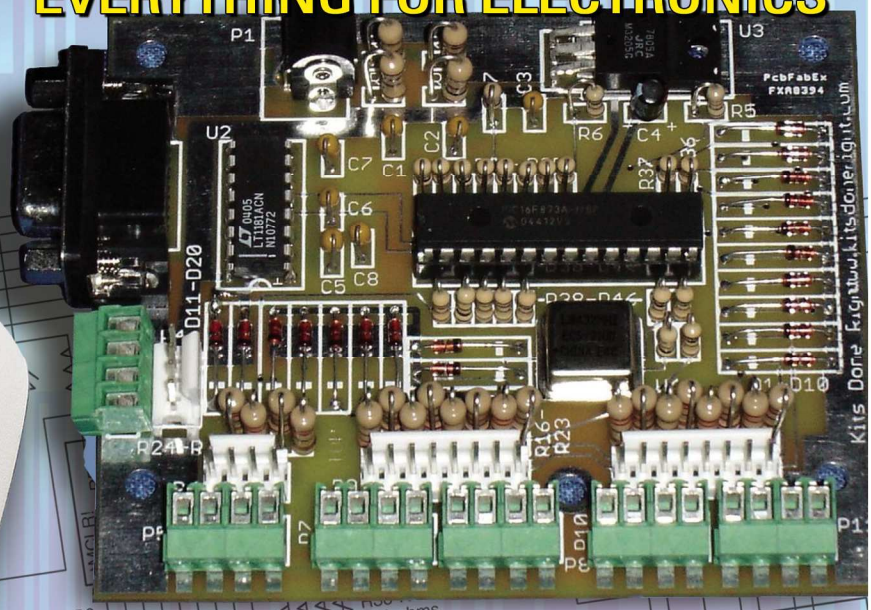
■ ROBOTICS ■ MICROCONTROLLERS ■ COMPUTER CONTROL ■ SPACE

NUTS AND VOLTS

NUTS AND VOLTS

MARCH 2006

EVERYTHING FOR ELECTRONICS

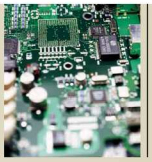


- **RESISTORS Tutorial**
- **Mini Sound Synthesizer**
- **Commercial Manned Spaceflight**

20 I/O Pin Data Acquisition Board

Easily Incorporate the Power of Your PC Into Any Project

www.nutsvolts.com



THIS MONTH'S PROJECTS

LEVEL RATING SYSTEM

To find out the level of difficulty for each of these projects, turn to our ratings for the answers.

●○○○ . . . . Beginner Level

●●○○ . . . . Intermediate Level

●●●○ . . . . Advanced Level

●●●● . . . . Professional Level

I remember the day when getting a single digital signal out of my computer was as easy as sticking a wire into the parallel port and remembering its address.

If you set a bit high, the corresponding pin on the parallel port would go high, and stay high until you told it to do otherwise. You could likewise read the state of any of the input pins, much like reading or writing to I/O pins on microcontrollers today.

20 I/O PIN DATA ACQUISITION BOARD

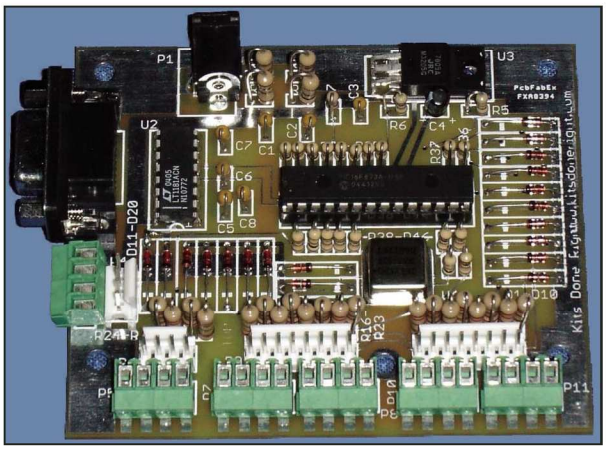
Easily Incorporate the Power of Your PC Into Any Project

When I first learned the art of manipulating the I/O pin on my computer's parallel port, I was in techno-heaven! Suddenly, connecting my projects to the computer and adding software became a breeze. I no longer had to try and build complex timers to operate my homebrew sprinkler system controller — I could simply use my computer to turn on a transistor which would drive my sprinkler valve solenoid.

I could then fully customize my sprinkler system to water the established grass on the right days at the right times (even with very strange watering restrictions), and also give my newly planted grass a five minute drink every half hour from 11am to 3pm. It seemed to me that any project could benefit from an interface with my computer's parallel port — home security systems, automatic lighting, and weather stations, to name a few.

This all changed with the advance of Microsoft Windows. No longer could I directly address my parallel port, and when I finally did figure out how to put data on it, I just got errors about not having a printer connected. I was able to make an LED blink on and off quite randomly, but it quickly became evident that I was going to need some other method for getting data into and out of my computer.

For a long time, I just didn't have a convenient (or inconvenient) way of utilizing the power of



■ FIGURE 1. Finished Data Acquisition Board.

my PC in my projects. I looked around at some other products and software, but none of them really seemed to fit my need. I needed something that I could use over and over again in a variety of different applications; a single board that could interface with several different projects at once, but was also cost-effective enough to justify dedicating it to a single project if that project demanded it.

I finally began to develop my own computer interface board, which would transform a standard serial port into something more like the old parallel ports from the days of DOS.

What Does the Board Do and How Does it Work?

The Data Acquisition Board (DAQ) converts standard serial I/O from a PC into a 20 pin parallel I/O bus. Each of the 20 I/O pins is individually configurable as an input or an output. You can read the state of each pin, or change the state if it's configured as an output, about a 1,000 times per second.

There is also the potential, with some additional firmware, to add up to five analog inputs with 10 bits of resolution. You can communicate with the board using binary commands or text commands. The text commands are easy to use, and can even be executed with Microsoft HyperTerminal. The binary commands require a better understanding of ones and zeros, but are much faster than the text commands. The board operates at bit rates up to 115.2K, and down to 300.

Why Did I Go With a Serial Port Interface Instead of a Parallel Port or USB?

Although many will disagree with me, there are a few advantages to a serial connection:

1. *Serial cables are cheap* — you can make your own without too much trouble, and if you're not worried about noise, you can sploog one together with just three wires: TX, RX, and GND.

2. *At the lower bit rates, you can use serial cables 50 ft or longer.* This has enabled me to connect some projects to the computer that would not otherwise have been practical without a wireless setup.

3. *A computer's serial port is easy to access*; you can use virtually any programming interface, and all you need are the standard drivers already installed on your PC.

Hardware Design

The DAQ is built around the PIC16F873A microcontroller (see the schematic in Figure 2). I used a 1.8432 MHz crystal oscillator for the clock because it's compatible with most of the standard serial port bit rates. You may notice that 300, 600, 1200, 2400, 4800, 9600, 19200, 38400, 57600, and 115200 all divide into 1.8432M evenly. Because of this, the DAQ can operate at all those bit rates with virtually no error.

RC6 and RC7 (pins 17 and 18 on the PIC) are the TX and RX lines for the serial communications; you can't just connect those two pins directly up to your PC however. The PIC is powered by a single +5V power supply, but a standard serial interface requires a negative voltage, as well as a positive voltage — a logical high is negative and a logical low is positive. Likewise, the transmit line coming from your computer swings positive and negative — if it was connected directly to the PIC, it would most certainly damage the chip.

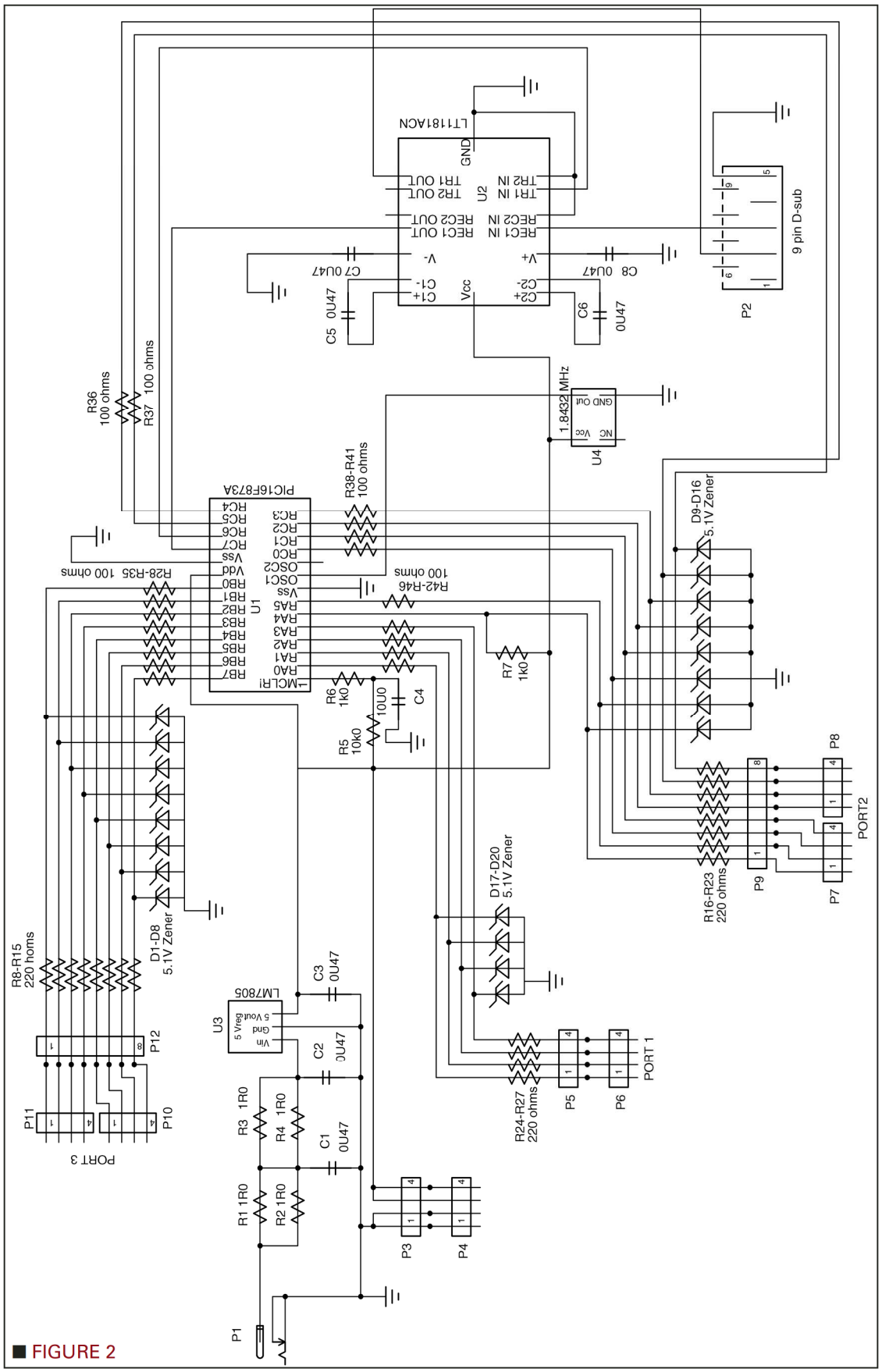
The TX signal from the PIC needs to be inverted and swing positive and negative, and the RX going to the PIC needs to be inverted and go from 0V to +5V. I decided to go with the LT1181ACN RS232 driver IC to interface between the microcontroller and the computer. This is a handy little IC! When the TX pin on the PIC goes high, the TX out pin on the LT1181 goes to about -8V. Likewise, when the TX pin on the PIC goes low, the LT1181 goes to about +8V. No additional power supply needed!

How do you get + and -8V out when you only have a +5V power supply? The LT1181 not only does the inverting for you, it also has a charge pump on the chip to generate a + and - voltage supply. Capacitors C5-C8 are filter caps for the switching network on the IC.

The MCLR pin on the PIC (pin 1) and its associated components (R5, R6, and C4) don't make a very interesting circuit. The circuit simply causes the MCLR pin to go high a short time after power-up which, in turn, causes the PIC to reset. I did want to pass on, however, that I tried a few different resistor and capacitor values before going with the ones I did.

A 10 µF cap for C4 intuitively seems very large for this application, however, smaller values for this cap cause a very frustrating problem. The DAQ would be operating just fine, and then it would freeze up for no apparent reason. I went over and over my firmware trying to find the never-ending loop I was sure was causing the problem, but I never found it. I checked and rechecked every solder joint trying to find the loose wire, but I never found that either.

After hours of troubleshooting and getting nowhere, it finally occurred to me that the PIC may be resetting intermittently. After a little more investigation, I found this to be the case. I then started experimenting with different values for R5 and C4; 10 kohms and 10 µF is what I ended up with. Again, a 10 µF electrolytic capacitor doesn't seem like the right answer to me, but I've used the DAQ board



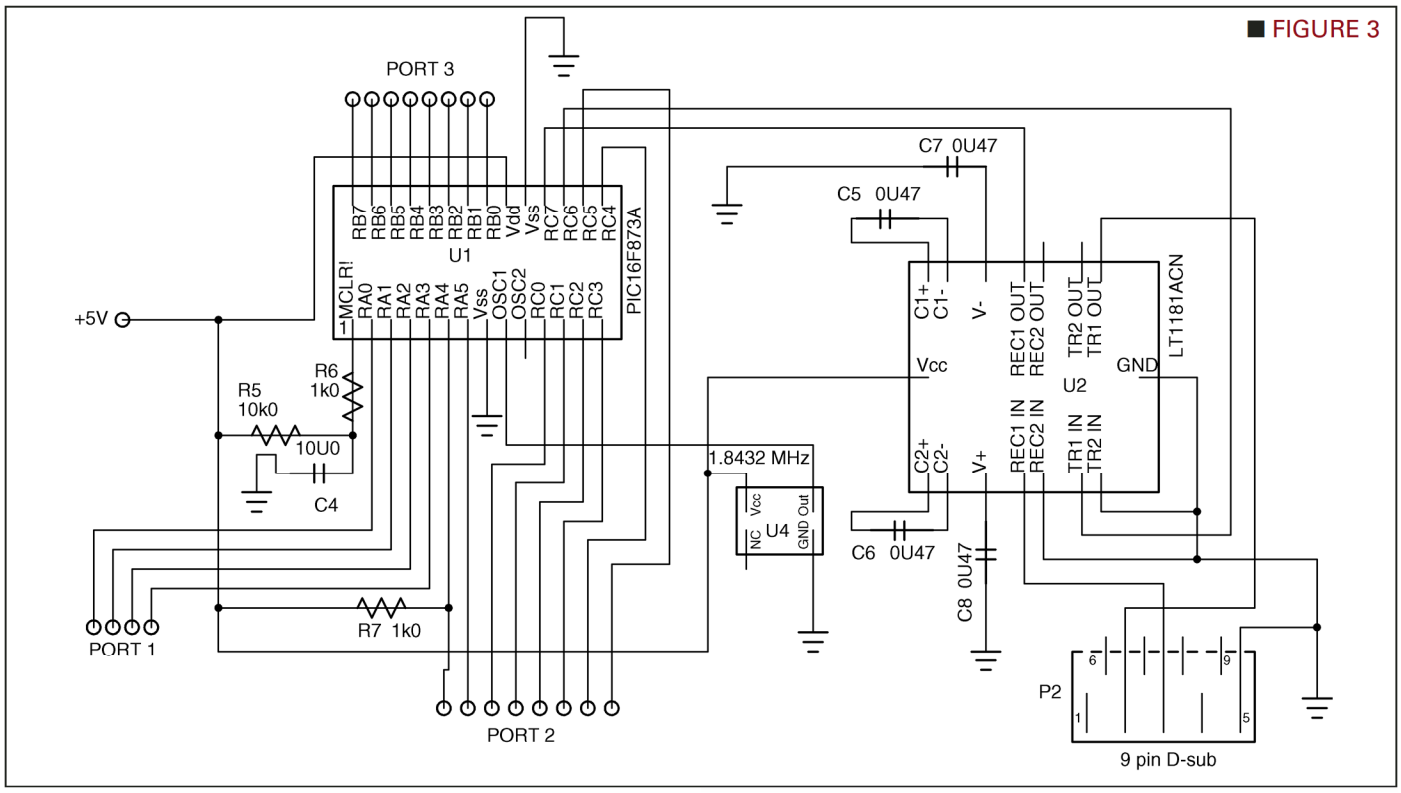
■ FIGURE 2

quite a bit in many different applications over the past year since I made the change, and I haven't run into that problem again.

The rest of the pins on the PIC (besides the power supply pins) are used as digital I/O. In the original design, each connector to the outside world was directly connected to its corresponding pin on the PIC. This worked very well and I didn't have any problems until the first time I accidentally applied a +12V signal to an input pin. The PIC didn't like that too much, and promptly quit working all together.

Since then, I've added some protection. As illustrated in the schematic, each I/O pin has two resistors in series between the connector and its pin on the PIC. Between the two resistors is a 5.1V zener diode to ground. The 220 ohm, 1/2 watt resistor in combination with the 5.1V zener should protect against a +15V signal applied to the input indefinitely, however the resistor will get hot. Any more voltage applied for an extended period of time may cause the resistor to "let out its smoke." The 100 ohm resistor limits the current from the PIC,

■ FIGURE 3



through the zener, to ground, if the pin is configured as an output.

Firmware Design

For an in-depth look at the firmware, it may be best for you to take a look at the full program. You can find the complete code by going to **www.kitsdoneright.com/daq.html** and clicking on the 'DAQ.asm file' link. It may also help to take a look at the user's manual and text commands (found on the same website) to get a feel for how the commands work before delving into the code. I wrote the code in assembly language using Microchip's MPLAB IDE.

The main program loop is relatively short and simple. It starts after setting up the serial port and I/O pins, and is labeled 'PoleFirstByte.' Most of the time, the processor just loops through the first three lines of the main loop ...

```
PoleFirstByte
     SEL_MEM_0
     btfss      PIR1, 5
     goto       PoleFirstByte
```

The first line you see — 'PoleFirstByte' — is not really a line of code, but just a label which marks the memory location of the first line of the main program which is 'SEL_MEM_0.' This stands for 'SELect MEMory bank 0' and is defined at the very beginning of the code; it can be replaced with 'bcf STATUS, 5.'

The PIC16F873A has two memory banks: bank 0 and bank 1. You have to tell the processor which memory bank you are using, or you may end up accessing the wrong memory locations. Clearing bit 5 of the STATUS register is how you tell the processor that you want to use bank 0, and the code 'bcf STATUS, 5' does just that. Next, you want to see if the PIC has received any data from the serial port.

Once a byte has been received on the serial port, the PIC will set bit 5 of the PIR1 register high. The second line of code instructs the processor to check the status of bit 5 in the PIR1 register and then skip the next line of code only if that bit is high. It's most likely that the bit is low, in which case the third line of code will be executed, which instructs the processor to go back to the memory location marked by 'PoleFirstByte' and start all over again.

These three lines of code are executed a few hundred thousand times a second until a byte is received on the serial port, then things get a little

Cost Cutting

■ If you're going to incorporate the DAQ into another project, you can significantly cut down on the cost by leaving out several of the unnecessary parts. If you're not concerned about applying an over-voltage to your inputs, you can eliminate R8-R46, and D1-D20. You can also eliminate P3-P12 if you want to hardwire your I/O. If you already have a +5V supply, you can eliminate P1, R1-R4, C1-C3, and U3. Altogether, that's 186 solder joints you don't have to worry about! By eliminating all those parts, it will also be much easer to put together without the PCB, further cutting down on cost. See Figure 3 above for the schematic.

more interesting.

The first byte in a serial transmission always has the command number (if you're using binary commands) or is a textual character. The rest of the main program loop just checks that first byte to see what to do next.

If the first byte is a binary command, then the program goes to the corresponding function which executes that command. If the first byte is a textual character, then the program goes to the memory location marked by 'cmd_text' where the rest of the command string is received, processed, and finally the command is executed by the processor.

Assembly Instructions

You can find detailed assembly and test instructions by going to www.kitsdoneright.com/daq.html and clicking on the 'Assembly instructions' link. You will also find a parts placement diagram, the images for the PCB (should you decide to make your own), and the parts list on the website. You can also download the complete firmware (DAQ.asm) so you can make your own custom modifications to the code.

The hex file is provided for download so you can just program the PIC without having to first compile the code. You can purchase a preprogrammed PIC, as well as the printed circuit board or the entire kit on the website. The website has many other resources, including programming examples and a utility program for testing your completed DAQ.

Possible Improvements

I use the DAQ board quite a bit, and it works very well for me, but there are a few improvements that could be made. The most obvious improvement would be to convert it to USB. This would greatly increase the speed of the DAQ, although it would also decrease the distance your project can be from the computer.

In regards to protection, you may have noticed that the DAQ inputs are protected relatively well to voltages over +5V, but not as well if you apply a negative voltage. In this case, the input pin would be clamped at -.6V, overstressing the PIC by .3V more than the manufacturer's specification allows for.

The addition of Schottky diodes in parallel with the zeners might be a good idea to clamp any negative voltages to -.2V. One other improvement that wouldn't take more than some additional firmware would be the addition of analog inputs. Turning PORT1 pins 0-3 and PORT2 pin 0 into analog inputs wouldn't be too difficult if you're using the binary commands. If you want to use the text commands, however, you will have to convert the binary number produced by the ADC into text — doable, but a little more difficult. The good news is, there's lots of room left in the PIC to add features. The current program only takes up about half of the program memory available. **NV**

PARTS LIST

RESISTORS	SUPPLIER	PART NO.	DESCRIPTION
❑ R1-R4	Panasonic	ERD-S1TJ1R0V	1 ohm, 1/2 watt, 5%, Carbon Film
❑ R6, R7	Panasonic	ERD-S2TJ102V	1K ohm, 1/4 watt, 5%, Carbon Film
❑ R5	Panasonic	ERD-S2TJ103V	10K ohms, 1/4 watt, 5%, Carbon Film
❑ R8-R27	Panasonic	ERD-S1TJ221V	220 ohm, 1/2 watt, 5%, Carbon Film
❑ R28-R46	Panasonic	ERD-S2TJ101V	100 ohm, 1/4 watt, 5%, Carbon Film
CAPACITORS			
❑ C1-C3	AVX Corporation	SA305E474MAA	0.47 µF, 50V, Ceramic
❑ C4	Panasonic	ECE-A1CKS100	10 µF, 16V, Electrolytic
❑ C5-C8	AVX Corporation	SA305E474MAA	0.47 µF, 50V, Ceramic
DIODES			
❑ D1-D20	Fairchild	1N751ATR	5.1V zener, 1/2 watt
ICs			
❑ U1	Microchip	PIC16F873A-I/SP	Microcontroller
❑ U2	LinearTechnology	LT1181ACN	RS232 Driver/Receiver
❑ U3	NJR	NJM7805FA	Voltage Regulator, 5V, 1.5A
❑ U4	ECS	ECS-2100A-018	1.8432 MHz Crystal Oscillator
CONNECTORS			
❑ P1	CUI, Inc.	PJ-202A	Power Jack, 2.1 mm, PCB mount
❑ P2	Kobiconn	152-3409	9 pin D-sub Receptacle
❑ P3, P5, P7 P8, P10, P11	Phoenix Contact	1984785	Terminal Block, 3.5 mm, Four Position
❑ P4, P6	AMP	640456-4	Vertical Header, Four Position, .100 in
❑ P9, P12	AMP	640456-8	Vertical Header, Eight Position, .100 in
MISCELLANEOUS			
❑ AC Adapter, 9 VDC, 300 mA			
❑ Xicon, 412-109033			
❑ Power Supply			