NXP 3725 - Panel Cutting Machine

Introduction

I design and build electronic equipment as a hobby. The part of every project that I dislike the most is making a front panel for the instrument. In a recent project, for example, there was a rectangular cut-out for the LCD and another large one for the keypad. I've always done these in the traditional way which is to drill around inside the desired opening, knock out the ragged interior and then file the opening to shape. I am not patient enough or skilled enough to do this well … and, I hate doing it!

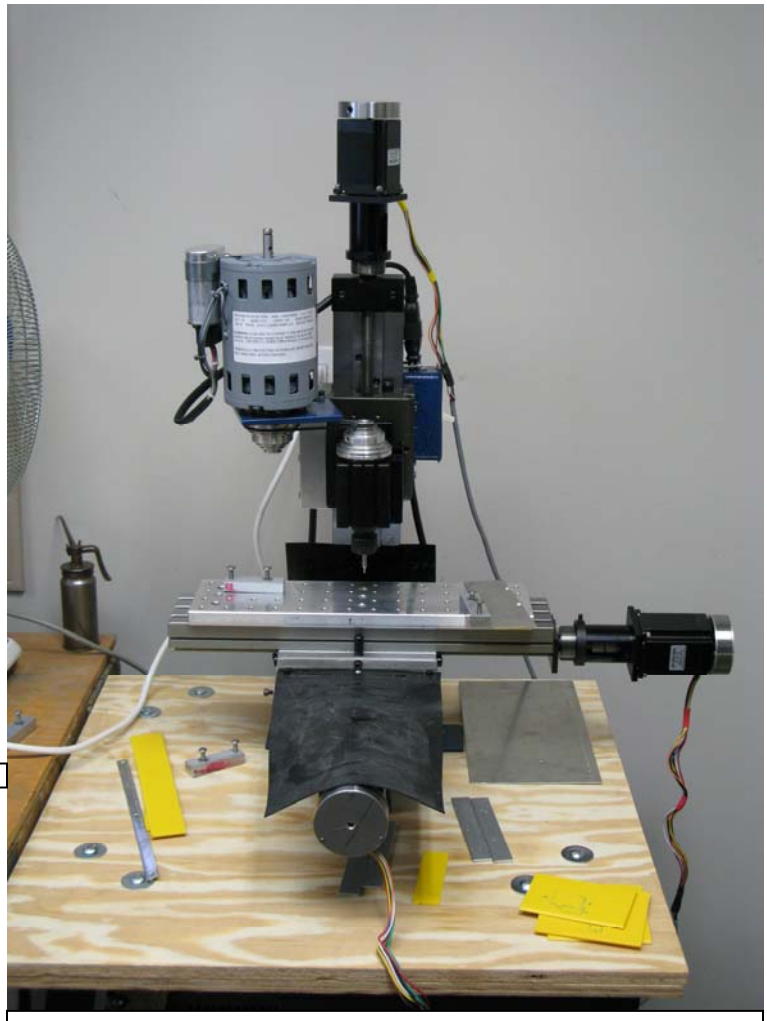

Figure 1 Taig micro mill with stepping motors on the leadscrews

I also do some other metal work for a hobby and, a few years ago, I bought a small milling machine which just had manual handles to turn the lead-screws which moved the table and the cutting tool. Each lead-screw had a graduated wheel with fifty divisions which, since each axis moved 0.05 inches per revolution, meant that it could be set with an accuracy of one-thousandth of an inch. However, if you wanted to move the table, say, 4.035 inches, it meant that you had to count 80 turns of the lead-screw plus another 35 divisions. It was surprisingly easy to loose count! After using this mill for a few projects, I resolved to add some electronic method of displaying the positions of the table and tool. It is possible to buy optical linear encoders designed for just this purpose but none was available for the particular mill that I had. I began to think

about making a controller which would keep track of the position by using stepper motors to set position rather than just passively detecting position. It was at this time that the mbed contest was announced. It seemed to be a natural.

Once you have a system in which the axes are driven by a controller, it is an obvious next step to add the capability for the automatic cutting of more complex shapes in sheet metal such as those needed for the panels of electronic equipment under the direction of a script of some sort. There are quite a few full-blown pc programs for controlling a milling machine using stepper motors but I don't like the idea of having a laptop or desktop computer in my workshop where metal chips are flying about. Also, in my workshop at least, there is very little vacant space. So, I have designed a physically small, self-contained controller, based on the mbed module, which allows me to run my milling machine either manually or, following a script, automatically.

Figure 2 Photo of the completed controller. The front panel was cut using this system.

System Design Considerations

Using stepping motors to move the lead-screws is an open-loop system because there is no sensor of position to ensure, once a step had been commanded,

that the shaft has really turned the correct amount. The remedy for this deficiency is to have strong enough stepper motors so that they will always be able to make the desired step no matter how much physical resistance there is. This means that the circuitry to drive the stepping motors must supply the proper source voltage and current to drive the larger motors. I decided, early on, that I did not want to have to learn all the knowledge necessary to make my own drivers so I bought a driver board from HobbyCNC[1]. I also bought the stepping motors from this company[2]. The stepping motors require a relatively high-voltage (high relative to logic circuitry) DC power supply capable of supplying several Amperes. Since voltage regulation is not required, it is a simple matter to build such a supply using a transformer, bridge

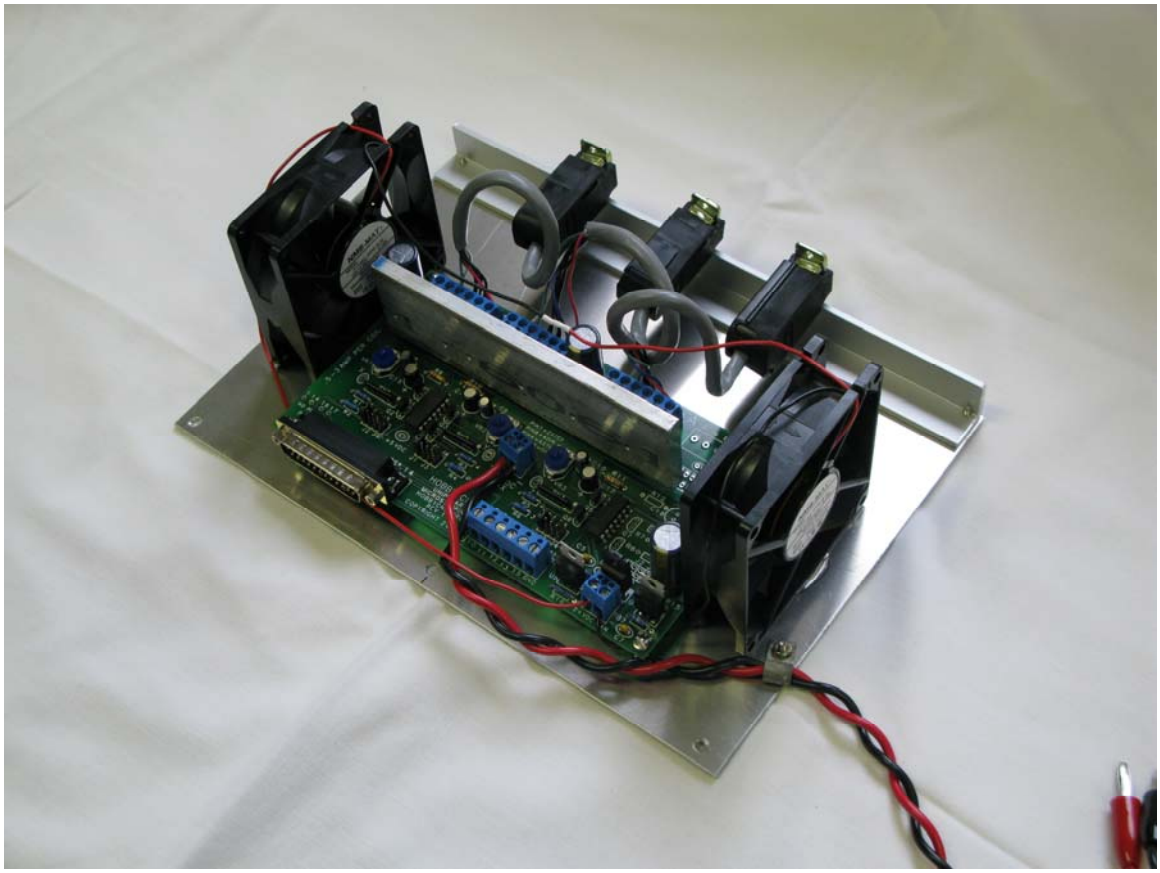

Figure 3 HobbyCNC stepper driver board

rectifier and large capacitor. Alternatively, switching power supplies capable of supplying sufficient voltage and current can be bought cheaply on eBay or from commercial electronics suppliers. I already had a large, heavy DC power supply[3] which is what I used. I have it set to provide 34V.

[1] http://www.hobbycnc.com/products/hobbycnc-pro-chopper-driver-board-kits/

[2] #23-205-DS8 285oz-in bipolar rating, 205oz-in unipolar rating. 3v, 3A, 200 S/R, 2.2mH, Size #23, Dual Shaft, 8 wire

[3] Sorensen SRL 40-6

Figure 4 Sorensen power supply with HobbyCNC board on top. This is on a shelf under the milling machine.

My milling machine was made by Taig[4]. I originally bought the manual model and subsequently added the kit which provided the mechanical coupling to stepper motors; it can be bought in the latter form as well. This is a rather small mill by normal standards and the torque needed for any of the three axes is well within the capability of the stepping motors which I had bought. No positional feedback is necessary.

On the Taig mill, the table moves in the X and Y directions by two lead-screws. The cutting tool with its motor moves on the Z-axis by its own lead-screw.

The Controller Design

For the controller, I wanted two basic modes of operation; manual and automatic. As we shall see, I later added an additional 'semi-automatic' mode.

[4] http://www.taigtools.com/mmill.html

In the manual mode, I wanted to display the position of the milling table and cutting tool at all times. The motion is controlled by three rotary encoders; one for each axis. Each encoder count causes a step of the stepping motor on the relevant axis. Most countries in the world use the metric system of measurement but, in Canada it is common to also use the Imperial system of units (inches, feet) so the display must show the positions in either of these measurement systems. I added an additional display mode of 'thou' because machinists often use that unit; a thou is a thousandth of an inch. The display units are controlled by a push-button switch, labelled UNITS; each press of the button cyclically changes the unit displayed. Internally, in the program, the positions are stored as a number of steps of the stepping motors.

Ideally the display should be a large, bright set of numbers at eye level when one is standing by the machine. However, I found that I usually sit on a stool while operating the machine so a bright, clear LCD on the same table as the milling machine can be used just as easily and quickly. I chose to use a back-lit 20x4 LCD; these are widely available from electronics distributors or, on eBay, from China. For the controller, I decided that one such LCD, labelled POSITION, would always be dedicated to the (x, y, z) position and that I would need a second display for other things like prompts for the operator.

Other than the units push-button, there are three other push-button switches. One is used to zero the displays; ZERO. This is very useful in machining projects where one might move the cutting tool to a starting position and then want to move specific distances from there. Pressing the zero button sets all the units to zero so that the displayed numbers after this will be relative to this starting position.

Another push-button switch, MODE, is used to cyclically change the mode of the controller from manual to semi-automatic to automatic. The fourth push-button switch is labelled GO and in either the semi-automatic or automatic mode, causes the controller to perform the relevant action.

Large industrial milling machines often have a rotary table holding the work piece so that, to make a horizontal pass of the tool at an angle to the normal X-Y directions, the table is rotated to the desired angle and then clamped before the cut is made. I wanted to be able to make angled cuts like that but it is virtually impossible to do this by hand using just the X-Y controls. I added what I called the semi-automatic mode in which a desired final position could be specified and then, when pressing GO, the tool would move in a straight line from where it is to this final desired position. In this mode, the rotary encoders only change the desired final

position as shown on the second LCD. After the desired position (only the x, y coordinates) is set, pressing the GO button will cause the table to move smoothly from the initial position to the desired position at a fixed rate in a straight line.

In the automatic mode, pressing the GO button causes the controller to follow a script which is written as a file on the SD card.

The control language

The scripts stored on the SD card are written in a very simple language describing the operations one needs to cut openings in panels. The file of these scripts must have the name 'setup.txt'. Each command line has the much same structure. It consists of a single letter command followed by up to three parameters. All spaces must use the 'space' character (ASCII 0x20). Everything following a ';' is considered to be a comment and does nothing. A list of commands will follow but let us consider the 'goto' command as an example.

```
g 2.35 1.23 i ; go to absolute location x=2.35, y=1.23 in inches.
```

If the 'i' is replaced by 'm', the units will be mm and if by 't', by thou's. The command or unit letters themselves may be in either upper or lower case. Some commands have no parameters. One of these is the 'start' command. It consists of a single 's' at the beginning of the line; the remainder of the line is considered as a comment. Some other commands may have just a single parameter; an example of this is the command which specifies the material thickness.

The commands are:

s	Start - this zeroes all the coordinates and lifts the tool from the surface. It is executed at the start of a program with the cutting tool at the top surface of the material and at the reference position in x and y. This command defines the origin of the absolute position of the cutting tool.
h	Home - this returns the tool to the starting position but with the tool lifted above the surface
u	Up - lifts the tool to the position for moving to another location
d	Down - lowers the tool to the cutting depth at the present position
g x y u	Goto - goes to position (x,y) with u being the units; i means inches, m means mm, t means thou's
r x y u	Rectangle - starting with the present position being the lower, left hand corner, cut a rectangle of dimensions x and y. It lowers the cutting tool and then cuts the rectangle and lifts at the end of the process. The tool diameter is compensated for.
c d x u	Circle - with the present position being the centre, cut a circular hole with diameter d; u is again the unit and x doesn't matter. Again, the tool is lowered to cutting depth, the circle is made (compensating for the tool diameter) and the tool is lifted at the end of the process.
a v	Thickness - v is the material thickness in inches. The cutting depth is set to be this thickness + 0.8 mm. When lowered, the tool will cut down to 0.8 mm below the surface of the working material. I normally put a thickness of plastic sheet under the working material. Z is zero at the surface of the material.
t v	Tool diameter - v is the tool diameter in inches
p v	Cutting speed -v is in inches per second. This sets the rate at which the material will be cut. When the tool is lifted, it moves at a faster speed.

So, for example, a file which would cut a 15 mm diameter circular hole at a location which is at x=1.0, y=1.0 inches from the starting location would look like this:

```
t  0.0625 ; the tool diameter is 1/16 inch
a 0.069 ; the material thickness is 0.069 inches – this is 16 gauge
p 0.02 ; cutting speed is 0.02 inches per second
;
;
s ; this lifts the tool after the coordinates are all set to zero
```

```
g 1.0 1.0 i ; go to location (1.0, 1.0) in inches
c 15.0 0.0 m ; cut a circular hole with a finished diameter of 15.0 mm
h ; return to the starting position but with the tool lifted
```

This file must be written by a simple text editor and named 'setup.txt'. If any or all of the first three commands were not there, the program would use the default values for these parameters.

Macros

To make the panel cut-outs for a device like a LCD, there are four mounting holes plus a central rectangle which must be cut. Normally you would want to place this device at some location on the panel - but, all the locations in the above commands are in absolute units. So, to put an LCD in a particular place, all the locations need to be referenced to a new starting position which is not the same as the original starting position. The problem is solved by the use of macros. If, in the command listing, a g, r or c command is preceded by a single 'm' command, all further positions will be referenced to the position where the tool was when the m command occurred. There is also a macro 'end' command consisting of a single e which turns off this relative position setting and all further positions are absolute.

At the moment, I have written macros for things like Cannon D connectors (9, 15 and 25 pin), the 20x4 LCD, and AC power sockets. These are text files and they are copy/pasted into the setup.txt file, preceded by a m command and followed by an e command. It is my ultimate intention to have these written on the same SD card which holds the setup.txt file and then just refer to them in the command file by name with a new 'macro' command.

As it is now, an example of one of these macros for a 15-pin type D connector is:

```
m
g 0.12 0.25 i   ; location of left-most mounting hole from bottom, left corner
c 0.125 0.0 i   ; drill 1/8" hole
g 1.42 0.25 i   ; location of right-most mounting hole
c 0.125 0.0 i   ; drill 1/8" hole
g 0.22 0.025 i  ; go to bottom-left corner of desired rectangular cut-out
r 1.11 0.45 i   ; cut the rectangle
e
```

The setup.txt file to cut a 15-pin type D connector with the lower left corner of the connector flange at absolute location (2.5,2.5) mm would be:

```
t  0.0625 ; the tool diameter is 1/16 inch
a 0.069 ; the material thickness is 0.069 inches – this is 16 gauge
p 0.02 ; cutting speed is 0.02 inches per second
;
;
s ; this lifts the tool after the coordinates are all set to zero
g 2.5 2.5 m ; go to location (2.5, 2.5) in mm
m
g 0.12 0.25 i   ; location of left-most mounting hole from bottom, left corner
c 0.125 0.0 i   ; drill 1/8" hole
g 1.42 0.25 i   ; location of right-most mounting hole
c 0.125 0.0 i   ; drill 1/8" hole
g 0.22 0.025 i  ; go to bottom-left corner of desired rectangular cut-out
r 1.11 0.45 i   ; cut the rectangle
e
h : home
```

Summary of controller functionality

In this manual mode, the system acts as a simple manual milling machine but with an electronic display of position and manual control of the three axes via the rotary encoders.

In the semi-automatic mode, one can set an end point (x, y) location and then, pressing GO, the milling machine tool will go from where it is now, in a straight line at a constant speed, to that location.

In the full automatic mode, pressing GO will cause the machine to execute a script written in a file named setup.txt on the SD card.

Controller Electronic Design

The first cut at the electronic design used the mbed module mounted on a prototype board and was used to test the procedures for running the LCD's and the SD file routines. This was incredibly easy because all the work had previously been done and it was just a matter of making use of library code written by others.

The controller needed digital logic level wires for the LCD's (eight lines for two displays), for four push-button switches (four lines), three rotary encoders (six lines), one SD card (three lines) and lines to the stepper driver (six lines); a total of twenty-seven. The mbed module only has twenty six free general purpose logic level lines. I decided to add a port-expander which uses up two lines for the I2C bus but gains eight new general purpose lines. A printed circuit board was designed, using Eagle, and manufactured by a company in China. The schematic diagram of this board is shown in Figure 14. The inter-connections to the various devices were by in-line headers. I wanted to be able to use an unregulated power supply for this board so I designed in a ubiquitous 7805 regulator. The regulator is on the bottom side of the board so that the mounting hex spacer will provide a short thermal path to the metal chassis.

The power supply inside the controller was built on a printed-circuit board which I designed and had made at least fifteen years ago. I have long since lost (mislaid?) the Eagle files used to make the board. It is a simple transformer followed by a full-wave bridge rectifier and a filter capacitor. I won't bother drawing a circuit diagram for it. It uses a Amveco Magnetics 70060K transformer with a 25 VA 7V secondary winding going to a Motorola MDA 970-2 rectifier followed by a 16V 82000 uF filter capacitor. Under load, this supplies about 10VDC. The transformer and filter capacitor are available from Digikey. The bridge rectifier is now obsolete but it can be replaced by any 50 PIV, 5 A bridge rectifier. I found that the mbed module would often not reset properly if it was turned on again shortly after it had been turned off. This was, I think, due to the fact that the filter capacitor would discharge very slowly after the voltage had gone down to below the regulator level on the mbed module. I connected a 25 Ohm 10W resistor, heat-sinked to the chassis, to the output of the power supply to cause the filter capacitor to discharge more rapidly and this fixed that problem.

The rotary encoders are optical types with 128 pulses per revolution. They are Bourns ENC1J-D28-L00128L.

Programming Considerations

The program is really quite simple. The structure of the main procedure is the quintessential controller; an initialization section followed by a 'while … forever' loop. The rotary encoders generate interrupts which change the value of global variables, one for each of the three. In the main loop, the push-button switches are polled to see if any have been pressed and then the globals changed by the interrupt procedures are examined to see if any of them have changed. If any of these actions have taken place, then the appropriate procedure or function is done.

I have deviated from what is considered good programming practises in two ways. One of these is in the way in which I treat *#include*'d files. The other is in the use of global variables. I feel I need to explain these differences because they are a departure from what is considered good practise and I want to assure the referees for this contest that I know the difference.

Conventional C or C++ programming is done with header files, indicated by the .h file name, which just contain the prototype statements of the procedures and functions. The implementation of these is then shown in a .c or .cpp file. This is mandatory if you have a large project which is being done either by more than one person or if it is very complex and must conform to some initial design. For small programs like the one here, it is, IMHO, unnecessary to do this and just entails more editing; when a change is made, you have to edit two files instead of one. For this reason, in my 'include'd .h files, I put the whole code and did not bother with separating the prototypes from the implementation. If I were publishing any of this code, for example as a library routine, I would separate the files.

Use of global variables is usually decried as making a program difficult to understand. Instead, one should normally make independent procedures and functions and pass all needed variables to them explicitly. However, passed variables are put on the stack and, if there are large numbers of them, it means that the procedures and functions which use them must first push them onto the stack and then, before exiting, pop them off the stack. I have been programming microprocessors since the mid-1970's and this pushing and popping was, for the earliest microprocessors at least, very wasteful of time and limited RAM space; often, especially, time. So, I have gotten into the habit of using global variables for many parameters. For this particular project, the demands on time and RAM space are not difficult to meet with the mbed module and, to conform to good programming practise, I could have done more parameters passing and only used global variables for things like in interrupt routines.

Indeed, I thought about it when beginning this project. I decided however, partly because this is a relatively small project, to just continue my normal programming habits in the interest of expediency.

The current state of the project

The system works well as it is. I originally made a very rough enclosure, shown in Figure 5, to develop the firmware. After the firmware was in its present state, I

Figure 5 Prototype unit used during development

used it, in its automatic mode, to make the finished enclosure shown in Figure 2. The back plate for this, as cut and without deburring the metal is shown in Figure 6. The rectangular opening, with two mounting holes is for the AC receptacle, there is a ½" hole for the fuse holder and a ¼" hole for the power switch. The AC receptacle hole

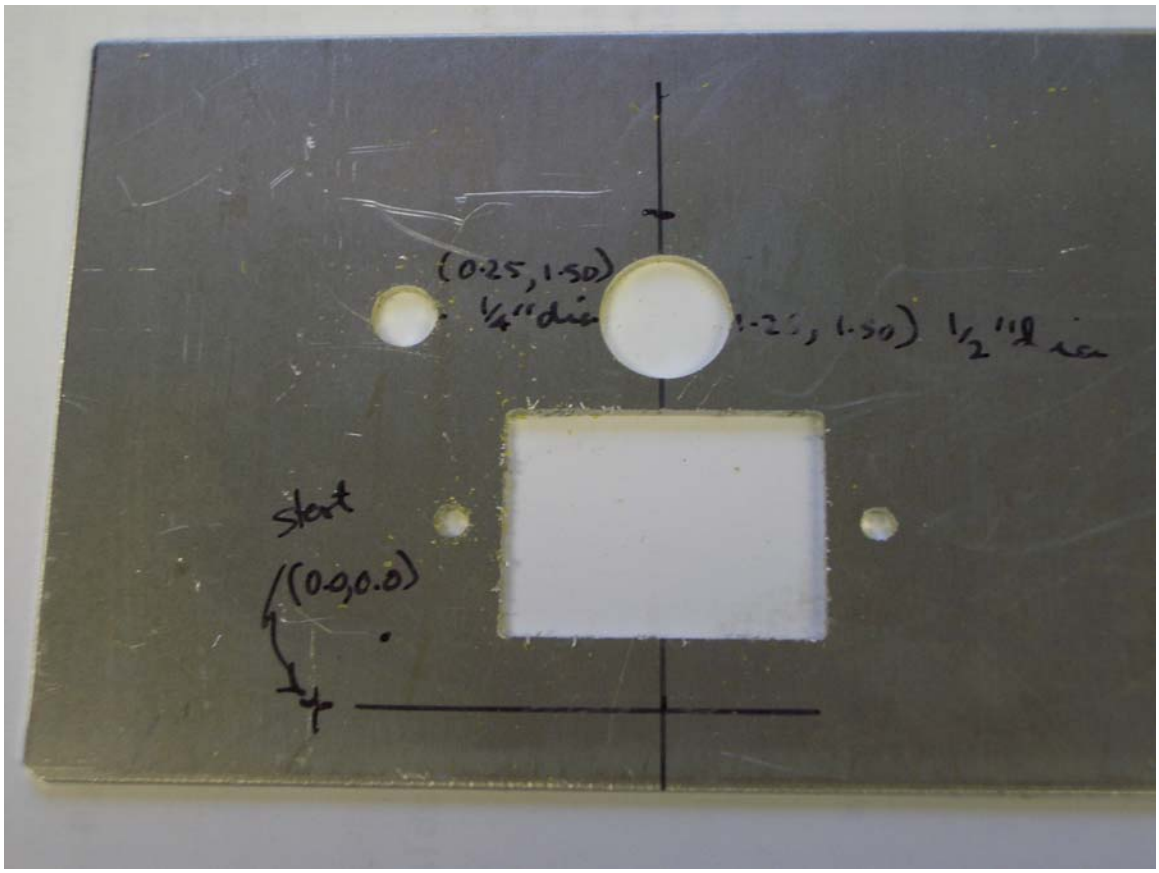

Figure 6 Rear panel after being cut but before deburring

is, really, quite rectangular; the camera angle makes it look a bit awry. The front panel was also cut automatically. Figure 7 shows the panel as cut; Figure 8 shows it

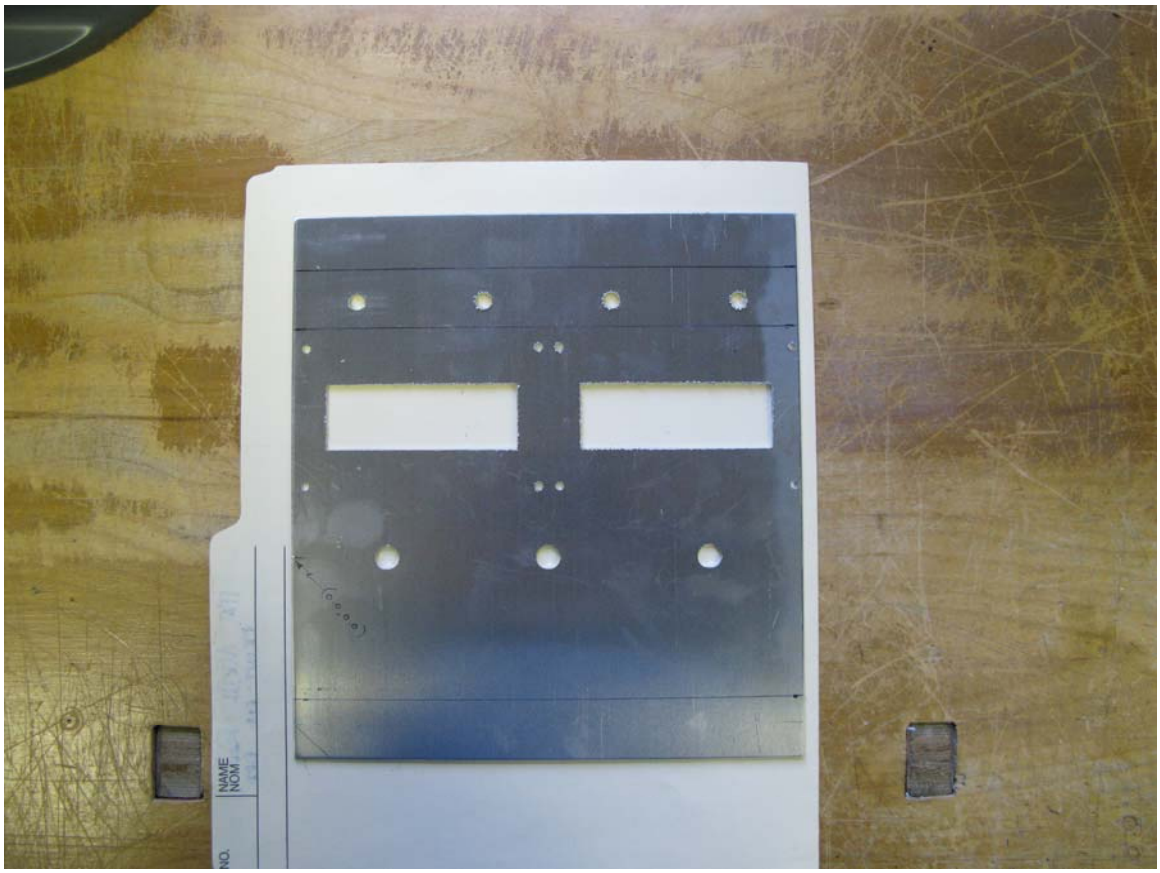

Figure 7 Front panel after being cut but before deburring

after bending but before painting. Figures 9 through 14 are some additional photos

Figure 8 Front panel after bending

showing the finished controller in its enclosure as well as some block diagrams and the schematic of the printed circuit board with the mbed module.

Future Plans

The first modification of the firmware will be to change the way the macros are done. Now, they are inserted into the script file using editor cut and paste. It isn't onerous to do it but it isn't really convenient. I will modify the parser in the program to allow macros to be included with a statement like:

```
m db25 ; macro to cut a 25 pin Cannon type D connector
```

On the SD card, I will have separate files for each macro with a filename, in this example, db25. When the parser encounters a macro statement like this, it will read the appropriate file and insert the instructions into the script. I do not anticipate any great difficulty in doing this.

In the longer term, I will modify the program to allow an 'engraving' mode where the cutting tool is replaced with a ball-end tool and the cutting depth adjusted

to allow the tool to be set to just engrave the surface. I will add some firmware to provide lettering so that front panels may be engraved. This isn't too difficult as, in the late 1970's, I wrote a library for doing lettering on a simple X-Y recorder where the recorder just moved to X-Y positions sent to it via a 300 baud serial interface. I was using a home-made CP/M system which had a Z80 processor, Zilog arithmetic co-processor, two 8" floppies holding, gasp!, an enormous 1 Mbyte each of storage. I was using a great compiler, for the time, called PascalMT+ and I even contributed this library to the PascalMT+ users group.

Addendum

I have included in this ZIPped documentation file a short video showing the system cutting, in automatic mode, the mounting holes for a Cannon DB09 connector. It is called db09.mp4. This was done on a scrap piece of 16 gauge aluminum sheet; there is a previously cut set of holes in the foreground.

The code files are in the bmm.zip file.

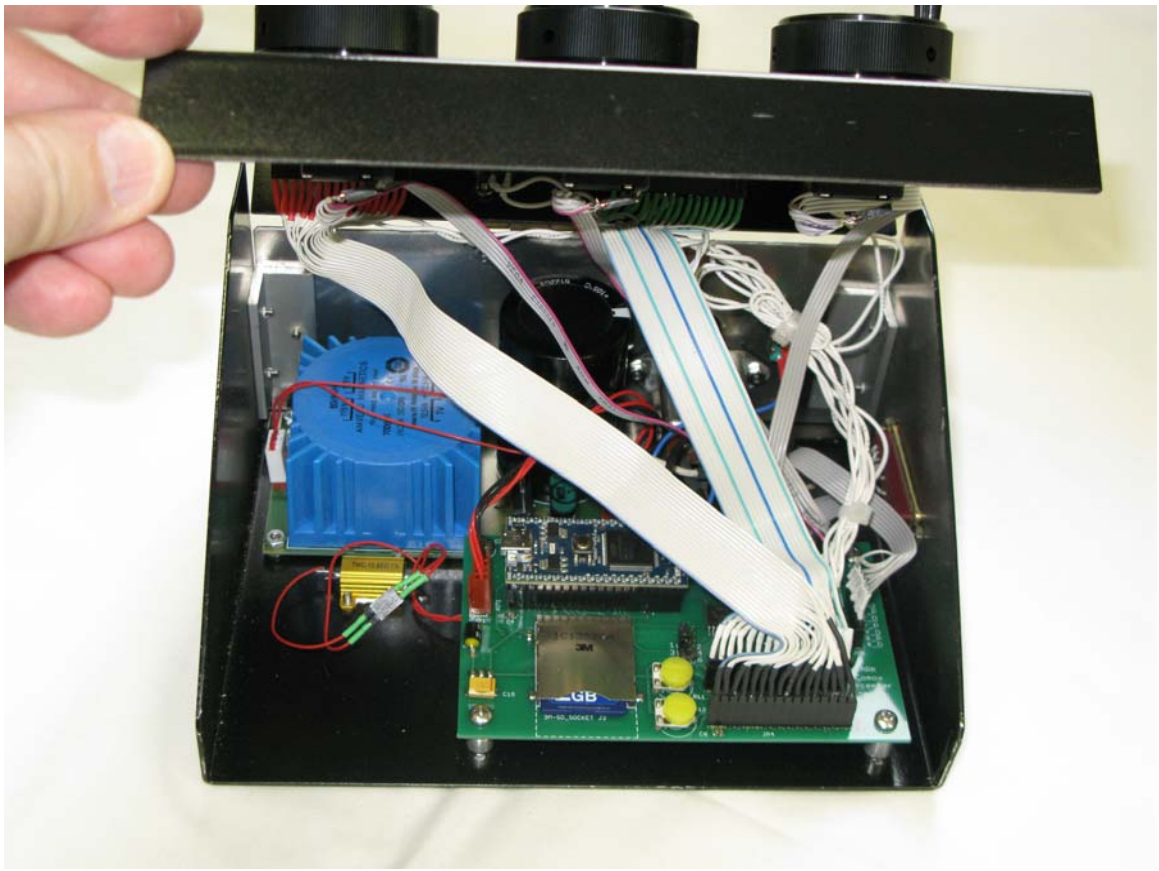

Figure 9 Photo of the controller with the hinged front lifted up to show power supply and printed circuit board holding the mbed module. Having a hinged front panel allows easy access to the SD card.

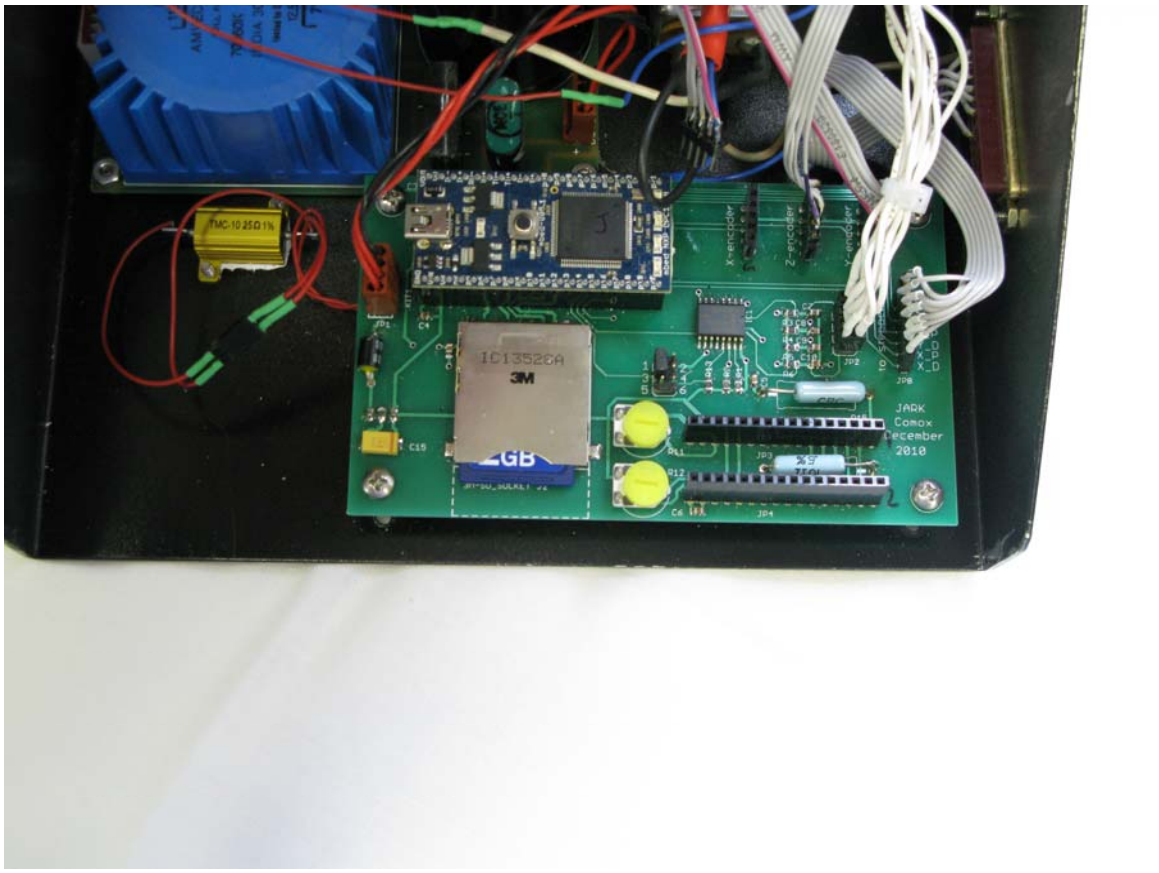

Figure 10 Photo showing the printed circuit board more clearly with some cables removed.

Figure 11 Photo showing prototype controller and milling machine during developmental phase of this project.

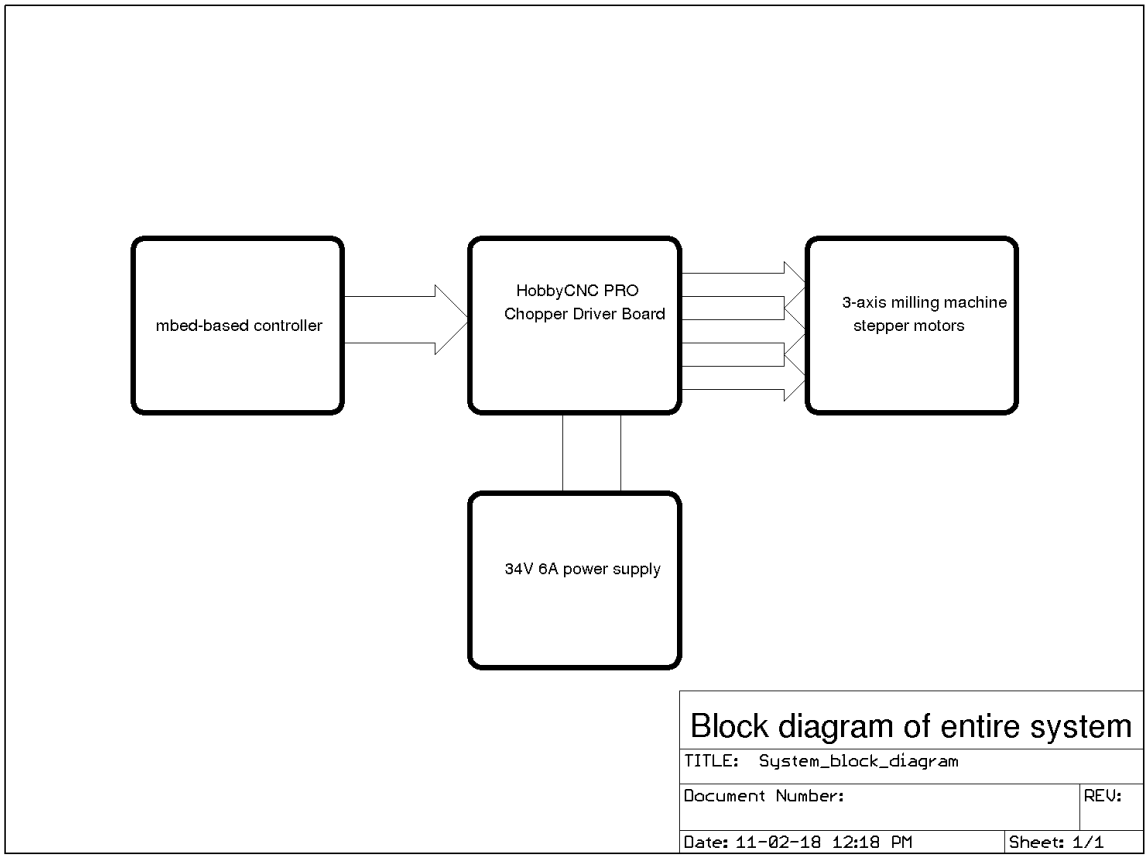

Figure 12 Block diagram of the whole system

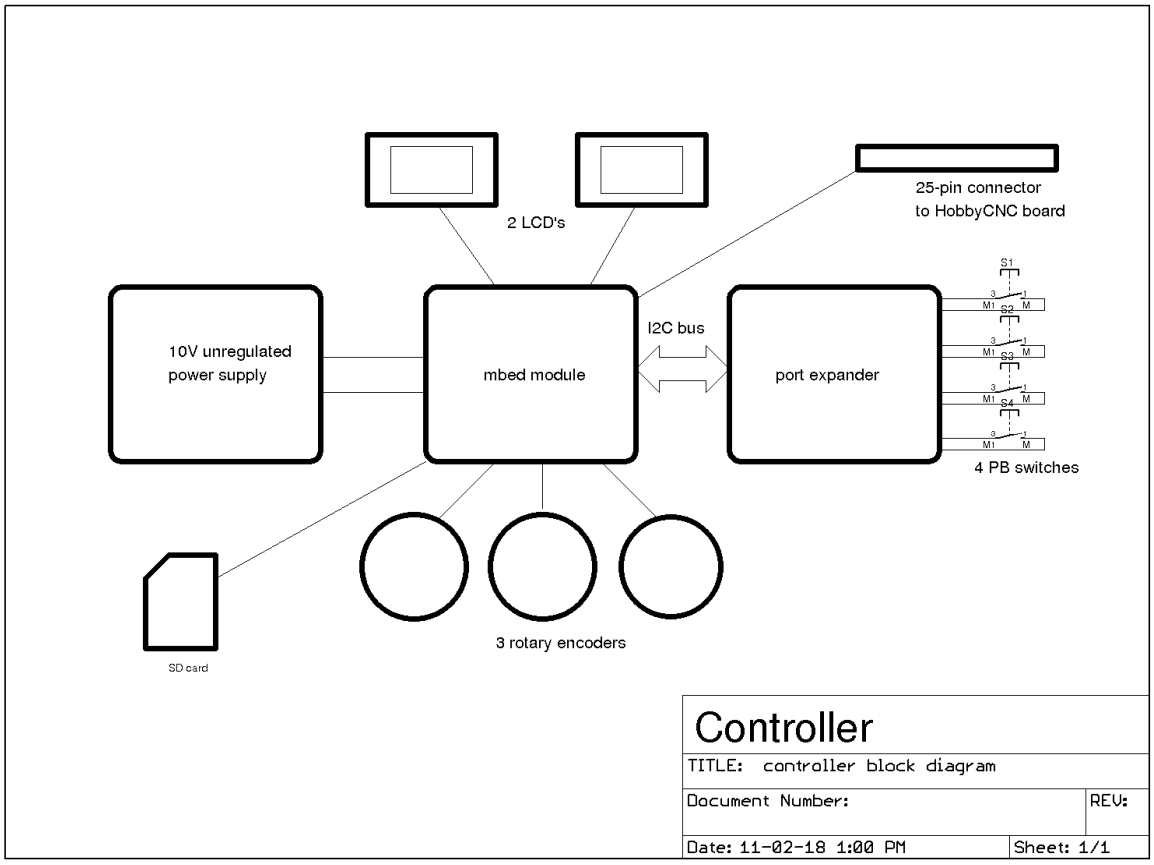

Figure 13 Block diagram of the controller

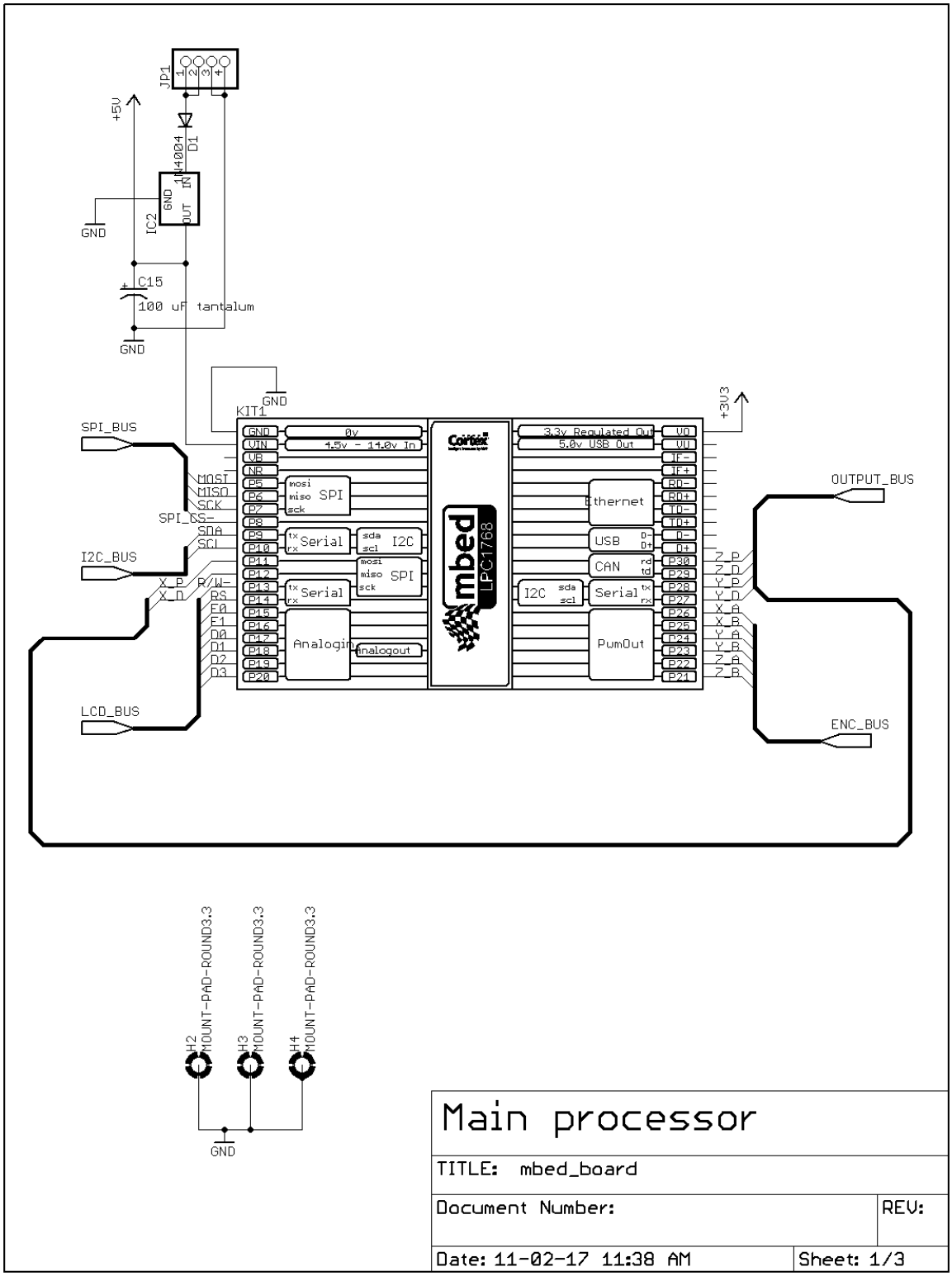

Figure 14a Schematic of the printed circuit board in the controller

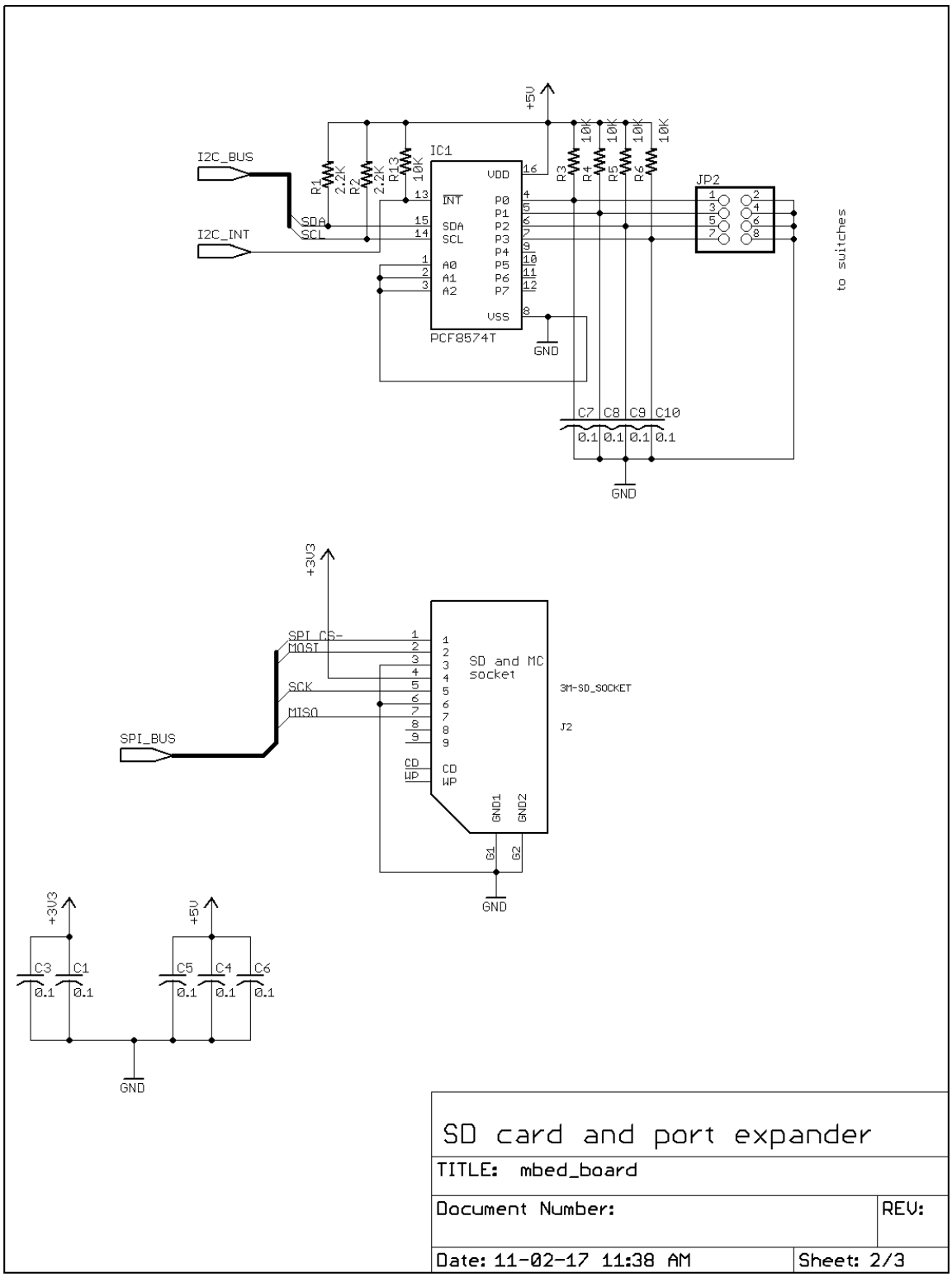

Figure 14b More of the schematic

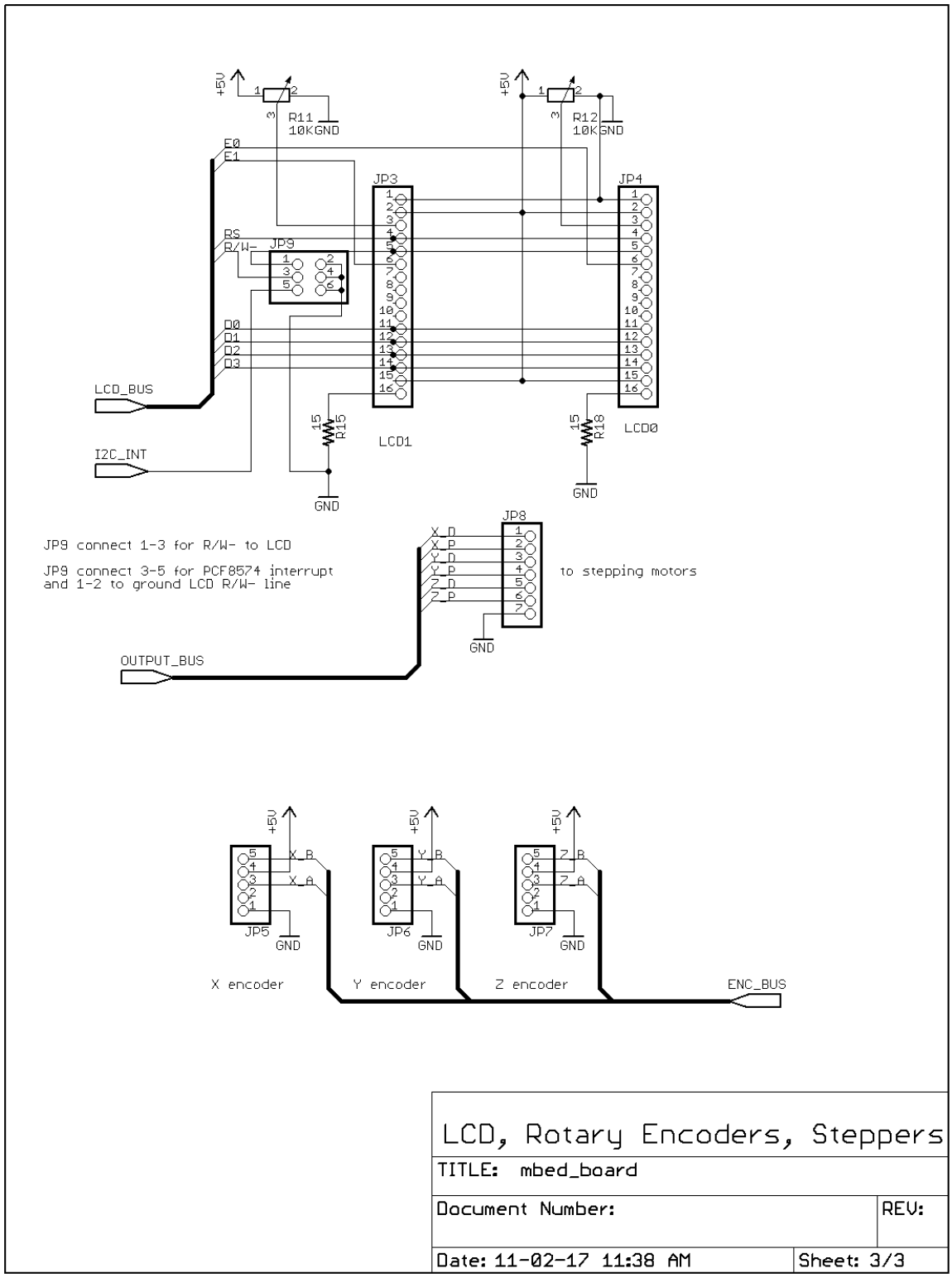

Figure 14c More schematic