I was using the machine, I became increasingly curious about the way the G-code (used to program the machine) was converted into three-axis motion using the machine’s three stepper motors. It seemed like the PC/Mach3 combination could be replaced by a microcontroller-based unit. I thought designing such a controller would be a challenging project and decided to give it a try. Photo 1 shows the end result.

I want to be upfront and say this is probably not the most practical project I have ever done. You can usually pick up a used PC for free, and the Mach3 software is professional-grade and handles much more complex G-code programs than my DIY controller will. However, it did provide me with a challenging programming task, and I learned a lot about designing a program with many concurrent tasks, all of which are time critical. Even if you are not interested in building such a controller, you may find some of the techniques and tricks I used to provide the multi-axis stepper-motor motion useful.

This project involved two main tasks. The first was to understand the G-code language used to program CNC machines well enough to write the firmware that would parse the G-code commands into...
something a microcontroller could use to control the stepper motors for each of the three axes. The second task was to design the hardware/firmware that would actually control the three stepper motors, all of which had to move synchronously at accurate, ramped speeds. First, let’s take a brief look at G-code.

G-CODE

Many books have been written about G-code programming, and I doubt any one particular book would be comprehensive enough for all types of milling machines. The G-code language is arguably as complicated as any computer language and probably harder for a computer to interpret because its syntax is not as strictly defined as most computer languages. I will only touch on the basics in this article. (For more information, search online for a comprehensive list of tutorials and so forth about G-code.)

G-code is a group of single-letter mnemonic codes that are used to define all the necessary operations required by any CNC machine tool. Because many operations are required by different milling tools, the single-letter codes are expanded in function by a numeric value that is appended to that code. Parameters (e.g., the position to which you want the cutting head to move) are expressed as ASCII numeric strings, containing as few or as many decimal places of accuracy as are needed for the operation. G-code command files are basically text files, with a CR/LF termination between each line of code. More than one G-code command can be on a single line, and there can also be comment lines. Command lines are not normally numbered, but it is acceptable to do so for your own purposes. The position coordinates’ order does not need to be specific (i.e., x, y, z). Not all target position coordinates need to be assigned with every command; only those that differ from the current position (in that axis) must be specified.

For the purposes of my controller, only linear motion commands (called linear interpolation in CNC) are implemented. The G-code language also contains commands for circular interpolation. These commands are more complex for the controller, so I did not include them.

That is not to say that this controller won’t handle curves, arcs, and circles. Many programs that provide G-code output do so using only the linear motion commands and use many small linear motions to generate curves and arcs. All the programs I use fall into this category, so my controller can handle this.

Among the linear motion commands, I manage two main commands. The G00 code, which is called the rapid code, is meant to move the tool rapidly to a target position at a rate independent from (and normally higher than) the programmed feed rate. In rapid moves, the tool is not cutting and the intent is to quickly get it to the target position. If the target position is different in more than one axis, the tool does not necessarily move at an angle. However, it may do one axis after another, until the desired target position is achieved. The G01 code is used for normal linear moves in which the tool is cutting. Therefore, up to three axes may be simultaneously moving if the required cut is either a 2-D or 3-D (compound) angle.

In addition to the above commands, I also manage the M3 and M2 commands, which turn the spindle motor (i.e., the cutter) on and off, respectively. The last command I handle is the F command, which sets the feed rate for cutting operations.

The host controller firmware is given a G-code file to run after the operator picks one out of the directory list for the inserted SD card. It then parses that file one line at a time, loading up the proper x, y, and z coordinates, feed rate, and so forth. After it finds either a rapid or a linear interpolation motion command, it initiates motion from one to three axes, as dictated by the parameters it has just encountered. I’ll describe how this is accomplished in the next section.

MOTION-CONTROL ELECTRONICS

While I was committed to using a microcontroller for the project, I still spent a lot of time trying to choose the best one for the job (with the criterion that I wanted to use a microcontroller in a DIP package because it would be easier to handle with the custom PCB I would design). Since I knew it was going to be a significant programming effort, I decided to use Atmel AVR microcontrollers so I could write the program in MCS Electronics’s BASCOM-AVR, the language with which I have the most experience.

Admittedly, this severely limited my choices. It eliminated many sophisticated microcontrollers that could have handled all the tasks, including the timing-critical motion-control algorithms for three axes. However, it didn’t seem reasonable to use an unfamiliar microcontroller to tackle a complex project such as this. After a lot of analysis, I decided the only way to achieve the critical timing needed for three-axis motion control was to dedicate an individual (i.e., slave) microcontroller for each axis. In addition to the three axis controllers, I would use a host microcontroller to read the G-code file, interpret it into motion commands for the three axes, and dispatch the tasks to the individual axis controllers. The host microcontroller would also be responsible for handling the various manual motion commands the user would enter while setting up the machine for the actual CNC milling job. It was cost effective to use the dedicated microcontroller approach. The slave axis controllers each use an Atmel ATMega88 and the host controller is an Atmel ATmega1284. The total cost of the four microcontrollers is about $15, which compares favorably to using a single, more complex microcontroller.

Another design choice I made early on involved the actual user interface. I needed some type of LCD screen to handle the various required functions. I could have chosen an inexpensive, alphanumeric LCD panel, but I decided to include a 4D Systems µLCD-32PT 3.2” thin film transistor (TFT) color touchscreen display with graphics capability. Using a touchscreen display meant I could eliminate many of the switches, as well as the numeric keypad that would normally be needed in this type of unit. TFT touchscreen displays are becoming inexpensive, while good-quality switches and numeric keypads are becoming more expensive.

Although it was tempting to display the three axis positions on the TFT display, this would have been less than satisfactory, as they would have been too small to comfortably read while using the machine itself. Therefore, I included a six-digit red
LED display for each axis, which is visible from many feet away.

The last major design choice involved the type of media I would use for the G-code files. There were two basic choices: USB flash drives or SD memory cards. I chose the latter as it is easy to interface SD cards to AVR microcontrollers using MCS Electronics’s AVR-DOS and the microcontroller’s SPI port.

While I have used Future Technology Devices International’s VDrive2 Vinculum modules to interface to USB flash drives, they use a UART port and my host microcontroller only had two UART ports, both of which were allocated to other devices. (Note: The VDrive2 modules also contain a SPI port, but it uses a nonstandard protocol that I haven’t yet used successfully.) On the PC side, there are small, inexpensive memory card adapters available that enable you to read and write many different memory cards via the PC’s USB port. Figure 1 is a block diagram of the complete CNC controller showing the host and one of the three-axis controllers.

BASIC CNC MOTION CONTROL

CNC machines use two drive methods: Servomotor/position encoder or stepper motor. Professional CNC machines, particularly larger ones, generally use the servomotor/position encoder method, as it enables much faster axis motion. However, servomotor/position encoder drives are expensive. Stepper motors have markedly dropped in price and they are now much more powerful, for a given frame size, than ever before. I used Automation Technologies’s KL23H276-30-8B stepper motors, which are rated for 282 oz-in of torque and cost less than $40.

I used Geckodrive’s G540 four-axis digital step drive, which contains the high-power drivers for four motors and the specialized digital circuitry needed for microstepping (i.e., the ability to move the stepper motor in steps smaller than the 200 steps per revolution, for which the Keling motors are rated). The Geckodrive drivers are an industry standard for small CNC machines and work well.

For any CNC machine, you must determine how many step pulses are required per inch of linear travel along the axis involved. This is determined by the formula:

\[ \text{Pulses per inch of linear travel} = \frac{\text{Motor Steps}}{\text{Revolution}} \times \frac{\text{Microstep Ratio}}{\text{Mechanical Ratio}} \]

In my case, the motor steps are 200 per revolution. The G540 uses a microstep ratio of 10. You feed it 10 pulses for every discrete step of the stepper and the motor makes 10 microsteps for every 1.8º physical step (i.e., 360º for every 200 motor steps).

In my CNC router, the stepper motors’ rotary motion is converted into linear motion using an Acme lead screw and an anti-backlash lead nut. Acme lead screws are commonly sold with 10 threads per inch (TPI). This would mean that the screw provided a linear motion of 0.1” per revolution.

However, there is another parameter called “start,” which can be one, two, or five. A one-start Acme lead screw would most resemble the standard machine screw you are familiar with (i.e., there is one continuous thread running the length of the screw). A two-start Acme lead screw, on the other hand, would have two discrete threads, each one of which would start at 180º from the other (if you were to look at a screw’s cross-section), and the thread’s pitch would be 0.5 that of a one-start screw. A five-start screw would have five discrete threads, so a 10-TPI screw would, in this particular case, move 0.2” per turn of the screw. You would use a five-start Acme lead screw for higher speeds, or a one-start screw for more resolution. I chose the latter, so my mechanical ratio was 10 turns per inch of linear travel.

Therefore, using the preceding formula, I needed 20,000 pulses per inch of linear motion. Because the stepper motor is only capable of turning a couple of revolutions per second, under load, my machine is capable of less than 1” per second of linear motion. But that is sufficient for my tasks (e.g., cutting wood, plastic, and relatively thin aluminum cabinets and panels).

In my microcontroller-based CNC-controller design, I aimed for a 25,000 per second maximum pulse rate, although this is somewhat higher than needed for my motor and lead screw setup.

To achieve good cutting results, the cutting tool’s feed rate must be tailored to several variables

---

*Figure 1*—This CNC controller block diagram includes an Atmel ATmega1284 host controller, a 4D Systems µLCD-32PT touchscreen display, and Micrel MIC5821 and MIC5891 latched drivers, which are used to drive the LED displays. The Geckodrive G540 shown in blue is a separate commercial unit.

*Figure 2*—Stepper motors can run at modest speeds, but they cannot accelerate or decelerate instantaneously. Therefore, it is customary to use a trapezoidal stepping profile to accommodate this limitation.
(e.g., the type and thickness of the material being machined, and the cutting tool’s rotational speed). There is a big difference in the feed rate you can use when cutting a groove in a thin piece of pine compared to a thick piece of steel. Because of this, it is important that you can vary the cutting tool’s feed rate in each of the three axes. However, for the above reason alone, it is not important that an exact feed rate be achieved. You must achieve this feed rate approximately.

In practice, you must be able to cut at any angle and cut a curved profile. To do so, you must specify a feed rate for each axis that has a precisely defined ratio with respect to the other two axes. If not, the angle cut will be wrong or the curve will be incorrect. For this reason, it is essential that the pulse rate produced by the controller is accurate and of high resolution. While a PC containing a CPU with a clock rate in the hundreds (or thousands) of megahertz can accomplish this for three axes simultaneously, I found it was also possible to achieve this using one low-cost microcontroller per axis.

While it has not been previously mentioned, it is not possible to run a stepper motor at anything other than really slow speeds, unless you gradually ramp up the speed at the start of a programmed move, and then gradually ramp down as you approach the end of that move. This is most commonly done using a trapezoidal stepping profile, as shown in Figure 2. Therefore, in addition to requiring an accurate, high-resolution source of stepper-motor pulses, those pulses must be delivered with up and down ramps, as shown in Figure 2.

In Part 2 of this article series, I’ll describe the design and provide details about the design’s two functional blocks: The axis controller block (powered by three small microcontrollers) and the host controller block (powered by a more powerful microcontroller).

Brian Millier (bmillier1@gmail.com) runs Computer Interface Consultants. He was an instrumentation engineer in the Department of Chemistry at Dalhousie University (Halifax, Canada) for 29 years.

RESOURCES

SOURCES
μLCD-32PT Touchscreen display
4D Systems, Ltd. | [www.4dsystems.com.au](http://www.4dsystems.com.au)

Mach3 CNC Control software
ArtSoft USA | [www.machsupport.com](http://www.machsupport.com)

ATmega88 and ATmega1284 Microcontrollers
Atmel Corp. | [www.atmel.com](http://www.atmel.com)

KL23H276-30-8B Stepper motor
Automation Technologies, Inc. | [www.automationtechnologies.com](http://www.automationtechnologies.com)

VDrive2 Vinculum modules
Future Technology Devices International, Ltd. | [www.ftdichip.com](http://www.ftdichip.com)

G540 Four-axis digital step drive
Geckodrive, Inc. | [www.geckodrive.com](http://www.geckodrive.com)

BASCOM-AVR and AVR-DOS
MCS Electronics | [www.mcselec.com](http://www.mcselec.com)

MIC5821 and MIC5891 Latched drivers
Micrel, Inc. | [www.micrel.com](http://www.micrel.com)

**TRACE32**®
Always one step ahead

Visit us at booth 2006!