# DSPs in Teaching Embedded Systems

Dr. Anita M. Flynn
MicroPropulsion Corp., Berkeley, CA
aflynn@micropropulsion.com
Visiting Professor, UC Berkeley Mechanical Engineering, '99-00

## INTRODUCTION

This past academic year I taught two courses in Mechanical Engineering at UC Berkeley, in which I incorporated DSPs.

The Fall '99 course was ME102B – Senior Design. All mechanical engineering undergraduate programs in the US must now have a senior design experience/course for accreditation. Prior to last Fall, ME102B was entirely mechanism design, and some semesters, it was just a paper exercise. The department wanted to upgrade ME102B into a year-long "Smart Machines" design sequence, which would teach the senior class not only mechanical design but also microprocessor control of their machines. The Fall '99 semester was a first attempt at incorporating electronic and software components into the teaching of mechanical design at the undergraduate level.

The Spring '00 course was a graduate course which I put together called Design of Embedded Systems, ME235. Here, I went "under the hood" and taught more electronics and less mechanics, giving the students a complete foundation in digital electronics using FPGAs, quite a bit of analog and power electronics, and also extensive exercises in real-time control using interrupts.

Both courses had very large final projects (actually, starting nearly from day one) in which the students had to dream up their own inventions and then go off and implement them. Also, both courses had final shows, or "Open Houses" which lasted six hours, during which the public could wander around to various stations to talk to the students, see their projects and witness live demonstrations of the projects working.

Neither course actually focused much on DSP as it is traditionally taught. Also, neither course had a written final report requirement. However, the students in ME235 were required to "hand in" a web page, complete with videos of working projects, schematics, layouts, and code in lieu of a paper report. The project web pages can all be found from the course URL:
http://www.me.berkeley.edu/ME235

## DSPs IN SENIOR DESIGN

I co-taught the Fall '99 ME102B course with Prof. H. Kazerooni, who had been in charge of the course for a number of years. We taught one section of ME102B with this new material, roughly 45 students. Our goal for this new course was to add material on microcontrollers to give the students exposure to computer-controlled electro-mechanical systems. We taught bearings

and pulleys, and brushed/brushless motors as usual, but had to cut out some of the traditional gear tooth design, etc., in order to incorporate the basics of digital controllers, A/D converters and power electronics for DC-DC converters and motor drives. When the course becomes a year-long sequence, more of the mechanical design will be put back in. Nevertheless, all the students had to go through some basic machine shop training, make 3-D drawings of their designs using SolidWorks or IronCAD, and then go into the shop and build their mechanisms before they added their controllers.

Where do DSPs come in? Essentially, I was looking for some decent software and found it at a TI booth at a trade show in Silicon Valley. As a staff scientist and graduate student at the MIT Mobile Robotics Lab, I had spent years debugging embedded systems and interrupt-laden assembly language code with one blinking LED. That type of debugging just wasn't going to be viable if we were to be successful in pulling together this new course.

In the Fall of the previous year, around September '98, I had seen a demonstration of Code Composer Studio running its Execution Graph window. A picture is worth a thousand words and that one real-time graph of interrupt processes running would be a perfect teaching tool, I thought. Interrupts are hard to explain to someone who doesn't know anything about microcontrollers – and they're even harder to debug.

I did some consulting later that year using the 6201 EVM and when I took the Visiting Professor position at Berkeley the next Fall, I went to TI's DSP Fest to see if they might have some development kits that would be useful for students doing embedded projects. While an EVM could theoretically run outside of a PCI slot, the emulator cable, at $4000, was far too pricey to be viable in a classroom situation. The TI '24x series of processors were small, cheap and some third-party '24x boards could be embedded – but Code Composer Studio didn't run on the '24x processors. Interestingly, after badgering enough TI folks about this dilemma, I found that the 6211 DSK would shortly be out with a parallel port emulator cable – but not in time for the Fall '99 ME102B course. What I did find was DMC Pro for the '24x by Technosoft, a TI third party member. Technosoft also sold small '24x boards which did not require an expensive emulator cable but could instead be downloaded via a serial port. Technosoft's DMC Pro software, while not able to communicate debugging information in real-time, could log data to internal RAM in real-time and then let you play it back later. DMC Pro came complete with graphing utilities for data and it looked like a fantastic teaching tool. We went with Technosoft, their MCK243 boards and DMC Pro for ME102B.

Our experience with Technosoft and their hardware and software was great. They provided excellent support, and plenty of code examples. One piece of technology was still missing however. We needed some power electronics for driving motors. A commercial drive for a motor could cost several hundred dollars, clearly not feasible for a class of 45 students. Fortunately, Frank Cheung, a Cal alum and former student of mine from a class I had taught in Electrical Engineering a few years earlier, offered

to design and lay out some dual, 7 amp, H-bridge drivers for our course. We had 20 copies of these boards fabricated.

The students' final projects, mostly done in groups of three, spanned a wide spectrum of problems critical in the lives of 21 year-olds. One was a machine for putting golf balls on a tee automatically so the golfer would not be required to bend down. (Tap your club near the tee and the machine puts the ball on the grass for an iron shot.) Another was an automatic transmission for a bicycle so the cyclist would not have to pull the gearshift lever by hand. Another project was a gecko feeder – so the owner would not be forced to pick up cockroaches individually and put them in the gecko cage. Clearly, the world will be a better place when this group hits industry…

A number of projects involved small mobile robots, either wheeled or walking. One was a vacuum cleaner, one was a sweeper, one was a wet-spot detector, one was a miniature walker and one was a cat toy. Another project was a canon that could rotate and tilt to fire marbles autonomously at a infrared beacon acting as a moving target. A few projects worked at higher power levels and required the students to build custom power electronics. Two of these projects included one force amplifier based on an RC car's internal combustion engine linked to an electronic clutch, while the other was an electronic timing valve for a engine based on a custom solenoid design.

A couple of problems stand out from that first experimental semester. First, the students didn't know C, and we didn't have time to teach it. They had to just figure it out. Most students had just one prior programming course, which was based on MATLAB. The second problem was that 14 out of the 15 groups blew up an H-bridge board at least once. Interestingly, once the students figured out C, they didn't have too many problems with the DSPs and the software. The '24x DSPs are basically microcontrollers – they have on-board A/Ds and PWM outputs. We gave the students functions for reading the A/Ds and for outputting PWM frequencies, which was all they needed. They also never had to use interrupts in any of their projects. It would have been much better however if the '24x processors had more PWM channels and more timers. These chips are geared towards single-motor applications such as washing machines and are not quite the right fit for robotics projects. However, the DMC Pro software was excellent and helped the students get through bugs quickly. They could also look at their sensor data directly - and that saves so much debugging time.

## DSPs in Embedded Systems

The Spring '00 ME235 course, Design of Embedded Systems, had a different goal than the Senior Design class. Because the '24x implementation was not quite the right hardware or software fit for ME102B, it seemed like it would be useful to teach students how to interface their own peripherals to a processor that didn't have on-board A/Ds or PWMs, but might have useful software tools or other features. In the end, the speed of the processor is not important for most of these types of applications – but the software environment and debugging tools are.

By the time one has all the programming done, there will probably be a faster version of the same processor out anyway. It is more important to account for all the time and money that is invested in software development. A software environment that will span future generations of processors, plus debugging features targeted specifically for real-time applications are the most critical concerns.

The 6211 DSK, which could run the Code Composer Studio software, became available just in time for the Spring '00 semester (the 5402 DSK came out later). It could run stand alone, rather than in a PC chassis, due to an on-board interface for JTAG control. That feature made the boards cheap enough for TI to be willing to donate lots of DSKs, and the students would be able to embed these boards in their projects and run them from the on-board flash. Furthermore, while earlier DSKs had to be programmed in assembly language, the 6211 DSKs could be programmed in C, which seemed essential for a one-semester course.

However, the 6211 DSPs did not have the on-chip A/Ds that the '24x DSPs had, nor any provision for motor control. I decided that I would teach the students how to interface their own A/Ds and how to memory map their own digital I/O using an FPGA. That way, they could have as many timers and PWM generators as they wanted. Because the 6211 DSK has a pair of connectors for expansion of peripherals onto a daughterboard, I centered the final projects around building a daughtercard containing a Xilinx FPGA and interfacing that daughtercard to the DSK.

Twenty five students took the course, 21 graduate students and 4 seniors. Most were mechanical engineers, but two of the undergraduates were from Materials Science and Engineering Physics respectively. One graduate student had an MD and had completed his surgical training but was going back to school to pick up a master's in electrical engineering. Another graduate student was from Bio-engineering. Most of the mechanical engineering graduate students were first or second year students working on their master's degrees. Two were starting on PhD theses. All of the students were warned that the course would not only be a course on real-time control, but also would be a real-time sink.

If I had been teaching this course in Electrical Engineering, I might have expected the students to have some background in digital logic, as most of the juniors in EECS take CS150 – Components and Design Techniques for Digital Systems. CS150 spans digital systems from MOSFETs, through gates and combinational logic, to finite state machines and some introduction to computer architecture. It's another time-sink course, in that the course has 7 labs, 12 homeworks and a major final project using a Xilinx FPGA (Spring '00 they built a MIDI synthesizer for an electronic keyboard).

However, none of the students in my class had this background and even if they wanted to, they could not have taken CS150 because it is always impacted. I had never actually used an FPGA either, but had done quite a bit of digital design and computer architecture. After discussions with Prof. Wawrzynek,

who was teaching EECS150, I decided to copy CS150 exactly and make the first half of my course the same as CS150. He had all his lectures, problem sets and labs online at:
http://www.eecs.berkeley.edu/~cs150

I went to his lectures, then regurgitated them to my class. I even used his midterm. I set up the same labs, borrowing 10 Xilinx 4005E FPGA evaluation boards from EECS and getting another 10 donated by Xilinx. We had our own lab in Mechanical Engineering with scopes, power supplies, etc., but no logic analyzers. I borrowed 5 large logic analyzers and 10 handhelds from EECS and got Hewlett-Packard to donate another 5 logic analyzers.

In the first half of the course, we followed along CS150, but I also incorporated quite a bit of analog and power electronics in the first two weeks before CS150 got under full steam (CS150 had 400 students). In addition, I made my students, in a group class effort, learn Orcad and lay out and ship a daughtercard for the 6211 DSK by the end of the first week of the term. Hence, two weeks later, we had 20 daughtercards that contained a socket for a Xilinx 4005E FPGA, along with plenty of prototyping space for students to add their own A/Ds, motor drivers, connectors, etc.

In the second half of the semester after they now knew everything there was to know about basic logic design, I veered off from the direction CS150 was going (towards VLSI design and internals of computer architecture, pipelining, etc.) and went into memory-mapped I/O and interfacing peripherals to a microprocessor. I went over computer architecture at the level of the instruction set and the programmer's model of a DSP vs. a general purpose microprocessor.

At this point, I then went into real-time software, interrupt service routines, booting from FLASH, software interrupts, multi-tasking and variations of schedulers. I had taken two of the TI workshops – one on the 6211 DSK and the other on DSP/BIOS. Each came with a copy of the lecture notes and a set of software labs. We had done 5 of the CS150 labs and so for the next 4 weeks, I had the students do four of the TI labs. These covered DSP/BIOS's real-time debugging tools and also some digital filtering of voice signals.

However, the real emphasis from day one was the final project. I had students work in teams of two, although three projects ended up being single-person projects. Although the students in my class did not have a background in digital logic per se, most had a very deep background in control theory and math. I could speak about FFTs, frequency responses, etc. and they were comfortable with that. Two students had even taken the senior level EECS course in digital signal processing. All had much more programming experience and general debugging experience than the seniors in the previous term's ME102B course. One thing that helped was that many of the mechanical engineering graduate students had taken ME230 – Real-time Applications of Microcomputers, the previous term which used C++ and the VentureCom real-time extension to NT to learn about schedulers, multi-tasking and structured programming. It was soft real-time and

all hardware was abstracted away into black boxes. Nevertheless, with that background, after going through the DSP/BIOS labs, the students were able to do a tremendous amount of debugging on their own in their projects in my course. I spent a lot of time in the lab helping them with some of the nastier bugs, but the Code Composer Studio tools were an incredible help. The Xilinx Foundation series software was also indispensable in pulling this all together. In addition, three groups went further with Orcad and laid out complete custom boards for their final projects, either because they had surface mount components, or they wanted to add megabytes of extra memory, or because they wanted a sturdier setup for their projects, which were also part of their master's theses. Most of the groups though, used the original daughtercards we had fabricated in the first week of the term, and simply wire-wrapped additional components.

I was very impressed with the level of sophistication of the projects the students were able to reach in one semester. Certainly, the tools available today allow them to go much further than what I was able years ago when I was in their position. Again, the students thought up their own projects. All used the 6211 DSK and the Xilinx 4005E FPGA. Some projects used the host-port interface on the DSK to bring output up to the PC's monitor, while other projects ran stand alone. Beyond that, students found their own A/Ds or other components required for their projects.

One project was an MP3 Studio – essentially a digital music mixer which could handle four channels of music, mix them and store the results in FLASH for MP3 translation. Another project was a data glove where a person would move their fingers and the system would recognize the various finger positions, printing out corresponding letters on the PC's monitor. In a similar vein, another project measured the height of a typist's wrists, for training purposes, to prevent carpal tunnel syndrome.

One group worked on an autonomous helicopter project that was part of a research effort. They redid all the electronics from an earlier version of the autonomous helicopter, porting the sensors and actuators from an embedded PC to a 6701 board from TI third-party member, D.SignT. They interfaced a digital compass, an eight-channel A/D for an inertial measurement unit and a radio modem, then added an extended Kalhman filter to generate a more precise estimate of the helicopter position, velocity, etc.

Yet another project involved internet-enabled devices controlled by the electrical activity in a person's muscles. By flexing either the left or right arm, the operator could turn a camera mounted on a pan-tilt head, which was located two computers away on the internet. The intent is for a paraplegic or otherwise disabled person to be able to control appliances using whatever muscles they still have. Along those lines, another group built a smart wheelchair, designed to be cheap and for use in developing countries. Consequently, they built their own motor controllers and custom analog electronics.

Another project focused primarily on analog circuitry – a smart

battery charger for the UC Berkeley Solar Car club. This was a high-voltage DC-DC converter, completely isolated, with analog control loops for over-voltage protection, short-circuit protection, etc.

Two groups (rather, two single-person projects) worked on force feedback devices. One used a joystick donated from Immersion Corp. that had two potentiometers and two motors inside, to create a 3-D flight simulator video game where the joystick would push back against the player's hand whenever they crashed their plane into an obstacle. The other force feedback device was part of a master's project that used a more kinematically complicated three degree-of-freedom joystick, built at NASA, to create a sensation of feeling the edges of a virtual sphere – both inside and out.

The sole mobile robot project was an RC car that the students modified with custom power electronics and to which they added their own voice recognition algorithm – essentially an FFT and some filtering to parse three verbal commands and a whistle, which made the car go forward, turn, speed up and stop.

Still another project included an airplane wing roll controller. An optical encoder measured the roll of the model plane about an axis to which it was mounted, a fan acted as a wind tunnel, and servo motors actuated flaps on each of the wings to keep the plane level despite disturbances.

Another group was interested in building their own digital camera from scratch. They bought a CMOS image sensor chip from Omnivision and interfaced it to the DMA on the 6211 DSK and created a motion sensor for security applications – that worked in the dark.

Finally, one group of budding rock stars created Air Drummer – a virtual drumset in the same vein as Air Guitar. An accelerometer in the drummer's hand, a bend sensor mounted inside one elbow and touch switches mounted on each foot produced signals that they encoded into different percussion instruments with different beats and volumes – then played back corresponding WAV files over a set of speakers connected to the 6211 DSK's codec.

All together the workload for the course encompassed 8 labs, 8 problem sets and these final projects – along with lots and lots of all-nighters. The course was only allocated one teaching assistant and so I did not have him bother to grade homeworks, but rather had him focus on getting the labs up and running and then getting the students through the labs. I spent quite a lot of time and personal attention in the lab with each group in the hopes of making as many projects as possible successful. Twelve out of fourteen final projects worked for the full 6-hour Open House.

## FEEDBACK

We found some silicon bugs in the TI hardware and ran into a few software glitches, but TI tech support was exceptional. One bit of feedback for TI - almost all the students picked Analog Devices A/Ds, because they could get free samples from Analog Devices' web page without having to

make any phone calls. The price point for the DSKs, at $195, was low enough that three students bought them for their personal use. Recently however, the DSKs have doubled in price.

Also, if the Active X technology in Code Composer Studio was better documented or if a simple MATLAB interface was available, that would make quite a lot of headway in the educational realm.

Finally, if TI had an extremely cheap, even low-end processor that could run Code Composer Studio with all of its beautiful real-time debugging tools, and if the processor was cheap enough for toys and had plenty of PWM generators, the technology could be brought down to the high-school and hobbyist level for mass-market education.

In that case, I can imagine a new low cost DSK which would contain an image sensor, an audio codec, a programmable logic chip, and some integrated H-bridges that would teach young people not only about robotics, but about the signal processing technologies that go into creating higher fidelity perception for the next generation of artificial creatures.

I imagine having a plug-in for Code Composer Studio that would contain a set of lessons and demonstrations of digital signal processing and real-time control techniques. I can also imagine creating a web page where hobbyists could go to download new files for the programmable logic chip if they wanted more timers, PWM drivers, etc. to build a twelve degree-of-freedom walking robot, for instance. There could also be add-on radio modules to buy, and programs to download for the DSP that would let the board communicate with the broadband in-home wireless networks that will soon be ubiquitous. Then the robot could be controlled remotely from a kid's browser.

Almost all of the microprocessor based toys today use a 4-bit Sun Plus processor which costs 15 cents. Next generation toys, however, will be much more sophisticated. If the toy technology was based on a processor platform which supported a Code Composer Studio-like real-time software environment, then there would be a tremendous opportunity for education.

In the courses I taught this past year, I took advantage of the declining costs of DSPs due to the communications industry. However, the toy industry similarly pushes costs down and the sensor/actuator interfaces for a toy-focused DSP would be a better fit for these classes in embedded systems and real-time control.

After all, one of TI's first DSP products was the Speak 'n Spell toy. It will be interesting to see what the next generation of DSPs will bring.