**EE 247 Programming for Embedded Applications Spring, 2020**

Text: (suggested, but not required) Fabio Pereira, *HCS08 Unleashed, Designer's Guide to the HCS08 Microcontrollers*, 2nd ed., 2009, ISBN 1-4196-8592-9
MC9HCS08SH8 Manual, NXP / Freescale Corp. (available online or as download)
Scheduled class times: Monday & Wednesday 10:00-11:20 SLC238 (SLC193?)
Instructor: John B. Gilmer Jr. Office hours: TBD Office: SLC220 Phone: 4885

This course is intended to provide practical programming experience applicable to embedded systems. Embedded systems are small scale computer systems that are part of a larger component or system. For example, a microwave oven or an automobile engine may use a small computer to control the system and interact with the user or other systems. Embedded programming is done to configure small computers to do particular jobs, like operate the microwave oven and its display, or to control the spark timing and other settings for an automobile engine. These computers are typically microcontrollers, with either no operating system or a specialized real-time operating system. In this context programs interact closely with hardware, in contrast with larger systems in which hardware interaction is typically mediated by the operating system. The typical personal computer has much more elaborate operating system supporting much more complex programming abstractions than are used in small embedded systems.

So, programming for embedded systems, at least at the smaller end, is characterized by:
1. The computer is small and inexpensive, often less than a dollar in consumer electronics.
2. The computer resources, especially memory, are very limited: perhaps only 4K bytes.
3. Because of small memory, simple programming languages and constructs are used.
4. The user interface, if there is one at all, is simple: maybe just pushbuttons and LEDs.
5. Timing is important because the computer must directly manipulate real physical things.
6. Programming is done using a different computer and downloading the code to the system.

This Programming for Embedded Applications course will emphasize the sort of small, even tiny, applications that are embedded in consumer goods, weapons, and industrial components. There are other embedded applications, such as a computer that controls a high performance fighter aircraft, for which the resource constraints may not be as tight and the cost may reach the hundreds of thousands of dollars. Yes, you can do embedded systems work with small personal computers running Unix or Windows, and using Labview or similar software and interface technology. Such embedded computers would cost hundreds to thousands, and would be appropriate for systems starting in the tens of thousands of dollars. But, the major growth in embedded systems, and the largest number of systems, will come from new applications for which including a computer is not practical now. Or perhaps the system and its capability does not yet exist because it has not yet been invented. Ask yourself what might be done if computation is very cheap and very small. Consider the cellular telephone and GPS units. These things are possible only because of small, very cheap computers. They cost tens to hundreds of dollars. What things could be possible if the price of a system with a computer in it is only a dollar to ten dollars? Or less? The imagination is the limit.

This course is intended to give the student a basic understanding of that context and the nature of what small embedded computers can do. It is an introductory level technical course, so it will include basic coverage of programming in the most popular computer language used for embedded applications: "C". But, since so much is going on at the hardware level, we will also be looking at what is going on in machine / assembly language, the actual code in binary that is operating on the computer. We won't normally need to write code in assembly, but we need to understand it. How the machine executes code is tied tightly to the small memory and speed issues. Since that's how the computer actually operates, we need to understand that. "C" is

merely a more convenient way to produce what we want to happen in machine language. (This is exactly the opposite of the usual "computer science" view, where you abstract things into more complex objects so you don't have to think about what's going on at the machine level.)

EGR247 Programming for Embedded Applications is a course expected to be a regular (every other year) technical elective. Some of it overlaps material introduced in EGR222 Mechatronics. We will be going into a lot more detail before that subject even comes up in Mechatronics. This course a prerequisite (one of a few options) for EE342, which has been revised to focus more on applications and more sophisticated use of peripherals.

This course assumes that the student has some familiarity with computer based algorithms, control, and storage. That familiarity should have been gained in either EGR140 (programming and representation of data in MATLAB, C or Python), or in CS125 Computer Science I, or equivalent level programming experience.

The schedule below is preliminary. I plan to cover the topics below, but we will have to feel our way along and make sure the course does not move too fast or too slow. I will bring some peripherals into the early weeks (especially the serial port) because we need a way to see our program executing, for example. I may cover an introduction to C++ before the end, but we will have to wait and see.

| Week | Date | Topics covered | Tests |
|---|---|---|---|
| 1 | Jan 13,25 | Representation of data in memory: constants and variables | |
| 2 | Jan 22 | Microcontroller structures and programming context | |
| 3 | Jan 27,28 | Microcontroller structures and programming context (continued) | |
| 4 | Feb 3,5 | Basic program control structures in C: branching and loops | |
| 5 | Feb 10,12 | Basic program control structures in C: branching and loops (cont.) | test #1 |
| 6 | Feb 17,19 | Machine language basics: addresses, instructions, registers | |
| 7 | Feb 24,26 | Machine language basics: addresses, instructions, registers (cont.) | |
| 8 | Mar 9,11 | Mapping of C statements to machine language, debugging | test#2 |
| 9 | Mar 16,18 | Subroutines, Functions and the Stack | |
| 10 | Mar 23,25 | Subroutines, Functions and the Stack(continued) | |
| 11 | Mar 30, Apr 1 | Program structure for embedded applications | |
| 12 | Apr 6,8 | Peripheral interfaces | |
| 13 | Apr 13,15 | Interrupts | test#3 |
| 14 | Apr 20,22 | Driving other (selected) peripherals – IIC etc. | |
| 15 | Apr 29 | Grander things: USB, interfaces, and more | |
| | | Final Examination | exam |

The schedule does not show reading assignments. These will be selected chapters and sections from the Microcontroller manual. Paper copies of some of this material will be made as we need to, when we get to various topics. Generally, you will know the topic and will dig out what you need to know by diving into the various sources of information. This is closer to what happens in industry. The Pereiro book is really more of a "getting started with these microcontrollers" reference, rather than a textbook in the usual sense. (That makes it cheaper!) There are additional support materials such as one or two old (somewhat out of date) Metrowerks Code Warrior getting-started books, and a copy of the Kernigan & Ritchie C book. I will put some of these materials in the laboratory for reference.

**Grading:**

Tests will cover all material through the previous week. Tests will normally be on Wednesday, unless an announcement is made setting the date differently. The test schedule is very tentative. We can move them if need be, if there are other important tests the same day affecting most of the class. Tests will be open book, open notes, use of calculators permitted. I

expect to have you at the computer, able to use your project files and other reference material, such as the Microcontroller manual. You still need to be well prepared.

Grading Allocation:

| | | | |
|---|---|---|---|
| 3 tests at 15% each : | 45% | Final examination: | 32% |
| 4 programming projects 5% each: | 20% | Class Participation: | 3% |

All material will be graded on a basis of 0-100, with most graded material allowing for grades higher than 100 with bonus questions (usually up to 10% extra) considered. On tests and the examinations some questions may be "compensated" if large numbers of students miss them (indicating possibly a badly posed question or inadequate coverage of the topic in class). On such questions, some proportion of the "lost" credit will be returned. This is the only form of "curving" of grades in the course. All written work is expected to be neat and well presented. A penalty of up to 20% will be assessed for poor presentation on any written work. The grades from all work will be weighted as given in the above table, totaled, and converted into the Wilkes 4.0 scale grading system using the following conversion:

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 91+: | 4.0 | 82-86: | 3.0 | 70-76: | 2.0 | 60-65: | 1.0 |
| 87-90: | 3.5 | 77-81: | 2.5 | 66-69: | 1.5 | below 60: | 0.0 |

Students are coming into this class with a great variety of backgrounds. I'm going to do my best to see that all students get something useful from it, which means some of those with more experience will be doing independent advanced projects. That means grading will have to be handled carefully. I intend to be as fair as possible. I will not punish a student for coming in with less background than others, nor for being more advanced.

**Homework and Class Participation:**

Homework will be assigned, and collected. The bulk of your learning will take place as you do the four software projects rather than homework. On the due date, the homework will be collected and a solution set passed out. Questions and discussion concerning the homework will follow. Homework will not be graded. It will be reviewed, and on a selective basis some students or some questions may be examined in detail. The only effect on a student's grade from homework is in the "class participation" category. I will be tracking who does the homework and the degree of seriousness with which it is taken. Along with attendance and tardiness, homework submission will be considered when assigning a number for a subjective 3% of the grade for "class participation." Class participation will also consider attendance and effort.

**Projects:**

The current plan is for each student to do four software projects of increasing sophistication over the course of the semester. (Not partnerships!) For each of these projects you will submit a report that includes text describing what you have done, diagrams (for example, showing register and bit usage), and documented software (code). Documentation of code is very important; undocumented code is almost useless once some time has elapsed and you don't remember what you were doing. I know this from personal experience! Worse, it's very, very difficult to understand someone else's undocumented code. We will be discussing documentation issues throughout the course. The project report is also to include a "Conclusions" section that states whether the project works or not. Generally you will demonstrate correct operation of your project. In the conclusions section of the report, you will state that it was demonstrated to the instructor (or failed to operate correctly, with details given), and give the date. Failure to document its operational status requires that the project and report be graded as if it does not work, in addition to the penalty for omitting this statement. The *Engineering Laboratory Reports Manual* is a useful reference for preparing these reports, although it does not specifically address reports on software-oriented projects.

**Collusion and help on assignments:**

All graded material handed in is to be the student's own work. (For this purpose, the homework exercises are ungraded. Projects are graded.) A student may get help from another student or anyone else in matters of technique or background, but answers and written text **and code** are not to be copied. At all. That means you do not copy and modify (without specific permission and attribution). Your submission should include no code or schematics, etc. directly or indirectly copied from anyone else's material, except for materials supplied to the class or from library materials or other approved and generally available sources. When you do include any kind of copied material (including that from the allowed sources mentioned), you must identify it as copied and identify the source. You must attribute it properly. You are to also attribute help received from classmates. That's a good thing to do, both as a professional courtesy and as good documentation of sources.

Occasionally I will allow and actually expect shared use of some class developed code or other resources. For example, I may allow an interrupt service routine developed by one student to be used by others (with correct attribution). Indeed, learning to work cooperatively in a software project is important, and will reinforce your perception of the value of documentation. There may even be cases where, say, two students develop similar functions or routines and compete to see which gets their product adopted by the most other members of the class. If in doubt about whether authorized (and attributed) use of someone else's code or other materials is allowed, ask. If you do use copied work, and it is properly attributed, I will not consider the use of the copied work a matter of academic dishonesty, but instead, merely having not followed directions to do your own version of whatever it is.

Evidence of inappropriately copied material not given proper attribution may be rewarded by zero grades on the concerned material or more serious sanction if the situation merits such action. If you are in doubt about whether some form of help to or from another student is allowed or not, ask me. If you cannot find me, at the least send e-mail to explain. Whoever receives the help must acknowledge it. It may even be worth some extra credit to the person giving help if the help given is proper and correct. I expect to assign project problems that will have individual variations, so that there should be less problem giving or receiving help.

**Notes:** A loose leaf notebook of class notes may be put in the library on reserve if requested. This will include lecture materials used, worked homework assignments, and other material as I may develop it. You are not obligated to copy any of this; it is merely meant to be helpful. Any material that is really needed will be distributed in the form of handouts in class. When I have put such materials on reserve in the past, they have sometimes never been used, so now I do not do so unless requested.

**Attendance:** Class session attendance is not optional. Missing a week's worth of classes, unless explicitly excused, is reason to award a zero grade for the class. If you expect to miss a class, inform me by email or phone message immediately. If you cannot reach me, in an emergency contact the EE/Physics administrative assistant, Mrs. Nikki Stapleton. In this course, some students who have had earlier coverage of some topics may be excused if requested; I'll be more flexible about this in this class than I normally would be.

**The Microcontroller and board:** We plan to use the Freescale HCS08SH8 microcontroller, mounted on a "DEMO" board. (Freescale is now part of "NXP" Corp.) This is similar to the processor and demo board being used in EGR222 Mechatronics, but it has 20 pins instead of 16 (adding a C port) and can operate on 5 Volts, convenient for integration with TTL. The 'SH8 processor is a 20 pin DIP. The demo board has two user controlled LEDs, two pushbuttons, a pot connected to an A/D port, an RS232 interface, and (most important) debugger support using a USB connection to a PC. The board has a connector that allows you to run wires to additional circuitry. Or, instead, you can put the 20 pin DIP HCS08SH8 on your breadboard and run three wires back to the prototyping board for debugging support. When you remove the debugging connections, it runs stand-alone. Every student will get one of these to use during the semester.

**Code Warrior:**
We will be using the Metrowerks Code Warrior "Integrated Development Environment" (IDE). There is some help for getting started with Code Warrior in the textbook, but it is far from comprehensive. You can get "help" books for Code Warrior but they get out of date rapidly, and are probably not a good investment. You'll learn the most just by poking around and trying things. (Remember to save a backup copy first for anything that's important!)

Here's a potential problem: The version of Code Warrior that ships on the CD with the microcontroller won't run on Windows 7. A version 10.x is available and can be downloaded. Once you have used the DEMO Board with Windows 7 and Code Warrior 10, you can't go back the other way without errors. Not a big deal. Windows 7 does not come with the terminal emulator "Hyperterminal", but we will have installed a better alternative, "PuTTY". I have seen students use CW10 on more recent Windows versions. However, we have been notified that now CW10 does not run on Windows 10, but there is a new CW11 that does. If we are to move to SLC193 (which I believe will be a more convenient venue), those computers are running Windows 10 and we will need to upgrade to CW11. (In fact, on the NXP web site, CW10 doesn't seem to be available anymore – it's CW11.) This is the one potential hold-up to moving the class to SLC193.

Here's something else to be aware of: We are using a DEMO ("special") version of Codewarrior for Microcontrollers. There is a limit on the size of the program that can be written under the demo version. The limit is large enough for any project we could want to do in C. They had to make the C limit large enough to hold the USB support and other bulky stuff that had to be in all the demos for bigger microcontrollers. However, the limit for a project in C++ is MUCH smaller – so small in fact that a CS student who wanted to work in C++ discovered it while doing a relatively simple project- a 4 function calculator. It was not at all obvious what the problem was, and it took a good bit of digging to discover the severe limit on C++ projects. (The limit concerns the number of bytes of memory for the code – something like only 1KB for C++.) So if you are a fan of C++, you will either need to put aside your enthusiasm for that language long enough to do this course in C, or buy your own individual license for the full version. (Don't!) In EE342 we will be moving to a different environment.

**The web site for the S08SH8 microcontroller:**
**http://www.nxp.com/products/automotive-products/microcontrollers-and-processors/8-bit-s08-5.5v-mcus/8-bit-general-purpose-sh-mcus:S08SH** (Yes! This link still works! Under the "Software and Tools" tab you can find: "CodeWarrior® for MCUs (Eclipse IDE) - ColdFire®, 56800/E DSC, Qorivva® 56xx, RS08/S08, S12Z - 11.1" with a button for download options.) After clicking that, you get asked to log in. I have not gone down that path yet myself (lacking a Windows 10 machine).

**Facilities:**
We will meet in SLC238 and then (I expect) SLC193. The latter is a laboratory with Windows 10 computers that we expect to have CodeWarrior 11 installed. Wednesday will normally be our "lab" day, although we may be doing lab stuff on Mondays as well as needed. The SLC 238 lab has computers at the respective workstations running Windows 7, with CodeWarrior 10 installed. There will not be enough computers for everybody in SLC238, and we may have a problem with that in SLC193. I do highly recommend you download CodeWarrior for Microcontrollers 10 from the Freescale web site and put it on a laptop, and maybe even bring it instead of using the lab computers if you want to. At the very least, have your projects on a flash drive you can plug in elsewhere. The CD that comes with the microcontroller demo board is useless for the CW version, but has useful documentation.

If you use your own computer, you will need a serial port. I'm expecting we will obtain USB to serial adapters needed for either your laptop or the SLC193 computers. (The SLC238 computers have serial ports.)

**Looking Forward:**
This class is the prerequisite for EE342, the microcontroller applications course. In that course we will also use NXP microcontrollers, but will shift to using the "Kinetis" 32 bit processors, specifically the KL43Z, and will communicate back to the PC using USB rather than serial ports. We will do some programming of the PC, developing (actually, adapting) a C++ application program that will allow the user to point and click on the PC to send information that the microcontroller acts on, and the microcontroller can send information back that is displayed on the screen. (The KL43Z demo board also has a 4 character LCD display.) We will apply these techniques to controlling an HO toy train set. EE342 is expected to be offered during the Fall 2020 semester.