Transitioning a Microcontroller Course from Assembly Language to C

Steve Menhart, Ph.D.

Dept. of Engineering Technology
University of Arkansas at Little Rock
2801 S. University Ave.
Little Rock, AR 72204

Abstract

This paper describes improvements made to an integrated lecture and laboratory course dealing with microcontrollers, taught in the Engineering Technology Department, at the University of Arkansas at Little Rock (UALR). This course initially used the Motorola 68HC11 microcontroller, but currently uses the Motorola MC9S12DP256B microcontroller. The course was previously taught using assembly language, but has just been updated so that students now program using C. The issues related to upgrading a course such as this from assembly language to C are addressed in this paper.

Introduction

The use of programmable devices has become the paradigm of choice for circuit designers. These devices result in cost-effective, flexible designs. There is an inherent appeal to circuits whose application may be changed via reprogramming, as opposed to a change in hardware components. Complex programmable logic devices (CPLDs) and field programmable gate arrays (FPGAs) may be programmed using a hardware description language such as VHDL. CPLDs and FPGAs provide very cost effective realizations of large combinational and sequential logic circuits. However, a point is reached, particularly when arithmetic or timed operations are involved when a microcontroller is a required. Indeed embedded microcontrollers are now ubiquitous. The applied nature of engineering technology (ET) requires that students in the electronics and computer ET fields be fully competent in the use of programmable devices, and microcontrollers should certainly stand at the pinnacle of such degree programs. In order for students to fully comprehend all aspects of embedded systems it is essential that they gain the hands-on experience of actually programming and interfacing hardware to a microcontroller. Assembly language is a very efficient method of programming small applications, and for those well versed its art, the preferred method. However, industry has steadily moved towards programming microcontrollers using high-level languages, most often using the C language. To program in C, and generate the machine code to download to the microcontroller, a cross-compiler is required.

Every engineering program has to be responsive to the needs of the employers of their students. The baccalaureate Electronics and Computer Engineering Technology program, here at the University of Arkansas at Little Rock has seen an increasing number of requests for us to teach our microcontrollers course using C-programming. As a result the course is now taught in C, although a minimal assembly language component is still included. It should be noted that the department still teaches a parallel microprocessor course sequence, using Intel assembly language.

Hardware

This course uses the Adapt9S12DP256 board, manufactured by Technological Arts[1]. This board uses the Motorola 9S12DP256 chip (a derivative of the 68HC12), which is an extremely feature-packed microcontroller. The 9S12DP256 has 12k of RAM, 4k of EEPROM, 256k of fast flash memory, and two 10-bit, eight-channel analog-to-digital converters, plus many other features, all on-chip. Although capable of interfacing to external memory so as to operate in an expanded mode, it seems very unlikely that this would ever be necessary, given the amount of single-chip mode memory available. All I/O lines and control signals are routed to two 50-pin interface connectors, allowing the full use of all of its numerous ports. Figure 1 shows the Adapt9S12DP256 development board. As can be seen the board is remarkably compact, and designed to be modular and stackable with other types of boards. For student lab use, printed circuit boards have been designed and constructed here at UALR. These boards plug into each of the header strips, with each board having a set of terminal strips mounted to it. This allows students to connect their experiments to the board, with minimized risk of damage to the board. The current text used in the course covers both assembly language and C-programming, as applied to the 68HC12 family[2]. The text supports an integrated hardware/software approach and is a good fit for a lab-intensive course such as this[3].

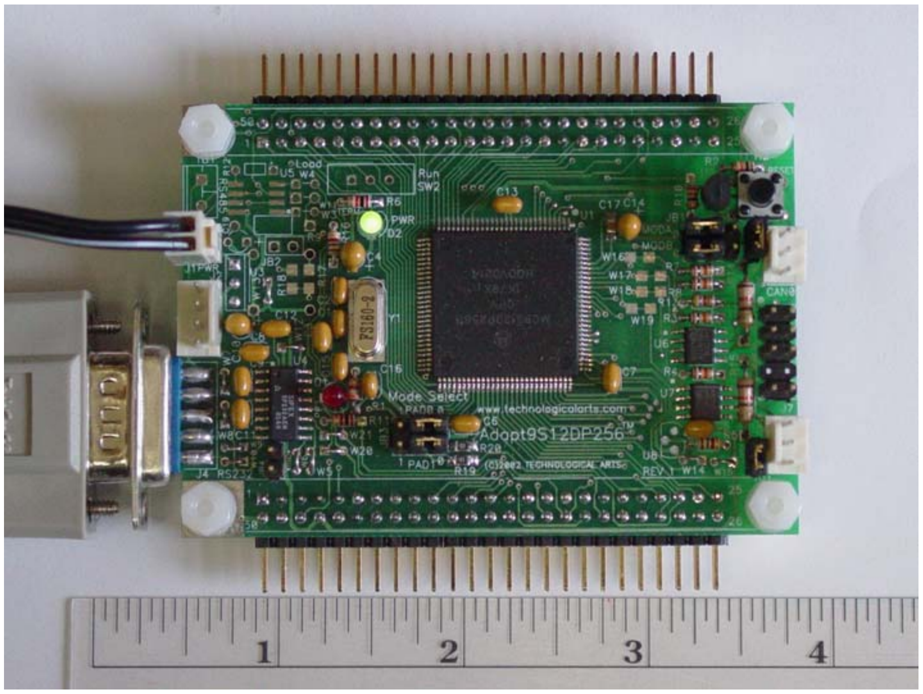

Figure1. The Adapt9S12DP256 development board.

Software and Lab Assignments

One advantage of programming in assembly language is that there are freeware tools available. (This course used to use the miniIDE assembler throughout the course, provided as freeware from MGTEK, which includes a built-in text editor, and integrated build and download features.) A free, usable, cross-compiler for the 68HC12 could not be found. The C cross-compiler selected and purchased is produced by ImageCraft, and lists for $199 for the standard version. This compiler was chosen for the following reasons: it is relatively inexpensive compared to other products on the market; it is supported in the course text; it is intuitive and easy to learn how to use.

When the ImageCraft program is started the user sees a graphical user interface, and several windows. The main window is the editor window, in which C-code is written and saved as a file. This file can be added to a newly created project. The target device (in this case the 9S12) is specified using the project options. When a target is specified the RAM and EEPROM addresses and the stack pointer are also defined. Usually, the project will also contain header files. The software also contains a terminal window, in which D-bug12 commands can be executed, and the status of the microcontroller's registers and memory can be displayed. There is another window that shows the status of any on-going event, such as compiling the code. Any error messages are also shown in this window. After the project has compiled successfully, an S19 file is generated. The S19 file can be downloaded to the 9S12 from within the ImageCraft program via a download button. The program is run by executing the “Go” command from within the terminal window.

One disadvantage of using the ImageCraft software, as compared to the miniIDE, is that after running the program the user cannot communicate with the 9S12 board unless the reset button is pressed. When using the miniIDE and programming in assembly language, control is returned to D-bug12, provided that the program is not of a continuous polling type. One consequence of not having control returned to D-bug12 is that the user cannot view the contents of the microcontroller’s registers, because pressing reset reinitializes them.   (It does not change values in RAM.) It also means that students cannot set user breakpoints or execute the program line by line using the “trace” command. For example, before adopting C programming, the following was a typical first student assignment using assembly language.

```
org $1000        ;place program beginning at address 1000₁₆
ldaa #$aa        ;load register ‘a’ with aa₁₆
ldab #$77        ;load register ‘b’ with 77₁₆
deca             ;decrement ‘a’
decb             ;decrement ‘b’
swi              ;software interrupt
stop             ;stop processing
end
```

Students were instructed to look at registers ‘A’ and ‘B’ before the program was run, and then to run the program and observe the changed data in ‘A’ and ‘B’. Students also executed the program using the “trace” command, and set user breakpoints. These methods were valuable debugging techniques. To get around the problem of not having control returned to D-bug12 when programming in C, printf statements can be used to display any desired information, such as internal register values. The following complete program shows part of the first C-language assignment.

```
#include <stdio.h>
main()
{
unsigned char save; //must define as unsigned char or char
asm("ldaa #100"); //in-line assembly language instructions
asm("deca");
asm("staa %save");  //% is used to access a local variable
printf("Register A = %d",save);//%i will not work
return 0;
}
```

In the short program above a local variable (it is within main) called ‘save’ is defined. The contents of register ‘A’ are assigned to ‘save’, using the in-line assembly instruction staa (store ‘A’). Note the required % operator immediately before ‘save’. The printf statement causes the value of register ‘A’, now stored in ‘save’, to be displayed on the terminal monitor as this part of the program is executed by the microcontroller. The program below is similar, except that the variable is now global (outside of main and accessible to any other functions). Note that in this case the underscore must precede the variable 'save' (used for global variables). In both programs the value of register 'A' will be displayed in a decimal format.

```
#include <stdio.h>
unsigned char save; //global variable
main()
{
asm("ldaa #$aa"); //hex aa=decimal 170
asm("deca");
asm("staa _save"); //underscore is used to access a global variable
printf("Register A = %d",save); //%i will not work
return 0;
}
```

As can be seen inline assembly language code can be inserted into a C-program using asm("….."). Assembly language is required to view the contents of registers. In the short programs above the printf statements are used for debugging purposes only, and would be removed from a finalized embedded application. In the next lab students learn how to control hardware (output port lines) using C. The following short program shows how bit zero of port 'A' may be turned on.

```
#include <stdio.h>
#define PORTA *(unsigned char volatile *)(0x0000)//physical address=0000(hex)
#define DDRA *(unsigned char volatile *)(0x0002)//"0x" prefix indicates hex
main()
{
DDRA=0x0f; //the lower 4 bits of port A are outputs
PORTA|=0x01;//PORTA=(PORTA OR 00000001), therefore the LSB bit = 1
return 0;
}
```

The "# define" statements are macros, which define pointers. For example, PORTA points to the physical address of port A, which is 0000_{16}. DDRA refers to the data direction register of port A. The "PORTA|=0x01" statement performs a logical bitwise OR operation with the current values of the 8 bits of PORTA and 01_{16} (00000001_2). Therefore, only the least significant bit is affected (turned on), all other bits are unchanged. Other bits that may be on remain on, and bits that are off remain off. (This is analogous to a BSET assembly instruction.) If PORTA=0x01 were to be used instead to turn-on the least significant bit, any other bits currently on would be turned off. The 0x prefix is used to indicate a hex value.

At this point in the course students are able to view the contents of any internal registers, and can set the values of any output ports or control registers. Subsequent labs deal with the proper configuration of control registers for specific functions, e.g. the timer, and analog-to-digital units. Students also learn how to interface various devices to the microcontroller including a keypad and an LCD display. The C required to accomplish these tasks is relatively straightforward, and at this point students can progress more rapidly than when the course was taught exclusively with assembly language. Additional references dealing specifically with the application of C for microcontroller programming are included[4,5].

Conclusions

The transition of this course from assembly language to C programming has proved very popular with students. A minimal number of Motorola assembly language instructions are still introduced in the course. Once students understand some basic fundamental concepts, they can realize the full potential of programming a microcontroller using C. Most of our students decide to use the 9S12 microcontroller in their capstone senior design project course, allowing them to tackle larger problems using C. The changes made to this course are primarily a consequence of requests from industry, and it is expected that the marketability of our students will be improved as a result.

References

1. Technological Arts, Toronto, Canada, http://www.technologicalarts.com/.
2. Han-Way Huang, MC68HC12 An Introduction: Software and Hardware Interfacing, Thomson, 2003.
3. S. Menhart, "Updating Microcontroller Laboratory Courses," proceedings of the 38th ASEE Midwest Section Meeting, University of Missouri-Rolla, Missouri, September, 2003.
4. Ted Van Sickle, Programming Microcontrollers in C, LLH Technology Publishing, 2001.
5. Jonathan W. Valvano, Developing Embedded Software in C Using ICC11/ICC12/Hiware, http://www.ece.utexas.edu/~valvano/embed/toc1.htm.

STEVE MENHART
Dr. Menhart joined the faculty of the University of Arkansas at Little Rock in 1989. He currently holds the rank of Professor in the Department of Engineering Technology (ET), teaching in the Electronics and Computer ET baccalaureate degree program. He obtained the Ph.D. and M.S. degrees in Electrical Engineering, from Texas Tech University in 1988 and 1985, respectively.