April 2012

Basic Concept of Embedded 'C': Review

S. C. Agrawal

Department of Electronics Design & Technology, DOEACC Society, Gorakhpur, UP, India,
arushi123@yahoo.com

Sanjay Singh

Department of Electronics design, (ED&T), Gorakhpur UP India, dcsanjaypm@gmail.com

Arun Kumar Gautam

Department of Electronics Design & Technology, DOEACC Society, Gorakhpur, UP, India,
gautam20045@gmail.com

Mahesh Kumar Singh

Dept. of Electronics Design & Technology, DOEACC Society, Gorakhpur, UP, India,
maheshvns@radiffmail.com

Follow this and additional works at: <https://www.interscience.in/ijcsi>

Part of the [Computer Engineering Commons](#), [Information Security Commons](#), and the [Systems and Communications Commons](#)

Recommended Citation

Agrawal, S. C.; Singh, Sanjay; Gautam, Arun Kumar; and Singh, Mahesh Kumar (2012) "Basic Concept of Embedded 'C': Review," *International Journal of Computer Science and Informatics*: Vol. 1 : Iss. 4 , Article 13.

DOL: 10.47893/IJCSI.2012.1055

Available at: <https://www.interscience.in/ijcsi/vol1/iss4/13>

This Article is brought to you for free and open access by the Interscience Journals at Interscience Research Network. It has been accepted for inclusion in International Journal of Computer Science and Informatics by an authorized editor of Interscience Research Network. For more information, please contact sritampatnaik@gmail.com.

Basic Concept of Embedded 'C': Review

S. C. Agrawal, Sanjay Singh, Arun Kumar Gautam & Mahesh Kumar Singh

Department of Electronics Design & Technology, DOEACC Society, Gorakhpur, UP, India

E-mail : arushi123@yahoo.com, dcsanjaypm@gmail.com, gautam20045@gmail.com, maheshvns@radiffmail.com

Abstract - The use of C language to program micro controllers is becoming too common. And most of the time its not easy to build an application in assembly which instead you can make easily in C. So its important that you know C language for micro controller which is commonly known as C remains a very popular language for micro-controller developers due to the code efficiency and reduced overhead and development time. C offers low-level control and is considered more readable than assembly. Many free C compilers are available for a wide variety of development platforms. The compilers are part of an IDEs with ICD support, breakpoints, single-stepping and an assembly window.

Key words - *Embedded C, Microcontroller, Compilers.*

I. INTRODUCTION

An embedded system is some combination of computer hardware and software, either fixed in capability or programmable, that is specifically designed for a particular function. Industrial machines, automobiles, medical equipment, cameras, household appliances, airplanes, vending machines and toys (as well as the more obvious cellular phone and PDA) are among the myriad possible hosts of an embedded system. Embedded systems that are programmable are provided with programming interfaces, and embedded systems programming is a specialized occupation. Certain operating systems or language platforms are tailored for the embedded market, such as Embedded Java and Windows XP Embedded. However, some low-end consumer products use very inexpensive microprocessors and limited storage, with the application and operating system both part of a single program. The program is written permanently into the system's memory in this case, rather than being loaded into RAM (random access memory) like programs on a personal computer.

Most C programmers are spoiled because they program in environments where not only is there a standard library implementation, but there are frequently a number of other libraries available for use. The cold fact is, that in embedded systems, there rarely are many of the libraries that programmers have grown used to, but occasionally an embedded system might not have a complete standard library, if there is a standard library at all. Few embedded systems have capability for dynamic linking, so if standard library functions are to be

available at all, they often need to be directly linked into the executable. Oftentimes, because of space concerns, it is not possible to link in an entire library file, and programmers are often forced to "brew their own" standard c library implementations if they want to use them at all. While some libraries are bulky and not well suited for use on microcontrollers, many development systems still include the standard libraries which are the most common for C programmers.

C remains a very popular language for micro-controller developers due to the code efficiency and reduced overhead and development time. C offers low-level control and is considered more readable than assembly. Many free C compilers are available for a wide variety of development platforms. The compilers are part of an IDEs with ICD support, breakpoints, single-stepping and an assembly window. The performance of C compilers has improved considerably in recent years, and they are claimed to be more or less as good as assembly, depending on who you ask. Most tools now offer options for customizing the compiler optimization. Additionally, using C increases portability, since C code can be compiled for different types of processors.

II. CONCEPT OF EMBEDDED SYSTEMS IN 'C'

An embedded system is a system that connects pre-programmed software on a controller embedded in the computer hardware. The software is installed on the controller, the brains of the electronic device. Each embedded system is used for one specific function. A complex device like a PDA or smart phone may have

embedded devices that can control several functions. However, embedded systems are not as complex or programmable as a personal computer. C is one of the most commonly used programming languages in embedded devices. Embedded systems are used to control electronic devices such as DVD players, cell phones, watches and medical devices. Video cards and network switches are embedded systems used in computer peripherals. The cruise control and anti-lock brakes in a car are also embedded systems. Embedded systems are cheaper than complex processors due to their simplicity and frequent mass production. Software written in a language like C is used to control the mechanical devices within the embedded system.

Reasons C is Common in Embedded Systems

C is one of the most commonly used software languages used on embedded device controllers. One reason is because it is one of the few software languages that operates on both 8 bit controllers and 64 bit PCs, meaning that many computer programmers can write C software for both personal computers and embedded devices. The C language can also use very simple commands to control the device, freeing up the limited memory of the device to hold many commands or parameters. C can be written for both microcontrollers and digital signal processors.

C Languages Used in Embedded Systems

C and C!! are used frequently in creation of embedded systems. Embedded systems are rarely programmed using the C++ because embedded systems rarely have the memory space for the complex programming used in C++. C# is occasionally used on embedded Linux systems.

How C Programs Are Created and Installed on Embedded Systems

Code is written in C on a programmer's PC. Code is run through a compiler on the programmer's PC to create a software program. The embedded system software may be run through a simulator on the programmer's computer. The software program is copied onto the controller using a "programmer." The controller is then tested on a "test bed" to ensure that it works properly.

Common Embedded Systems that Use C

Bluetooth devices are programmed in C. PIC microcontrollers such as those used in web cameras are frequently programmed in C. PIC microcontrollers programmed in C have also been used in LED (light emitting diodes) devices and LCD (liquid crystal display) monitors. USB devices are embedded devices frequently coded in C.

Standards for Embedded System Programming in C

The American National Standards Institute (ANSI) has written standards for the C programming language. The International Standards Organization wrote standard ISO/IEC 9899 for the C programming language. The Motor Industry Software Reliability Association has created a proprietary set of standards for programming in C for embedded devices in automobiles.

III. HOW 'C' IS USED DIFFERENTLY FOR EMBEDDED PROGRAMMING

If you are new to embedded C programming, you will notice that there are only subtle differences between a regular C program and an embedded C program. The following is a list of the most important differences between C programming for embedded systems and C programming for PCs.

Writing low-level code

In embedded programming, it is necessary to directly access the underlying hardware. For example, we might need to access a port, timer or a memory location. Similarly, we might need to do low-level programming activities like accessing the job queue, raise some signals, interrupts, etc. So, we need to write low-level programs that directly access, modify or update hardware; this is an important characteristic of embedded C programs. How do we write low-level code? C features such as pointers and bit-manipulation facilities enable us to program at the hardware level directly; so these features are used extensively in embedded programming.

Writing in-line assembly code

The C language provides a limited set of features, so it is not possible to use high-level C code to perform a specific function. For example, the device might have some instructions for which there is no direct equivalent in C code (for example, bit-wise rotation). In such cases, we can write assembly code embedded within C programs called 'inline assembly'. The exact syntax depends on the compiler; here is an example for GCC:

```
int a=10, b;
asm ("movl %1, %%eax;
movl %%eax, %0;"
:"=r"(b) /* output */
:"r"(a) /* input */
: "%eax" /* clobbered register */
);
```

This code just assigns a to b (!), but this is just an example to show how in-line assembly code looks; usually it will have the ASM keyword prefixed and/or suffixed by under scores. The assembly code is written in C program itself and we can use C variables to refer to data; the register allocation and other details will be taken care of by the compiler. We can write high-level functionality such as looping in C syntax, and only when we require processor specific functionality, we can write in-line assembly code. In this way, writing in-line assembly is more convenient than writing full-fledged assembly code.

No recursion, no heap

Many of the devices are small, so the heap area (where dynamic memory segments will be allocated) might be limited or might not even exist. So, we would need to program without using malloc or its variations. This poses difficulties for those new to embedded systems: how to use a linked list or a tree data structure which we conventionally program using dynamic memory allocation? Fortunately, many of the data structures can be implemented by using static allocation itself. Here is a simple illustration of a doubly linked list made up of three nodes that are allocated statically (instead of dynamic memory allocation):

```
struct node {
    struct node * prev;
    int data;
    struct node * next;
};
int main(){
    struct node one, two, three;
    one.prev = &three; one.next = &two; one.data = 100;
    two.prev = &one; two.next = &three; two.data = 200;
    three.prev = &two; three.next = &one; three.data = 300;
    struct node* temp = &one;
    do {
        printf("%d ", temp->data);
        temp = temp->next;
    } while(temp != &one);
}
```

This program prints '100 200 300'. Using the basic idea from this program, if we pre-allocate the number of expected nodes statically, then we can write our own my_node_malloc() function that will return a node that was allocated statically. Using such techniques, it is possible to implement dynamic data structures such as linked lists and trees that still work under the limitations

of embedded devices. Similarly, features such as recursion are not allowed in most of the embedded devices. One reason is that the recursion is inefficient in terms of space and time compared to iterative code. Fortunately, it is possible to write any recursive code as iterative code (though writing or understanding iterative versions of the code is not usually intuitive). To put it simply, costly language features (in terms of space and time) are either not available, or recommended in embedded programming. As programmers, we should find alternative ways of achieving the same functionality.

Using limited C features or new language extensions

Some of the language features that we take for granted in regular programming might simply not be available for embedded programming. For example, if the device does not support floating point numbers, then we cannot use floats or doubles in the programs. Problems frequently occur the other way also: often, the underlying devices have hardware features that don't have direct support in the C programming language. For example, a device might support fixed-point numbers; however, C language doesn't have any support for this datatype, so there is no direct way of programming it. Embedded C compilers differ from regular C compilers in many ways; one important difference is the language features they support. Most of the embedded C compilers will not have full conformance to the ANSI C standard (because it is heavy-weight); rather, they will support the embedded C specification (check the 'Reference' section for more details). This specification is meant for use in embedded systems, and the most common extensions and features to enhance performance and convenience for accessing underlying hardware resources are provided in it. The advantage in using this specification is that a large number of compilers implement the specification, and thus it is easy to port embedded programs with it.

Fig.1 Code Generation Flow for assembly and C

IV. PROCESSOR USED IN EMBEDDED 'C'

When desktop developers first think about working with embedded systems, there is a natural inclination to stick with what they know and look for a book which uses Pentium processors or other devices from this family (such as the 80486, or the Intel 188). However, if you open up the engine management unit or the air bag release system in your car, or take the back off your dishwasher, you will not find any of these processors sitting inside, nor will there be anywhere to plug in a keyboard, graphics display or mouse. Typical desktop processors cost more than US \$100.00 a piece (often much more). This cost puts them out of reach of all but the most expensive embedded application. (Who would pay more than US \$100 for a TV remote-control unit?) In addition, a desktop processor requires numerous external support chips in order to function: this further increases the cost. The additional components also increase the physical size of the system, and the power consumption: both of these factors are major problems for battery-powered embedded devices. (Who would buy a portable music player that requires ten large batteries to run, and needs a trolley to transport it?)

Overall, the state-of-the art technology used in desktop processors matches the needs of the PC user very well: however, their key features – an ability to execute industry-standard code at a rate of more than 1000 million instructions per second – come with a heavy price tag and are simply not required in most embedded systems.

The 8051 device is very different. It is a well-tested design, introduced in its original form by Intel in 1980. The development costs of this device have now been fully recovered, and prices of modern 8051 devices now start at less than US \$1.00. At this price, you get a performance of around 1 million instructions per second, and 256 **bytes** (not megabytes!) of on-chip RAM. You also get 32 port pins and a serial interface. The 8051's profile (price, performance, available memory, serial interface) match the needs of many embedded systems very well. As a result, it is now produced in more than 400 different forms by a diverse range of companies including Philips, Infineon, Atmel and Dallas. Sales of this vast family are estimated to have the largest share (around 60%) of the microcontroller market as a whole, and to make up more than 50% of the 8-bit microcontroller market. Versions of the 8051 are currently used in a long list of embedded products, from children's toys to automotive systems.

V. OPERATING SYSTEM USED IN EMBEDDED 'C'

Having opted to create our 8051-based applications using C, we can now begin to consider how this

language can be used. In doing so, we will begin to probe some of the differences between software development for desktop and embedded systems. In the desktop environment, the program the user requires (such as a word processor program) is usually loaded from disk on demand, along with any required data (such as a word processor file). Figure 1.5 shows a typical operating environment for such a word processor. Here the system is well insulated from the underlying hardware. For example, when the user wishes to save his or her latest novel on disk, the word processor delegates most of the necessary work to the operating system, which in turn may delegate many of the hardware-specific commands to the BIOS (basic input/output system). The desktop PC does not *require* an operating system (or BIOS). However, for most users, the main advantage of a personal computer is its flexibility: that is, that the same piece of equipment has the potential to run many thousands of different programs. If the PC had no operating system, each of these programs would need to be able to carry out all the low-level functions for itself. This would be very inefficient and would tend to make systems more expensive. It would also be likely to lead to errors, as many simple functions would have to be duplicated in even the smallest of programs. One way of viewing this is that a desktop PC is used to run multiple programs, and the operating system provides the 'common code' (for printing, file storage, graphics, and so forth) that is required by this set of programs.

Fig. 2 : A schematic representation of the BIOS/OS sandwich from a desk-bound computer system running some word processor software

There are two fundamental differences between the embedded systems we are concerned with in this book and desktop computer systems:

- The vast majority of embedded systems are required to run only one program: this program will start running when the microcontroller is powered up, and will stop running when the power is removed.
- Many of the facilities provided by the modern desktop OS – such as the ability to display high-resolution graphics, printing facilities and efficient

disk access –are of little value in embedded systems, where sophisticated graphics screens, printers and disks are generally unavailable.

VI. ADVANTAGES OF EMBEDDED 'C' IN THE PROGRAMMING

Programming in C makes the Embedded systems more reliable. C code written for a specific micro controller can easily be transferred to systems using different micro controllers of different vendors without little or no modification. It can be reused, easy to maintain and easy to debug and extend.

The advantages of C as shown in the following

- Of higher level languages, C is the closest to assembly languages.
- Most micro controllers have available C compilers.
- Writing in C simplifies code development for large projects.
- It is easier and less time consuming to write in C language than assembly.
- C is easier to modify and update.
- We can use code available in function libraries.
- C code is portable to other micro controllers with little or no change.
- Assembly is the fastest, however, it is difficult to find or train assembly experts. Then if a new processor is required, we have to start over!
- C is mid-level, lots of good C programmers are available, C compilers are available. C can be used on 8-, 16-, 32-, and 64-bits processors.

VII. CONCLUSION

An embedded system is some combination of computer hardware and software, either fixed in capability or programmable. C remains a very popular language for micro-controller developers due to the code efficiency and reduced overhead and development time.

C offers low-level control and is considered more readable than assembly. Many free C compilers are available for a wide variety of development platforms. The compilers are part of an IDEs with ICD support, breakpoints, single-stepping and an assembly window. The performance of C compilers has improved considerably in recent years, and they are claimed to be more or less as good as assembly, depending on who you ask. Most tools now offer options for customizing the compiler optimization. Additionally, using C increases portability, since C code can be compiled for different types of processors.

REFERENCES

- [1] Victor Yodaiken and Michael Barabanov “design document about RTLinux in SMLabs 1997.
- [2] Cramer, Michael Lynn: “A Representation for the Adaptive Generation of Simple Sequential Programs” Proceedings, International Conference on Genetic Algo, July 1985 [CMU], pp183-187.
- [3] Wayne Wolf, Computers as Components: Principle of Embedded Computing System Design, Morgan Kaufmann Publishers, 2001.
- [4] Frank Vahid and Tony Givargis, Embedded System Design: A Unified hardware/ Software introduction, John Wiley & Sons, 2002.
- [5] Scott Rosenthal, Selecting an embedded processor involves both simple and non-technical criteria, Lume, 1997.
- [6] King, K. N. (April 2008). C Programming: A Modern Approach (2nd ed.). Norton. ISBN 978-0-393-97950-3.
- [7] Thompson, Ken. A New C Compiler. Murray Hill, New Jersey: AT&T Bell Laboratories. http://doc.cat-v.org/bell_labs/new_c_compilers/new_c_compiler.pdf