An embedded C language Target Code level Unit Test method based on CPU Simulator

Heng Wang*, Xinrui Chen

School of Electronics and Information Engineering, Jingchu University of Technology, Jingmen 448000, China

*wanghengwh@126.cn

Abstract. For the 8bit MCU of RISC architecture which is widely used at present, the internal resources such as Rom Ram stack are all limited. If written in C language, a C language instruction will become a lot of machine code after compiling, and it is easy to find that the ROM space is not enough. Stack overflow, etc. One of the biggest advantages of C is that it has nothing to do with any particular hardware or system. This allows a user to write programs that run on almost all machines without any modification. This paper presents an embedded C language target code level unit test method based on CPU simulator.

1. Introduction

C language and assembly language in the development of single-chip computer what are the advantages and disadvantages, assembly language is a character mnemonic to express machine instructions symbolic language. It is a language closest to machine code. Its main advantage is that it takes less resources and has high efficiency of program execution [1]. However, different CPUs may have different assembly languages, so it is not easy to transplant.

C language is a structured high-level language, which has the advantages of good readability, easy transplantation and is widely used as a computer language.

For the 8bit MCU, which is currently widely used in RISC architecture, its internal resources, such as Rom Ram stack, are limited, if written in C language. After compiling a C language instruction, it will become a lot of machine code, it is easy to appear ROM space is not enough, stack overflow and so on. And some single-chip computer manufacturers may not be able to provide C compiler, and assembly language. One instruction corresponds to a machine code, and each step is very clear about what action to perform, and the program size and stack call are easy to control, and debugging is relatively convenient, so it is in the development of a single chip microcomputer. We'd better recommend assembly language.

This simple question is rarely answered entirely. In C, the keyword static has three distinct functions: in the body of a function. A variable declared as static remains constant during the call to the function. (but in the function outside, a variable declared as static can be queried by the function used in the module. It is a local global variable. In a module, a function declared as static can only be called by other functions within this module. That is. This function is limited to the local scope of the module that declares it. Most candidates can answer the first part correctly and part the second part correctly. Very few people understand the third part. This is a serious disadvantage of an examinee, who obviously does not understand the benefits and repeatability of localized data and code scope.
The C'programming language was originally developed by Dennis. One of the biggest advantages of C that Ritchie developed and implemented for UNIX in 1971 is that it is independent of any particular hardware or system. This makes it possible for a user to write a program without any modification [2]. On almost all machines, C is often referred to as Intermediate computer language because it combines the elements of high-level languages with the functions of assembly languages.

C is very flexible, and can do whatever you want. This freedom gives C very powerful functions that can be mastered by experienced users; C is a relatively small language, but it is durable; C is sometimes referred to as "advanced assembly language"; Low-level (bit operation) programming is also easy to implement; Loose type (unlike other high-level languages); C is a structured programming language. C allows you to create any tasks that you already have in mind. C preserves the basic system that programmers know what they are doing; It only requires that they express their intentions clearly.

The instruction system generated from development to the present has formed five kinds of style, which is a set of complex instructions (CISC), reduced instruction set (RISC), very long instruction word (VLIW), a data signal processing instruction set (DSP) and the special instruction set (ASIP). The five styles have different characteristics, can not be vague to say what kind of what kind of good or bad, they are suitable for different applications.

The number of instructions and the way of addressing are few. A single machine cycle operation. In a RISC machine, the execution of the overwhelming majority of instructions requires only one machine cycle.

RISC is characterized by high speed, reliability and low cost.

The essence of the concept of CISC is the processor design complexity, thereby simplifying procedures, and then make the required memory capacity is reduced to the minimum. And the idea of RISC is exactly opposite to CISC, because it allows the processor to execute simple instructions, more than one instruction in the same machine cycle, so the complex instruction performs faster than CISC.

In general, perform the same function, a RISC program is better than a CISC program 30%, which makes higher demands on the processor of RISC compiler, a very complex and technically demanding work optimizing compiler designers, so CISC's supporters pointed out that although the RISC simplifies the hardware design, but greatly increased the software the burden.

Because most instructions in the RISC architecture is register to register operations, thus reducing the memory access speed of the speed, but this also requires intermediate results within the processor must generate a general register enough to save in the computer. Usually the RISC processor by using at least 32 general-purpose registers, general register not so much to occupy the space of the chip, and the addressing time increased. While the CISC processor in the same performance conditions do not need so many general-purpose registers.

2. Code level Unit Test method for embedded C language Target

Pattern based static code analysis, runtime memory monitoring. Software verification techniques such as unit testing and data flow analysis are effective ways to find embedded C language programs / software defects. Each of the above techniques can find a particular type of errors. Even so. If the user uses only one or more of the above techniques for verification [3]. It is very likely that such a verification method will miss the examination of some defects in the program. A safe and effective strategy for solving such problems is to use all the complementary techniques in the above software verification simultaneously. Set up a solid framework to help users detect defects that may avoid a particular technology. At the same time. Users also naturally create an environment that detects critical and hard-to-find functional errors.

This article will elaborate the static code analysis based on the pattern, the memory error detection at run time. How to find out the defects in embedded C program / software when the automation technology such as unit test and data flow analysis are used together. In this article, we will use Parasoft C. Test demonstrates that the above techniques. C test is an automated integration solution that has been proven to improve the efficiency and quality of software development teams through a wide range of best practices.
It is important to pay attention to screenshots when reading this article and thinking about the defects you find at any time. Automated detection of defects such as memory crashes and deadlocks. There is no doubt that this is an essential task for any development team. However, the most fatal flaw is a functional error. This is often difficult to detect automatically. In the conclusion of this article, we will briefly discuss techniques for finding these defects.

Both UPDATE and DELETE statements are set operations, and if you only want to modify or delete one of the records, you need to use a cursor SELECT statement to identify all the conditional records. Further identify the records to be modified or deleted, and then process them with UPDATE and DELETE statements in the form of CURRENT.

1: use DELCARE statements to describe cursors. If it is prepared for UPDATE statements in the form of CURRENT. To use the FOR UPDATE in the SELECT statement. The of column name is used to indicate that the queried data is modifiable in the specified column if it is prepared for an DELETE statement in the form of CURRENT. You do not need to use the above clause.

2: check if the record is a modified or deleted record. If so. The clause WHERE CURRENT of <cursor name > is used in the UPDATE and DELETE statements. Represents a modification or deletion of a record that was last retrieved, that is, a record pointed to by a cursor pointer.

C or assembly language can be used in single-chip computer C can, as is shown by figure1.

![Figure 1](image_url)

Figure 1. In the development of single chip computer, it is mainly assembly and C

In the development of single chip computer, it is mainly assembly and C, but not with C. Assembly language is a symbolic language that uses character mnemonic to express machine instruction, and it is the language closest to machine code. Its main advantage is that it takes up less resources. Program execution efficiency. But different CPU, its assembly language may be different, so difficult to transplant.

For the 8bit MCU of RISC architecture which is widely used at present, the internal resources such as Rom Ram stack are all limited. If written in C language, a C language instruction will become a lot of machine code after compiling, and it is easy to find that the ROM space is not enough. Stack overflow and other problems. And some single-chip computer manufacturers may not be able to provide a C compiler. And assembly language, an instruction corresponding to a machine code, each step of what action is very clear, And the program size and stack call are easy to control, debugging is relatively convenient. So in the less resources of the development of single-chip computer, we recommend the use of assembly language is better [4].

C language is a compiler programming language, which takes into account the characteristics of many high-level languages, and has the function of assembly language. C language has rich library functions, fast operation, high compiling efficiency and good portability. Furthermore, it is possible to control the hardware of the system directly. C is a structured programming language. It supports the top-down structured programming technology widely used in current programming. In addition, C language programs have a perfect modular program structure. Therefore, using C language to program design has become a mainstream of software development. It can greatly shorten the development cycle, increase the readability of the software obviously, and make it easy to improve and expand, thus the system with larger scale and more complete performance can be developed.
In summary, it is an inexorable trend for the development and application of single chip computer to use C language to design the program of single chip computer, so as to develop a large scale software system, it is better for the single chip computer developer to master the basic C language programming [5].

A global variable (external variable) is a static global variable before it is described by static. The global variable itself is a static storage method. Static global variables are of course static storage. There is no difference between the two. The difference between the two is that the scope of non-static global variables is the whole source program. When a source program consists of multiple source files, non-static global variables are valid in each source file, while static global variables limit their scope. That is, it is valid only in the source file that defines the variable and cannot be used in other source files of the same source program. Because the scope of the static global variable is limited to one source file. Can only be used for functions within this source file, so errors can be avoided in other source files.

From the above analysis, we can see that changing the local variable to the static variable is changing its storage mode, that is, changing its lifetime, and changing the global variable to the static variable changes the scope of its scope. It limits its scope of use.

The static function is different from the normal function scope. Only in this file. The function used only in the current source file should be described as an internal function. Internal functions should be described and defined in the current source file. For functions that can be used outside the current source file, it should be stated in a header file that the source file to use these functions should contain the header file.

What is the difference between a static global variable and a normal global variable?

What's the difference between a static local variable and an ordinary local variable? the: static local variable is initialized only once, and the next one is based on the previous result value.

3. Embedded C language based on CPU Simulator

T Stack: as long as the remaining space in the stack is larger than the requested space, the system will provide memory for the program, otherwise it will report an exception to the stack overflow.

Heap: first of all, you should know that the operating system has a list of free memory addresses, and when the system receives an application from the program,

The list will be traversed to find the first heap node with a space larger than the requested space, and then the node will be removed from the free node list, and the space of the node will be allocated to the program, in addition, for most systems. The size of this allocation is recorded at the first address in this memory space, so that the delete statement in the code correctly frees the local memory space. Since the size of the found heap node is not necessarily equal to the size of the application, the system automatically replaces the excess part in the free list.

(III) restrictions on the size of applications

Stack: in Windows, a stack is a data structure that extends to a low address and is a continuous area of memory. This phrase means that the address at the top of the stack and the maximum capacity of the stack are predefined by the system. Under WINDOWS, the stack size is 2m (or, in some cases, 1m, a compile-time constant), if the requested space exceeds the stack's remaining space. Will prompt overflow. therefore, there is less space available from the stack, as is shown by figure2.
Figure 2. Heap: a heap is a data structure that extends to high addresses and is a discontinuous area of memory. This is because the system is free memory addresses stored in linked lists, which are naturally not contiguous. The traversal direction of the linked list is from low address to high address. The size of heap is limited by the effective virtual memory in computer system [6].

When we think about moving to C, we have some common problems:

- Produce large and inefficient code; standard IO code printf, scanf, strcpy et al.; memory location using: mallocator; stack usage, not very direct in C; data declarations in RAM and ROM; difficulty in writing interrupt service programs.

- ANSI C of 4-bit 8-bit Microcontroller.

For embedded systems, pure ANSI C is not convenient because.

- Embedded systems work with hardware. ANSI C provides a poor tool for addressing registers in fixed storage space; almost all embedded systems use interrupt ANSI C with various types of facilitation rules. It is definitely a performance killer for 8-bit machines; some microcontroller structures do not have hardware to support the C stack; many microcontrollers have multiple storage spaces.

- When using the C language on a low-end 8-bit microcontroller, try to make the code smaller. This means breaking some programming rules.

Open / close global interrupt; use GOTO statement; global label; global register segment; pointer support.

- The main features of embedded programming environment.

  - Limited RAM; limited stack space; hardware-oriented programming; strict timing ISRs, tasks, etc.; many different kinds of pointers are far / new / [7].

- The compiler is just a capable good assembler programmer.

Writing good C code that can be converted into efficient assembly code is much easier than manually writing efficient assembly code. C is the ultimate solution, but it is not the end in itself.

To give a concrete example, we will introduce and demonstrate our recommended defect lookup strategy for a recent case that we have encountered: a bug search strategy running in ARM. Simple sensor application on board.

Suppose we have created the application, but when we upload the program to the system target board and try to run it, we do not see the expected output on the LCD screen.

![Figure 3. a bug search strategy running in ARM](image-url)
We are not sure why the system is not working properly, so we try to debug the system. But debugging on the target board is time consuming and annoying, because we have to manually analyze the results of the debugger and try to figure out the real cause of the problem manually. Identify the wrong tools or techniques to help us lighten the burden. From this point of view, we can either expect to use the debugger to debug the program to bring good luck. Either we try to use an automated test strategy to find errors in our code if automation techniques still don't help us find errors. Then we have to go back to using the debugger as a last resort.

There are many points for attention in the design of high frequency circuit board, especially the high frequency circuit of GHz grade. More attention should be paid to the influence of the length of pad and printed pattern on the circuit characteristics. In recent years, high frequency circuits and digital circuits share the same circuit board. It seems that the so-called mixed-load circuit system has an increasing trend, such a design will often cause digital circuit action, high-frequency circuit action instability and other phenomena. One of the reasons is the noise generated by the digital circuit, which affects the normal operation of the high-frequency circuit. In order to avoid the above problems, besides trying to divide the two circuits block. Before the design of the circuit board to fully review the design concept, is the fundamental approach.

Stack: automatically assigned by the system, faster. But the programmer is out of control.

Heap: memory allocated by the new, generally slower, and easy to generate memory fragmentation, but most convenient to use.

In addition, under WINDOWS, the best way to allocate memory is to use Virtual Alloc, which is not on the heap or on the stack, but to keep a block of memory directly in the process's address space. Although the most inconvenient to use.

Storage content in the heap and stack

Stack: when a function is called, the first step into the stack is the address of the next instruction in the main function (the next executable statement of the function call statement), followed by the various parameters of the function, in most C compilers. Arguments go from right to left to the stack, then to the local variables in the function. Note that static variables are not stacks.

When the function call is over, it is the local variable goes out of the stack first, then the parameter. Finally, the top pointer of the stack points to the first stored address, that is, the next instruction in the main function, and the program continues to run from this point.

Heap: the heap size is usually stored in a byte in the header of the heap. The details of the heap are arranged by the programmer.

4. **An embedded C language Target Code level Unit Test method based on CPU Simulator**

We only find an instance of a memory write overflow in the path of the code we execute. How can we conclude that there will be no memory writes overflow errors in code that we have not yet executed? If we examine the coverage analysis, we will find that this function has never been executed. We need to test this function. But how exactly? Creating a unit test case that calls this function is a good idea.

Use runtime memory monitoring in unit tests: we use the C test test case wizard to create a test case framework. And add some test code to it. Then run the test case to check the untested functions mentioned above, while turning on runtime memory monitoring. Use C test. The whole process takes about a few seconds.

The result indicates that the function has been overwritten, but also finds a new error:

Our test case found more memory-related errors. Obviously, when the failed handler was called, we had problems initializing our memory (null pointers). We found that in the reportSensor value) function there was an error in the order in which the function was called. Function is called. But the finalize#en0#) function frees up the memory needed for the printMessagePin function.

Computer memory is divided into two parts, one part holds instructions and the other part holds data. Each of them has its own address space and access instruction, which can be accessed independently. So this kind of computer structure is called Harvard architecture, called Harvard structure.
Von Neumann Architecture (Princeton structure): there's only one main memory, and the contents in the main memory can be either data or instructions.

The data bus in the von Neumann architecture is divided into the instruction bus and the data bus in the Harvard architecture [8]. So that the data throughput of the Harvard processor is about twice as high as that of the von Neumann processor, so that the Harvard processor is usually more efficient than that of the Von Neumann processor.

When we ran the test case again, only one task was reported: the unverified unit test case and an unvalidated unit test case. This is not really an error. We just need to validate the output to convert the test case into a regression test by creating the appropriate assertion. C test will automatically complete these steps for us.

![Figure4. an unvalidated unit test case](image)

Next we run the whole program again. The coverage analysis tells us that almost the entire program has been overwritten and no memory errors have been found.

Is this over? Although we have run the entire program and created unit test cases for unoverridden functions, there are still paths that are not overwritten. We can still continue to create unit test cases. But it would take a long time to expect to cover all the paths in the program in such a way, or we could use another method, using data flow analysis, to simulate these paths.

The first two have the same effect that it is a constant integer. The third means is a pointer to a constant integer (that is, an integer is not modifiable.). The fourth meaning it is a constant pointer to an integer number (that is, the number of integers the pointer points to ismodifiable. But the pointer is not modifiable. The last one means that a is a constant pointer to a constant integer number (that is, the integer number the pointer points to is immutable). The pointer is also unmodifiable. If the candidate can answer these questions correctly, he will make a good impression on me. By the way, you may ask.

Even without the Const keyword, it's easy to write a program with the right functionality, so why do I value the keyword const so much? I also have the following reasons:

The function of the keyword const is to send very useful information to the person reading your code, in fact. Declare a parameter as a constant to tell the user the purpose of the application of the parameter. If you have spent a lot of time cleaning up the garbage left by others, you will soon learn to appreciate this redundant information. Programmers is who know how to use const leave little garbage for others to clean up.)

By giving the optimizer some additional information, using the keyword const may result in more compact code.

The rational use of the keyword const allows the compiler to naturally protect parameters that do not want to be modified by unintentional code. This can reduce the appearance of bug.

5. Summary
The IBIS model core consists of a list of information about current, voltage, and timing. The simulation speed of the IBIS model is much faster than that of the SPICE. The accuracy is only slightly reduced. Non-convergence is a problem for SPICE models and emulators, which is eliminated in IBIS simulations.
All EDA vendors now support the IBIS model. Most device IBIS models are available on the Internet for free. Several different manufacturers can be simulated on the same board.

IBIS model is a fast and accurate modeling method for I/O BUFFER based on the V/I curve. It is an international standard to reflect the electrical characteristics of chip driving and receiving. It provides a standard file format for recording parameters such as drive source output impedance, rise/drop time and input load. It is very suitable for the calculation and simulation of high-frequency effects such as oscillations and crosstalk.

Acknowledgements

2019 Jingmen Guiding Scientific Research Project “Research of Medical Image Encryption on Scrambling and composite segmentation chaotic mapping based on dynamic update”, Number: 2019YDKY078.

References

[1] S. Lee, B.G. Kim and K. Lee, Fuzzy cognitive map-based approach to evaluate EDI performance: a test of causal model, Expert Systems with Applications 27 (2) (2014), pp. 287–299.
[2] Ma Jun, "Research of Electronic Business Security Based On Public Key Encryption Methods", IJACT, Vol. 4, No. 2, pp. 50 ~ 57, 2012.
[3] A.K. Tsadiras and K.G. Margaritis, A study of the two unit certainty neuron fuzzy cognitive map, Neural, Parallel & Scientific Computations 9 (1) (2011), pp. 67–90.
[4] Xiangya Tao, Changyun Jia, Cheng Yang, "Security Analysis on China Campus Network and Construction on Network Security System", JCIT, Vol. 6, No. 11, pp. 166 ~ 174, 2014.
[5] Fensel D. Ontologies:Silver Bullet for Knowledge Management and Electronic Commerce[M]. Springer-Verlag, Berlin (2013).
[6] ZANG Weihua, GUO Rui, "The Application of Neural Network based on Evolutionary Strategy in Network Security Quantification Analysis", AISS, Vol. 4, No. 2, pp. 151 ~ 159, 2015.
[7] Ye Du, Jiqiang Liu, Ruhui Zhang, Jianguvuan Li,A Dynamic Security Mechanism for Web Services Based on NDIS Intermediate Drivers, Journal of Computers, Vol 6, No 10 (2016), 2021-2028
[8] Qing Wei, Jitaq Zhao, Jingjing Song, A Nonreversing FP-tree Algorithm Mining with Frequent Closed Item Sets, International Review on Computers and Software, Vol. 7, N. 2, pp. 822-829, 2012