Semiautomatic Porting of the C Library

Ludek Dolihal and Tomas Hruska
Faculty of Information Technology, BUT
Bozetechova 1/2,
612 66 Brno, Czech Republic
{idolihal, hruska}@fit.vutbr.cz

ABSTRACT

For testing of automatically generated C compiler for embedded systems on simulator, it is useful to have a C library support. Testing programs written in C very often use I/O operations and other functions provided by the C library. Hence not having the library, the number of programs that can be executed is very limited. In this paper, we describe a method that provides semiautomatic method of porting the C library. The processors for which we port the library are mainly 32-bit. For this reason, we have chosen the Newlib library.

KEYWORDS

Porting of a library, C library, compiler testing, simulation, hardware/software co-design, Newlib, Lissom, Codasip.

1 INTRODUCTION

This article will discuss the problematic of testing of the automatically generated compiler. Please note that this work is still in progress, and we are still dealing with some difficulties. To be more precise, the paper will discuss the semiautomatic porting of the C library that is essential for the thorough testing of the compiler. As the main aim of the Lissom project [1] (now also Codasip project-www.codasip.com) is hardware software co-design, we need to ensure that the compiler has the full functionality and contains a minimum of errors.

One of the goals of our research group is to deliver, together with the other tools, also the automatically generated compiler for various architectures. Currently, we are able to generate the compiler for various architectures including Microprocessor without Interlocked Pipeline Stages (MIPS) , Acorn RISC Machine (ARM) in the seventh version and also AVR32 from Atmel. The compiler is generated from the processors description.

To minimize the number of errors in the automatically generated compilers as much as possible, it is necessary to put the generated compilers under thorough testing. As we do not have all the platforms for which we develop available for testing, we use the generated simulators instead in majority of cases. Note that this simulator is also automatically generated from the processors description as well as the compiler.

The role of the compiler in the whole process of testing is crucial. We commonly use tests written in C for the compiler testing and the tests frequently use I/O functions, functions for memory management etc. Should we not possess the C library , we could not use over 50% of our test cases.

This paper is going to discuss the problematic of semiautomatic porting of the Newlib[4] library and is organized in the following way. The second section gives a brief overview of the Lissom project, section three provides some information about related work, section four describes the mechanism of the semiautomatic testing, and section five concludes this paper.

2 OVERVIEW OF THE PROJECT

The Lissom project focuses mainly on the hardware software co-design. As we try to deliver the best possible solution, we deliver also the C compiler, because the C is still the main development language for the embedded systems. Besides the C compiler there are various other tools generated from the processors description. The processor is described in the ISAC language [2]. Amongst the generated tools are:

- simulators,
- assembler,
- disassembler,
- debugger and
- verification environment.

The generated simulator was thoroughly described in the article [2] and verification environment description can be found in [3].

As is apparent, the tools are usually after several steps generated from the specification in the ISAC language. This description will play a crucial part also in case of semiautomatic library port. The primary role of the C library is to enlarge the range of constructions that can be used during the process of testing. It is without all doubts important to test the basic constructions such as if statement, loops, function calls etc. On the other hand, it is highly desirable to have a possibility of printing either to standard or error outputs or exiting program with different exit values or reading the the input from the file. And this cannot be done without C library support. Exit values are the basic notification of program evaluation and debugging dumps are also one of the core methods of debugging. Note that all the tests are designed for the given embedded system, and the tests are run on the simulator.

For the use in the embedded systems, we looked for the library that was small, modular, provided the basic functionality and was still in the process of development. After trying several of them we decided to use the library called Newlib.

3 RELATED WORK

Some basic information about the porting of the Newlib library can be found at the official pages of the project [4]. It is a quite detailed guide to the process of the creation of the port to the new architecture. However, the description contained on the project pages is far from being automatic. The user has to conduct quite a lot of manual steps to get the functional version of the library.

The automatic process of porting is in the center of the paper Automatic Porting of Binary File Descriptor Library [5]. This approach is quite similar to the process that is going to be described in this paper. Nevertheless, the mentioned paper describes port of the GNU Binutils core. The BFD library is used for the manipulation of the object files, so its purpose is completely different of the Newlib library. The process takes as the inputs, the architecture description and templates. But the process of generation is not described in details in this paper.

Unisim project [6] was developed as an open simulation environment which should deal with several crucial problems of today simulators. It looks at the problematic from the different perspective. One of the problems is a lack of interoperability. This could be solved, according to the article, by a library of compatible modules and also by the ability to inter-operate with other simulators by wrapping them into modules. Though this may seem to be a little out of our concern the idea of the interface within the simulator that allows adding any library is quite interesting. In our case we will have the possibility to add or remove modules from the library in a simple way. But the idea from the Unisim project would make the import of any other library far easier than it is now.

4 SEMIAUTOMATIC PORTING

We have modified the Newlib so it is now using a CMake system. We have divided it into two parts that are placed in a separate directories. One part is common for all platforms. This part is placed in the directory called newlib. The directories that contain platform dependent files are stored in the directory with the model. This is done to have all the platform dependent files in one place in the strictly given directory structure.

Let’s have a look at the platform dependent files. Strictly spoken, the directories does not contain only the platform dependent files. There are also files that are the same for all our platforms but the division is done on the level of directories and not on the level of files itself. The directories that are kept together with the model are the directories libgloss and directory newlib (this is the subdirectory of the directory newlib mentioned the above paragraph).

While the directory newlib contains mainly header files with various settings and definition of the setjmp.S the directory libgloss takes care of the syscalls handling. The syscalls are very important for our project, because this mechanism allows us to get the information in and out of the simulator. In this paper we will focus on the way how to automatize the process of syscalls creation.

There are several ways how to cope with syscalls porting. After we gathered all the necessary information about what syscalls are necessary for our simulation and tried several ways of implementation we found out, that only very small part of the syscalls must be done in assembler. The rest can be written in C and that makes the code platform independent. The Newlib defines 20 syscalls, but we need just 6 of them. Nevertheless, the rest of the syscalls could be implemented the same way as the 6 supported ones. The syscalls are defined in the header file, and have numbers from 1 to 20. The first 6 are the supported ones and the rest of the numbers is assigned to the unsupported ones.

For the syscalls itself, we have defined the structure called params. This structure contains the parameters that are needed for each syscall. This structure slightly varies depending on the actual syscall. But it is written in C, which makes it also platform independent. What is only done in the the assembler and is hence platform dependent is the PERFORM_SYSCALL function. In fact, it is not a function but multiple line macro defined in inline assembler. Assume, that multiple line macro can have the following form:

```
   #define PERFORM_SYSCALL(ADDR) \
      __asm__( "REGr1=add
REG0,%0" : :"r"(ADDR)); \
      __asm__( "syscall");
```

This macro is not taken from any existing processor. We have defined it just for model purpose. Now let’s have a closer look at the macro itself. This macro takes only one parameter. The ADDR parameter is the address of the structure that contains the parameters of the syscall as mentioned above. This address is assigned to the register that is used for passing of the parameters. This register can be specially marked as we will see later on. Then there is the special syscall instruction in this case it has the name syscall.

This two lines can be determined from the description of the core performed in the ISAC language. We will introduce the constructions that are necessary for that. The PERFORM_SYSCALL macro itself is a template. The necessary information is filled into this generic template before the compilation time of the library.

First lets have a look at the syscall instruction. We simply scan the model for the instruction that bears this name. In case the instruction is not found, we search the model for the construction in the following form:

```
#define syscall instruction
perform_syscall
```

In case this construction is found, we use this instruction in the second line of the multiple line macro. Please note that in this case, the instruction does not take any parameters. If this instruction was parameterized we would determine the parameters from the syntax. Nevertheless, this instruction does not have to be found. In such case, the template would be incomplete and error should be reported.

The process is shown on the following picture.

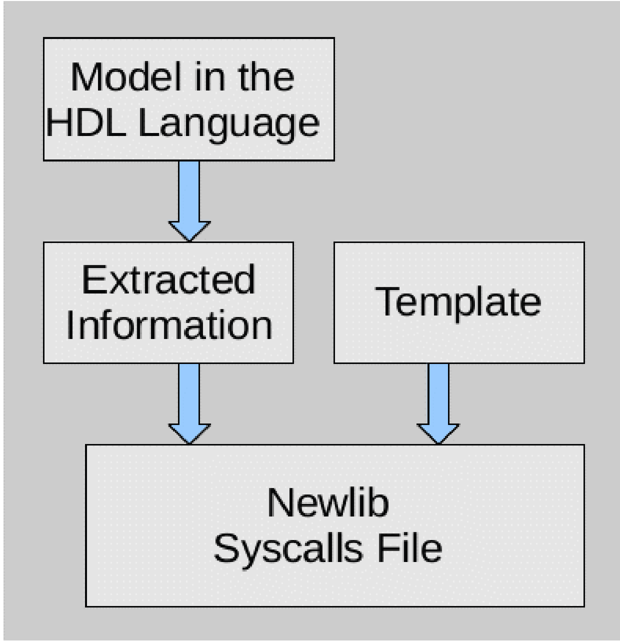

Figure 1. Scheme of the system

As far as the first line of the macro is concerned, we need to assure, that in the register which is used for passing the parameters we assign the address of the structure with the parameters. So we search the model for the instruction add or instruction with similar functionality. In the syntax section of the instruction, we find the actual form. Then we find the register for passing parameters in our model that also bears special description. From this parts of information, we should be able to put together the first line of the macro. This approach works for standard architectures. But there may occur architectures, for which might arise difficulties. The Newlib library in the current version support only 32-bit architectures.

5 CONCLUSION

In this short paper, we proposed the way how to automatically port the syscalls in the C library called Newlib. This approach saves time to the developer when creating the model of the microcontroller and needs the support of the C library. But this is just the part of the whole process of the Newlib porting. In the future, we would like to focus on automatic creation of the crt0.s and other platform dependent files. After this process is finished, the port should be fully automatized. We hope that this approach should shorten the time needed for the porting of syscalls significantly and at the same time it should widen rapidly the amount of programs that can be tested.

ACKNOWLEDGEMENTS

This work was supported by the IT4 Innovations Centre of Excellence CZ.1.05/1.1.00/02.0070.

REFERENCES

[1] Lissom Project, http://www.fit.vutbr.cz/research/groups/lissom

[2] Z. Přikryl, K. Masařík, T. Hruška, A. Husár, “Generated cycle-accurate profiler for C language”, 13th EUROMICRO Conference on Digital System Design, DSD'2010, Lille, France, pp. 263—268.

[3] M . Simkova, Z. Prikryl, T. Hruska and Z. Kotasek, “Automated functional verification of application specific instruction-set processors,” IFIP Advances in Information and Communication Technology, Heidelberg: Springer Verlag, 2013, vol. 4, no. 403, pp. 128-138. ISSN 1868-4238

[4] Newlib Project. http://www.embecosm.com/appnotes/ean9/ean9-howto-newlib-1.0.html

[5] M. Abbaspour, J. Zhu, “Automatic porting of Binary File Descriptor library”, www.eecg.toronto.edu/~jzhu/.../doc/tr-09-01.pdf

[6] S. Onder, R. Gupta, “Automatic generation of microarchitecture simulators”, Computer Languages, 1998. Proceedings. 1998 International Conference on , vol., no., pp.80-89, 14-16 May 1998