A MINIMAL DEVELOPMENT ENVIRONMENT FOR THE AVR PROCESSOR

Federico de Ceballos
Universidad de Cantabria
federico.ceballos@unican.es

November, 2001

Abstract

This paper presents a development environment based on the AVR processor and a few other discrete components, looking not only for minimum cost but also for ease of use. The system uses an external PC with a cross compiler that incorporates the necessary tools for loading the code in the target system.

The first priority in the development phase has been the search for a self-contained system in which all parts are as simple as possible, so that all tools needed can be completely mastered by a single person with a moderate effort.

Even if the system is presented as a good starting point in learning embedded systems, the compiler is an industrial grade product capable of being validated and used in more demanding environments.

Introduction

As it is widely acknowledged, one of the main advantages of the Forth language is its ability to adapt itself to different problems.

In this paper we shall consider a family of compilers developed in a Forth dialect (called NF, for Nuclear Forth) that produce code for different processors. These compilers have been developed by the author as part of his doctorate research, looking for a flexible system that can be subjected to severe analysis in order to meet the quality requirements of safety-critical applications.

These compilers, however, can also be viewed in a different light: Because of their simplicity, they are good candidates for a practical course in compilers and they can also be used for real work without an overwhelming environment or lots of resources.

At present four such compilers are available, each of them for a completely different environment: the Windows operating system, a single board computer based on a PSC1000, a single board computer based on a 68HC11 and an AVR microprocessor.

The last of them has been chosen for this presentation as it illustrates in the greatest detail the idea of compiler simplicity and lack of redundant elements.

Description of the AVR AT90S2313

The AVR, from Atmel, is a family of microcontrollers with a RISC core that combines a rich instructions set with 32 general-purpose working registers, all of them directly connected to the ALU. As a result, throughputs approaching 1 MIPS per MHz can be easily achieved.

Of all these processors, the AT90S2313 is the one that gives the most power in a package of no more than 20 pins. It is available in PDIP and SOIC packages, the former of which has been chosen for easier handing.

This microcontroller provides 2K bytes of In-System Programmable Flash, 128 bytes EEPROM, 128 bytes SRAM, 15 general-purpose I/O lines, timer/counters with compare modes, internal and external interrupts, a programmable serial UART, a programmable Watchdog Timer with internal oscillator, an SPI serial port for Flash memory downloading and two power-saving modules. Further details can be found in [2].

According to the manufacturer "*AVR instructions are tuned to decrease the size of the program whether the code is written in C or Assembly*". The same remark applies to Forth.

To operate, the microcontroller has to be put in a breadboard, connected to a suitable crystal or resonator and powered with a single +5V supply. A simple cable from the PC parallel port can be used for downloading the software.

Description of the Compiler

As stated before, the compiler family is composed of four distinct members that produce code suitable for execution in four different platforms. All of them have been developed on top of a single program: a conventional interpreter / compiler running on Windows.

One of the most arcane aspects in Forth is the meta-compilation process. Because of this, several authors strongly prefer coding a system in assembler (see, for example [10]). Arguments can be found in both sides (see [8]) and this issue is not yet resolved.

In this case, the author has preferred working towards a cross compiler that is simple enough to be comprehended, checked or modified rather than justifying the use of external resources that can lead to new errors.

This way, all the compilers work in a similar way, accepting a Forth dialect called NF (for Nuclear Forth, see [6]). This dialect takes advantage of using the new registers available in the Machine Forth virtual machine (described in [5] and [9]) and leaves aside all parts of ANF Forth not needed in the compilers or in the expected user applications.

As an example of this simplification, double number are not provided, as they are neither needed in the host system (as it is a 32-bit machine) nor used frequently in embedded applications.

Whereas it is not claimed that NF is the simplest Forth available, every effort has been taken in leaving out unneeded features.

Files and Programs involved in the System

Figure 1 shows the different parts that comprise the system. Groups of source files are shown in grey rectangles and programs are represented as rounded boxes.

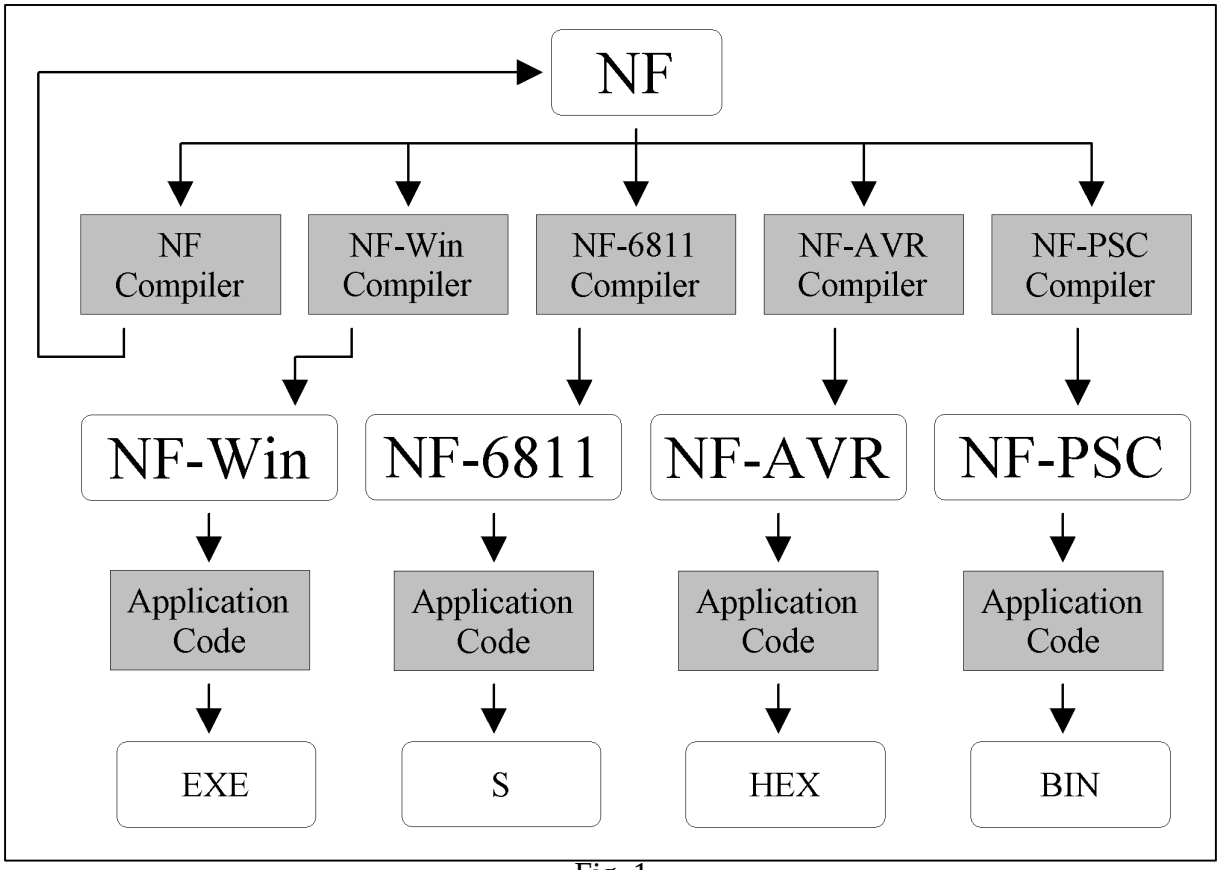

Fig. 1

As it can be seen, there are several versions of the NF compiler, one for each target. These versions have some code in common and the rest is specific of each target. When the new compiler is available as a separate program, it can be used to accept the application code and generate a file suitable for execution or for being downloaded to the target system. (To simplify matters further, each compiler can download the kind of files it produces.)

For Windows, two versions are available. NF-Win is a cross compiler that produces a final file without headers or other unneeded info (these files can be as little as 3K bytes). NF is the meta compiler used to produce a new version of the main compiler. This new version, as usual in Forth, will have headers, a compiler / interpreter and the ability to extend and save itself.

The ARV Compiler

This paper is centred on the NF-AVR compiler as it accepts source code that deviates most from ANS-Forth practice.

This compiler, as the rest of the family, follows the native code path with some inlining. In order to avoid including unneeded functions in the target system, it is possible to refer to previously declared functions. These functions, available in source code, are only included if used.

Its most outstanding feature is that it runs an 8-bit virtual machine.

According to ANS-Forth, 16 bits is the minimum allowed for an implementation to be compliant. However, there are lots of problems around that can be solved with less than that.

The use of 8 bits per cell, on the other hand, allows for less memory usage (something of the utmost important in a system with only 128 bytes of RAM available), simplifies the code (using less Flash, another scarce resource) and speeds the system up.

Using the Environment

It can be argued that the simplest environment is no environment at all. With this adage in mind, the author has tried to use the facilities provided by the host system, in the idea that the user should be already familiar with them.

Figure 2 shows part of the Windows Desktop (as of Windows 2000, the Spanish Edition). A folder labelled LEDS contains several text files with different tests and a link to the compiler.

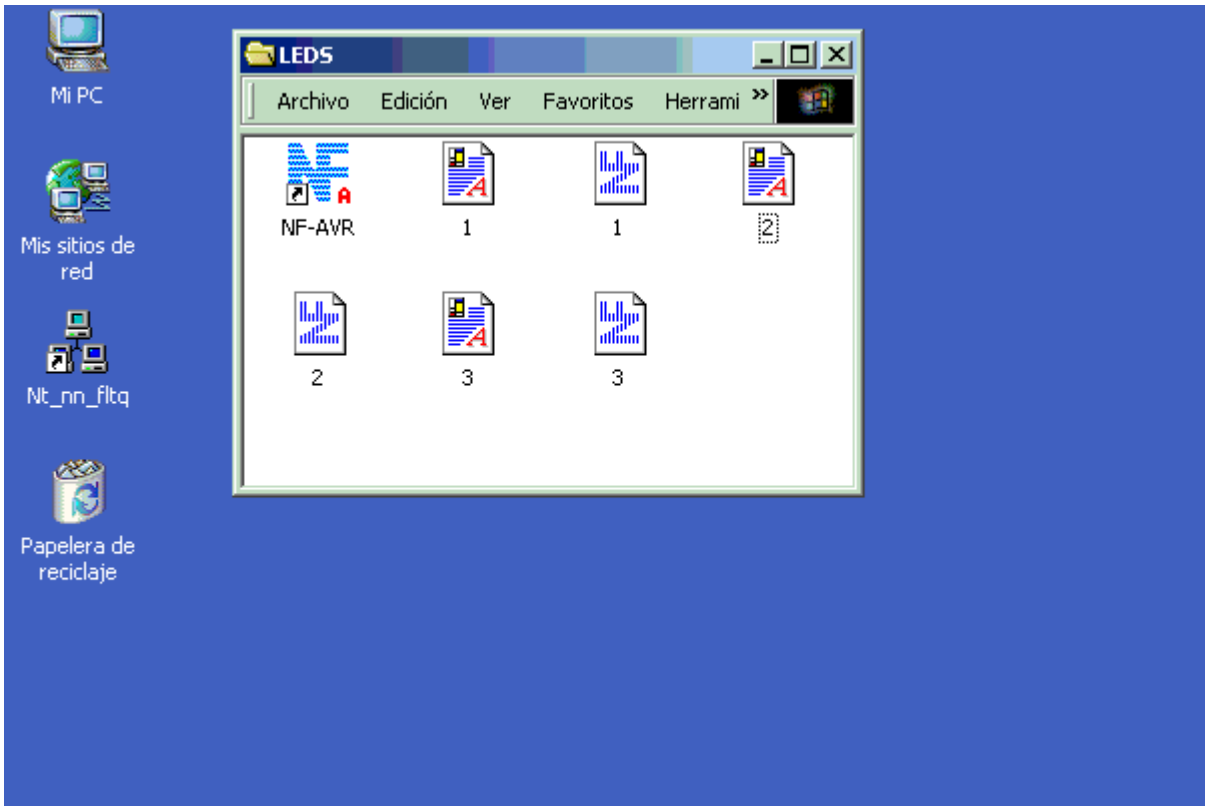

Fig 2

By double-clicking a source file (with the NF icon) it is opened with an editor (NotePad from Windows or any other of choice). By dragging and dropping a source file into the link, it is compiled and the corresponding HEX file is generated. It is also possible, inside some editors, to execute a program passing as parameter the name of the file being edited. This would also work here.

By double-clicking a HEX file it is opened with AVR Studio[1]. Dragging and dropping it into the link sends it to the target system.

[1] AVR Studio is a professional Integrated Development Environment (IDE) for writing and debugging AVR applications in Windows 9x/NT/2000 environments. It can be downloaded free from [4].

Resources Needed for the Students

If this environment is to be used in a classroom for programming or system design purposes, the following material is needed:

- A PC running Windows with the compiler.
- A 5 V regulated power supply.
- Other electronic equipment (optional): soldering iron, multimeter...
- A cable for accessing the target system (~ €4).
- A breadboard (~ €10).
- Assorted components (~ €6).
- A processor and resonator (~ €10 from [7]).

If both the PC and the power supply are available in the laboratory, the rest of the parts come to a total cost of around €30.

The students should also be provided with some documentation similar to the following:

- The NF language description (included in [6]).
- Databooks for the AVR processor (such as [1], [2] and [3]). This depends on the depth of the work to do.
- Some standard book on digital design (such as [11]). Several good books with examples and suggestions are also available.

Conclusions

The compiler presented here offers a good alternative to other solutions when beginning microcontroller programming, its main advantages being a short learning curve, fast code turnout and minimal expenses.

The students can be encouraged to experiment with the system and, because of the low cost of the components involved, breaking or burning a design does not represent a mayor mishap.

The relative simplicity of the different parts of the code makes the system also suitable for a first course on compiler generators.

On the other hand, this compiler does not fully comply with ANS-Forth. Therefore, if compatibility with code already available is a mayor concern, it should not be used. However, the technique used in developing the compiler can equally produce an ANS-Forth system.

References

[1] Atmel Corporation. *8-bit AVR Microcontroller Application Notes*. http://www.atmel.com/

[2] Atmel Corporation. *8-bit AVR Microcontroller AT90S2313*. http://www.atmel.com/

[3] Atmel Corporation. *AVR Instruction Set*. http://www.atmel.com/

[4] Atmel Corporation. *AVR Studio*. http://www.atmel.com/

[5] de Ceballos, Federico. *A Machine Forth Specification based on the PSC1000 capabilities*. EuroForth 2000. http://dec.bournemouth.ac.uk/forth/euro/

[6] de Ceballos, Federico. *Un Entorno de Desarrollo para Aplicaciones de Alta Fiabilidad*. Universidad de Cantabria. 2001.

[7] Dontronics. *Atmel/AVR Microcontrollers*. http://www.dontronics.com/

[8] Rodriguez, Brad. *Moving Forth: a series on writing Forth kernels*. http://www.zetetics.com/bj/papers/

[9] Tasgal, John. *An Introduction to Machine Forth*. Forthwrite, Special Issue. 2000. http://www.fig-uk.org/

[10] Ting, C.H. *Improving eForth*. FORML 21. 1999.

[11] Wakerly, John F. *Digital Design. Principles & Practices*. Prentice Hall. 2001.