2. Documentation and Tools

Documentation

The amount of documentation that is related to Ada and ARM is huge. I will try to carefully explain all the bits that are used in this tutorial, but it is still a good idea to know where this knowledge comes from and where to look for more information.

For Ada, a good starting point is the Ada Information Clearinghouse:

http://www.adaic.org/

Depending on your personal learning style, you can find links to tutorials, books and even the complete Ada Reference Manual on this web site.

The GNAT Ada compiler comes with its own set of documentation (and that can be also found on-line) - please locate the GNAT User Guide after installing the package, you might want to check it for the complate explanation of all available compiler options. The GNAT package also contains documentation for the other tools that we will use.

For ARM in general, feel free to explore the ARM Infocenter:

http://infocenter.arm.com/

Note that Arduino/Genuino Zero, Arduino M0 and Nucleo-32 use the Cortex-M0 microcontroller, Arduino Due use Cortex-M3 and Nucleo-144 is based on Cortex-M7; there are dedicated branches for these core variants in the documentation tree.

We will also heavily use the documentation (datasheets and reference manuals) that is specific for each microcontroller in our development boards. These are:

- For Arduino/Genuino Zero or Arduino M0 - Atmel SAMD21 documentation.
- For Arduino Due - Atmel SAM3X8E documentation.
- For STM32 Nucleo-32 - STM32F031K6 documentation.
- for STM32 Nucleo-144 - STM32F746ZG documentation.

All of these documents are periodically revised by the vendors and they also happen to change file names and URL locations occasionally. It is therefore necessary to find them when needed, it is not useful to present their fixed paths in this tutorial.

Note that such documents typically have more than 1000 pages - there is no need for you to read it all (I will give more exact references whenever we will need them), but just having a quick look over the whole will help to navigate the document later on.

We will also need to know the pin mapping for our development boards - that is, the mapping between microcontroller pins and the board pins. Such mappings can be derived from the on-line schematics files in the case of Arduino/Genuino boards and are cleanly described in the leaflets that are part of the ST Microelectronics packages.

Last but not least, if you decide to learn a bit of SPARK and use it in your projects, the tool together with the language documentation is available from the AdaCore website, see below.

Just like with any other engineering activity, programming embedded systems requires a couple of tools. Ada has a very similar deployment model to that of C and C++ in that it relies on compilers and linkers to prepare the executable program, which can then be executed directly on some target platform. Similarly to C and C++, programs can be compiled either natively or in the cross-compilation scheme. Both approaches are explained below.

Native tools

Native tools are those that are available on your workstation and which produce executables that can be executed on the same workstation. Every major operating system provides such tools and in the case of Linux and Mac OS X they are available out of the box together with the system. The advantage of native tools is that they are straightforward to use, and they are well integrated with the working environment. In addition, the resulting executable or object files can be used directly and executed on the same computer.

Is it possible to program embedded systems using native tools? Some time ago the obvious answer to this question

"no", as typical embedded systems did not have enough capacity to run any development tools and they usually did not support any interaction at the level that programmers are used to (with text editors, file system, commands, etc.). But ARM has changed that and there is a growing number of mini-computers that are based on ARM processors and have reasonable support for interactive use. Raspberry Pi is one example, but in fact it represents the whole family of products with similar properties: small, relatively cheap, with Linux as an operating system and enough power to be used as small desktops.

Curiously, most Linux distributions (and the Raspbian system in particular, which is based on Debian) include compilers that can generate executable code that is appropriate for themselves as well as for other ARM processors, including the microcontrollers that are used in Arduino and Nucleo boards. In other words, it is perfectly possible to use Raspberry Pi (or other similar ARM-based computer) as a compilation system for ARM microcontrollers. Whether you will use it as a full desktop system with monitor and a keyboard or a remote resource to which you can connect via SSH from your main computer is only a matter of taste - in both cases you can treat Raspberry Pi as a valid compilation system.

Using Raspberry Pi as a software development platform might sound a bit "geeky", but I think this should be treated really as a pilot of what is coming to the market in the near future. ARM-based laptops and desktops might become first-class citizens in the computing landscape and with their natural (native!) support for ARM programming, we can treat them as perfect solutions for our needs.

Another reason to consider Raspberry Pi (or similar) is when you worry about connecting your Arduino prototypes directly to your main computer. These are electronic toys and with enough cables sticking out of the board in every direction some things can always go wrong. Having a relatively cheap Raspberry as a host for connecting your prototypes can provide some safety net in case of short-cuts and overloads.

In any case, Raspberry Pi (or similar) supports the GNAT compiler, which can be installed with this simple command:

```
$ sudo apt-get install gnat
```

After this, all necessary commands for compiling and linking programs (explained below) are available directly, with no additional configuration steps. This is a very comfortable solution.

Cross compilation

If you already have the Arduino Zero/M0 or Due boards, there is also a high chance that you have downloaded the Arduino IDE. This is a software package that contains tools which can produce executable files for use on the board. There are also IDEs that can be used with STM32 chips and several of them are recommended on the STMicroelectronics website and they also have the capability to produce executables for the target platform. The important thing here is that those final executable files cannot be executed on your desktop computer - the compiler produces programs for some other platform and this is where the "cross-" in "cross-compilation" comes from.

The advantage of cross-compilation is that you can do all the work on your single favourite operating system. The disadvantage is that the necessary tools might be available on some paths that are difficult to locate or might have names that are slightly changed in order not to conflict with the native tools. For example, the linker, which would be natively available with this simple command:

```
$ ld
```

is named `arm-eabi-ld` if installed from the GNAT package on Windows and in Arduino IDE installed on Mac OS X the same linker is available under the path:

```
~/Library/Arduino15/packages/arduino/tools/
    arm-none-eabi-gcc/4.8.3-2014q1/bin/arm-none-eabi-ld
```

As you see, there is some price to be paid with cross-compilation tools and with no single naming standard it takes a bit of exploration to locate all of them. Fortunately, the GNAT package keeps all the tools in the same directory (for example, `C:\GNAT\2015\bin`), so if you find one of them, the others are there as well.

The GNAT cross-compiler that can compile Ada code and produce executables for ARM can be downloaded from several places. One source is the libre website:

http://libre.adacore.com/

which contains tools for several major operating systems - you should select the "ARM ELF format" package for your operating system. There are also independent packages contributed by community. If you decide to use any of these packages, please follow installation instructions that are provided for them. The Libre site is also where you will find the SPARK tool, together with its documentation.

In any case, if you decide to use cross-compilation tools, note that in this tutorial I will always use short and natural names (as for the native use) - you will have to find their equivalents in your particular cross-compilation package.

Compiler

The compiler that we will use is part of GNAT, and can be invoked with this command:

```
$ gcc
```

Yes, this looks like a C compiler, but in fact it is a driver for a whole family of compilers. GNAT is a package that allows one to use the same command to compile C, C++ and Ada programs. This is very convenient.

Make sure that you can find and run this command on your system (either natively or as a cross-compiler, which in GNAT is named `arm-eabi-gcc`).

Linker

With GNAT, the linker is common for C, C++ and Ada (we will make use of this later on) and is available as:

```
$ ld
```

The linker is a tool that can gather many compiled files and, with the directions from a dedicated script file, can manage layout of all program elements in memory, as appropriate for the given chip. Make sure that you can find and run this tool.

Name inspector

This is a simple program that can list all symbols used in object and executable files. We will occasionally use it to trace how our source code is compiled and linked into executable programs. This tool can be run as:

```
$ nm
```

Make sure you can run it.

Object copy

The object copy tool will be used to transform binary files from one format to another. It is normally part of the compiler suite and can be executed with this command:

```
$ objcopy
```

Object dump

We will also occasionally verify that our binary image files have exactly the layout that we expect. This program can be run with the simple command (on Linux and Mac OS X):

```
$ od
```

Unfortunately, there is no standard dump utility on Windows (and no such tool is provided in the GNAT package), but there are plenty of utilities that you can find on-line by searching "Windows hex viewer" or "hex editor". In this tutorial I will consistently use `od` for the purpose of viewing binary images.

Again, note that all commands above have short names that can be naturally available in a native environment and will be usually different (like hidden in some directories or having some prefix) in a cross-compilation scheme. Make sure that you can locate and run all of them.

Loader

Last but not least, we will need some tool to upload our binary images to the flash memory on the target board.

For Arduino M0 we will use the Avrdude loader, which exists as a standalone package and is also distributed as part of the Arduino IDE.

For Arduino Due the loader is called bossa and can be downloaded from here:

http://www.shumatech.com/web/products/bossa

It is also available as a package in most Linux distributions (the package might be called bossac or bossa-cli or something similar). You can also use the one that comes as part of Arduino IDE, if you decide to upload images from your desktop

computer.

In the case of Nucleo boards, we will use the OpenOCD utility to upload executable images to the target board (this can be also used for Arduino/Genuino Zero boards). The OpenOCD toolset can be installed either as a package on Linux or as part of the Arduino IDE (if the Zero board is installed via the board manager) or downloaded directly from the OpenOCD site:

http://www.openocd.org

Later in this book I will use either the OpenOCD package from the Arduino IDE, as it contains proper configuration script for the Zero board (in this case, it is convenient to copy the `openocd` binary together with all configuration scripts to some place that is easy to navigate, as the Arduino IDE packaging system bury its dependent tools in a horribly deep directory structure), or the recent package from the official OpenOCD repository. Note, however, that at the time of writing this tutorial the most recent official release of OpenOCD did not support the STM32F7 chip used in the Nucleo-144 board and I had to use the version compiled from the development source repository, combined with a bit of tinkering in the board configuration scripts. It should be expected that the official releases from 0.10 and higher will support the most recent STM32 microcontrollers out of the box.

Previous: Introduction, next: First Program.
See also Table of Contents.
