

universidade de aveiro



theoria poiesis praxis

GNU TOOLCHAIN FOR EMBEDDED DEVELOPMENT: BUILD OR BUY?

MARK MITCHEL, DIRECTOR OF OPEN SOURCE TOOLS, MENTOR GRAPHICS

António Valente

Introduction

Increasingly, embedded software developers are choosing the GNU toolchains for open source development.

Actually, Mentor® Embedded Sourcery™ CodeBench is a complete, reliable, and convenient GNU Toolchain

Of course , the core components of the toolchain are available as open source software, so developers, have the option to build the toolchain themselves .

Introduction: Toolchain

A GNU Toolchain is more than a compiler, it generally includes :

- C/C++ compilers
- Assembler
- Linker
- Runtime Libraries
- Debugger
- Debug Stub(s)
- Integrated Development Environment (IDE)

Building the toolchain

Building a complete GNU toolchain is a complex process.
We must:

- Create our build environment
- Obtain the source code
- Make decision about configuration options
- Understand and perform the build process
- Package the toolchain
- Validating the toolchain

Create our Build environment

- We must ask ourselves if we have the right tools to build our toolchain.
- Because the libraries on our built system will be linked into the toolchain, our toolchain will only run on machines compatible with ours.
- In order to support multiple host systems, we have two options:
 - a) Set up multiple environments
 - b) Build “Canadian Cross” toolchains

Obtain the Source Code

Before we can build the toolchain, we must obtain all the relevant source code. The first step should be to determine which version of each component will suit our needs.

While there is a single FSF source release of GCC (or any other toolchain component) with a given version number, there are a myriad patches to that release available, from various locations.

- Some of these patches may be useful for our toolchain, but others only introduce new problems of conflict.
- Unfortunately, there is no way to know if we have all the patches that we need

Configuration Options

- Some configuration options are purely a matter of preference, some affect performance but not correctness, and still others have a definitive right answer on a particular system.
- After building a GNU toolchain, we must validate it and ensure that we picked options that works correctly in our target environment.

Build Process

- The GNU coding standards suggest that all packages should be built with an uniform “Configure, Make, Make install” sequence.
- But we can find some variations:
 - between the compiler and the C library there are an interdependency, the GCC examines the C library to determine certain configuration parameter, but, of course, we can't build the C library until we have the compiler

Packing

Microsoft Windows :

- It's easy for the users to have a graphical installer that place the tools in the path, and manage an uninstallation

GNU/Linux:

- We may want support for Red Hat Package Manager(RPM) packages, witch are used for the most popular GNU/ Linux distributions.

Validating the toolchain

- Validating the toolchain is the next step to ensure that our toolchain will work, usually we use some techniques to test the compiler and related tools, such as :
 1. Compiling programs and running those programs in the target environment
 2. Compiling programs and inspecting the generated object file or executable image, without running the generated code
 3. Compiling fragments of a program with multiple compilers and linking the fragments together
 4. Compiling invalid program fragments and checking for appropriate error messages

Validating the toolchain

- Additional techniques that we should apply to test other components of the toolchain:
 - Interactive testing of the debugger
 - We should compile and debug programs using the debugger with our target system to ensure that all debugger functionality works as expected
 - Interactive testing of the IDE
 - We should check that all desired IDE functionality (including, in particular, integration with GDB, the GDB stub, and the target system) works as intended

GNU Regression Testsuits

Many of the GNU toolchain components include testsuits based on DejaGNU framework.

The DejaGNU GDB test suite performs live tests of debugger on a running target system to ensure appropriate interactive behavior.

To use each test suites, we must first develop a DejaGNU board configuration file for our target system.

Conformance Testsuits

We should use conformance testsuits to validate the behavior of our toolchain relative to published specification.

We may wish to seek out additional conformance testsuits to validate functionality specific to our intended use of the compiler, such as:

- Plum-Hall C and C++ Validations
- The OpenPOSIX Testsuite
- Mentor Embedded
- C++ ABI

Performance Testsuites

Performance testsuites are designed to provide data about the speed at which the code generated by the toolchain executes.

- It is helpful to find misconfigurations of the toolchain.
- It's useful to compare newer versions of the toolchain with older ones

The EEMBC benchmarks are widely used as measurements of embedded system performance

Testing Multiple Options

Once we validated the compiler using a single set of options, we should expand our “validation matrix”.

There are three important dimensions to the validation matrix

1. Target System
2. Level of Optimization
3. Host System

Analyzing & Correcting Failures

Having gathered data about which tests pass and which test fail, we must now evaluate the results.

We should categorize each of the failures we observe so that we can fully evaluate the quality of our toolchain:

1. Defects in tests themselves
2. Defects in the hardware platform
3. Resource limitations
4. Defects in the toolchain

Analyzing & Correcting Failures

If we found problems in the validation process or in the course of using the toolchain, we will need to :

1. Identify the component that is causing the problem
2. Debug that component
3. Get familiar with the code for the toolchain
4. Post the description of our problem to the public mailing list for the affected component asking for help
5. Correct the defect

When we have fixed the problem, we should consider contributing the change, that we made, to the public source repository, for the affected component, so that other can benefit from our improvement, just as we have benefited from theirs.

Conclusions

Actually there are pre-built toolchains (such as Mentor® Embedded Sourcery™ CodeBench) that fulfill the needs of embedded software developers.

Anyway if we really want to build ourselves a GNU toolchain we must look for all possible technical issues involved in building and validating the toolchain.