Building DSP/BIOS Programs in UNIX

Stephen Lau *Software Development Systems*

ABSTRACT

This application report presents a method for building DSP/BIOS™ programs on the UNIX™ platform. This application report indicates all the environmental variables and changes that need to be configured for proper UNIX-based DSP/BIOS program builds. Special care is taken to illustrate how configuration management systems interact with the UNIX tools. A case study based on Code Composer Studio™ tutorial code is included.

Contents

List of Figures

List of Tables

1 Introduction

Applications are typically created in a team environment. The modules of the application are architectured and assigned to team members. Typically, team members create a small program to test their module independent from the entire application. For purposes of revision control, modules are checked into a code repository in preparation for an application build. There are many revision control software packages available for UNIX™, which help make it a popular choice for application-build environments.

Once the application is built, it is often transferred to a different machine, usually Windows™-based, which provides a better environment for debugging on the target (see Figure 1).

This application report addresses how to integrate individual development into the team application development environment. This application report utilizes the “hello2” tutorial example as a case study. The case study target code is based on a TMS320C5000™ DSP platform, but is also applicable to the TMS320C6000™ DSP platform. The case study is done with a Windows machine equipped with Code Composer Studio™ IDE. The UNIX examples are based on Solaris™ running the C Shell. The revision control system is RCS. The code generation tools on both the Windows and UNIX machines are the same version.

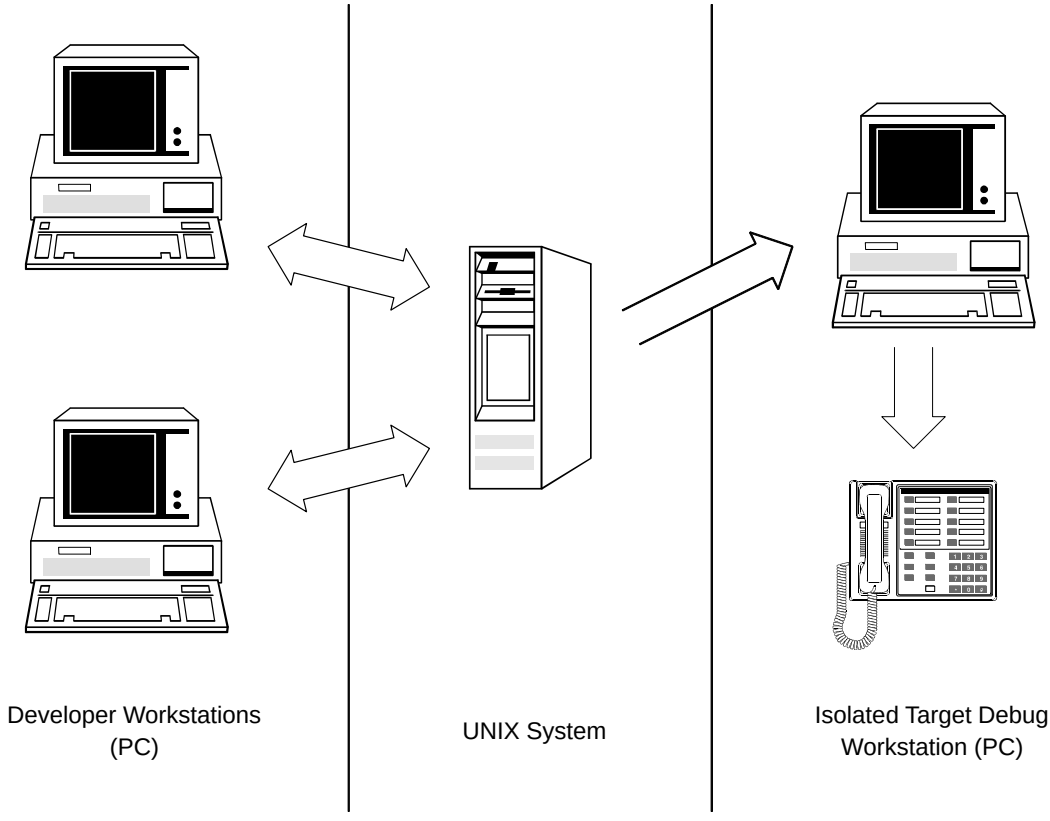

Figure 1. Development Overview

Code Composer Studio, TMS320C5000, and TMS320C6000 are trademarks of Texas Instruments.

Trademarks are the property of their respective owners.

2 Creation of a DSP/BIOS Program

DSP/BIOS™ programs are developed in Code Composer Studio IDE. The Configuration tool configures DSP/BIOS by allowing you to statically create objects and set parameters that are used in the application program. The tool actually generates code that is the framework for the user application, according to parameters such as memory configuration, interrupts, scheduling, statistics, and real time analysis.

2.1 Code Composer Studio IDE Versions

Code Composer Studio IDE is a mature product that has increased in functionality through several versions. The version of Code Composer Studio IDE can be found by going to the Help menu and selecting About Code Composer Studio. Table 1 shows some of the different versions and their associated chip support library feature.

Table 1. Code Composer Studio IDE Versions

Code Composer Studio Version	DSP/BIOS Processor Support	Chip Support Library
Code Composer Studio C6000 v1.0	C6000	No
Code Composer Studio C5000 v1.1	C5000	No
Code Composer Studio IDE v1.2x	C5000, C6000	No
Code Composer Studio IDE v2.0	C5000, C6000	Yes

This application report focuses on building DSP/BIOS programs for Code Composer Studio IDE v2.0. An Appendix section discusses the method for building legacy DSP/BIOS programs for prior versions. The functionality of DSP/BIOS in Code Composer Studio IDE v2.0 has been expanded through the addition of the Chip Support Library.

This application report does not discuss:

- Code Composer Studio IDE Version 2 exported make files.
- Code Composer Studio IDE Version 2 integrated configuration management.

DSP/BIOS is a trademark of Texas Instruments.

2.2 Case Study Introduction

The “hello2” example is one of the most basic DSP/BIOS programs available and is a good test of a code development environment. The “hello.c” program contains the “main” loop and a LOG_Printf to a trace window on the host.

The configuration tool generates six files (see Figure 2):

1. hello.cdb – The Configuration DataBase stores all of the parameters and information necessary to create a DSP/BIOS program. This does not have any linkable code, but contains information used to generate necessary code for the application. Code Composer Studio IDE uses this file for some of the real-time analysis displays (for example, execution graph).
2. hellocfg.cmd – Defines the memory map, sections, and libraries necessary for DSP/BIOS.
3. hellocfg.s54 – This assembly file defines the vector table and all of the statically created DSP/BIOS objects.
4. hellocfg.h54 – Definitions used by DSP/BIOS and statically created objects.
5. hellocfg.h – Definitions created by the Chip Support Library (CSL) and used by the statically created DSP/BIOS objects and handles.
6. hellocfg_c.c – Code for use by the Chip Support Library for use by DSP/BIOS or user applications.

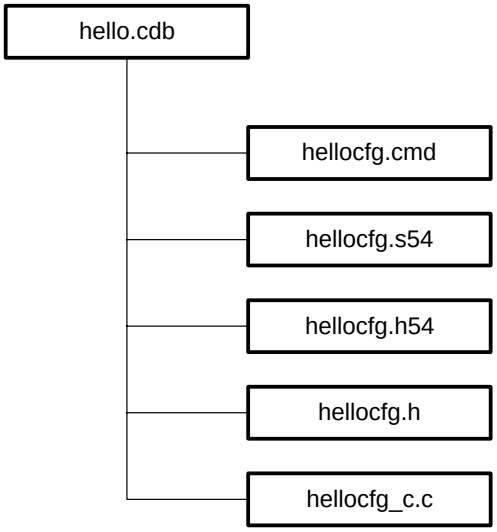

Figure 2. Configuration Tool Generated Files

If you are using the TMS320C6000 DSP, the file extensions would be: hello.s62 and hellocfg.h62.

Thus to create a DSP/BIOS program, the build process needs:

- User-generated code and header files. In this case, the only user-generated code is hello.c.
- Configuration tool generated code and header files. In this case, there are four files: hellocfg.s54, hellocfg.h54, hellocfg.h, and hellocfg_c.c.
- Linker command files. The linker command files instruct the linker how to map program code to memory. The Configuration tool generates a linker command file that must be used, hellocfg.cmd. If necessary, linker command files in addition to the generated one can be used.

3 Transfer Process

To move files from the Windows environment to the UNIX environment, the widely available File Transfer Protocol (FTP) or some other means can be used. Graphical FTP tools are available, as are command line tools. FTP operates in two modes, binary and text. All of our program files are text based, but all files generated by the compiler will be binary. The explicit details of the transfer process are in Appendix A.

4 Configuration Management

Configuration management is centered on controlling and tracking the changes that occur during development. The most basic unit in configuration management is the artifacts that are developed and worked with during development. Examples of artifacts include source (C, ASM) and header files. A revision control system provides artifact versioning, change control and coordination, and grouping. For the purposes of this application report, let us assume that all of the code has been checked into the database and checked back out without locks. Details of the configuration management process can be found in Appendix A.

5 Build Process

5.1 UNIX Setup

Before beginning the build process, we need to review the setup of our UNIX code generation tools. The only difference between the PC based code generation tools and the UNIX version are the actual executables. The DSP libraries are interchangeable. Thus, we can add the DSP/BIOS, CSL, and RTDX libraries to the UNIX code generation tools. This can be accomplished by transferring (through FTP or some other means) the appropriate library and header files. My code generation tools are installed into the “/usr/me/dsp/tool_dir”, so I have chosen to install my DSP/BIOS and CSL libraries into “/usr/me/dsp/tool_dir/bios” with two subdirectories “/include” and “/lib”. The RTDX files have also been transferred from my PC to the UNIX platform into the “/usr/me/dsp/tool_dir/rtdx” directory, again with two subdirectories. Figure 3 shows the UNIX directory structure used in this case study.

5.1.1 Path

In order to access the code generation tools from “/usr/me/dsp” we need to append the “/usr/me/dsp/tool_dir” to the path:

```
unixmachine{me}</user/me/dsp>13# set path=(~/dsp/tool_dir $path)
```

5.1.2 Environment Variables

Next, we need to be sure that the environment variables are properly setup for the C compiler (C_DIR) and Assembler (A_DIR):

```
unixmachine{me}</user/me/dsp>2# setenv C_DIR “tool_dir tool_dir/bios/include
tool_dir/rtdx/include”
```

```
unixmachine{me}</user/me/dsp>2# setenv A_DIR “tool_dir tool_dir/bios/include
tool_dir/rtdx/include”
```

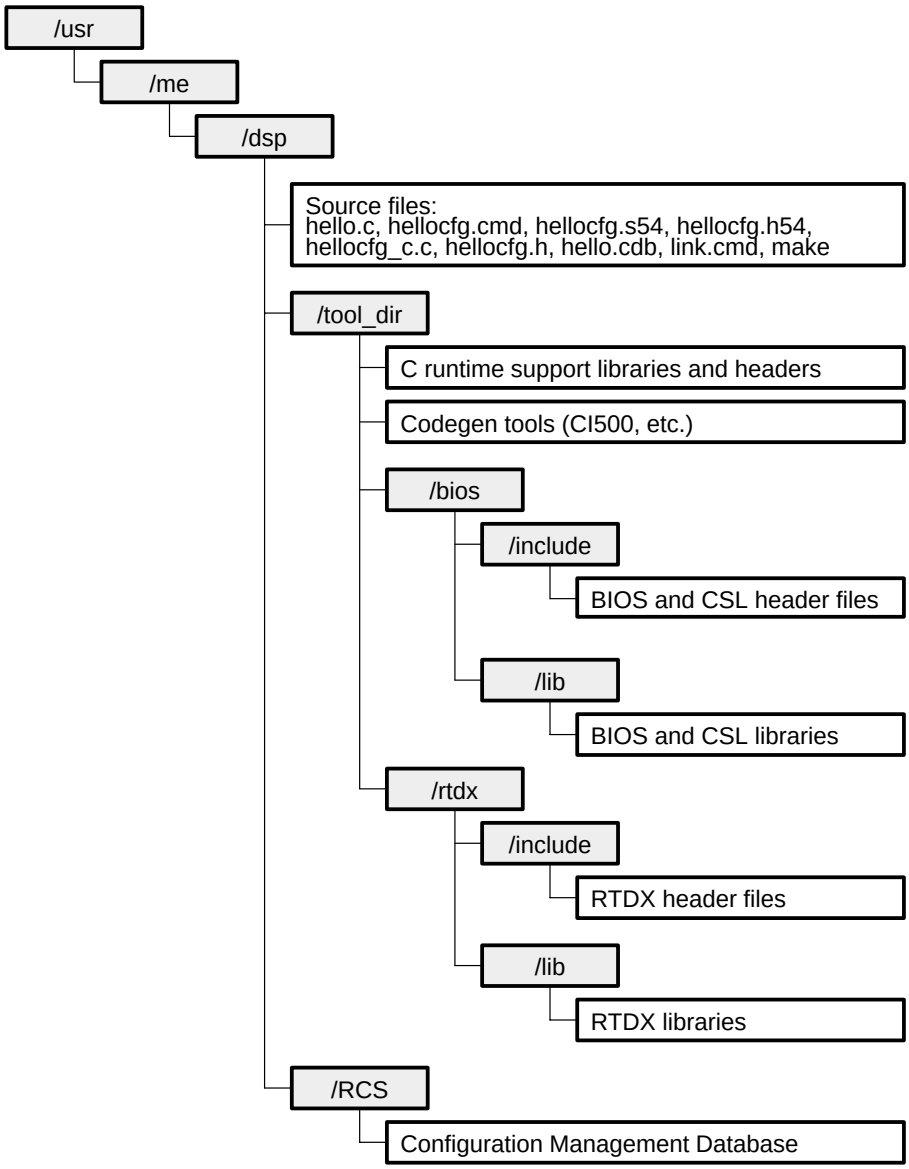

Figure 3. UNIX Directory Structure

5.2 Automating Builds

We want specific linker commands, so we use our own linker command file that calls the Configuration-tool-generated linker command file. We will create a linker command file called link.cmd and place it into the “/usr/me/dsp” directory. This linker command file specifies the name of our output file, and informs the linker of additional library directories.

```
–i tool_dir/bios/lib
–i tool_dir/rtdx/lib
–o hello.out
–x
–l hellocfg.cmd
```

The UNIX code generation tools are command line based. To further automate the build process, we create a makefile. GMAKE could have been used to control the make process, but for this program, a simple file that controls the compiler shell is sufficient. We name this file “make” and place it into the “/usr/me/dsp” directory. This file controls the shell and tells it which files to use for our application build. It also instructs the linker on which linker command file(s) to use:

```
/* Shell commands */
–g
–as
/* Files to process */
hello.c
hellocfg_c.c
hellocfg.s54
/* Linker command file */
–z link.cmd
```

These files should be checked into the configuration management database and checked back out unlocked. The application build is begun by invoking the compiler shell with our make file:

```
unixmachine{me}</user/me/dsp>93# cl500 –@make

[hello.c]
TMS320C54x ANSI C/C++ Compiler    Version 3.50
Copyright (c) 1996–1999 Texas Instruments Incorporated
   “hello.c”   ==> main
TMS320C54x ANSI C Codegen         Version 3.50
Copyright (c) 1996–1999 Texas Instruments Incorporated
   “hello.c”   ==> main
TMS320C54x COFF Assembler         Version 3.50
Copyright (c) 1996–1999 Texas Instruments Incorporated
 PASS 1
 PASS 2
 No Errors, No Warnings
```

```
[hellocfg_c.c]
TMS320C54x ANSI C/C++ Compiler    Version 3.50
Copyright (c) 1996–1999 Texas Instruments Incorporated
   ”hellocfg_c.c”   ==> CSL_cfgInit
TMS320C54x ANSI C Codegen         Version 3.50
Copyright (c) 1996–1999 Texas Instruments Incorporated
   ”hellocfg_c.c”   ==> CSL_cfgInit
TMS320C54x COFF Assembler         Version 3.50
Copyright (c) 1996–1999 Texas Instruments Incorporated
 PASS 1
 PASS 2

<hellocfg.s54>
TMS320C54x COFF Assembler         Version 3.50
Copyright (c) 1996–1999 Texas Instruments Incorporated
 PASS 1
 PASS 2
 No Errors, No Warnings
<Linking>
TMS320C54x COFF Linker            Version 3.50
Copyright (c) 1996–1999 Texas Instruments Incorporated
```

At the end of this process a binary file named hello.out is created in the “/usr/me/dsp” directory. This file may now undergo further processing (hex conversion) or used for debugging.

6 Transfer Back to Debug Station

Module developers usually work on standalone stations, with evaluation module hardware, or simulators. But the application build is a team effort, and is usually intended for particular target hardware. This hardware is usually connected to an isolated standalone station, to protect it from being loaded incorrectly.

Files can be moved from the UNIX environment to the Windows environment with FTP or some other means. All of the program files are text based, but all files generated by the compiler are binary. The details of the transfer process are in Appendix A.

The files we have on the standalone PC are:

- hello.c
- hello.cdb
- hellocfg.s54
- hellocfg.h54
- hellocfg_c.c
- hellocfg.h
- hello.out

The files are ready to be used by Code Composer Studio IDE for debugging. Alternatively, the code could be further processed (by Hex500 or some other tool) to become a ROM image or file loadable by a host processor.

Appendix A Transferring Files from the PC to UNIX

We need to transfer five files to build our application: hello.c, hellocfg.s54, hellocfg.h54, hellocfg_c.c, hellocfg.h, and hellocfg.cmd. In addition, we will transfer the text file hello.cdb to our UNIX machine for configuration management purposes.

To transfer files to the UNIX system, we change into the directory where the four files are located and begin the FTP process.

```
C:\ cd C:\ti\tutorial\sim54xx\hello2
C:\ti\tutorial\sim54xx\hello2>ftp unixmachine.ti.com
Connected to unixmachine.ti.com.
220 unixmachine FTP server (UNIX(r) System V Release 4.0) ready.
User (unixmachine.ti.com:(none)): me
331 Password required for me.
Password:
230 User me logged in.
```

Then, to actually transfer the files, we utilize the PUT instruction. PUT initiates a transfer from the Windows machine to the UNIX machine. We could change directories by using the CD command. We put our source files into the “/usr/me/dsp” directory. We need to transfer all four files:

```
ftp> cd dsp
250 CWD command successful.
ftp> put hello.c
200 PORT command successful.
150 ASCII data connection for hello.c (unixmachine.ti.com).
226 Transfer complete.
ftp: 1177 bytes sent in 0.00Seconds 1177000.00Kbytes/sec.
ftp> put hellocfg.s54
200 PORT command successful.
150 ASCII data connection for hellocfg.s54 (unixmachine.ti.com).
226 Transfer complete.
ftp: 38461 bytes sent in 7.75Seconds 4.96Kbytes/sec.
ftp> put hellocfg.h54
200 PORT command successful.
150 ASCII data connection for hellocfg.h54 (unixmachine.ti.com).
226 Transfer complete.
ftp: 2383 bytes sent in 0.00Seconds 2383000.00Kbytes/sec.
ftp> put hellocfg_c.c
200 PORT command successful.
150 ASCII data connection for hellocfg_c.c (unixmachine.ti.com).
226 Transfer complete.
ftp: 4232 bytes sent in 0.00Seconds 2331000.00Kbytes/sec.
ftp> put hellocfg.h
200 PORT command successful.
150 ASCII data connection for hellocfg.h (unixmachine.ti.com).
226 Transfer complete.
ftp: 1732 bytes sent in 0.33Seconds 17.02Kbytes/sec.
ftp> put hellocfg.cmd
200 PORT command successful.
150 ASCII data connection for hellocfg.cmd (unixmachine.ti.com).
226 Transfer complete.
ftp: 6279 bytes sent in 0.33Seconds 19.03Kbytes/sec.
```

```
ftp> put hello.cdb
200 PORT command successful.
150 ASCII data connection for hello.cdb (unixmachine.ti.com).
226 Transfer complete.
ftp: 213123 bytes sent in 0.99Seconds 19.03Kbytes/sec.
After we are done transferring, we can quit:
ftp> quit
221 Goodbye.
C:\ti\tutorial\sim54xx\hello2>
```

A.1 Preparation for Application Build

We have created two files, make and link.cmd that need to be added to the RCS database:

```
unixmachine{me}</user/me/dsp>5# ci make link.cmd
RCS/link.cmd,v  <-- link.cmd
enter description, terminated with single '.' or end of file:
NOTE: This is NOT the log message!
>> my own linker command
>> .
initial revision: 1.1
done
RCS/make,v  <--  make
enter description, terminated with single '.' or end of file:
NOTE: This is NOT the log message!
>> Header.
>> Compiler shell make file (not gmake)
initial revision: 1.1
done
unixmachine{me}</user/me/dsp>105# co make
RCS/make,v  -->  make
revision 1.1
done
unixmachine{me}</user/me/dsp>106# co link.cmd
RCS/link.cmd,v  -->  link.cmd
revision 1.1
done
```

Now that all of the files necessary for the application build are in the configuration management database, we tag all of these files with a group name. This group name uniquely identifies this particular application build. In this case, we have checked out all of the files before tagging them with a group name, but since they are not locked (read only), this is acceptable.

```
unixmachine{me}</user/me/dsp>109# rcs –nVersion_1 RCS/*
RCS file: RCS/hello.c,v
done
RCS file: RCS/hello.cdb,v
done
RCS file: RCS/hellocfg.cmd,v
Done
RCS file: RCS/hellocfg.h54,v
done
RCS file: RCS/hellocfg.s54,v
done
RCS file: RCS/hellocfg_c.c,v
```

```
done
RCS file: RCS/hellocfg.h,v
done
RCS file: RCS/link.cmd,v
done
RCS file: RCS/make,v
```

A.2 Transfer Back to Debug Station

The file generated by the UNIX code generation tools, hello.out, is binary and should be transferred in that mode. To debug on the debug station, we also need to transfer the following files: hello.cdb, hello.c, hellocfg.s54, hellocfg.h54, hellocfg_c.c, hellocfg.h, and hellocfg.cmd (optional). For this case study, we use FTP to transfer the files from the UNIX station to the PC. We utilize the GET instruction that initiates a transfer from the UNIX station to the PC. Also, the BINARY keyword can change the transfer mode to binary.

```
D:\debugcomputer >ftp unixmachine.rtc.sc.ti.com
Connected to unixmachine.rtc.sc.ti.com.
220 unixmachine FTP server (UNIX(r) System V Release 4.0) ready.
User (unixmachine.rtc.sc.ti.com:(none)): me
331 Password required for me.
Password:
230 User me logged in.
ftp> cd dsp
250 CWD command successful.
ftp> get hello.c
200 PORT command successful.
150 ASCII data connection for hello.c (1146 bytes).
226 ASCII Transfer complete.
ftp: 1179 bytes received in 0.06Seconds 19.65Kbytes/sec.
ftp> get hellocfg.cdb
200 PORT command successful.
150 ASCII data connection for hellocfg.cdb (11146 bytes).
226 ASCII Transfer complete.
ftp: 11179 bytes received in 0.06Seconds 19.65Kbytes/sec.
ftp> get hellocfg.s54
200 PORT command successful.
150 ASCII data connection for hellocfg.s54 (30266 bytes).
226 ASCII Transfer complete.
ftp: 31337 bytes received in 1.15Seconds 27.25Kbytes/sec.
ftp> get hellocfg.h54
200 PORT command successful.
150 ASCII data connection for hellocfg.h54 (2994 bytes).
226 ASCII Transfer complete.
ftp: 3113 bytes received in 0.22Seconds 14.15Kbytes/sec.
ftp> get hellocfg_c.c
200 PORT command successful.
150 ASCII data connection for hellocfg_c.c (2994 bytes).
226 ASCII Transfer complete.
ftp: 3113 bytes received in 0.22Seconds 14.15Kbytes/sec.
ftp> get hellocfg.h
200 PORT command successful.
150 ASCII data connection for hellocfg.h (30266 bytes).
226 ASCII Transfer complete.
```

```
ftp: 31337 bytes received in 1.15Seconds 27.25Kbytes/sec.
```

Now, we transfer the executable in binary mode:

```
ftp> bin
200 Type set to I.
ftp> get hello.out
200 PORT command successful.
150 Binary data connection for hello.out (158.218.96.107,4273) (54944 bytes).
226 Binary Transfer complete.
ftp: 54944 bytes received in 1.71Seconds 32.13Kbytes/sec.
ftp> quit
221 Goodbye.
```

A.3 Configuration Management

Artifact versioning manages the different revisions of a particular artifact in the database. For example, artifact versioning would allow us to differentiate between version 1.1 and 1.2 of a particular file. In addition to assigning a version number, artifact versioning typically allows a brief description to be included.

Change control and coordination tracks the creation of the various revisions of files from different team members, usually through a check-in/check-out mechanism. With the check-in/check-out mechanism, each team member has a personal workspace with copies of the files necessary for their work. In order to change a file, the person must check out the file to keep anyone else from modifying the file. When they are done, they can check the file back into the database and create a new revision of the file.

Artifacts are usually grouped together to form a finished product. Thus, in addition to an artifact version, a group of artifacts can be assigned a group revision (or tag).

In this case study, we are using the Revision Control System (RCS) that provides for check-in/check-out change control and versioning. Thus, after transferring all of the files to the UNIX build environment, we check the files into the RCS database while in a Telnet session:

```
unixmachine{me}</user/me/dsp>4# ci –l hello.c
RCS/hello.c,v  <––  hello.c
enter description, terminated with single ’.’ or end of file:
NOTE: This is NOT the log message!
>> Main user source file.
.>>
initial revision: 1.1
done
```

We used `ci -l` to specify that we wish to check in the hello.c file and immediately check it back out locked. This has been done to show how we can make small changes and have the RCS track them. Also note that this was the first time we have done a program build, so the initial message should be a description of the file, and not of the changes. Next, we need to check in all of the other artifacts:

```
unixmachine{me}</user/me/dsp>5# ci hellocfg.cmd hellocfg.h54 hellocfg.s54
hellocfg_c.c hellocfg.h hello.cdb
RCS/hellocfg.cmd,v  <--  hellocfg.cmd
enter description, terminated with single '.' or end of file:
NOTE: This is NOT the log message!
>> Generated linker command file
>>.
initial revision: 1.1
done
RCS/hellocfg.h54,v  <--  hellocfg.h54
enter description, terminated with single '.' or end of file:
NOTE: This is NOT the log message!
>> Generated Header.
>>.
initial revision: 1.1
done
RCS/hellocfg.s54,v  <--  hellocfg.s54
enter description, terminated with single '.' or end of file:
NOTE: This is NOT the log message!
>> Generated Assembly code.
.
>> initial revision: 1.1
done
RCS/hellocfg_c.c,v  <--  hellocfg_c.c
enter description, terminated with single '.' or end of file:
NOTE: This is NOT the log message!
>> Generated C code.
.
>> initial revision: 1.1
done
RCS/hellocfg.h,v  <--  hellocfg.h
enter description, terminated with single '.' or end of file:
NOTE: This is NOT the log message!
>> Generated Header file.
.
>> initial revision: 1.1
done
RCS/hello.cdb,v  <--  hello.cdb
enter description, terminated with single '.' or end of file:
NOTE: This is NOT the log message!
>> Generated CDB file.
.
>> initial revision: 1.1
done
```

Now, we check out all the previous artifacts without locking (read only):

```
unixmachine{me}</user/me/dsp>7# co hellocfg.cmd
RCS/hellocfg.cmd,v  -->  hellocfg.cmd
revision 1.1
done
unixmachine{me}</user/me/dsp>8# co hellocfg.h54
RCS/hellocfg.h54,v  -->  hellocfg.h54
revision 1.1
done
unixmachine{me}</user/me/dsp>9# co hellocfg.s54
RCS/hellocfg.s54,v  -->  hellocfg.s54
revision 1.1
done
unixmachine{me}</user/me/dsp>8# co hellocfg_c.c
RCS/hellocfg_c.c,v  -->  hellocfg_c.c
revision 1.1
done
unixmachine{me}</user/me/dsp>9# co hellocfg.h
RCS/hellocfg.h,v  -->  hellocfg.h
revision 1.1
done
unixmachine{me}</user/me/dsp>9# co hello.cdb
RCS/hello.cdb,v  -->  hello.cdb
revision 1.1
done
```

Now, if there were any last minute changes to the hello.c file, we could check that file back in and add a log message. Then, we check the hello.c file back out without locking (read only):

```
unixmachine{me}</user/me/dsp>11# ci hello.c
RCS/hello.c,v  <--  hello.c
new revision: 1.2; previous revision: 1.1
enter log message, terminated with single '.' or end of file:
>> I added some graffitti to the header to show I'd been there.
.>>
done
unixmachine{me}</user/me/dsp>12# co hello.c
RCS/hello.c,v  -->  hello.c
revision 1.2
done
```

Note that hello.c is now version 1.2. The database that RCS is using to store our code in our case study is located under "/usr/me/dsp/RCS", which is shown in Figure 3.

Now, all of our source files are in the database and are ready to be used for the application build process. There are many different configuration management tools available, many with a graphical interface.

IMPORTANT NOTICE

Texas Instruments and its subsidiaries (TI) reserve the right to make changes to their products or to discontinue any product or service without notice, and advise customers to obtain the latest version of relevant information to verify, before placing orders, that information being relied on is current and complete. All products are sold subject to the terms and conditions of sale supplied at the time of order acknowledgment, including those pertaining to warranty, patent infringement, and limitation of liability.

TI warrants performance of its products to the specifications applicable at the time of sale in accordance with TI's standard warranty. Testing and other quality control techniques are utilized to the extent TI deems necessary to support this warranty. Specific testing of all parameters of each device is not necessarily performed, except those mandated by government requirements.

Customers are responsible for their applications using TI components.

In order to minimize risks associated with the customer's applications, adequate design and operating safeguards must be provided by the customer to minimize inherent or procedural hazards.

TI assumes no liability for applications assistance or customer product design. TI does not warrant or represent that any license, either express or implied, is granted under any patent right, copyright, mask work right, or other intellectual property right of TI covering or relating to any combination, machine, or process in which such products or services might be or are used. TI's publication of information regarding any third party's products or services does not constitute TI's approval, license, warranty or endorsement thereof.

Reproduction of information in TI data books or data sheets is permissible only if reproduction is without alteration and is accompanied by all associated warranties, conditions, limitations and notices. Representation or reproduction of this information with alteration voids all warranties provided for an associated TI product or service, is an unfair and deceptive business practice, and TI is not responsible nor liable for any such use.

Resale of TI's products or services with *statements different from or beyond the parameters* stated by TI for that product or service voids all express and any implied warranties for the associated TI product or service, is an unfair and deceptive business practice, and TI is not responsible nor liable for any such use.

Also see: Standard Terms and Conditions of Sale for Semiconductor Products. www.ti.com/sc/docs/stdterms.htm

Mailing Address:

Texas Instruments
Post Office Box 655303
Dallas, Texas 75265