
ATC-10B Vocoder – Analyzing TI C54 Assembly Code Coverage using TI C55 Code Composer Studio (CCS) Tools

CONTENTS

- INTRODUCTION
- CODE COVERAGE REQUIREMENTS
- PORTING SOFTWARE (C54 to C55)
 - Porting Overview
- TEST TOOL OVERVIEW
 - TI Code Composer Studio
 - MASM55 Mnemonic Assembler
 - C5000 Analysis Toolkit (with Code Coverage Analysis)
- ATC10B CODE COVERAGE ANALYSIS
 - Proposed Approach
 - Requirements Compliance
 - Benefits/Limitations
- TEST TOOL VERIFICATION
- SUMMARY

This white paper explores the use of TI C55 Code Composer Studio (CCS) tools to perform code coverage analysis for the ATC-10B vocoder software, which is written entirely in TI C54 assembly.

INTRODUCTION

The ATC-10B voice compression algorithm, developed by Digital Voice Systems, Inc (Westford, MA), is specified by DO-224A for use in VDL Mode 3 air traffic communication equipment. The software is distributed by DVSI as a Texas Instruments' TMS320C5416 object library with accompanying header files, C54 assembly source files, and documentation. For DO-178B Level C software certification, a code coverage analysis must be completed on the ATC10B software.

The FAA is presently working with three avionic vendors to develop prototype VDL Mode 3 avionic radios. While two of these vendors have selected the Texas Instruments (TI) C54 platform for the vocoder implementation, the remaining vendor has selected a TI C55 platform. These two processor platforms are not identical, but are closely related. TI has provided tools that facilitate code porting from the C54 to the C55 platform. As a result of the vendor platform selections and processor similarities, DVSI has provided a common code set that can be built to run on either platform. This code is provided as C54 assembly.

Although a number of structural analysis tools exist for C-language software, a tool for native C54 assembly is not commercially available. However, a C55 processor family tool is available from TI that works with assembly source files. This white paper explores the use of TI C55 Code Composer Studio Tools to perform code coverage analysis on the ATC10B code.

CODE COVERAGE REQUIREMENTS

The requirements for the code coverage tool are presented in Table 1. The FAA, in a recent document aimed at acquiring or developing a C54 assembly language code coverage tool, listed these requirements.

Table 1: Code Coverage Requirements

#	Requirement
1	Coverage Analysis. The tool shall analyze TMS320C54x assembly language code for statement coverage against any given set of input test vectors.
2	Executed Statements. The tool shall identify, as executed all statements that did execute as a result of the input test vectors.
3	Not-Executed Statements. The tool shall identify, as not executed, any statements that did not execute as a result of the input test vectors.
4	Not-Evaluated Statements. Any statements, except source code comments, that cannot be evaluated as executed or not executed shall be identified as not evaluated.
5	Electronic Output. The tool output shall be in electronic form.
6	Output Listing Format. The tool output shall consist of annotated assembly language listings, or equivalent, to allow direct correlation of coverage results to the original assembly language source code.
7	Summary Information. The tool output shall summarize the total number of statements in the source code, the number of statements determined to be executed, the total number of statements not executed and the source files where they are contained, the total number of statements that could not be evaluated, and a list of all files that did not achieve 100% structural coverage. The tool shall have a feature to provide this information on a file-by-file basis as well as a feature to provide this information for the whole program.

PORTING SOFTWARE (C54 to C55)

The first (and most important) statement about porting the ATC10B C54 assembly to the C55 platform is that it already has been done and tested by DVSI and the C55 platform avionic vendor. Thus, the toolsets and process are already (or soon will be) validated. Notwithstanding, it is important for any system to properly integrate C54 source code on to a C55 platform. For example, the C55 must be setup to enable C54 Compatibility Mode. These integration issues must be addressed for the ATC10B code coverage test program.

The rest of this section provides background information for those who may not be familiar with the processors and/or the porting process. It also touches upon some of the system related integration requirements. For more information, see the list of reference documentation provided at the end of this white paper.

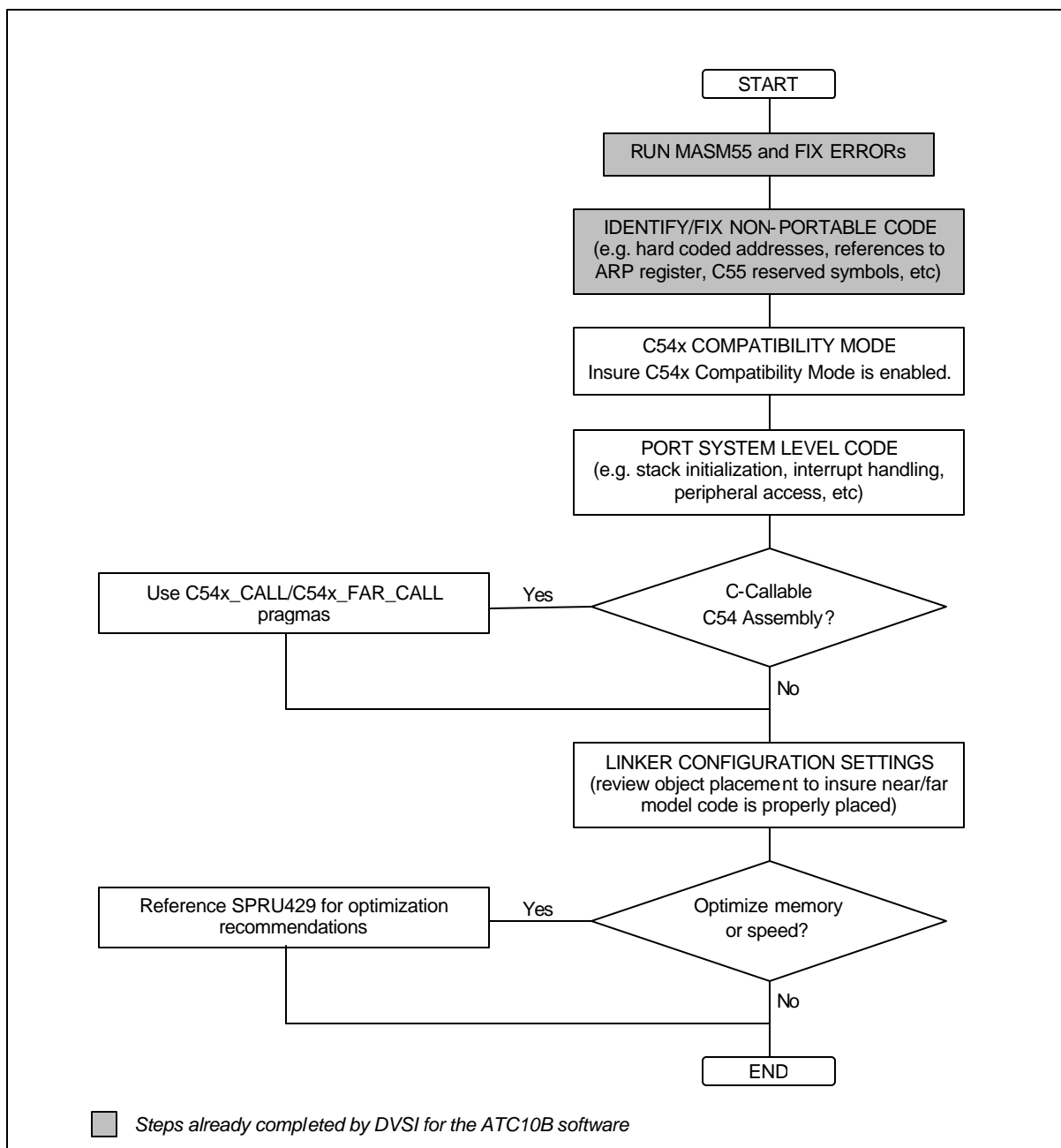
Porting Overview

The C54 and C55 platform processors are very similar. Texas Instruments designed the C55 platform was designed to support the migration and execution of C54 code. There are two key elements that facilitate the porting of C54 code:

- **Silicon Support:** The C55 processor can be configured to run in C54 Compatibility Mode. In this mode, the C55 mimics the operation of the C54 processor in executing instructions. For example, the saturation methods used are compliant with the C54 processor. The native C55 saturation modes are slightly different. In addition, special C54 bits in the C55 processor are enabled for use. The C16 bit, which covers either single wide or dual short arithmetic, is an example of a C54 bit that is replicated on the C55 platform.
- **Tool Support:** The MASM55 Mnemonic Assembler reads C54 and C55 assembly source instructions to produce C55 executable output objects. The assembly source languages are not identical, but very similar. For example, the C54 uses STx (store) instructions while the C55 uses MOV (move) instructions to accomplish the same objective. The C55 also includes an expanded register set, so there is a required mapping of C54 register mnemonics to C55 register mnemonics. Fortunately, the MASM55 tool automatically maps all of the C54 instructions to C55 instructions.

Texas Instruments provides a guide to assist with the migration of C54 code to the C55 platform (see *TMS320C54x-to-TMS320C55x Code Migration Reference Guide, SPRU429*). While the process is mostly automated, there are a few issues that must be addressed to insure the resultant port is successful. For example, hard-coded program addresses and offsets should be avoided in the C54 code, since the C54 processor uses word-based program addressing and the C55 uses byte-based program addressing. Hard-coded addresses and offsets will not point to the correct locations for code ported by the MASM55 assembler. In addition, system-related code (like stack initialization, interrupt handling, and peripheral I/O access code) must be ported outside of the MASM55 assembler. Figure 1 depicts the porting process flow.

Figure 1: Porting Process Flowchart



Fortunately, DVSI has already completed the significant portion of the porting process (see shaded areas of Figure 1). The ATC10B software consists of C-callable vocoder functions, but does not contain any internal system related code. Thus, the step related to porting of system level code is not applicable.

TEST TOOL OVERVIEW

This section provides an overview of the TI C55 Code Composer Studio (CCS) tools including the Integrated Development Environment (IDE), MASM55 Mnemonic Assembler, the C55 Simulator, and the Analysis Toolkit (performs code coverage analysis).

This section uses an example project, TST1, which includes ported C54 assembly code. The example program does the following:

- Calls a native C55 assembly program (System_vlinit) to insure that C54 Compatibility Mode is enabled
- Calls two C54 assembly functions (TestAdd and TestFill)
- Calls C-Language functions that read computer input files, execute a binary search tree (BST) algorithm, and print out results.

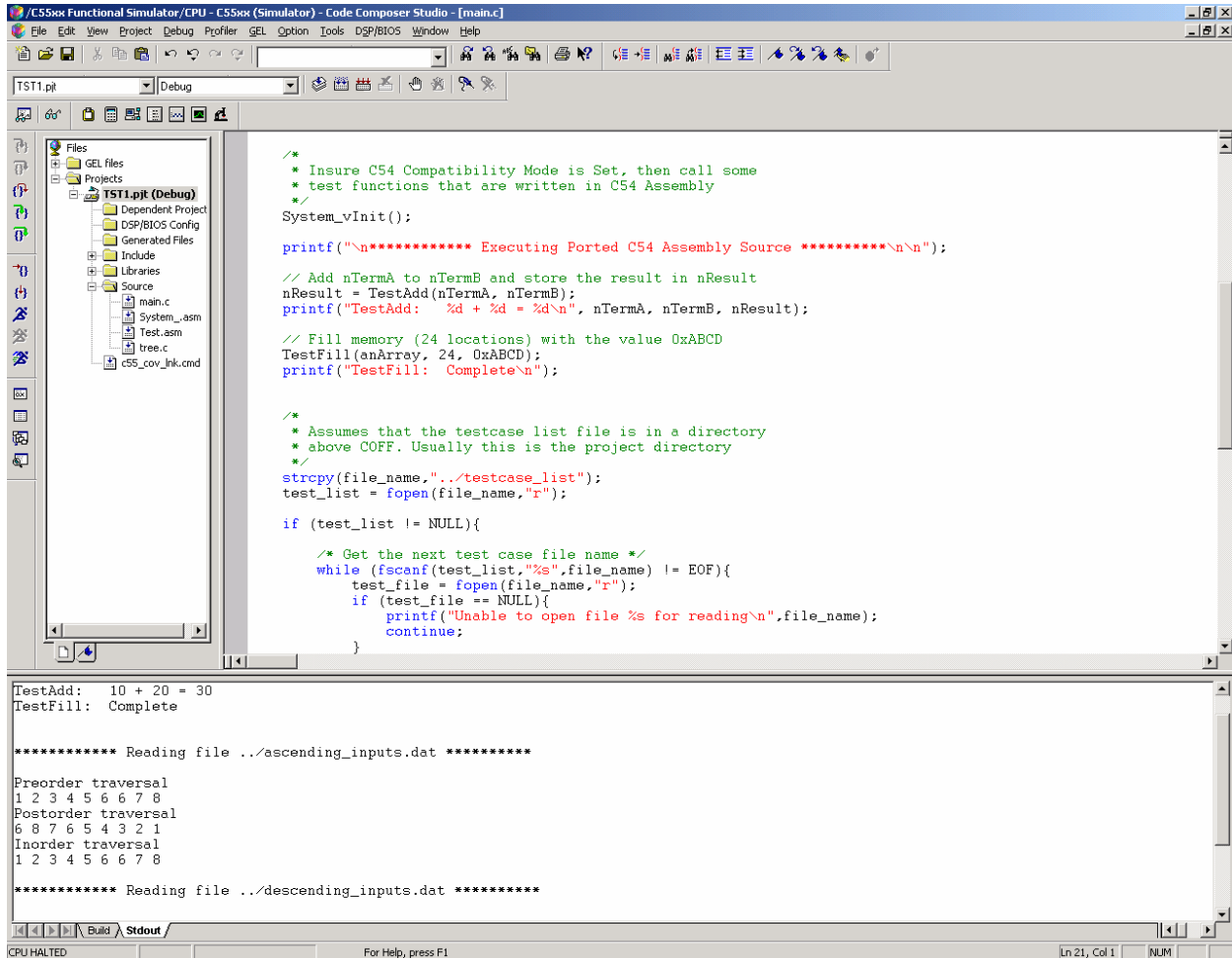
TI Code Composer Studio

Texas Instruments provides an Integrated Development Environment (IDE) called Code Composer Studio (CCS) that allows developers to code, build, and test TI DSP software programs.

The IDE is similar to other IDEs (like Microsoft Visual Studio) in that it supports the concept of a project. Files can be added to the project, edited using a built-in editor, and subsequently compiled, assembled and linked to create an operational executable. The built executable can be tested using either a built-in simulator or using a real hardware target (facilitated with TI's Real Time Embedded Exchange toolset). The IDE supports single stepping, breakpoints, and memory display/modification.

Figure 2 is a screen capture of CCS with the opened example project, TST1. Note: The example project includes the file, TEST.ASM, which includes C54 assembly functions. The bottom pane shows the printed output statements resulting from the execution of the test program using the C55 simulator.

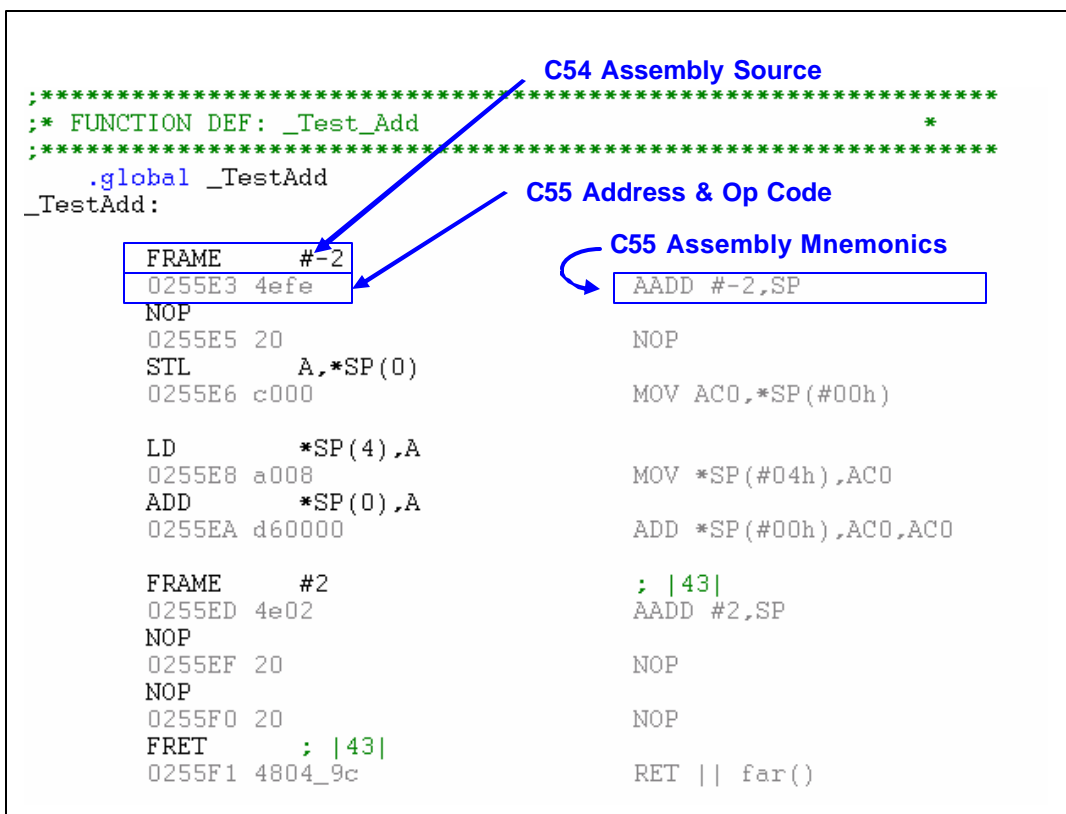
Figure 2: TI Code Composer Studio (CCS) Screen Capture



MASM55 Mnemonic Assembler

The MASM55 Mnemonic Assembler is automatically called by the CCS IDE when a C54 assembly source file is detected. Output listing files can be created that show the mapping of C54 instructions to C55 instructions. Figure 3 shows a view of mixed source and assembly that is available from within the CCS IDE after a project is built and loaded.

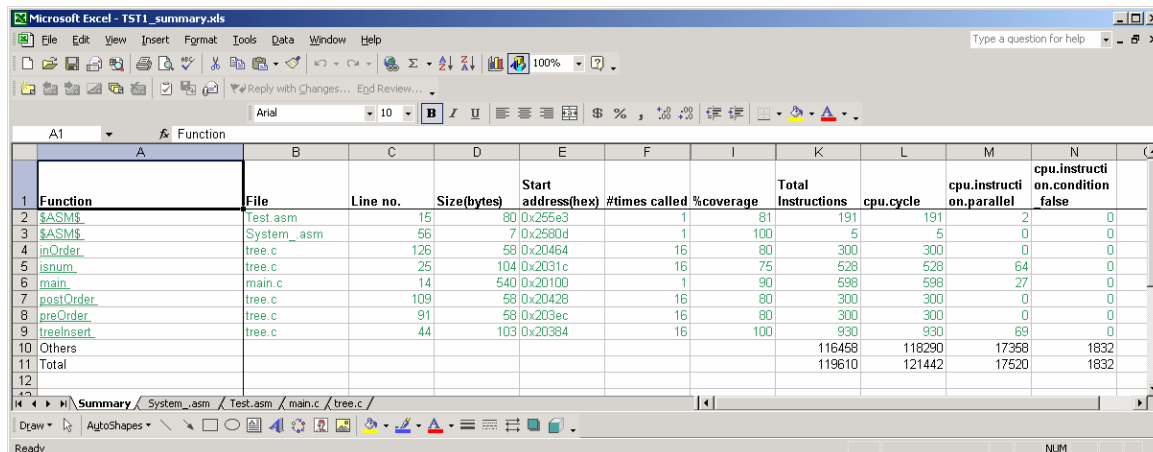
Figure 3: MASM55 Mnemonic Assembler (viewed from within IDE)



C5000 Analysis Toolkit (with Code Coverage Analysis)

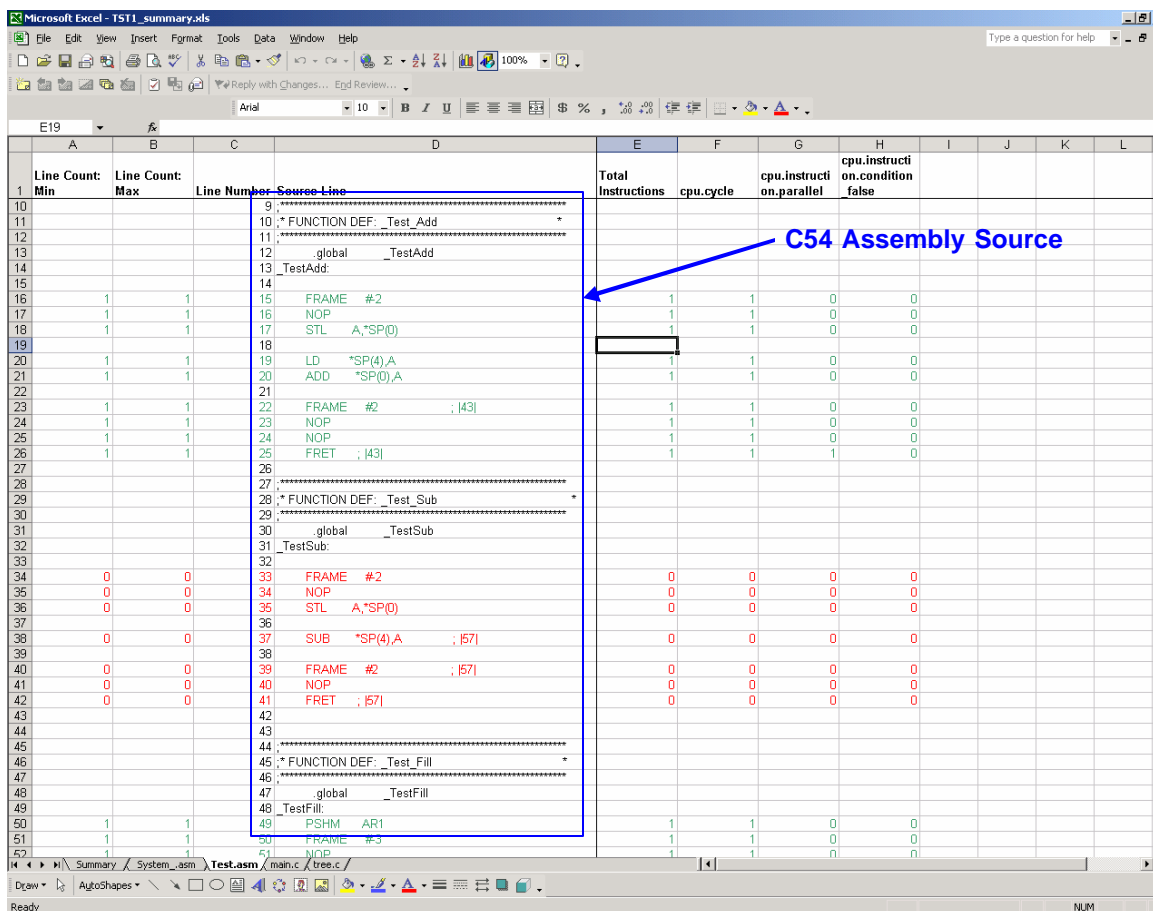
The CCS Analysis Toolkit (for C55 and C6000 platforms only) includes Code Coverage Analysis. The tool kit is not distributed with the CCS IDE, but is available as a free plug-in to registered users of CCS. When installed, the tool captures and analyses information in the trace buffers created by the C55 Simulator. It produces an Excel spreadsheet as well as comma separated variable (CSV) text files for use with post processing tools. Figure 4 is a screen shot of the SUMMARY worksheet created by the toolkit. The summary sheet has an entry row for each file in the project. A column is provided listing the code coverage percentage resulting from the simulated execution of the program. Clicking on the Function column entry takes you to the applicable file worksheet. Figure 5 is a screen shot of a portion of the TEST.ASM file worksheet. The executed code is shown in green. The not executed code is shown in red.

Figure 4: Code Coverage Output (SUMMARY Worksheet)



Function	File	Line no.	Size(bytes)	Start address(hex)	#times called	%coverage	Total Instructions	cpu.cycle	cpu.instruction.parallel	cpu.instruction.condition.false
\$ASM\$	Test.asm	15	80	0x255e3	1	81	191	191	2	0
\$ASM\$	System_.asm	56	7	0x2580d	1	100	5	5	0	0
InOrder	tree.c	126	58	0x20464	16	80	300	300	0	0
isnum	tree.c	25	104	0x2031c	16	75	528	528	64	0
main	main.c	14	540	0x20100	1	90	598	598	27	0
postOrder	tree.c	109	58	0x20428	16	80	300	300	0	0
preOrder	tree.c	91	58	0x203ec	16	80	300	300	0	0
treeInsert	tree.c	44	103	0x20384	16	100	930	930	69	0
Others							116458	118290	17358	1832
Total							119610	121442	17520	1832

Figure 5: Code Coverage Output (TEST.ASM Worksheet)



Line Count: Min	Line Count: Max	Line Number	Source Line	Total Instructions	cpu.cycle	cpu.instruction.parallel	cpu.instruction.condition.false
		9	*****				
		10	* FUNCTION DEF: _Test_Add				
		11	*****				
		12	global _TestAdd				
		13	_TestAdd:				
		14	*****				
		15	FRAME #2	1	1	0	0
		16	NOP	1	1	0	0
		17	STL A,*SP(0)	1	1	0	0
		18	*****				
		19	LD *SP(4),A	1	1	0	0
		20	ADD *SP(0),A	1	1	0	0
		21	*****				
		22	FRAME #2 ; 43	1	1	0	0
		23	NOP	1	1	0	0
		24	NOP	1	1	0	0
		25	FRET ; 43	1	1	1	0
		26	*****				
		27	*****				
		28	* FUNCTION DEF: _Test_Sub				
		29	*****				
		30	global _TestSub				
		31	_TestSub:				
		32	*****				
		33	FRAME #2	0	0	0	0
		34	NOP	0	0	0	0
		35	STL A,*SP(0)	0	0	0	0
		36	*****				
		37	SUB *SP(4),A ; 57	0	0	0	0
		38	*****				
		39	FRAME #2 ; 57	0	0	0	0
		40	NOP	0	0	0	0
		41	FRET ; 57	0	0	0	0
		42	*****				
		43	*****				
		44	*****				
		45	* FUNCTION DEF: _Test_Fill				
		46	*****				
		47	global _TestFill				
		48	_TestFill:				
		49	PSHM AR1	1	1	0	0
		50	FRAME #3	1	1	0	0
		51	NOP	1	1	0	0
		52	*****				

ATC10B CODE COVERAGE ANALYSIS

Proposed Approach

The proposed approach for verifying ATC10B vocoder code coverage requires the creation of a code test program (CTP) using the TI CCS IDE. The CTP will be written to input/output linear voice (PCM) and/or compressed voice (CVM) test vector files and to execute the ATC10B encoder/decoder software within the C55 simulator. The ATC10B C54 assembly source files will be added directly to the C55 project. The IDE automatically calls the MASM55 assembler for these files. In addition, the C55 simulator libraries support computer file input/output (IO) operations.

The CTP will initially read an external control file that includes a line for enabling encoding and/or decoding operations followed by a list of test input vector files. The generated voice output files provide a means of verifying the test program system implementation by using the bit exact properties of the algorithm.

To use the CTP, a user needs to have TI Code Composer Studio C5000 version 2.2 with the Analysis Toolkit Plug-In. The user modifies the CTP control file using a text editor to run the desired set of input vectors. Then, the user starts Code Composer Studio IDE, loads the CTP program into the simulator and executes the CTP program. The TI Analysis Toolkit Plug-In automatically generates an Excel spreadsheet listing the C54 assembly source files and code coverage results.

The ATC10B source code uses macros in the assembly source. Since the source file uses the macro, the code coverage tool lists the macro (and not the expanded assembly instruction list). If desired, the C54 preprocessor can be executed to expand all macros. These expanded assembly source files could then be added to C55 project to view code coverage within the macro code.

Finally, the CTP program and associated ATC10B code could be modified within the IDE simulated environment, if necessary, to stub out functions and force exception events to extend code coverage beyond that exercised by the available PCM/CVM input voice vectors.

Requirements Compliance

Use of TI CCS with the code test program fully meets the requirements identified by the FAA.

Table 2: Compliance with Requirements

#	Requirement	Compliance Commentary
1	Coverage Analysis. The tool shall analyze TMS320C54x assembly language code for statement coverage against any given set of input test vectors.	Compliant The analysis toolkit automatically reviews the trace file(s) generated by the simulation and analyzes code coverage.
2	Executed Statements. The tool shall identify, as executed all statements that did execute as a result of the input test vectors.	Compliant Executed statements are displayed in green along with information about the number of times the statement was executed.
3	Not-Executed Statements. The tool shall identify, as not executed, any statements that did not execute as a result of the input test vectors.	Compliant Not-Executed statements are displayed in red .
4	Not-Evaluated Statements. Any statements, except source code comments, that cannot be evaluated as executed or not executed shall be identified as not evaluated.	Compliant Not-Evaluated statements are displayed in black .
5	Electronic Output. The tool output shall be in electronic form.	Compliant An output Excel spreadsheet and corresponding set of comma separated variable (CSV) files are automatically generated documenting the test results.
6	Output Listing Format. The tool output shall consist of annotated assembly language listings, or equivalent, to allow direct correlation of coverage results to the original assembly language source code.	Compliant The Excel spreadsheet includes a row for every line in the source code. The C54 assembly source code statement and the number of times executed are listed on the same row.
7	Summary Information The tool output shall summarize the total number of statements in the source code, the number of statements determined to be executed, the total number of statements not executed and the source files where they are contained, the total number of statements that could not be evaluated, and a list of all files that did not achieve 100% structural coverage. The tool shall have a feature to provide this information on a file-by-file basis as well as a feature to provide this information for the whole program.	Compliant The Excel spreadsheet includes summary table listing each source file and the code coverage percentage. Clicking on a file automatically brings up the file details. While the total number of statements executed, not-executed, and not evaluated aren't specifically listed by the analysis toolkit output, this information can easily be derived from the spreadsheet information. A post processing program can be written to read in the CSV files to automatically generate this information.

Benefits/Limitations

Benefits Include:

- **Base C55 Port Complete:** DVSI has already reviewed the C54 assembly code and removed/modified any problematic code associated with C54 to C55 porting. The resultant code has already been tested on the C55 platform one of the avionic radio vendors. Hence, all of the porting issues with the application code have already been addressed. Note: The CTP still needs to address system integration issues.
- **Commercial Tool Use:** Given the TI C55 Code Composer Studio Tools are available as commercial products with an established user base, there can already be a level of confidence in the reported results. In addition, since the tool is already available, the tool verification can be initiated immediately.
- **Deterministic Rather Statistical Coverage:** Since the analysis tool views the execution trace buffer, all executed code is detected on a single run of the program. Statistical methods that use interrupts to capture program counter states must be run over a period of time. Further, there is no guarantee that all executed instructions will be captured by the interrupting process.
- **Completely Automated:** The code coverage analysis tool is completed automated. One or more input vector files can be processed and electronic output files are automatically created.
- **Source Code Instrumentation Not Required.** Unlike other approaches, special instrumentation code does not need to be compiled and linked with the target code.

Limitations include:

- **Operating System and Peripheral Access Coverage:** While application code is straight forward to port, operating system and peripheral IO code may be difficult or impossible to port from the C54 and/or test. Note: Other code coverage alternatives may also have difficulty in achieving effective operating system and peripheral access coverage. Fortunately, the ATC10B code does not utilize any system or peripheral access instructions. This tool may be helpful for system-related C55 code (used by one avionic radio vendor), but will be less applicable for testing system-related C54 code (used by the other two vendors).

In addition, to the benefits and limitations described above, it should be noted that the TI C55 Code Composer Studio Toolset does have some reported bugs (see the TI website for more information: www.ti.com). Fortunately, none of the bugs have appeared to hinder the initial efforts associated with ATC10B code coverage analysis.

Note: Initial testing of the ATC10B files have already been successfully tested (with a zero vector linear input buffer) for code coverage analysis using this approach.

TEST TOOL VERIFICATION

It is anticipated that verification of TI C55 Code Composer Studio Toolset may be easier to accomplish (with a higher level of confidence) than verification of a tool created from scratch. While tool verification is not necessarily a foregone conclusion, commercial use of the toolset probably has likely resulted in the identification and correction any significant tool problems. In addition, the MASM55 Mnemonic Assembler (part of the C55 Code Composer Studio Toolset) will already need to be approved to support the certification efforts of the C55 platform avionic vendor.

The additional tools that will require verification include:

- C55 Simulator
- C5000 Analysis Toolkit.
- Code Test Program (used to exercise the ATC10B code)

Given that the ATC10B algorithm is bit-exact, the CTP voice output files can be compared with the other vocoder algorithm implementations to insure the code test program (CTP) has properly integrated the ATC10B code.

SUMMARY

In summary, use of the TI C55 Code Composer Studio Tools should expedite ATC10B C54 assembly code coverage analysis. The simulated environment and detailed output listings will facilitate identification of a suitable set test vectors and will support software unit level testing to expand code coverage (if required). A copy of the Excel output file for the example project (referenced in this paper) is available by email by sending a request to: kdevito@cie-eng.com.

For more information, reference the following documentation:

- Software Considerations in Airborne Systems and Equipment Certification (RTCA/DO-178B), RTCA Incorporation
- TMS320C54x-to-TMS320C55x Code Migration Reference Guide (SPRU429), Texas Instruments
- TMS320C55x Instruction Set Simulator Technical Overview (SPRU599), Texas Instruments
- Analysis Toolkit for Code Composer Studio – v2.2 User's Guide (SPRU623), Texas Instruments
- Code Coverage and Multi-event Profiler User's Guide (SPRU624), Texas Instruments

CIE Engineering, Inc.
600 Maryland Avenue, S.W.,
Suite 740
Washington, DC 20024
www.cie-eng.com
(202) 484-2298