Instructions to Benchmark C55 DSP Library

ABSTRACT

DSP Library (DSPLIB) is a collection of optimized C55x assembly functions that implement a wide variety of DSP functions. The *TMS320C55x DSP Library Programmer’s Reference User's Guide* (SPRU422) provides cycle estimation for the execution of each function in the library. This work enables the user to measure the performances of a sub-set of the DSPLIB functions executed on real hardware, and with some additional hardware support to measure the actual power consumption of the DSP device when it executes any of these functions.

Project collateral and source code discussed in this application report can be downloaded from the https://git.ti.com/apps/c55x-benchmarks. The C55x DSP library functions can be downloaded from http://software-dl.ti.com/libs/c55_dsplib/latest/index_FDS.html.

Contents

List of Figures

1 Introduction

There is a dedicated Code Composer Studio™ (CCS) project for each benchmarked function, the hardware that is used is TI EZDSP5535 (a low-cost platform). Each project is based on the DSPLIB unit test for the corresponding optimized function.

To ensure easy portability of the projects. The code is distributed as source code. This document provides very detailed step-by-step instructions on how to build and run the various projects. The only requirements to build and run the projects are Code Composer Studio (CCSv6), library source code and the compressed file that contains the project sources, as well as the EZDSP5535 EVM.

The following is a list of the functions that are part of this benchmark project:

- Complex values FFT (two cases): software only and using of the FFT accelerator
- Real values FIR (two versions): a regular version and a faster version with some limitations
- Real values Convolution
- Real Values Auto-Correlation
- Two Cases of the Real Value maximum function: finding the maximum value only and finding the maximum value and its index
- Two Cases of the Real Value delay LMS filter: regular one and faster one with some limitations

Adding additional functions for benchmarking is a simple straightforward procedure.

1.1 System Requirements

- TMS320C5535 eZdsp USB kit
- CCS v6 or newer needs to be installed on the Windows machine (including support for the C55 family)
- Compressed-projects-source file - that is a file with the compressed source
- C55x DSPLIB optimized routines loaded from https://git.ti.com/apps/c55x-benchmarks

2 Algorithm Flow

All benchmark projects share the same algorithm flow. Figure 1 describes the algorithm flow.

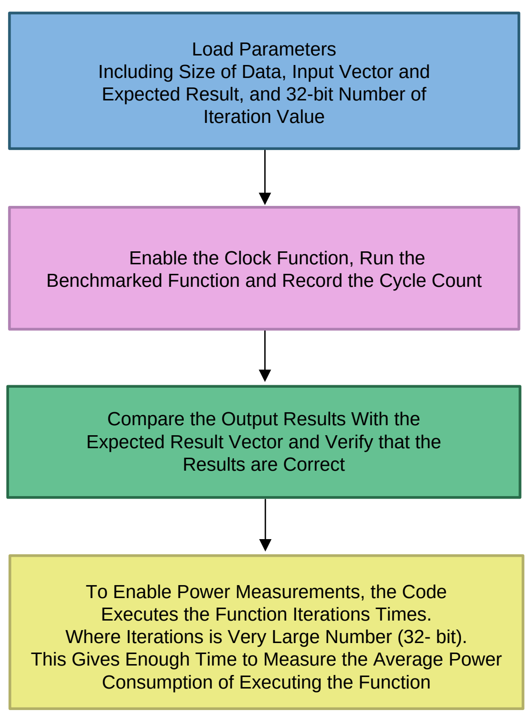

Figure 1. Algorithm Flow

3 Load and Uncompress the Project File

Load benchmark code from the TI public git server. From https://git.ti.com/apps/c55x-benchmarks, click on the master branch at the upper part of the page. At the right side of the page just below Source tree, a tar.gz file of the release is available to download.

The project can be installed in any directory of the Windows 7 machine. The code was tested only in the Windows 7 environment. After the user uncompresses the downloaded file, apps-c55x-benchmark-master.tar.gz, the following directories will be created (see Figure 2).

NOTE: Several tools are available to uncompress tar files (7_zip, tar utility on LINUX emulation, winzip and others).

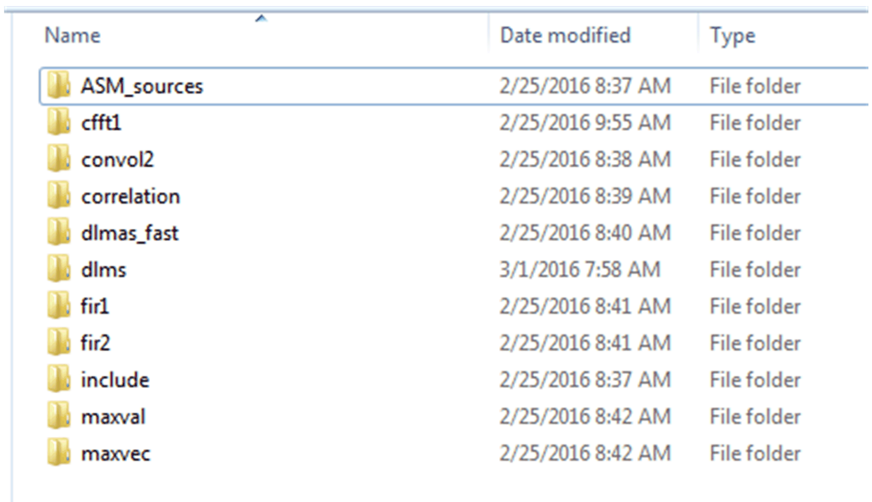

Figure 2. Directories From Uncompressed apps-c55x-benchmark-master.tar.gz File

Table 1. Description of the Sub-Directories

ASM_sources	Contains sources of all the library optimized functions. The sources of all the optimized functions must be loaded into this directory.
cfft1	Contains the code and data for all FFT benchmarks, including software only and using the HWAFFT accelerator
convolve2	Contains the code and data for convolution
correlation	Contains the code and data for auto-correlation
Dlmas_fast	Contains the code and data for fast delayed least mean square (LMS) filter
Dlma	Contains the code and data for standard delayed LMS filter
fir1	Contains the code and data for the regular fir filter
fir2	Contains the code and data for fast fir filter (even number of elements)
include	Contains three include file. The path to this directory should be defined in the project properties
maxval	Contains the code and data for finding the maximum value in a sequence
maxVec	Contains the code and data for finding the maximum value and the index of maximum value in a sequence

Load the optimized library functions from http://software-dl.ti.com/libs/c55_dsplib/latest/index_FDS.html to the ASM_sources directory. Follow the directions from *SPRC100-C55_DSPLIB-03.00.00.03-Setup.exe* on how to install the DSPLIB library on your computer. After installing the library, go to *directory c55_dsplib_03.00.00.03\55x_src* and copy all of the asm files into the ASM_sources directory.

4 Building the CFFT Project

The following instructions show how to build and run the CFFT project. As was mentioned Section 1, to ensure easy porting, the software package contains only source code and auxiliary files (linker command files).

Assume that the software package is uncompressed into a directory called myDirectory and it looks like what is shown in Figure 2. Use the following steps to build and run the CFFT project.

1. Start CCS and define a new project.

   Make sure that the target is EZDSP5535, the project is empty (no main.c file), and that you are using the latest TI compiler version available to you, see Figure 3.

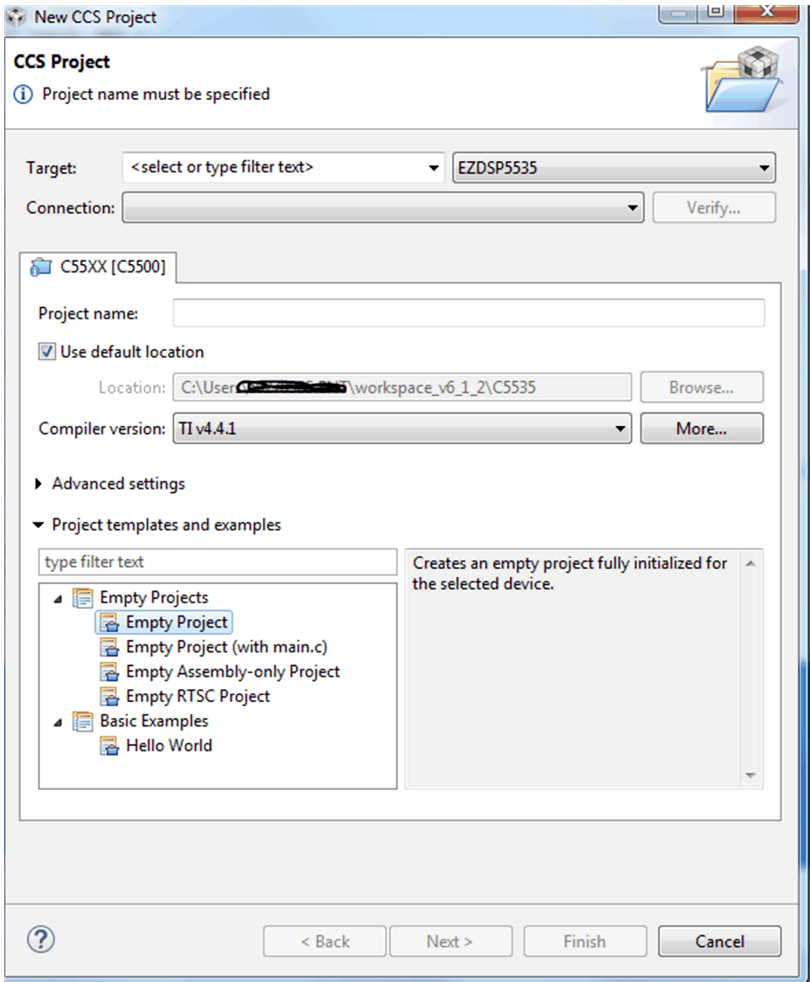

Figure 3. New CCS Project

   Name the project (cfft1 was chosen in the screenshots, but any name can be used) and click finish.

2. Add files to the project.

   Open the project that was just created and delete the linker command file. The source code includes the linker command file for each project. Click on the *c5535.cmd* file and delete it.

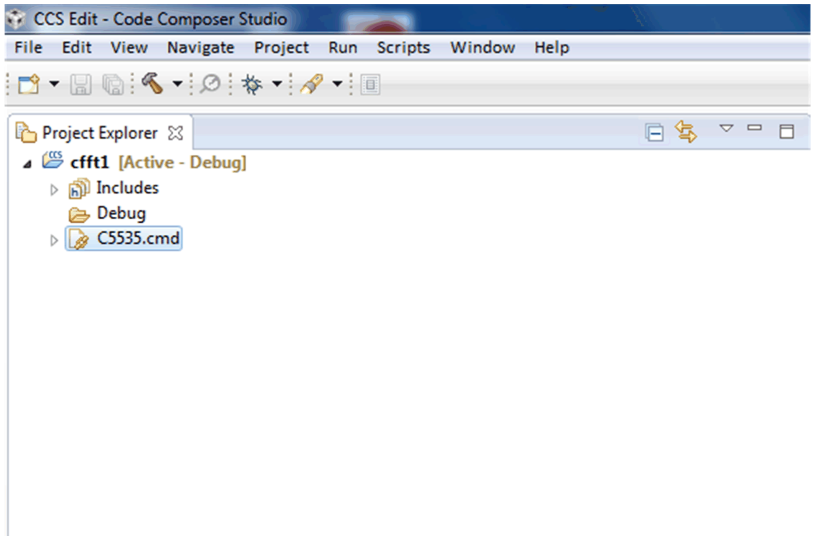

Figure 4. Remove Linker Command File

Next, add all of the source files in the cfft1 directory to the project. There are two ways to add files to the project: copy the files or link the files. For these projects, it is recommended to copy the source files. The ability to change the project files without affecting the original source files is allowed. To do so, right clock on the project and choose 'add files'. In the dialogue box, navigate to the cfft1 directory, select all the files, and click open.

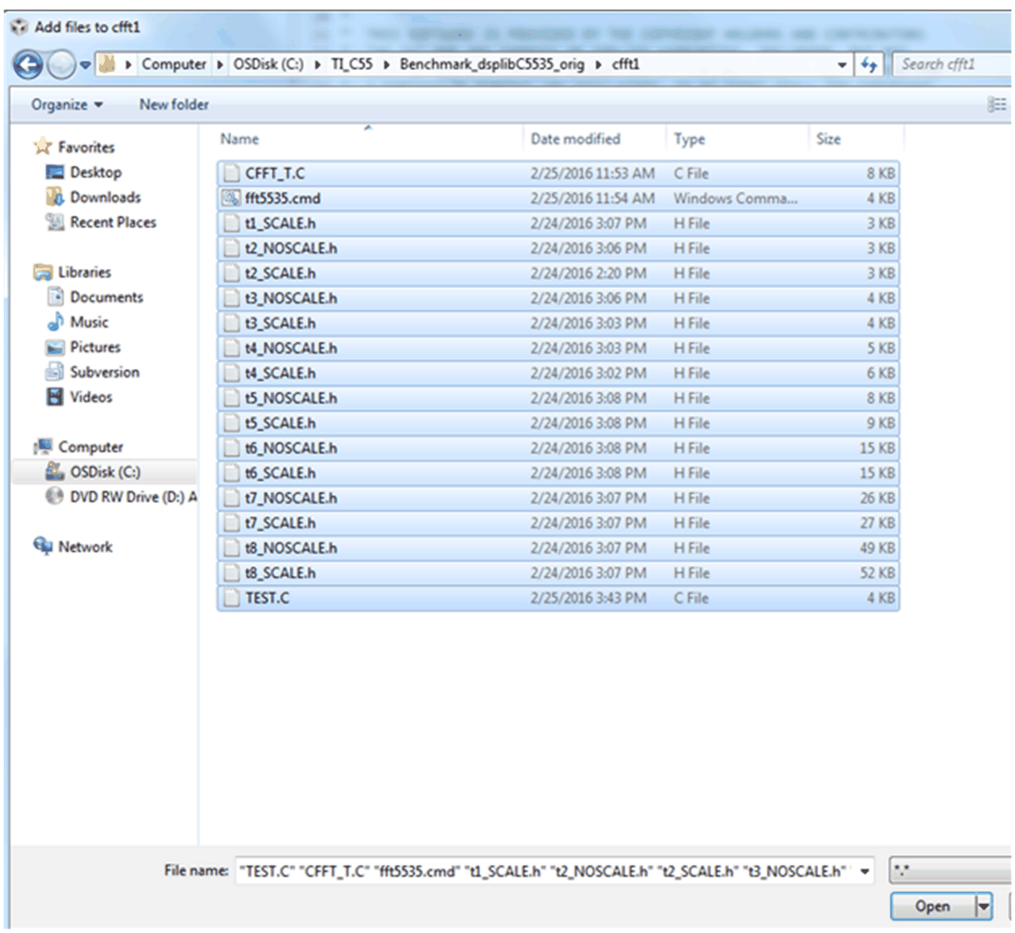

Figure 5. Add Files to cfft1

A new dialogue box will open. Choose between linking the files and copying them. 'Copy' is selected in Figure 6. Click OK.

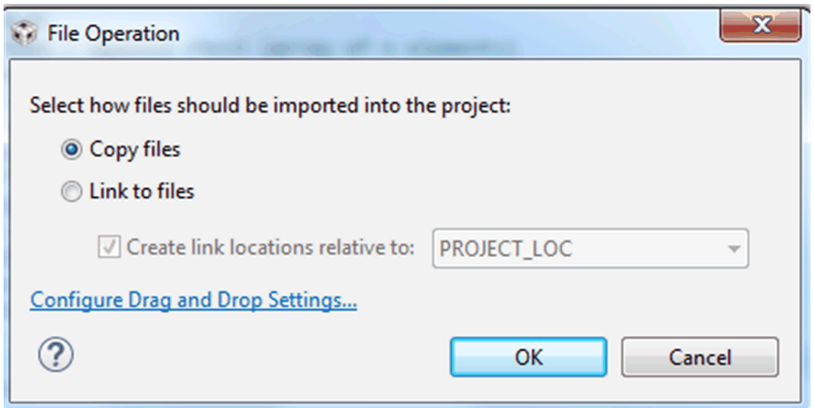

Figure 6. Link or Copy Files

The test code files were added to the project. The optimized library files that are written in assembly are needed. If the DSPLIB was already downloaded, the assembly files are in the c55_dsplib_03.00.00.03\55x_src directory. If not, all assembly routines are part of the project packages in the ASM_sources directory. Regardless of the location of the original optimized function files, these files should be linked to the project and not copied. There is no need to modify the optimized assembly code functions. For the CFFT project, four files (shown in Figure 7 and Figure 8) are needed (cbrev.asm is the bit reversal function, cfft_scale.asm and cfft_noscale.asm are two fft routines for executing fft with or without block scaling). Block scaling consumes more cycles, but provides a better dynamic range and lower truncation errors. The assembly code twiddle.asm contains the twiddle factors.

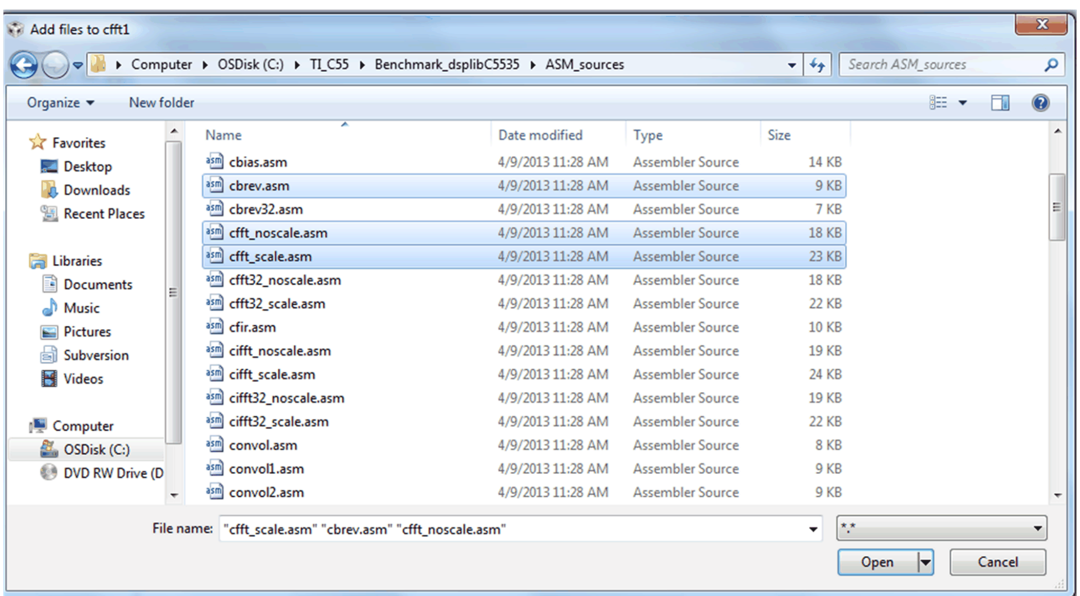

Figure 7. Optimized Files to Link to the Project

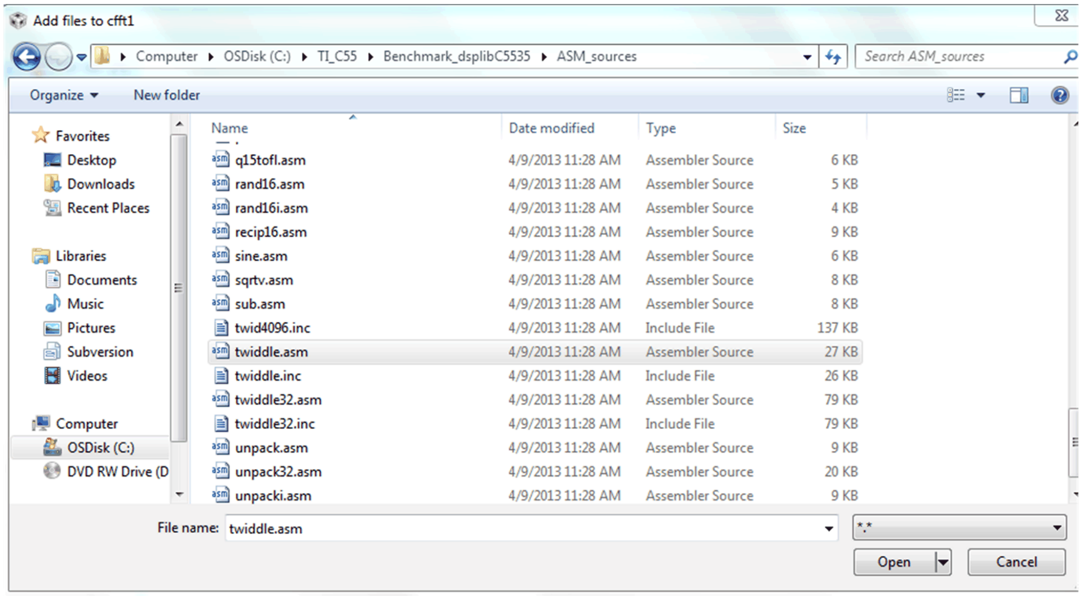

Figure 8. Optimized Files to Link to the Project

Remember that the assembly files should be linked and not copied.

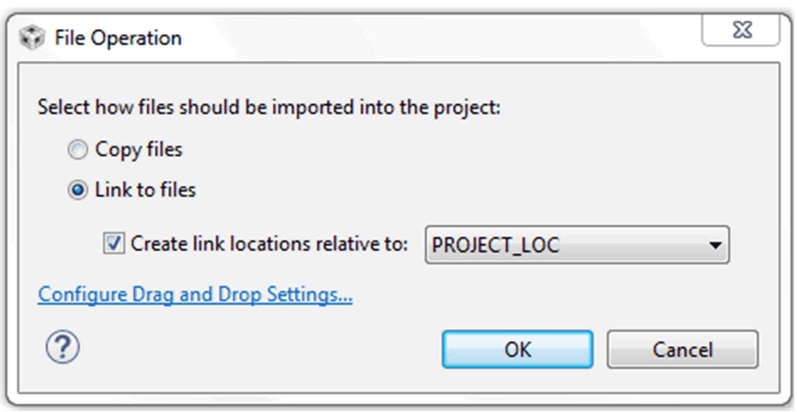

Figure 9. Copy or Link Dialogue Box

3. Set project properties.

It is assumed that CCS v6 was installed to support the C55 family, therefore, a version of the TI code generation tools for C55 was installed. Right click on the project name and choose properties. In properties go to Resources → Linked Resources tab. Figure 10 shows what resources are already linked into the CCS.

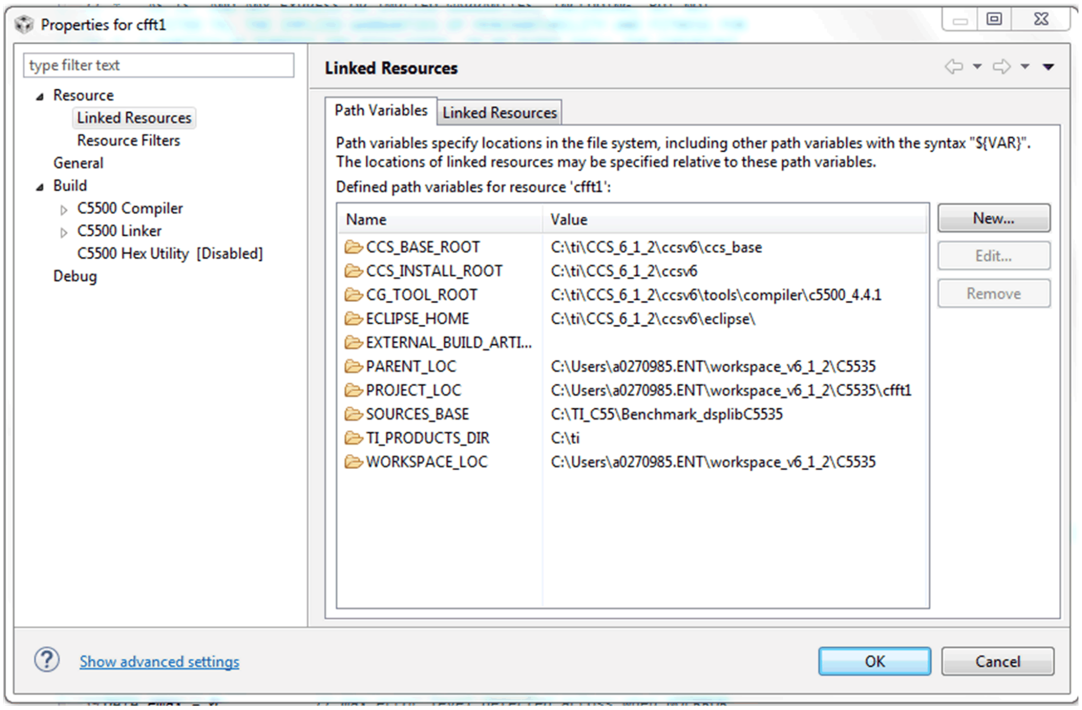

Figure 10. Properties for cfft1

Note that SOURCES_BASE from Figure 10 is not set in the Linked Resources tab by default; it must be added.

Figure 11 shows how to add the SOURCES_BASE linked resource. Click on new, enter the resource name “SOURCES_BASE” in the dialogue box. Click on the folder tab and navigate to the directory where the project was installed “myDirectory”. Figure 11 shows how to add SOURCES_BASE if the project was installed in the \TI_C55\Benchmark_dsplibC5535 directory.

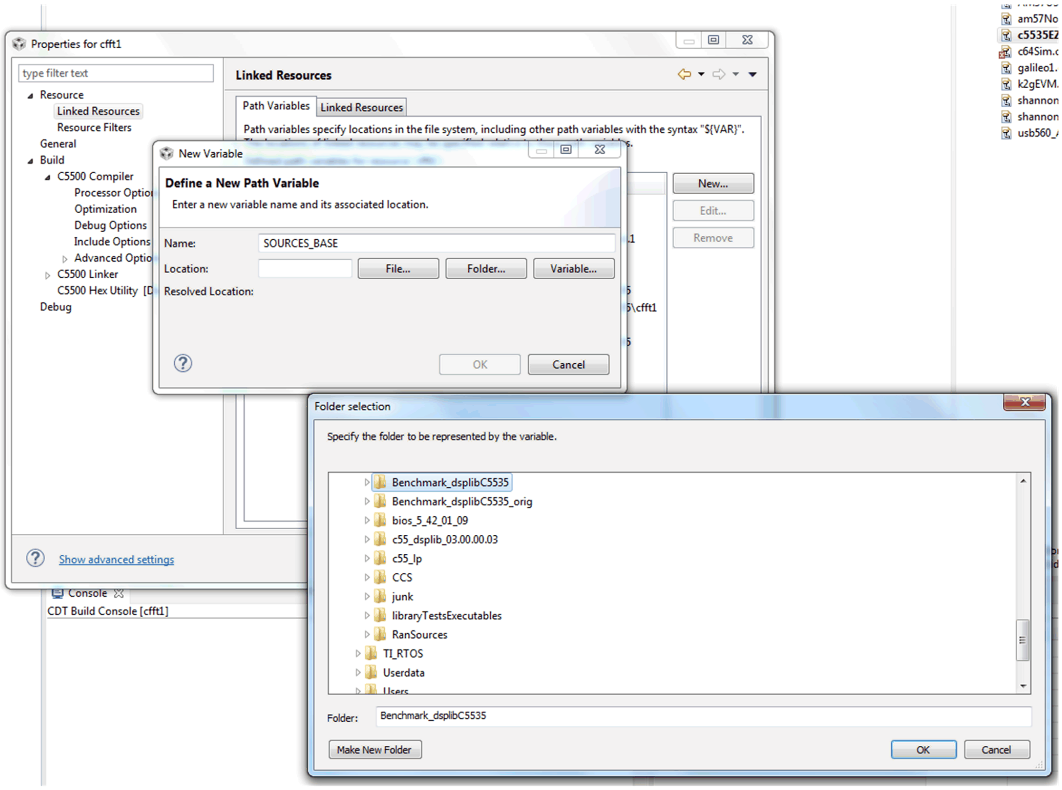

Figure 11. Define a New Resource

Next, a path to the common include files should be added. The common include files are in the sub-directory include of “myDirectory”. Figure 12 shows how to add the path ${SOURCES_BASE}\include to the project property.

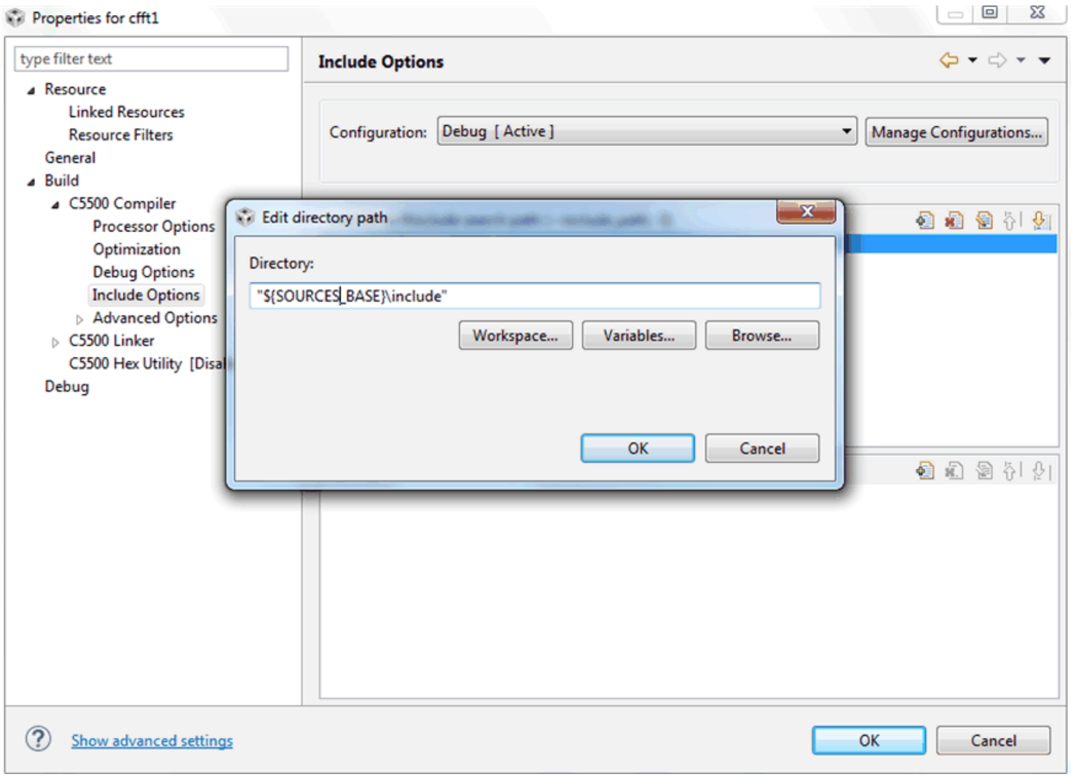

Figure 12. Adding Include Path

After clicking OK to the new include file and to the property dialogue box, the configuration of the project is done.

4. Build the CFFT Project.

The *CFFT_T.c* file enables the ability to define multiple test scenarios by changing #define or #include values at the top of the file.

NUMBER_OF_ITERATIONS determines how many times the optimized library function executed to enable stable power measurements. This is a long variable and, thus, can be configured to take a very long time.

FFT_HARDWARE If the FFT_hardware is set to 1, the Hardware FFT (HWAFFT) accelerator is used. If the value is zero, the DSP core optimized FFT routine will be used.

A set of include files are used to change the size of FFT and to determine whether block scaling or not block scaling is used in the FFT calculation. The user should comment on all of the include files except the one that is chosen based on scale or no scale and the sizes.

1000000 iterations, no hardware FFT accelerator and 256 values complex FFT with scaling were chosen in the following code:

```
#define  NUMBER_OF_ITERATIONS   1000000l
#define FFT_HARDWARE   1
//#include "t1_SCALE.h"
//#include "t2_SCALE.h" //16
//#include "t3_SCALE.h" //32
//#include "t4_SCALE.h" //64
//#include "t5_SCALE.h" //128
#include "t6_SCALE.h" //256
//#include "t7_SCALE.h" //512
//#include "t8_SCALE.h" //1024
//#include "t2_NOSCALE.h"
```

```
//#include "t3_NOSCALE.h"
//#include "t4_NOSCALE.h"
//#include "t5_NOSCALE.h"
//#include "t6_NOSCALE.h"
//#include "t7_NOSCALE.h"
//#include "t8_NOSCALE.h"
```

After configuring the iteration, the hardware accelerator usage, and the size, right click on the project and choose build. A correct build is shown below.

```
'Finished building: C:/TI_C55/Benchmark_dsplibC5535/ASM_sources/twiddle.asm'
' '
'Building target: cfft1.out'
'Invoking: C5500 Linker'
"C:/ti/CCS_6_1_2/ccsv6/tools/compiler/c5500_4.4.1/bin/cl55" -v5515 --memory_model=large -g --
define=c5535 --display_error_number --diag_warning=225 --ptrdiff_size=16 -z -m"cfft1.map" --
stack_size=0x200 --heap_size=0x400 -i"C:/ti/CCS_6_1_2/ccsv6/tools/compiler/c5500_4.4.1/lib" -
i"C:/ti/CCS_6_1_2/ccsv6/tools/compiler/c5500_4.4.1/include" --reread_libs --
display_error_number --warn_sections --xml_link_info="cfft1_linkInfo.xml" --rom_model --
sys_stacksize=0x200 -
o "cfft1.out" "./CFFT_T.obj" "./TEST.obj" "./cbrev.obj" "./cfft_noscale.obj"
"./cfft_scale.obj" "./twiddle.obj" "../fft5535.cmd"  -l"libc.a"
<Linking>
'Finished building target: cfft1.out'
' '

**** Build Finished ****
```

5. Define the platform.

   If not already done, the platform must be defined. In the “Target Configuration” tab (which can be set from the view tab in edit /C prospective of CCS), right click on User Defined and select “New Target Configuration”. At the dialogue box, provide a name to the new target (for example, EZsdp5535) and click finish.

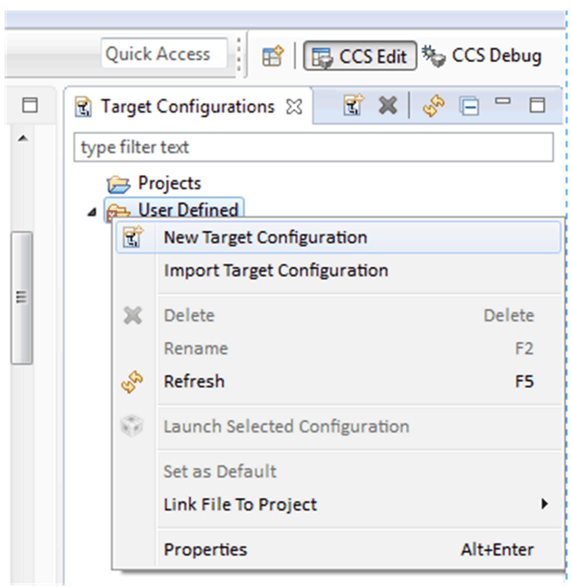

Figure 13. New Target Configuration

Figure 14 shows how to configure the new target. If the system uses the on-board USB emulator, set the connection to Texas Instruments XDS100v2 USB Debug Probe. Then, set a filter on the platform with 5535 in its name by typing 5535 in the Board or Device tab (where it says “type filter test”). Select the EZDSP5535 boards and click save.

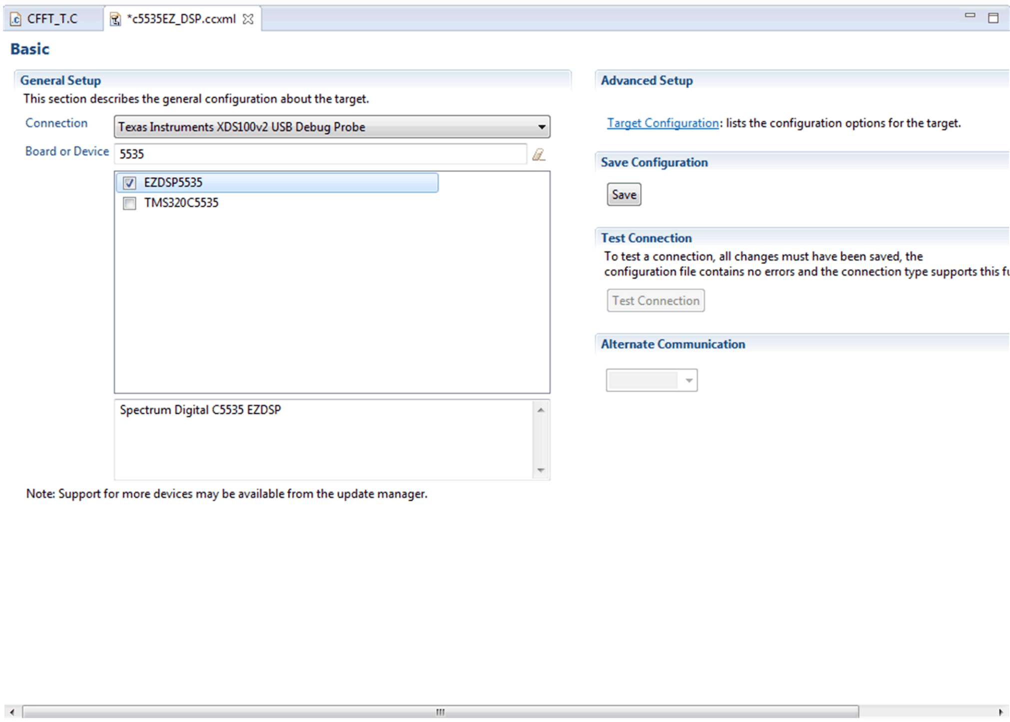

Figure 14. Configuring the New Target

Verify that the gel file was linked. Click on the Advanced tab (at the bottom of the dialogue box) and select C55xx as illustrated in Figure 15. Make sure that the initialization script (on the right side of the screen shot) is set.

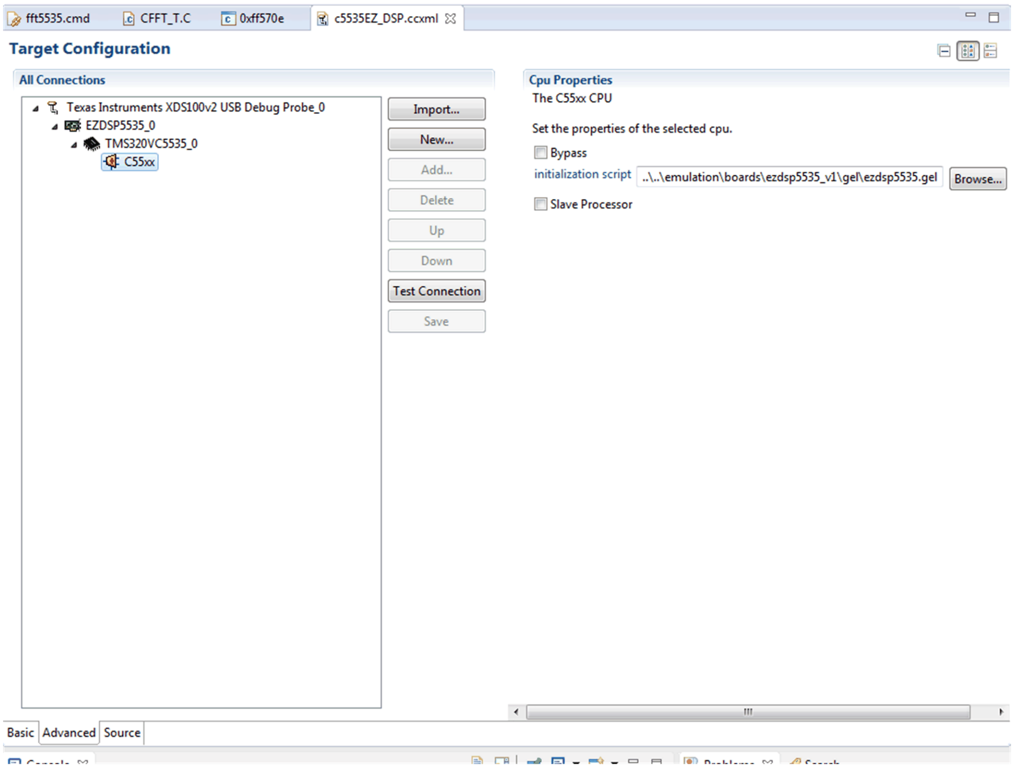

Figure 15. Target Configuration Dialogue Box

6. Launch and connect.

   In the target configuration dialogue box, select the EZDSP5535 target that was defined. Right click and select “launch select configuration”. CCS will change to Debug Prospective, see Figure 16.

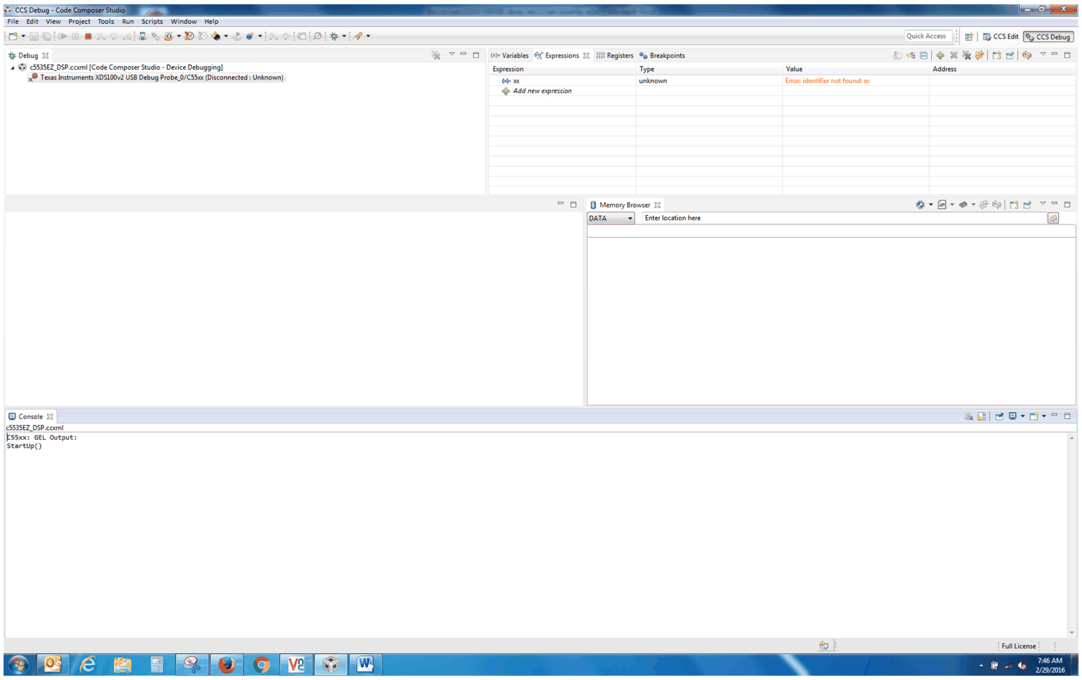

Figure 16. Debug Prospective

   Right click on the emulator name (at the top of the prospective) and select connect. The console will show the initialization steps:

```
C55xx: GEL Output:
StartUp()
C55xx: GEL Output:
OnTargetConnect()
C55xx: GEL Output:
OnReset()
C55xx: GEL Output: Reset Peripherals is complete.
C55xx: GEL Output: Configuring PLL (100 MHz).
C55xx: GEL Output: PLL Init Done.
C55xx: GEL Output: Target Connection Complete.
```

7. Load, enable the clock and run.

   From the Run tab, select Load → Load Program. In the dialogue box that is opened, select “Browse project” and navigate to the project name (shown as cfft1 in the screenshots); debug and select the out file, see Figure 17.

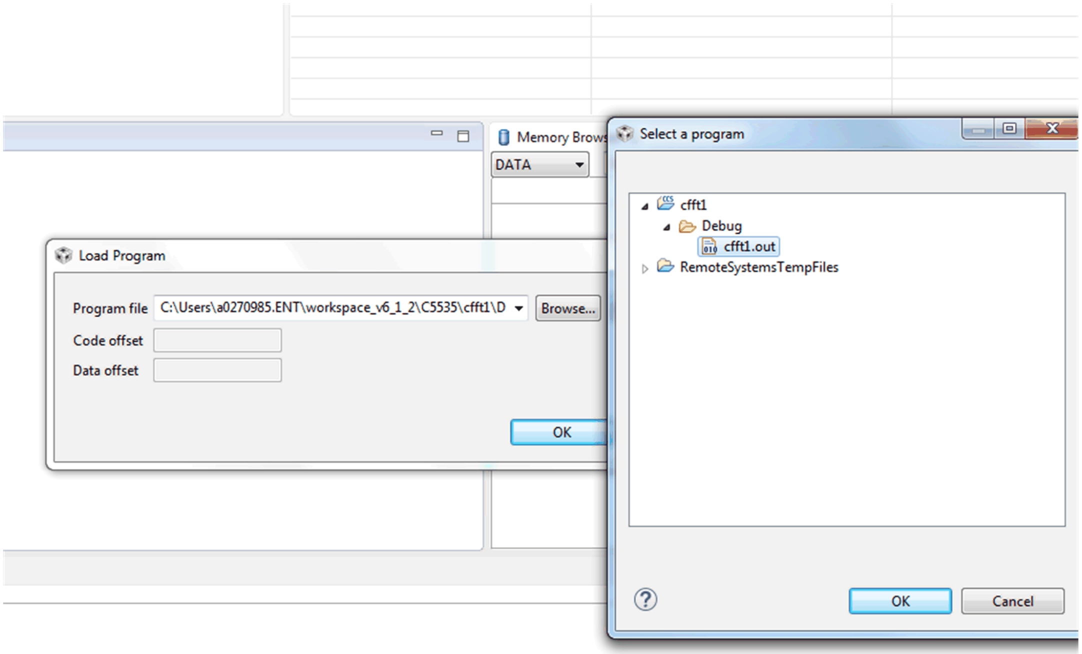

Figure 17. Loading the Project Executable

Select OK and OK. The executable will be loaded into the device and the main function will appear in the Edit window.

Next, the clock is enabled. From the Run tab, right click to select Clock → Enable and verify that the clock icon and the value 0 appear at the bottom of the CCS screen. Figure 18 and Figure 19 show Clock Enable and the clock icon.

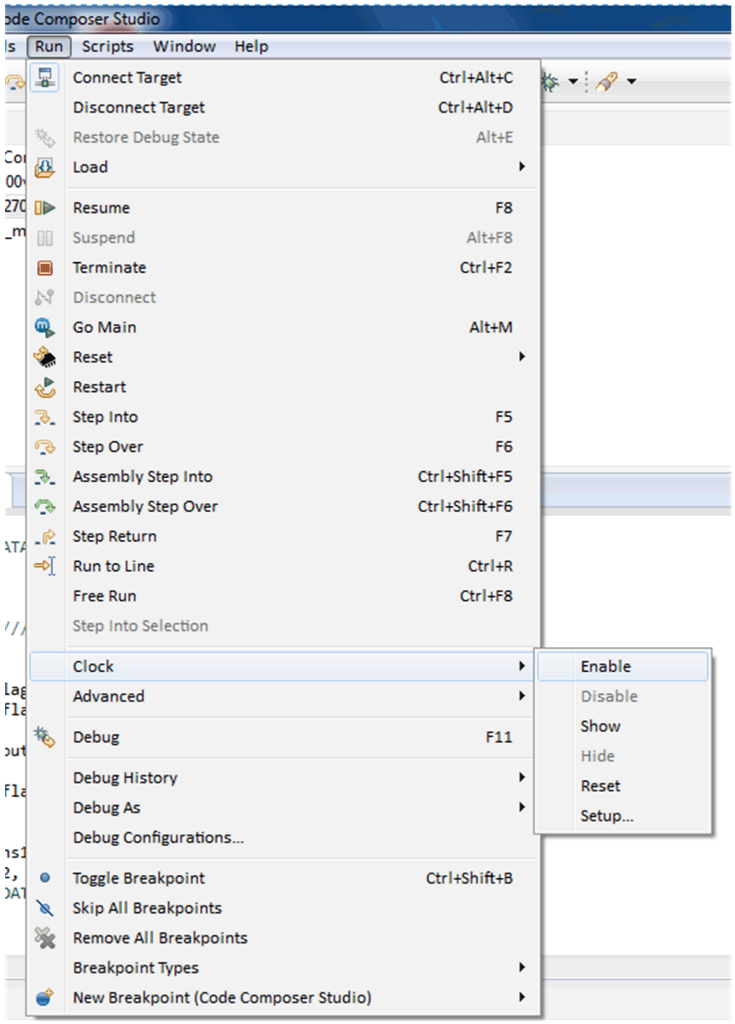

Figure 18. Clock Enable

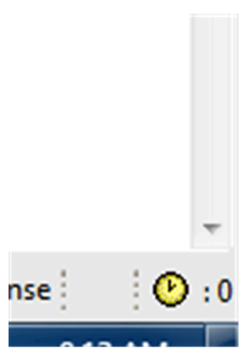

Figure 19. Clock Icon

There are multiple ways to run the code. From the Run menu, click on the green arrow (at the top of CCS window) or type F8.

For 256 16-bit fix-point complex FFT, if the hardware accelerator is not used (`#define FFT_HARDWARE 0`), the printing on the console will be as shown below. Note that the value of "iteration" determines how long the program runs to completion.

```
Complex FFT number of elements is 256
fft time (in cycles)  5366
bit reverse time (in cycles)  521
Done with 1000000 iteration
```

If the hardware accelerator is used (`#define FFT_HARDWARE 1`), the printing on the console will be as shown below:

```
Using bit reversal Number of elements  is 256
Bit reversal accelerator  time (in cycles)  538
 Complex FFT number of elements is 256
fft time (in cycles)  1136
 max Error =   3
 number of errors  21023
Done with 1000000 iteration
```

Note that the time consumed by the bit reversal function for 256 complex points is less for the DSP implementation than the hardware accelerator.

5 Build and Run all Other Projects

The instructions to build and run all other projects are similar to the instructions for the CFFT project discussed in Section 4. The following is a summary of the steps that are needed and then a short description of the expected output for each project.

1. Define new project. This follows the instructions in defining the CFFT project (step 1 from Section 4).
2. Delete default linker command and add Project files (step 2 from Section 4). It is important to delete the default linker command file that comes with the new project definition. All of the test source files and data must be added to the project directory following step , then the appropriate optimized library function needs to be linked to the project.
3. Set Project Priorities. Follow CFFT step 3 from Section 4 of the CFFT build exactly.
4. Build the project. Similar to CFFT step 4 from Section 4, define the number of iterations (the value of iteration) and what include file to un-comment. The include file determines the parameters of the test.
5. Step 5 from Section 4 is already done for the CFFT project.
6. Launch and connect. The same as CFFT step 6.
7. Load, enable the clock, and Run. If the clock is already enabled, no need to enable it again. Load and run the same as step 7 from Section 4 of CFFT. There is a screenshot of the output for each project.

5.1 FIR2

In this benchmark, the test program is in the *fir2_t.c* file and the optimized library assembly routine name is fir2.asm. The linker command file is fir2.cmd.

The include *t5_ran.h* file was built to have 256 taps filter. However, the result vector is good only up to 32 elements. Using this include file with more than 32 input values (still 256 Taps filter) will not compare with the result vector. Using any other include file requires changing the definition of nx in the *fir2_t.c* file.

Building and running the code provides the following printout:

```
256 tap, 32 values FIR Real 16-bit
fir2 time (in cycles)  4247
Done with 1000 iteration
```

Changing the nx value to 2 (fir2 must have even number of elements) in the *ft2_t.c* file provides the following results:

```
256 tap, 2 values FIR Real 16-bit
fir2 time (in cycles)  330
Done with 1000 iteration
```

5.2 FIR1

FIR1 can process both an even and odd number of output elements; however, it is slower than FIR2. FIR2 can process only even number of elements. This benchmark processes a single output value with 256 taps FIR1 filter and 32 output values with the same filter.

In this benchmark, the test program is in the *fir_t.c* file and the optimized library assembly routine name is fir.asm. The linker command file is fir1.cmd.

The include *t5_ran.h* file was built to have 256 taps filter. However, the result vector is good only up to 32 elements. Using this include file with more than 32 input values (still 256 Taps filter) will not compare with the result vector. Using any other include file requires changing the definition of nx in the *fir_t.c* file. The test program runs the code for 32 output values and a single output value. The following printouts are for 32 output values and a single output value.

```
number of elements in the vector is  is 1  and corefficients  256
FIR1 filter time (in cycles)  310
number of elements in the vector is  is 32  and corefficients  256
FIR1 filter time (in cycles)  8309
Done with 1000 iteration
```

5.3 Convol2

Convol2 is a test project for the convolution function. There are three convolution optimized functions in the DSPLIB library. Convol2 is the fastest one; convol1.asm and convol.asm can be substituted for convol2.asm and run on benchmark with one of the other functions.

In this benchmark, the test program is in the *conv2_T_ran.c* and the optimized library assembly routine name is convol2.asm. The linker command file is 55xConvolve2.cmd.

The include *t4_ran.h* file was built to have the sum of 80 elements for each vector of the convolution, and 80 output values. As with FIR2, the result vector is good only up to 80 elements. Using this include file with more than 80 input values (still 80 Taps convolution) will not compare with the result vector. The test program runs the code for 80 output values. The following is the printout of executing the test.

```
80 tap, 80 values convuolution Real 16-bit
  Convolv2  time (in cycles)  3285
Done with 1000 iteration
```

5.4 Auto-Correlation

Correlation is a test project for the auto-correlation function. In this benchmark, the test program is ARAW_T.c and a c model of the optimized assembly function routine is araw.c. The optimized library assembly routine name is araw.asm. The linker command file is correlation.cmd.

The include *t3_ran.h* file was built to have the sum of 80 elements for each output of the auto-correlation, and 80 output values. As with convolution, the result vector is good only up to 80 elements. The results are compared to the results of the C model function in the file araw_c.c. The following is the printout of executing the test.

```
80 tap, 80 values Auto-Correlation Real 16-bit
  time (in cycles)  3456
Done with 1000000 iteration
```

5.5 Delay LMS Filter

There are two optimized library functions for delay LMS filter: the standard function dlms and the fast version dlms_fast. Thus, there are two benchmark projects: one for the standard version and one for the fast version. The faster function has some limitations on the size of the filter and the location of the data in the device memory. For a detailed description of the requirements, see the *TMS320C55x DSP Library Programmer's Reference User's Guide* (SPRU422).

The test programs are dlms_fast_T.c and dlms_T.c for the fast and the standard versions. The optimized library assembly routine names are dlms_fast.asm and DLMS.asm for the fast version and the standard version. The linker command file is dlms.cmd.

The include *t4.h* file was built to have filter of 32 coefficients and 64 element input and output values. The following is a printout from the standard and fast test code when *t4.h* is included. Note that the cycle's advantage of the comparison between the fast vs standard processing depends on the size of the filter and the data.

```
standard LMS  Delay filter #values 64  #of taps 32  step 327
  time (in cycles)  4474
Done with 1000 iteration
```

```
fast LMS  Delay filter #values 64  #of taps 32  step 327
  time (in cycles)  4372
FAIL THE TEST
Done with 1000 iteration
```

5.6 Vector Max Value – Value Only

Maxval finds the maximum value of a real vector. The test code is MAXVAL_T.c; the linker command is maxval.cmd.

The include *t8.h* and *t8_original.h* files have the same values in different order. This is done to demonstrate that the cycle consumption does not depend on the maximum order. Both include files have 100 elements. The following is the printout of executing the test.

```
number of elements in the vector is  is 100
max value  time (in cycles)  77
Done with 100 iteration
```

5.7 Vector Max Value – Value and Index

Maxvec finds the maximum value and the index of the maximum value of a real vector. The test code is Maxvec_T.c and the linker command is maxvec.cmd.

The include *t8.h* file has 100 elements. The following is the printout of executing the test:

```
number of elements in the vector is  is 100
max (index and value)  time (in cycles)  322
Done with 1000 iteration
```

6 Power Measurements

Power measurements require special hardware settings and the ability to run the measured function for a long period of time. At the end of each benchmark project, the measured library function runs multiple times. The number of times that the measured function runs is defined by a 32-bit long variable iteration. The value can be set to a small value (for example, 1000 during debug session) and converted to a long value (for example, 10,000,000 for power measurements). The maximum iteration value is 0x7fffffff = approximately 2147 million iterations (for 100 MHZ system) and a function that consumes 200 cycles will last more than an hour.

7 Benchmark More Library Functions

All test code and data files are based on the DSPLIB library unit test. Each library function has its own unit test. Do the following to port a unit test to a benchmark project:

1. Go to the directory where the C55x DSPLIB library was loaded using the information in Section 3.
2. Copy the data files, source code files and the linker command file from the Library Examples directory to a new directory.
3. Modify the test code as follows:
   (a) Add #include <time.h> to the list of include files.
   (b) Add #define NUMBER_OF_ITERATIONS. For debug time, get a small value. For power measurement time, the value should be large.
   (c) Add a set of variables for the number of elements and other variables that are associated with the test (number of taps, and so forth). Add a long iteration1 counter and a set of clock_t type time measurements (clock_t t1,t2, t11,t22 ,total1_t ,total2_t,diff).
   (d) Measure the overhead that is associated with the time measurements as seen in the following:
```
t1 = clock()   ;
t2 = clock()  ;
diff = t2 - t1 ;   ///   overhead of calling
```
   (e) t1 = clock() ; t2 = clock() ; diff = t2 - t1 ; /// overhead of calling
   (f) Calculate the time consumed by the library routine by adding the following code (replace libraryRoutineFunction with the real function that is benchmarked and list the real parameters).
```
t1 = clock ()   ;
LibraryRoutineFunction ();
t2 = clock ();
 total1_t = (double) (t2 - t1-diff)   ;
  Printf ("Function Parameters are %d  %d %d  %d \n" , list,of,parameters) ;
  printf("  time (in cycles)  %ld \n", total1_t) ;
```
   (g) Add the iteration part to run the library routine for a long time (replace the library function with the real function call and parameter and list the real parameters)
   for (iterations1 = 0; iterations1 < NUMBER_OF_ITERATIONS;iterations1++)
```
{
       libraryFunction(parameter list);
   }
   printf("Done with %ld iteration  \n",iterations1 );
```
   (h) Repeat the steps from Section 5.

8 References

TMS320C55x DSP Library Programmer's Reference User's Guide (SPRU422)

IMPORTANT NOTICE

Texas Instruments Incorporated and its subsidiaries (TI) reserve the right to make corrections, enhancements, improvements and other changes to its semiconductor products and services per JESD46, latest issue, and to discontinue any product or service per JESD48, latest issue. Buyers should obtain the latest relevant information before placing orders and should verify that such information is current and complete. All semiconductor products (also referred to herein as “components”) are sold subject to TI’s terms and conditions of sale supplied at the time of order acknowledgment.

TI warrants performance of its components to the specifications applicable at the time of sale, in accordance with the warranty in TI’s terms and conditions of sale of semiconductor products. Testing and other quality control techniques are used to the extent TI deems necessary to support this warranty. Except where mandated by applicable law, testing of all parameters of each component is not necessarily performed.

TI assumes no liability for applications assistance or the design of Buyers’ products. Buyers are responsible for their products and applications using TI components. To minimize the risks associated with Buyers’ products and applications, Buyers should provide adequate design and operating safeguards.

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 relating to any combination, machine, or process in which TI components or services are used. Information published by TI regarding third-party products or services does not constitute a license to use such products or services or a warranty or endorsement thereof. Use of such information may require a license from a third party under the patents or other intellectual property of the third party, or a license from TI under the patents or other intellectual property of TI.

Reproduction of significant portions of TI 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. TI is not responsible or liable for such altered documentation. Information of third parties may be subject to additional restrictions.

Resale of TI components or services with statements different from or beyond the parameters stated by TI for that component or service voids all express and any implied warranties for the associated TI component or service and is an unfair and deceptive business practice. TI is not responsible or liable for any such statements.

Buyer acknowledges and agrees that it is solely responsible for compliance with all legal, regulatory and safety-related requirements concerning its products, and any use of TI components in its applications, notwithstanding any applications-related information or support that may be provided by TI. Buyer represents and agrees that it has all the necessary expertise to create and implement safeguards which anticipate dangerous consequences of failures, monitor failures and their consequences, lessen the likelihood of failures that might cause harm and take appropriate remedial actions. Buyer will fully indemnify TI and its representatives against any damages arising out of the use of any TI components in safety-critical applications.

In some cases, TI components may be promoted specifically to facilitate safety-related applications. With such components, TI’s goal is to help enable customers to design and create their own end-product solutions that meet applicable functional safety standards and requirements. Nonetheless, such components are subject to these terms.

No TI components are authorized for use in FDA Class III (or similar life-critical medical equipment) unless authorized officers of the parties have executed a special agreement specifically governing such use.

Only those TI components which TI has specifically designated as military grade or “enhanced plastic” are designed and intended for use in military/aerospace applications or environments. Buyer acknowledges and agrees that any military or aerospace use of TI components which have ***not*** been so designated is solely at the Buyer's risk, and that Buyer is solely responsible for compliance with all legal and regulatory requirements in connection with such use.

TI has specifically designated certain components as meeting ISO/TS16949 requirements, mainly for automotive use. In any case of use of non-designated products, TI will not be responsible for any failure to meet ISO/TS16949.

Products		**Applications**	
Audio	www.ti.com/audio	Automotive and Transportation	www.ti.com/automotive
Amplifiers	amplifier.ti.com	Communications and Telecom	www.ti.com/communications
Data Converters	dataconverter.ti.com	Computers and Peripherals	www.ti.com/computers
DLP® Products	www.dlp.com	Consumer Electronics	www.ti.com/consumer-apps
DSP	dsp.ti.com	Energy and Lighting	www.ti.com/energy
Clocks and Timers	www.ti.com/clocks	Industrial	www.ti.com/industrial
Interface	interface.ti.com	Medical	www.ti.com/medical
Logic	logic.ti.com	Security	www.ti.com/security
Power Mgmt	power.ti.com	Space, Avionics and Defense	www.ti.com/space-avionics-defense
Microcontrollers	microcontroller.ti.com	Video and Imaging	www.ti.com/video
RFID	www.ti-rfid.com		
OMAP Applications Processors	www.ti.com/omap	**TI E2E Community**	e2e.ti.com
Wireless Connectivity	www.ti.com/wirelessconnectivity		

Mailing Address: Texas Instruments, Post Office Box 655303, Dallas, Texas 75265