Freescale Semiconductor
Application NoteWhite Paper

Document Number: AN4207
Rev. 2, 05/2012

Kernel Examples for the StarCore SC3850 DSP Core

The StarCore SC3850 DSP core addresses the key market needs of mainstream DSP applications, such as communications, video, audio, and medical imaging. The SC3850 core efficiently deploys a variable-length execution set (VLES) execution model that utilizes maximum parallelism by allowing multiple address generation and data arithmetic logic units to execute multiple instructions in a single clock cycle, while delivering very compact code.

The SC3850 digital signal processing library provides a set of C-callable, optimized ASM functions commonly used in signal processing applications. This application note presents the usage and performance of five key signal processing kernels, that is, complex Fast Fourier Transform (complex FFT), complex Discrete Fourier Transform (complex DFT), complex Finite Impulse Response (complex FIR), Cyclic Redundancy Check (CRC), and Infinite Impulse Response (IIR), to help users better utilize the DSP library in their system development.

NOTE

The supporting software library is available as zip file AN4207SW on freescale.com.

Contents

1 References

- *MSC8156 Reference Manual* (MSC8156RM)
- *Software Optimization of FFTs and IFFTs Using the SC3850 Core* (AN3666)
- *Software Optimization of DFTs and IDFTs Using the StarCore SC3850 DSP Core* (AN3680)

2 SC3850 core overview

The SC3850 core technology is a generation of Freescale StarCore DSP cores. Key enhancements to the previous generation include a twofold increase in DSP multiplication capacity, significantly higher compiled code performance, and an improved memory hierarchy. Micro-architectural improvements and new instructions enable the core to accelerate control functions, memory management, and DSP code performance.

This figure shows the SC3850 block diagram.

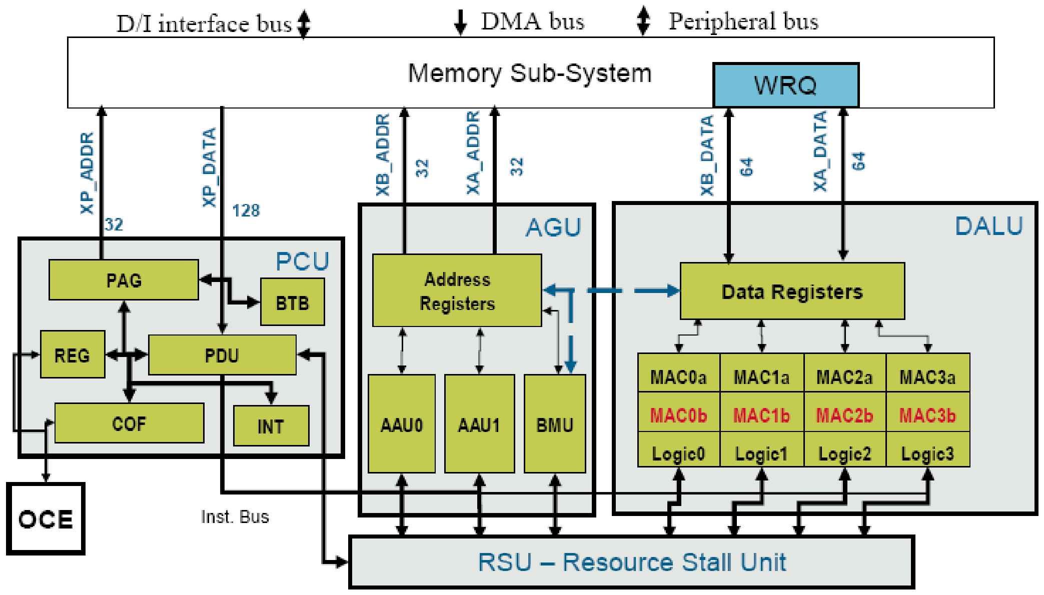


Figure 1. StarCore SC3850 block diagram

There are four parallel arithmetic logic units (ALUs) in the data arithmetic-logic unit (DALU), where most of the arithmetic and logical operations are performed on data operands. Each data ALU can perform two 16 × 16 multiplications per cycle (total of 8 multiplications for all ALUs, and up to 8 GMACs per cycle at 1 GHz core frequency). There are two address arithmetic units (AAUs) in the address generation unit (AGU), which performs effective address calculations using the integer arithmetic necessary to address data operands in memory. The program control unit (PCU) performs instruction fetch, instruction dispatch, hardware loop control, and exception processing. The resource stall unit (RSU) controls the hardware interlocks. It collects information from the instruction bus, holds the status for all resources in the core, and inserts enough stalls to resolve the pipeline hazards. Memory interface provides data exchange between the core and the other on-chip blocks, which has one 128-bit program bus and two 64-bit data buses.

This figure shows the SC3850 programming model. There are sixteen 40-bit data registers in DALU, sixteen 32-bit address registers, four 32-bit offset registers, and four 32-bit modulo registers in AGU. In addition, there are control and configuration registers that contain fields for indicating and controlling specific aspects of the core operation, status flags, and fields for indicating certain exception conditions.

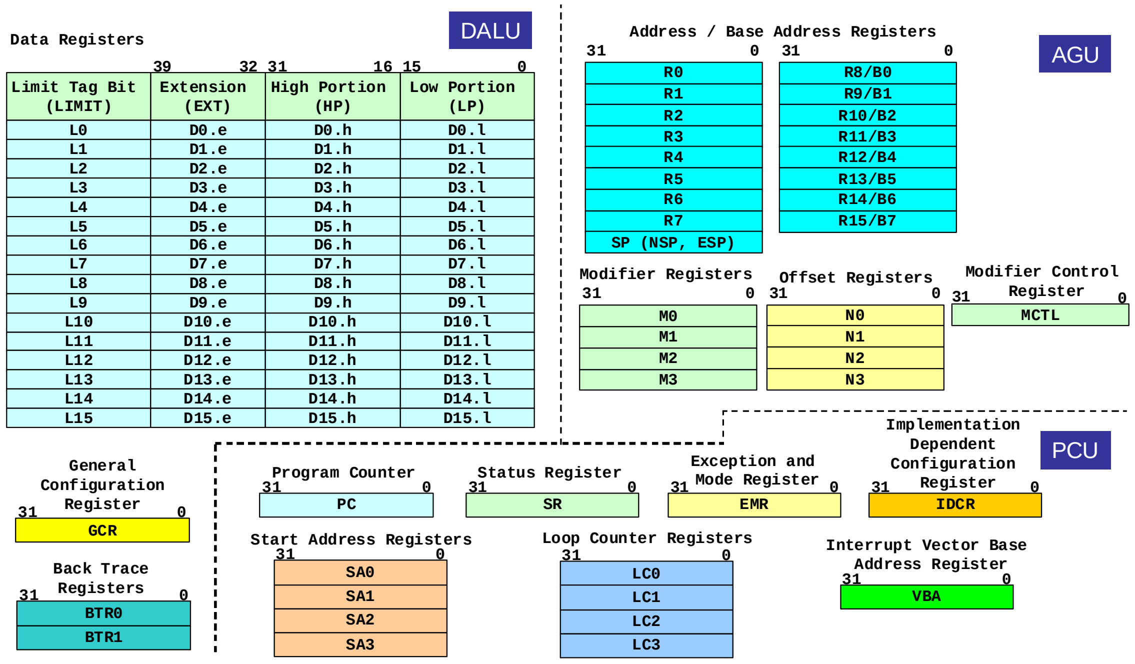


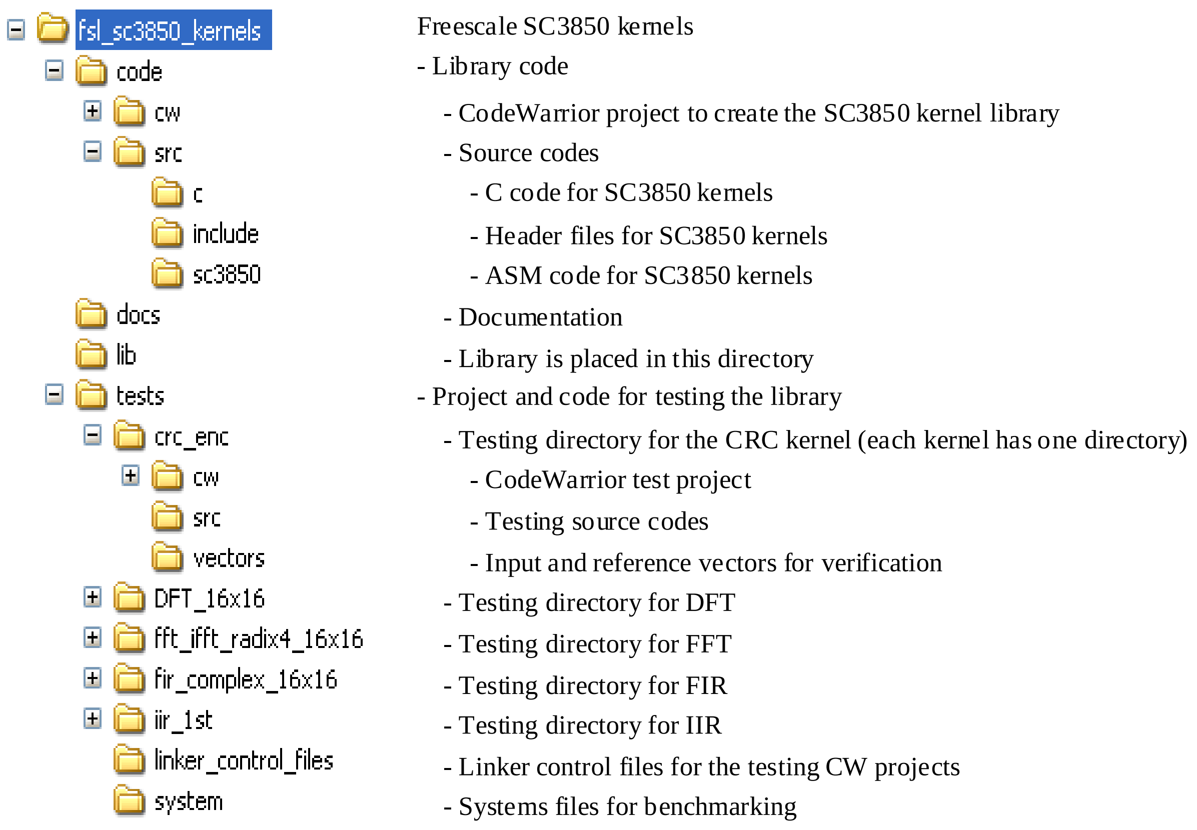
Figure 2. SC3850 programming model

The challenge facing DSP programmers is to use all the resources available in the advanced SC3850 architectures effectively. Ideally, the design should maximize the use of the program and data buses, and all six operational units (4 ALUs and 2 AAUs) simultaneously. To help users shorten the time-to-market in system development, Freescale provides a set of optimized functions in C and ASM, named SC3850 DSP library. Each function in the library is designed to produce the best performance possible by optimally utilizing available resources.

This application note presents the usage and performance of five key signal processing kernels, that is, complex Fast Fourier Transform (complex FFT), complex Discrete Fourier Transform (complex DFT), complex Finite Impulse Response (complex FIR), Cyclic Redundancy Check (CRC), and Infinite Impulse Response (IIR), to help users better utilize the DSP library in their system development.

3 Library organization

Figure 3 shows the directory structure of the SC3850 DSP library. To build the library, open `fsl_sc3850_kernels\code\cw\sc3850_kernels\.project` in CodeWarrior 10.1.x IDE. This project contains all the SC3850 kernels and produces `.elb` library output. To test each kernel, open the test projects located in `fsl_sc3850_kernels\tests`. For example, open complex FFT test project located in `fsl_sc3850_kernels\tests\fft_ifft_radix4_16x16\cw\test_sc3850_fft_ifft_radix4_complex_16x16\.project` to run/debug the complex FFT kernel.



```
fsl_sc3850_kernels              Freescale SC3850 kernels
  code                          - Library code
    cw                            - CodeWarrior project to create the SC3850 kernel library
    src                           - Source codes
      c                             - C code for SC3850 kernels
      include                       - Header files for SC3850 kernels
      sc3850                        - ASM code for SC3850 kernels
    docs                          - Documentation
    lib                           - Library is placed in this directory
  tests                         - Project and code for testing the library
    crc_enc                       - Testing directory for the CRC kernel (each kernel has one directory)
      cw                            - CodeWarrior test project
      src                           - Testing source codes
      vectors                       - Input and reference vectors for verification
    DFT_16x16                     - Testing directory for DFT
    fft_ifft_radix4_16x16         - Testing directory for FFT
    fir_complex_16x16             - Testing directory for FIR
    iir_1st                       - Testing directory for IIR
    linker_control_files          - Linker control files for the testing CW projects
    system                        - Systems files for benchmarking
```

Figure 3. SC3850 DSP library directory structure

4 Benchmarking

The MSC8156 application development system (MSC8156ADS) is a complete debugging environment for engineers developing applications for the MSC8156 Freescale digital signal processor (DSP). A MSC8156ADS board is used in this application note to measure cycle counts. The MSC8156 device is a highly integrated DSP processor that contains six StarCore SC3850 DSP subsystems (48 GMACS at 1 GHz). Each DSP core is part of an SC3850 subsystem that includes:

- 32 KB DCache
- 32 KB ICache
- 512 KB unified L2 cache/M2 shared memory

In addition, the MSC8156 contains:

- 1 MB of M3 shared memory
- Two DDR2/DDR3 memory controllers with a 64-bit data rate up to 800 Mbps

The MSC8156ADS board is connected to the USB port of a Host Debug Machine through USB cable. The connection configuration can be set in the Debug Configurations window to let users to run applications on MSC8156 ADS or on simulator. The clock difference between SC3850 PACC (Performance Accurate) simulator and MSC8156 hardware is up to 5%. Running and debugging applications on hardware and simulator is discussed in Section 6, *Using the SC3850 DSP library and test projects* on page 8. A core debug block, called on-chip emulator (OCE), can be used to measure cycle counts for SC3850 DSP kernel examples. Example 1 lists sample code to initialize the OCE and measure cycle counts. `InitEonce` and `ReadCountEonce` are defined in `eonce.h` and `eonce.c`.

Example 1. Sample code to initial the OCE and measure cycle counts

```
#define INIT_CYCLE InitEonceAll()
#define GET_CYCLE  ReadCountEonce()
    INIT_CYCLE;
    //Compute the overhead of calling eonce
    overhead = GET_CYCLE;
    overhead = GET_CYCLE - overhead;

    // Warm the Cache if necessary by calling kernel function
    #ifdef WARMCACHE
    sc3850_fft_radix4_complex_16x16_asm(...);
    #endif

    cycleCount_curr = GET_CYCLE;
    // Call kernel function
    sc3850_fft_radix4_complex_16x16_asm(...);
    cycleCount_curr = GET_CYCLE - cycleCount_curr - overhead;
    printf("My cycleCount is: %8i cycles\n", cycleCount_curr);
```

It is important to understand that data/instruction access overhead can affect the benchmarking. In real DSP applications, the memory access overhead can be minimized by overlapping the data transfer time and the computation time. Usually the DSP kernels are benchmarked with Warm cache. The data and instructions are preloaded into cache to minimize the memory access overhead. The Zero Wait State (ZWS) cycle count is provided in AN3666 (Ref [2]) for FFT and in AN3680 (Ref [3]) for DFT. To utilize the bus width fully and maximize the performance, some DSP kernels require data and program code to be aligned to a specific boundary.

5 CodeWarrior Trace and Profile tools

There are powerful trace and profile tools in CodeWarrior Development Studio that can be used to improve the application performance using either the simulator or the hardware. This section provides a short introduction to the Trace and Profile tools.

5.1 Analysis tools features

The CodeWarrior Profiling and Analysis tools can be used to improve the application performance using either the simulator or the hardware. The tools allow the following:

- Easy setup via the Trace and Profile tab in the Debug configuration panel.
- In the case of hardware debug, the advanced setup can be configured via the advanced Settings dialogue window. The developer can set complex configurations for the Debug and Profiling Unit (DPU), On Chip Emulator (OCE), and Virtual Trace Buffer (VTB).
- Profiling of any routine, group of routines or an entire project and display the results in a intuitive manner in the TRACE, CODE, and PERFORMANCE views.
- Track the time spent in any routine and the subroutines called and provide an easy way of displaying the data in forms of graphs while the application is in debug mode or offline.
- Provide data viewing features such as export data to Excel file, export the trace and function data generated by simulator or target hardware in CVS format, apply multi-level searches/filters and find/isolate specific data, and show/hide and copy/paste columns and cells.

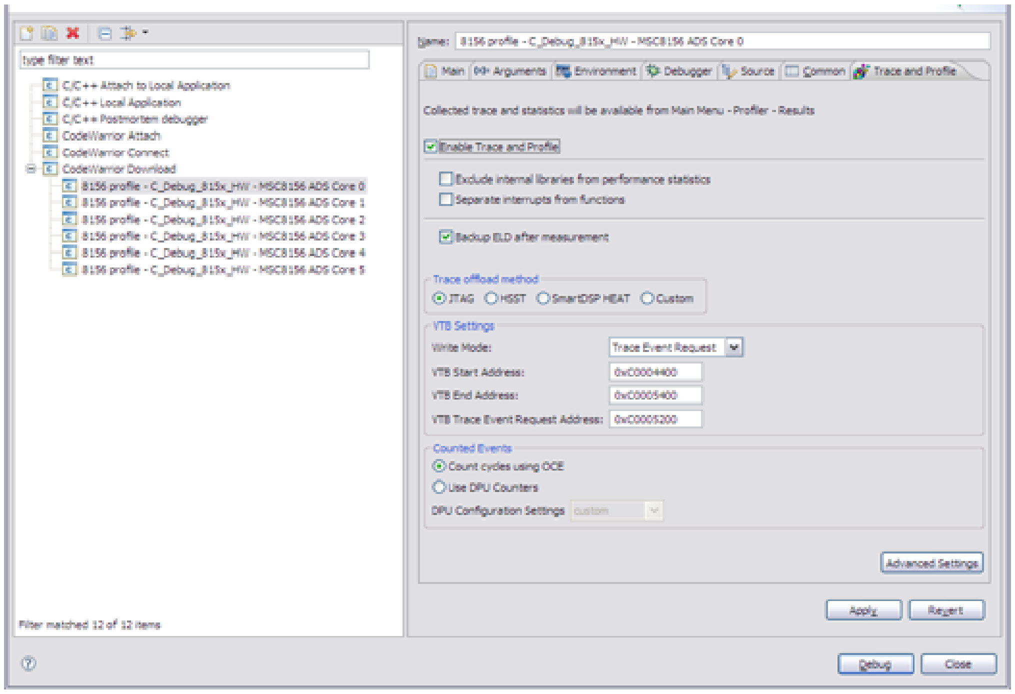
5.2 Start a new profiling session within the CodeWarrior environment

CodeWarrior profiler launch configuration can be performed using a few simple steps. After the project is created (for example, an MSC8156 stationary), use the following steps to start a profiling session:

1. Open the Debug dialog window. In the CodeWarrior Project view, right-click on the project name (for example, the 8156 profile: `C_Debug_815x_HW`) and select the Debug as/Debug Configurations... option in the context menu;
2. Select CodeWarrior Download configuration in the tree-view panel on the left and select the launch configuration corresponding to the project you are using (for example, `8156 profiling:-C_Debug_815x_HW - MSC8156 ADS Core 0`);
3. Select Enable Trace and Profile in the Trace and Profile tab. After the check box is selected, modify the Trace and Profile basic settings. Also, on this tab you can find the Advanced Settings button that allows you to adjust/modify the settings for DPU, OCE, and VTB.

NOTE

Depending on the target configuration chosen, SC3x50, MSC814x or MSC815x, Figure 4 could be slightly different. More or less options may be shown in the Trace and Profile tab.



type filter text

- C/C++ Attach to Local Application
- C/C++ Local Application
- C/C++ Postmortem debugger
- CodeWarrior Attach
- CodeWarrior Connect
- CodeWarrior Download
  - 8156 profile - C_Debug_815x_HW - MSC8156 ADS Core 0
  - 8156 profile - C_Debug_815x_HW - MSC8156 ADS Core 1
  - 8156 profile - C_Debug_815x_HW - MSC8156 ADS Core 2
  - 8156 profile - C_Debug_815x_HW - MSC8156 ADS Core 3
  - 8156 profile - C_Debug_815x_HW - MSC8156 ADS Core 4
  - 8156 profile - C_Debug_815x_HW - MSC8156 ADS Core 5

Filter matched 12 of 12 items

Name: 8156 profile - C_Debug_815x_HW - MSC8156 ADS Core 0

Collected trace and statistics will be available from Main Menu - Profiler - Results

[✓] Enable Trace and Profile

[ ] Exclude internal libraries from performance statistics
[ ] Separate interrupts from functions

[✓] Backup ELD after measurement

Trace offload method
[● JTAG] ○ HSST ○ SmartDSP HEAT ○ Custom

VTB Settings
Write Mode: Trace Event Request
VTB Start Address: 0xC0004400
VTB End Address: 0xC0005400
VTB Trace Event Request Address: 0xC0005200

Counted Events
[● Count cycles using OCE]
○ Use DPU Counters
DPU Configuration Settings: custom

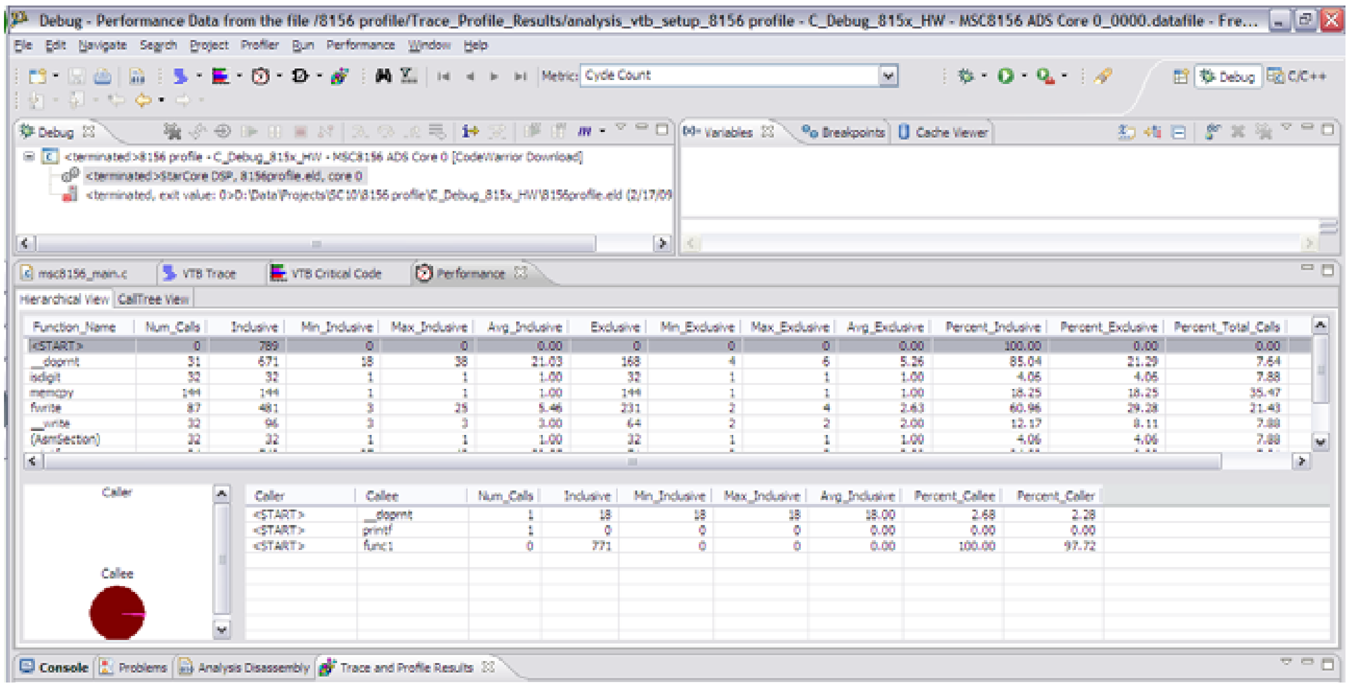
Figure 4. Debug Trace and Profile view

At this point, if you click on the Apply and then Debug buttons, the basic profiler initialization is done and results are ready to collect.

5.3 Viewing Trace and Profile results

Pressing the icon (*Profiler/All results menu can be used as an alternative*) shows profiler results. A new Trace and Profile Results view is displayed on the lower part of the screen. After the data is collected the name of the data file is listed under VTB Profile results data source. After the debug session is terminated, you can expand the data file in order to view the data sets:

- *Trace*. Detailed information about trace event such as type of event occurred (VLES, Branch, Routine Call/Return), symbol name of the current event belong to and symbol name and address for current event in case of a Branch/Call/Return and also the destination symbol name and address; All the other data (Critical code and Performance) are also generated based on trace data;
- *Critical code*. A combination of stalls and cache information. Offers information and statistics for all the instructions executed in the debug session. It display the OCE cycles and values for each counter;
- *Performance*. The metric and invocation information for each function that was/is executed during the debug session. It enables you to compare the relative efficiency of different portions of your target program;
- *ALU-AGU data* (only generated by the simulator). Displays the usage of hardware blocks by the application and the average usage of ALUs and AGUs by the each function instruction.


Debug - Performance Data from the file /8156 profile/Trace_Profile_Results/analysis_vtb_setup_8156 profile - C_Debug_815x_HW - MSC8156 ADS Core 0_0000.datafile - Fre...

Metric: Cycle Count

<terminated>8156 profile - C_Debug_815x_HW - MSC8156 ADS Core 0 [CodeWarrior Download]
<terminated>StarCore DSP, 8156profile.eld, core 0
<terminated, exit value: 0>D:\Data\Projects\SC10\8156 profile\C_Debug_815x_HW\8156profile.eld (2/17/09

Function_Name	Num_Calls	Inclusive	Min_Inclusive	Max_Inclusive	Avg_Inclusive	Exclusive	Min_Exclusive	Max_Exclusive	Avg_Exclusive	Percent_Inclusive	Percent_Exclusive	Percent_Total_Calls
<START>	0	789	0	0	0.00	0	0	0	0.00	100.00	0.00	0.00
__doprnt	31	671	18	38	21.03	168	4	6	5.26	85.04	21.29	7.64
isdigit	32	32	1	1	1.00	32	1	1	1.00	4.06	4.06	7.88
memcpy	144	144	1	1	1.00	144	1	1	1.00	18.25	18.25	35.47
fwrite	87	481	3	25	5.46	231	2	4	2.63	60.96	29.28	21.43
__write	32	96	3	3	3.00	64	2	2	2.00	12.17	8.11	7.88
(AsmSection)	32	32	1	1	1.00	32	1	1	1.00	4.06	4.06	7.88

Caller	Callee	Num_Calls	Inclusive	Min_Inclusive	Max_Inclusive	Avg_Inclusive	Percent_Callee	Percent_Caller
<START>	__doprnt	1	18	18	18	18.00	2.68	2.28
<START>	printf	1	0	0	0	0.00	0.00	0.00
<START>	func1	0	771	0	0	0.00	100.00	97.72

Figure 5. Debug Trace and Profile results view

NOTE

The VTB Profile results are available ONLY after the debug session is terminated

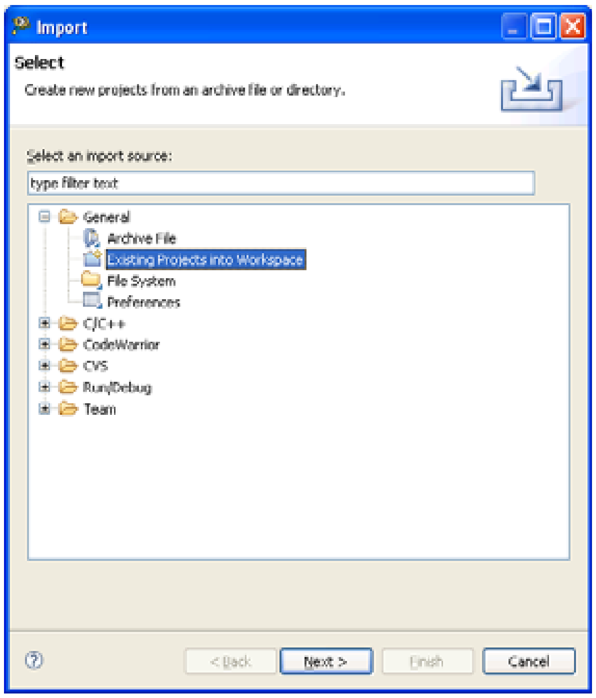
6 Using the SC3850 DSP library and test projects

This section demonstrates how to build the SC3850 kernel library in CodeWarrior IDE and how to link the library into the test projects. The application interfaces (APIs) and performance are presented for five key signal processing kernels: complex FFT, complex DFT, complex FIR, CRC, and IIR.

6.1 Build a SC3850 DSP library

The SC3850 DSP library is in a zip file. The library structure after unzipping it described in Section 3, *Library organization* on page 3. Use the following steps to open the library project in CodeWarrior.

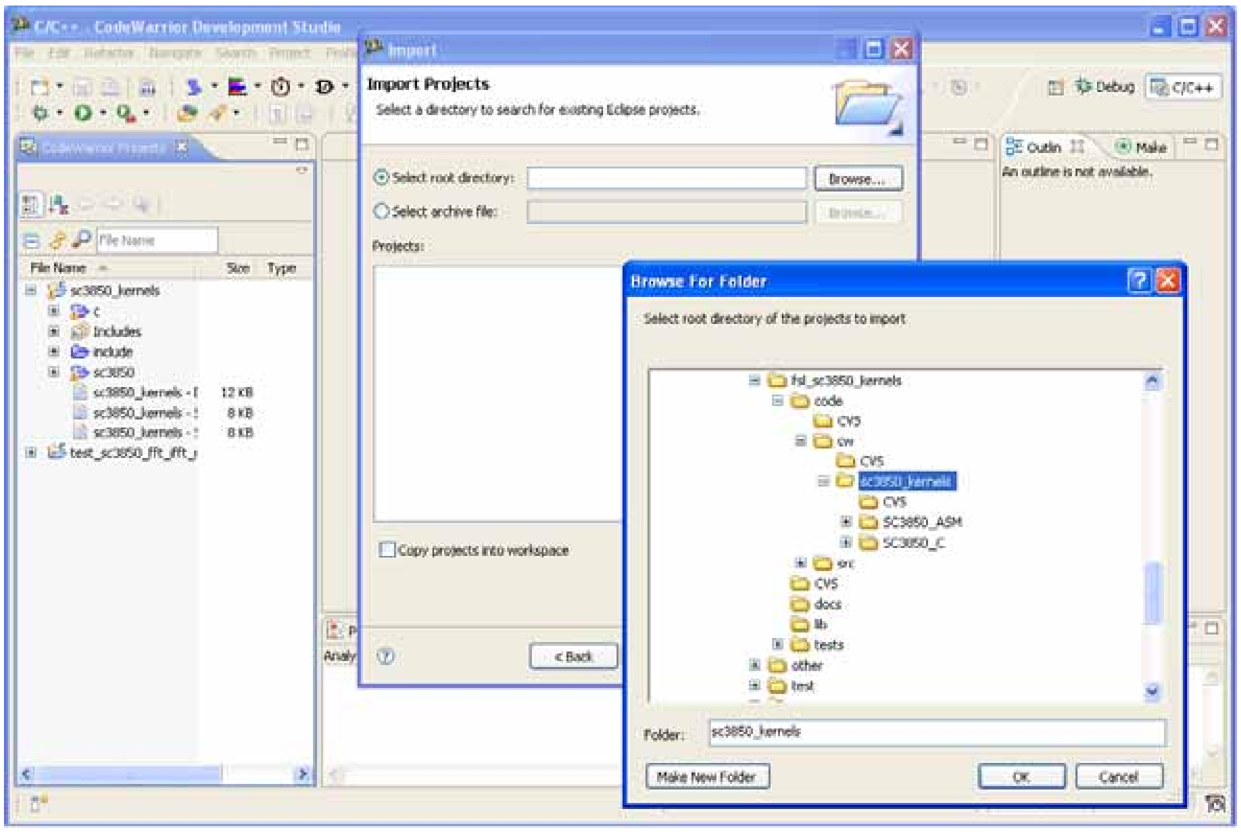
1. Select File > Import from the C/C++ Perspective toolbar. The Import dialog box appears.
2. Select General Folder > Existing Project into Workspace. Please see Figure 6.
3. Click Next. The Import Projects dialog box appears.
4. Click Browse. The Browse For Folder dialog box appears. Browse to root directory of the library project (fsl_sc3850_kernels\code\cw\sc3850_kernels), as shown in Figure 7.
5. Click OK. The imported project appears in the Projects window.
6. Click Finish.



Create new projects from an archive file or directory.

type filter text

Figure 6. Open the import window in the CodeWarrior IDE



Select a directory to search for existing Eclipse projects.

An outline is not available.

Select root directory of the projects to import

Folder: sc3850_kernels

Figure 7. Browse to root directory of the library project

NOTE

You can also drag-and-drop a project (`.project`) file from Windows Explorer to the CodeWarrior project window to open the project.

To build the library, select `sc3850_kernels` in the project window and click “SC3850_ASM” or “SC3850_C” to create ASM or C library, as shown in Figure 8. The library files (`.elb`) for C and ASM code are generated in `fsl_sc3850_kernels\lib`, respectively.

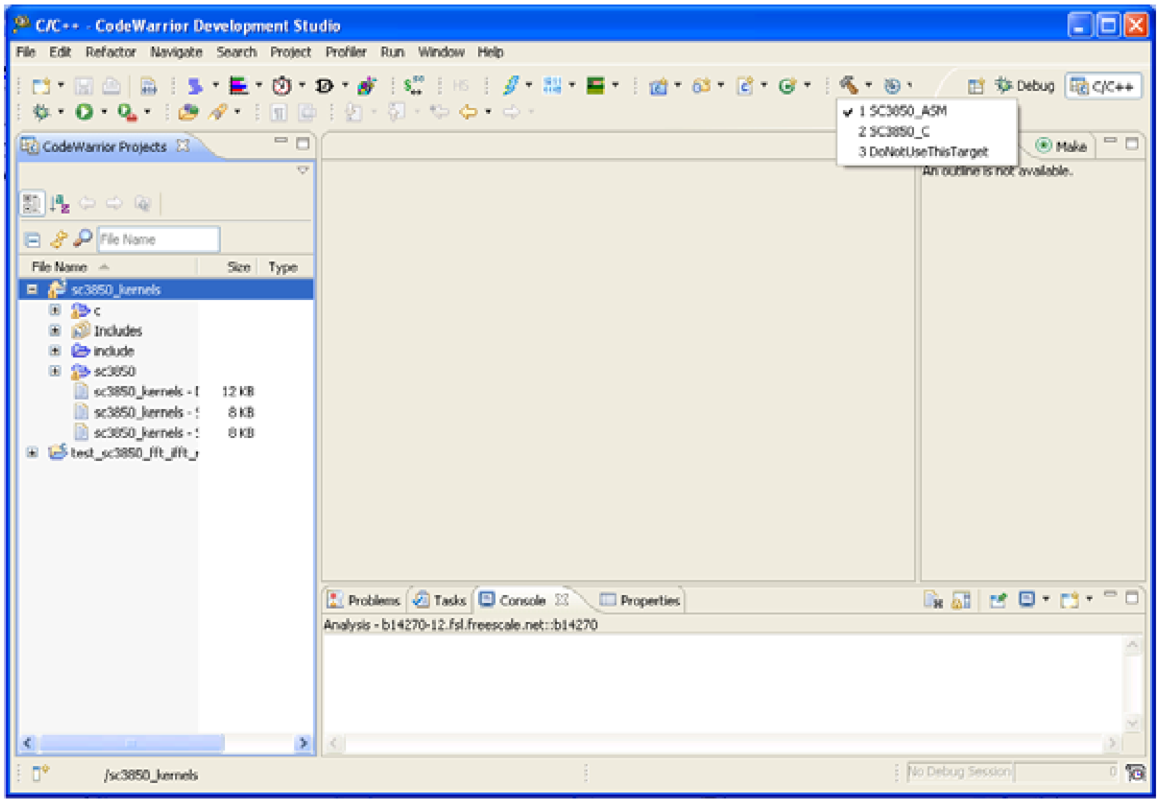


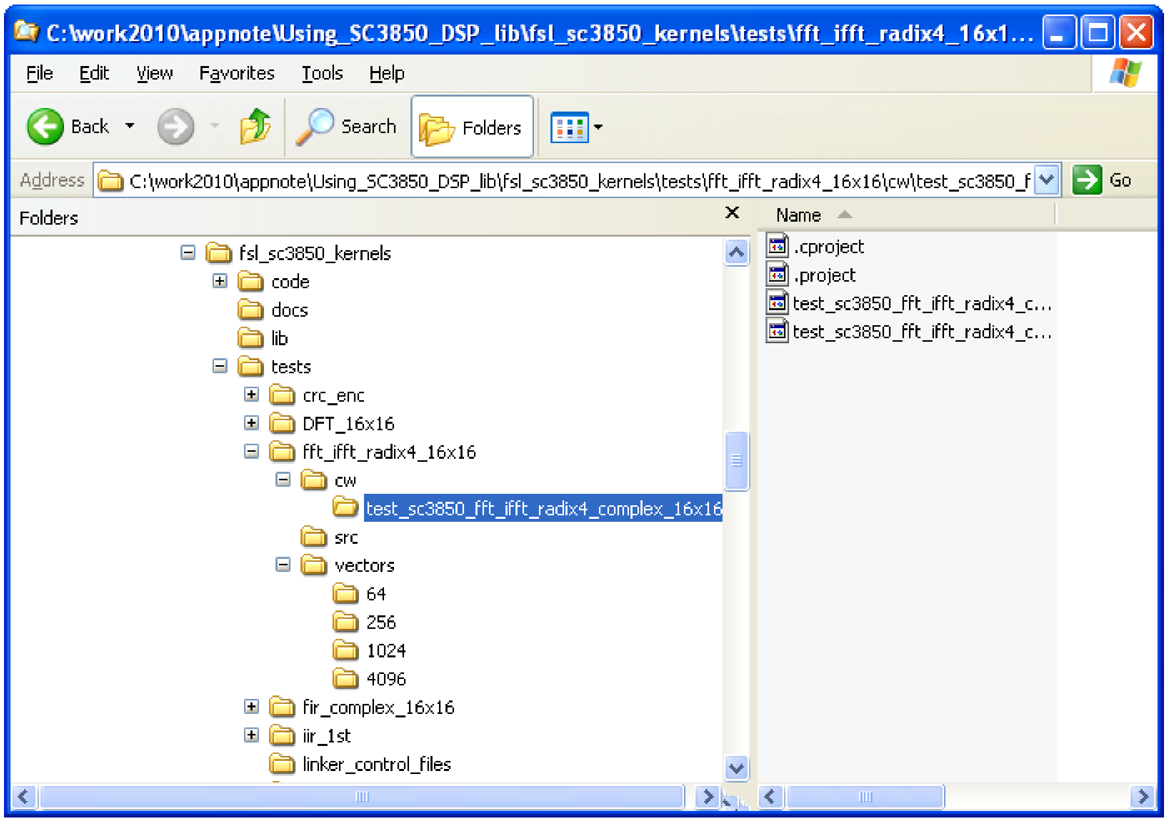
Figure 8. Build the ASM library (SC3850_ASM) or C library (SC3850_C)

6.2 Using the complex Fast Fourier Transform (complex FFT)

This section presents how to use complex radix-4 FFT/IFFT with 16 bit precision. Since the implementation of complex FFT and complex IFFT kernels are very similar, we will focus on the complex FFT kernel.

6.2.1 Test project structure

To demonstrate how to use the kernels and show their performance, CodeWarrior test projects are provided in the package. The CodeWarrior test project for complex FFT/IFFT kernels is located in `\fsl_sc3850_kernels\tests`. Each test project has project files, test harness source code, and test vectors subdirectories, as shown in this figure.


C:\work2010\appnote\Using_SC3850_DSP_lib\fsl_sc3850_kernels\tests\fft_ifft_radix4_16x1...
File Edit View Favorites Tools Help
Back
Search
Folders
Address
C:\work2010\appnote\Using_SC3850_DSP_lib\fsl_sc3850_kernels\tests\fft_ifft_radix4_16x16\cw\test_sc3850_f
Go
Folders
Name

```
⊟ fsl_sc3850_kernels
    ⊞ code
      docs
      lib
    ⊟ tests
        ⊞ crc_enc
        ⊞ DFT_16x16
        ⊟ fft_ifft_radix4_16x16
            ⊟ cw
                  test_sc3850_fft_ifft_radix4_complex_16x16
              src
            ⊟ vectors
                  64
                  256
                  1024
                  4096
        ⊞ fir_complex_16x16
        ⊞ iir_1st
          linker_control_files
```

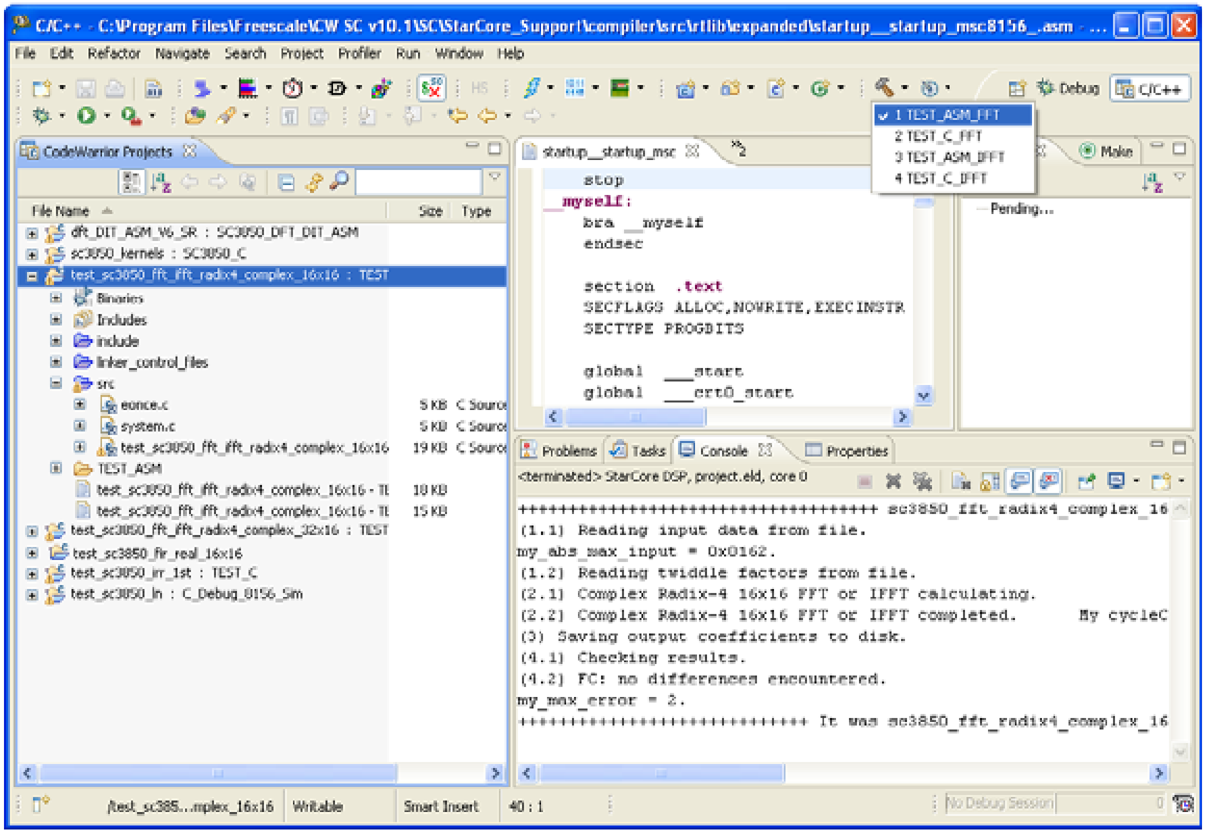
```
.cproject
.project
test_sc3850_fft_ifft_radix4_c...
test_sc3850_fft_ifft_radix4_c...
```

Figure 9. Complex FFT/IFFT kernel test project structure

Users are referred to [•] for detailed algorithms, implementations, source codes, and test vectors of the complex FFT/IFFT kernels

6.2.2 Build test projects

To build complex FFT/IFFT test project, select the corresponding test project in the project window and the interested build configuration. Figure 10 shows how to build the test project with the `sc3850_fft_radix4_complex_16x16_asm` kernel. A binary file (`.eld`) are created in `fsl_sc3850_kernels\tests\fft_ifft_radix4_16x16\cw` after building.



```
    stop
__myself:
    bra __myself
    endsec

    section  .text
    SECFLAGS ALLOC,NOWRITE,EXECINSTR
    SECTYPE PROGBITS

    global  ___start
    global  ___crt0_start
```

<terminated> StarCore DSP, project.eld, core 0

```
++++++++++++++++++++++++++++++++++++ sc3850_fft_radix4_complex_16
(1.1) Reading input data from file.
my_abs_max_input = 0x0162.
(1.2) Reading twiddle factors from file.
(2.1) Complex Radix-4 16x16 FFT or IFFT calculating.
(2.2) Complex Radix-4 16x16 FFT or IFFT completed.      My cycleC
(3) Saving output coefficients to disk.
(4.1) Checking results.
(4.2) FC: no differences encountered.
my_max_error = 2.
+++++++++++++++++++++++++++++ It was sc3850_fft_radix4_complex_16
```

Figure 10. Build test project

6.2.3 Debug and run test projects on MSC8156 ADS

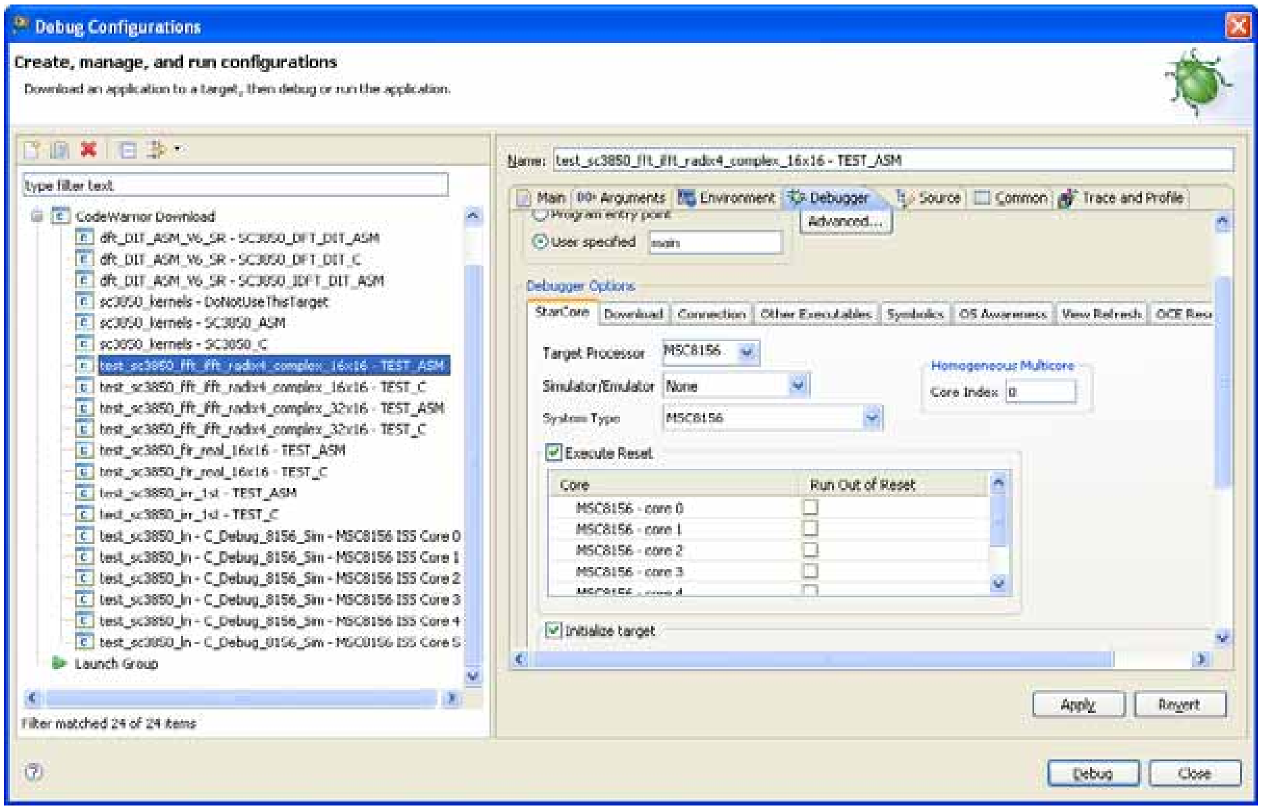
Use the following steps to debug and run test projects:

- Select `Run > Debug Configurations`. The Debug Configurations dialog box appears.
- Select `CodeWarrior Download > test_sc3850_fft_ifft_radix4_complex_16x16 - TEST_ASM` in the left pane if you want to test the `sc3850_fft_radix4_complex_16x16_asm` kernel. You must build the test project with a specific kernel before running it.
- Click the Debugger tab. The Debugger page appears in the right pane. Change the settings on this page as per your requirements. For example, select the required target processor and simulator/emulator. The default setting is shown in Figure 11, and the configurations are set to run on MSC8156 ADS board.
- Click the Connection tab in the Debugger page.
- Make sure Ethernet TAP is selected from the Connection drop-down list.
- If you make any changes, click the Apply button.
- Click Debug. The program downloads to hardware. The Debug perspective appears and the execution halts at the first statement of main().
- Control program:

— Click Step Over . The debugger executes the current statement and halts at the next statement.

— Click Resume . The debugger executes all statements if there are no breakpoints.

— Click Terminate . The debug session ends.



Debug Configurations

Create, manage, and run configurations

Download an application to a target, then debug or run the application.

type filter text

- CodeWarrior Download
  - dft_DIT_ASM_V6_SR - SC3850_DFT_DIT_ASM
  - dft_DIT_ASM_V6_SR - SC3850_DFT_DIT_C
  - dft_DIT_ASM_V6_SR - SC3850_IDFT_DIT_ASM
  - sc3850_kernels - DoNotUseThisTarget
  - sc3850_kernels - SC3850_ASM
  - sc3850_kernels - SC3850_C
  - test_sc3850_fft_ifft_radix4_complex_16x16 - TEST_ASM
  - test_sc3850_fft_ifft_radix4_complex_16x16 - TEST_C
  - test_sc3850_fft_ifft_radix4_complex_32x16 - TEST_ASM
  - test_sc3850_fft_ifft_radix4_complex_32x16 - TEST_C
  - test_sc3850_fir_real_16x16 - TEST_ASM
  - test_sc3850_fir_real_16x16 - TEST_C
  - test_sc3850_iir_1st - TEST_ASM
  - test_sc3850_iir_1st - TEST_C
  - test_sc3850_ln - C_Debug_8156_Sim - MSC8156 ISS Core 0
  - test_sc3850_ln - C_Debug_8156_Sim - MSC8156 ISS Core 1
  - test_sc3850_ln - C_Debug_8156_Sim - MSC8156 ISS Core 2
  - test_sc3850_ln - C_Debug_8156_Sim - MSC8156 ISS Core 3
  - test_sc3850_ln - C_Debug_8156_Sim - MSC8156 ISS Core 4
  - test_sc3850_ln - C_Debug_8156_Sim - MSC8156 ISS Core 5
- Launch Group

Filter matched 24 of 24 items

Name: test_sc3850_fft_ifft_radix4_complex_16x16 - TEST_ASM

Main | Arguments | Environment | Debugger | Source | Common | Trace and Profile

Program entry point

User specified: main

Advanced...

Debugger Options

StarCore | Download | Connection | Other Executables | Symbolics | OS Awareness | View Refresh | OCE Reg

Target Processor: MSC8156

Simulator/Emulator: None

System Type: MSC8156

Homogeneous Multicore

Core Index: 0

[✓] Execute Reset

Core	Run Out of Reset
MSC8156 - core 0	☐
MSC8156 - core 1	☐
MSC8156 - core 2	☐
MSC8156 - core 3	☐
MSC8156 - core 4	☐

[✓] Initialize target

Apply | Revert

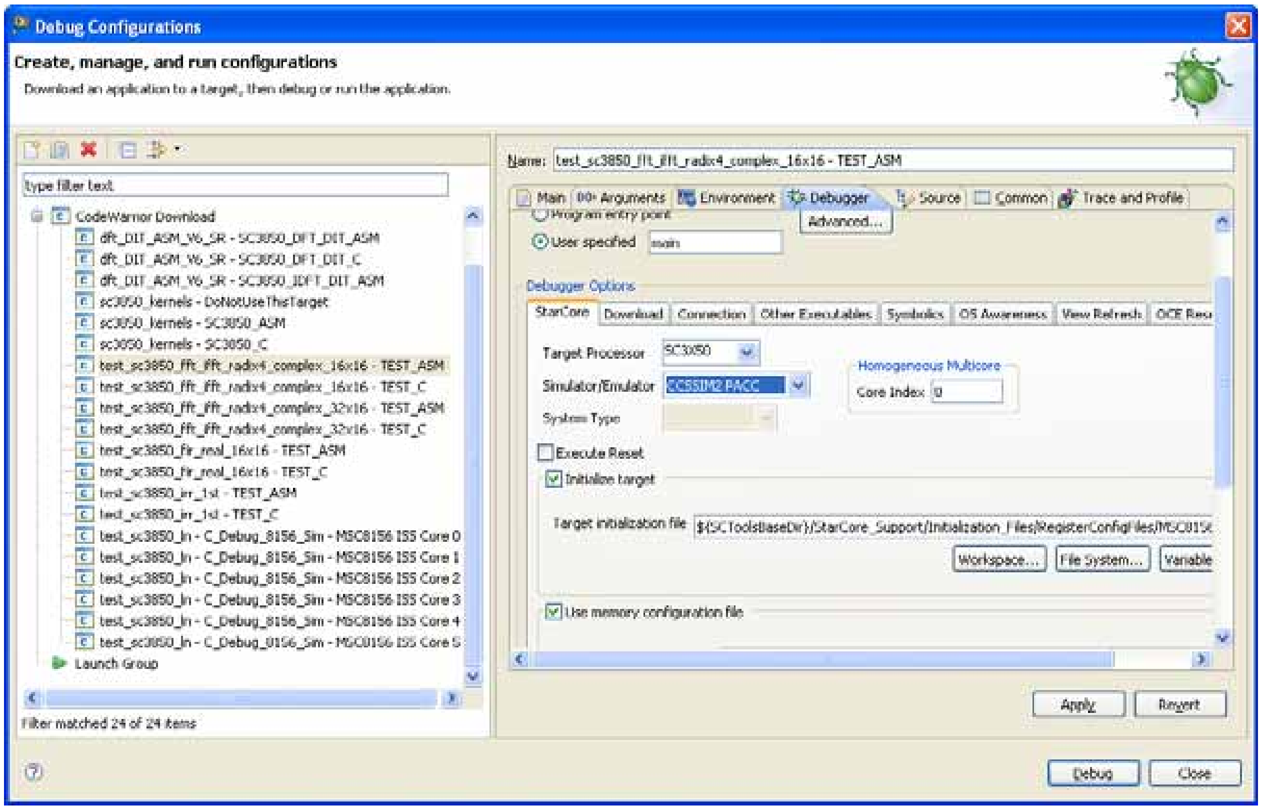
Debug | Close

Figure 11. Debug configuration on MSC8156 ADS board

6.2.4 Debug and run test projects on SC3850 PACC simulator

Use the following steps to debug and run test projects on the simulator:

- Select `Run > Debug Configurations`. The Debug Configurations dialog box appears.
- Select `CodeWarrior Download > test_sc3850_fft_ifft_radix4_complex_16x16 - TEST_ASM` in the left pane if you want to test the `sc3850_fft_radix4_complex_16x16_asm` kernel.
- Click the Debugger tab. The Debugger page appears in the right pane.
- Click the StarCore tab in the Debugger page. Select `SC3x50` for target processor and `CCSSIM2 PACC` for simulator/emulator, as shown in Figure 12.
- Click the Connection tab in the Debugger page. Make sure `Generic` is selected from the Connection drop-down list.
- Click the Apply button.
- Click `Debug.` The program downloads to hardware. The Debug perspective appears and you can debug the program from this window.


Debug Configurations

Create, manage, and run configurations

Download an application to a target, then debug or run the application.

type filter text

- CodeWarrior Download
  - dft_DIT_ASM_V6_SR - SC3850_DFT_DIT_ASM
  - dft_DIT_ASM_V6_SR - SC3850_DFT_DIT_C
  - dft_DIT_ASM_V6_SR - SC3850_IDFT_DIT_ASM
  - sc3850_kernels - DoNotUseThisTarget
  - sc3850_kernels - SC3850_ASM
  - sc3850_kernels - SC3850_C
  - test_sc3850_fft_ifft_radix4_complex_16x16 - TEST_ASM
  - test_sc3850_fft_ifft_radix4_complex_16x16 - TEST_C
  - test_sc3850_fft_ifft_radix4_complex_32x16 - TEST_ASM
  - test_sc3850_fft_ifft_radix4_complex_32x16 - TEST_C
  - test_sc3850_fir_real_16x16 - TEST_ASM
  - test_sc3850_fir_real_16x16 - TEST_C
  - test_sc3850_iir_1st - TEST_ASM
  - test_sc3850_iir_1st - TEST_C
  - test_sc3850_ln - C_Debug_8156_Sim - MSC8156 ISS Core 0
  - test_sc3850_ln - C_Debug_8156_Sim - MSC8156 ISS Core 1
  - test_sc3850_ln - C_Debug_8156_Sim - MSC8156 ISS Core 2
  - test_sc3850_ln - C_Debug_8156_Sim - MSC8156 ISS Core 3
  - test_sc3850_ln - C_Debug_8156_Sim - MSC8156 ISS Core 4
  - test_sc3850_ln - C_Debug_8156_Sim - MSC8156 ISS Core 5
- Launch Group

Filter matched 24 of 24 items

Name: test_sc3850_fft_ifft_radix4_complex_16x16 - TEST_ASM

Main | Arguments | Environment | Debugger | Source | Common | Trace and Profile

Program entry point

User specified: main

Advanced...

Debugger Options

StarCore | Download | Connection | Other Executables | Symbolics | OS Awareness | View Refresh | OCE Reg

Target Processor: SC3X50

Simulator/Emulator: CCSSIM2 PACC

System Type:

Homogeneous Multicore

Core Index: 0

[ ] Execute Reset

[✓] Initialize target

Target initialization file: ${SCToolsBaseDir}/StarCore_Support/Initialization_Files/RegisterConfigFiles/MSC815

Workspace... | File System... | Variable

[✓] Use memory configuration file

Apply | Revert

Debug | Close

Figure 12. Debug configuration on SC3850 PACC simulator

6.2.5 In-place implementation

The in-place complex FFT/IFFT algorithms are implemented on the SC3850 core. An in-place FFT is simply an FFT that is calculated entirely inside its original sample memory. In our complex FFT/IFFT kernels, one data buffer is used for input and output data. Thus, after calculation, the input data is not valid because the input data are overwritten by intermediate data or output data. The complex FFT/IFFT kernels can run on cold cache and warm cache. Warm cache gives us better performance because it minimizes the memory access overhead. In the warm cache case, we simply run the function twice, where the first function run is used to warm up the cache. Since an in-place FFT is implemented, the input data are not valid for the second run of the function. Thus, warm cache approach is used for cycle count measurement. On the other hand, cold cache approach is used for bit exactness measurement. Note that in real applications, other methods can be used to warm up the cache, such as data/instruction prefetching.

6.2.6 Function API complex Radix-4 FFT 16 $\times$ 16

Radix-4 complex FFT with 16-bit input and 16-bit output. Input and output complex data are stored in the form `[real][imag]`. Radix-4 supports 64, 256, 1024, and 4096 point FFTs. Please refer to [•] for the complex FFT algorithms and implementation details. The API of the complex IFFT kernel is very similar to the complex FFT kernel, and, therefore, is not described in this application note.

Example 2. Radix-4 FFT 16 × 16 API

```
void sc3850_fft_radix4_complex_16x16_asm (Word16 data_buffer[],
    Word16 wctwiddles[],
    Word16 wbdtwiddles[],
    Word16 n,
    Word16 ln,
    Word16 Shift_down);
```

Input:

```
Word16 data_buffer[]        // Address of Input and Output Buffer. Input and
                     // output share one memory area pointed by data_buffer.
Word16 wctwiddles[]              // Address of the array of twiddle factor Wc
Word16 wbdtwiddles[]     // Address of the array of twiddle factor Wb and Wd
Word16 n                                                         // FFT point
Word16 ln                             // Base 4 Log(N). Number of FFT stages
Word16 Shift_down                    // Scaling down parameter at each stage
                                               // Shift_down = 0; No scaling
                        // Shift_down = 1; scaling down (dividing) by 2
                        // Shift_down = 2; scaling down (dividing) by 4
```

Output:

```
Word16 data_buffer[]// Address of Input and Output Buffer. Input and output share
                               // one big memory area pointed by data_buffer.
```

Data alignment requirement:

```
data_buffer                                                            4N
wctwiddles                                                              N
wbdtwiddles                                                            2N
```

The input parameter `Shift_down` is used to avoid possible overflow in the FFT calculation by normalizing the input before the 1st stage and then scaling it down by 4 at the output of every stage. `Scaling by 2` and `no scaling` modes can be used for short FFTs only, that is, for N = 64, 256, and for vectors with appropriate low amplitude values.

6.2.7 Check results

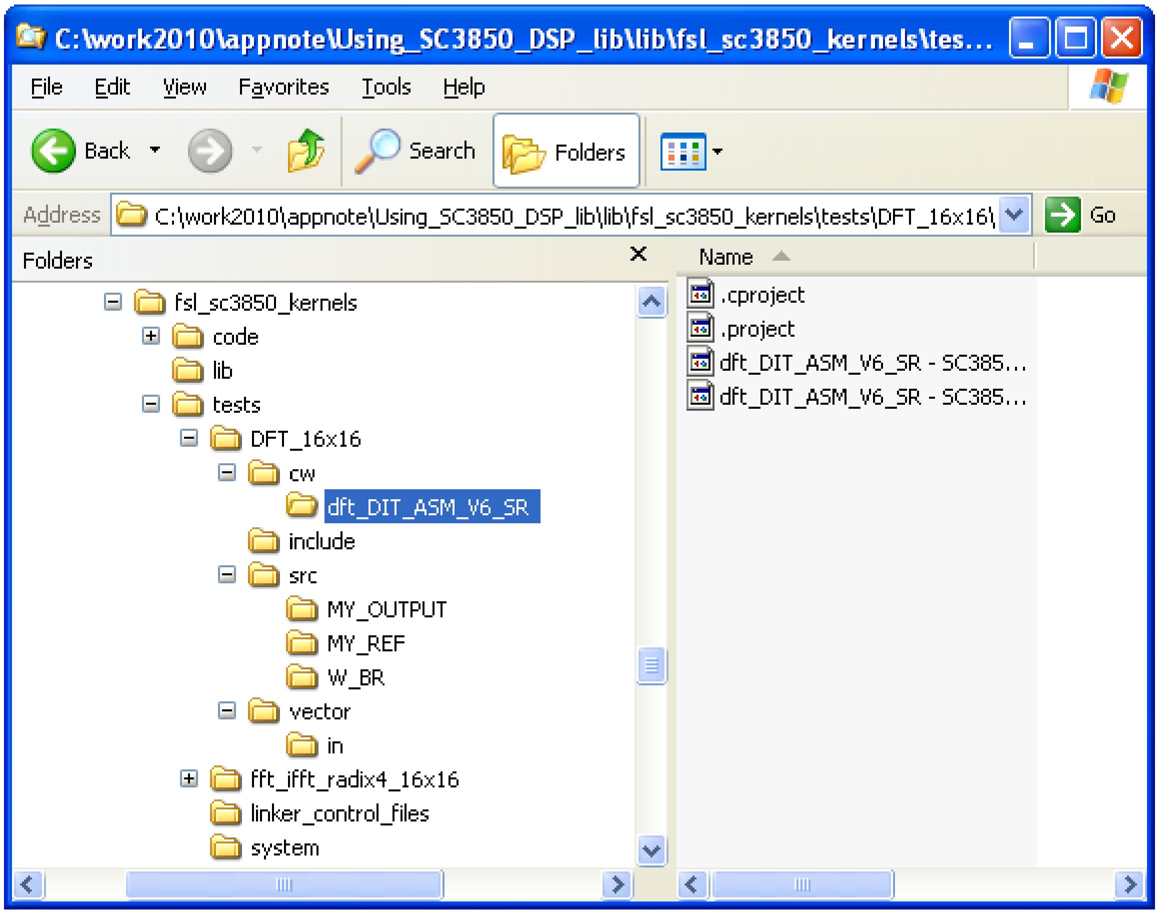
Reference complex FFT/IFFT outputs are pre-calculated and stored in data files. The test code loads the reference outputs and compares the outputs of the complex FFT/IFFT kernel with the reference ones. If the maximum absolute difference is smaller than a threshold, then we say the kernel passes result checking.

6.3 Complex Discrete Fourier Transform (complex DFT)

This section presents how to use complex DFT kernel with 16 bit precision.

6.3.1 Test project structure

The CodeWarrior test project for complex DFT kernels is located in `\fsl_sc3850_kernels\tests`. Each test project has project files, test harness source code, and test vectors subdirectories, as shown in this figure.


C:\work2010\appnote\Using_SC3850_DSP_lib\lib\fsl_sc3850_kernels\tes...
File
Edit
View
Favorites
Tools
Help
Back
Search
Folders
Address
Go
Folders
Name

```
Address  C:\work2010\appnote\Using_SC3850_DSP_lib\lib\fsl_sc3850_kernels\tests\DFT_16x16\

Folders
⊟ fsl_sc3850_kernels
    ⊞ code
      lib
    ⊟ tests
        ⊟ DFT_16x16
            ⊟ cw
                  dft_DIT_ASM_V6_SR
              include
            ⊟ src
                  MY_OUTPUT
                  MY_REF
                  W_BR
            ⊟ vector
                  in
        ⊞ fft_ifft_radix4_16x16
          linker_control_files
          system

Name
.cproject
.project
dft_DIT_ASM_V6_SR - SC385...
dft_DIT_ASM_V6_SR - SC385...
```

Figure 13. Complex DFT/IDFT kernel test project structure

Users are referred to [•] for detailed algorithms, implementations, source codes, and test vectors of the complex DFT/IDFT kernels

6.3.2 Build, debug and run test projects on MSC8156 ADS

Users can follow the same steps described in Section 6.2.2, Section 6.2.3 and Section 6.2.4 to build, debug and run the DFT test projects on MSC8156 ADS and SC3850 PACC simulator.

6.3.3 Function API complex DFT 16 $\times$ 16

Complex DFT has 16-bit input and 16-bit output. Input and output complex data are stored in the form [real][imag]. It implements mixed radix-2/3/4/5 FFTs with reduced computation and combines them together to support DFT of size $N = 2^k\, 3^l\, 4^m\, 5^n$ (k, l, m, and n are positive integers). For example, a DFT of size 1200 can be calculated in five stages using radices of 5, 4, and 3 (1,200 = 5 x 5 x 4 x 4 x 3). Please refer to [•] for the complex DFT algorithms and implementation details. The API of the complex IDFT kernel is very similar to the complex DFT kernel, and, therefore, is not described in this application note. Twiddle factors are precalculated. Bit-reversed addressing is not performed in this implementation. The Digit Reversed Address "psiDigitReversedAddress" is an offset for digit reversed address.

Example 3. Complex DFT 16 × 16 API

```
void sc3850_dft_dit_complex_16x16_auto_scale_asm( fft_arg *psFit );
```

Structure Definition:

```
typedef struct fft_arg_t
{
   short *psiIn;// Pointer to Input Buffer
   short *psiOut;// Pointer to Output Buffer
   short *psiNumRadix;// Pointer to the array of radix every stage
   short *psiNumButterfly;// Pointer to the array of # of butterflies every subgroup
   short *psiNumSubgroup;// Pointer to the array of number of subgroups every stage
   short *psiNumRadixOffset;// Pointer to the array of # of each radix in/out offset
   short *psiDigitReversedAddress;// Pointer to the Digit Reversed Address
   short *psiWb;// Pointer to the array of twiddle factor Wb
   short *psiWc;// Pointer to the array of twiddle factor Wc
   short *psiWd;// Pointer to the array of twiddle factor Wd
   short *psiWe;// Pointer to the array of twiddle factor We
   short *psiDFTpoint;// Pointer to the DFT point
   short *psiScale;// Pointer to the scaling factor every butterfly stage
} fft_arg;
```

Input:

```
short *psiIn;              // Pointer to Input Buffer
short *psiNumRadix;        // Pointer to the array of radix every stage
short *psiNumButterfly;   // Pointer to the array of # of butterflies every subgroup
short *psiNumSubgroup;    // Pointer to the array of number of subgroups every stage
short *psiNumRadixOffset; // Pointer to the array of # of each radix in/out offset
short *psiDigitReversedAddress; // Pointer to the Digit Reversed Address
short *psiWb;              // Pointer to the array of twiddle factor Wb
short *psiWc;              // Pointer to the array of twiddle factor Wc
short *psiWd;              // Pointer to the array of twiddle factor Wd
short *psiWe;              // Pointer to the array of twiddle factor We
short *psiDFTpoint;        // Pointer to the DFT point
short *psiScale;           // Pointer to the scaling factor every butterfly stage
```

Output:

```
          short *psiOut;            // Pointer to Output Buffer
```

Two scaling methods are implemented in the complex DFT kernel to prevent overflowing. The first method is to scale down the input data by a fixed number of bits at each stage without considering the range of the input data. This is called fixed scaling method. Another method is to detect the bit growth at the output of each DFT stage. Once the number of bit growth is known, the number of scaling bits can be determined to reverse the bit growth. This method is called automatic scaling method, which provides better precision over the fixed scaling method.

6.3.4 Check results

Reference complex DFT/IDFT outputs are pre-calculated and stored in data files. The outputs of the DFT/IDFT kernel are compared with the reference outputs. If the maximum absolute difference is smaller than a threshold, then we say the kernel passes result checking.

7 Revision history

This table provides a revision history for this document.

Table 1. Document revision history

Rev. number	Date	Substantive change(s)
2	05/2012	Updated note on first page from “The supporting software library is available as a zip file under NDA only. Contact your local sales office or representative for details” to “The supporting software library is available as zip file AN4207SW on freescale.com.”
1	12/2010	Added note on first page.
0	11/2010	Public release

How to Reach Us:

Home Page:
freescale.com

Web Support:
freescale.com/support

Document Number: AN4207
Rev. 2
05/2012