CodeWarrior Development Studio for StarCore 3900FP DSP Architectures Simulator User Guide

Document Number: CWSCSIMUG
Rev. 10.9.0, 06/2015

Contents

Chapter 1
Introduction

This manual describes the software simulation models released with CodeWarrior Development Studio for StarCore Architecture.

This chapter describes:

- Overview
- Accompanying Documentation
- Simulator System Requirements

1.1 Overview

This section provides details about the simulator models supported by CodeWarrior Development Studio for StarCore Architecture.

Following is the list of supported simulator models:

- Instruction Set Simulators (ISS): The output and behavior of the instruction-set simulators matches that of the architecture respectively. These models can execute 10 to 100 times faster than the cycle-accurate models (in case a corresponding cycle accurate model exists). However, these models do not produce cycle counts, which are useful for system performance analysis. The instruction-set simulators supported by the CodeWarrior Development Studio for StarCore DSPs are:
  - SC3850_Platform_ISS
  - SC3900_Platform_ISS
  - B4860 ISS
  - B4420 ISS
  - MSC8154_ISS simulator for SC3850 DSP platforms
  - MSC8156_ISS simulator for SC3850 DSP platforms
  - MSC8157_ISS simulator for SC3850 DSP platforms
  - MSC8158_ISS simulator for SC3850 DSP platforms

- Performance Accurate Simulators (PACC): These are clock-accurate platform simulators. For 3850, the clock difference between PACC and the corresponding chip is about 5%. Refer the Accuracy Benchmarks section for more information. The performance-accurate simulators supported by the CodeWarrior Development Studio for StarCore DSPs are:
  - SC3850_Platform_PACC
  - SC3900plat_PACC
- Unified simulators: These simulators allow switching between platform PACC and platform ISS modes during the application execution. These devices resolve the typical disadvantages of platform ISS (absent of clock accuracy) and platform PACC (slow speed). The unified simulators supported by the CodeWarrior Development Studio for StarCore DSPs are:
  - SC3850plat

NOTE

The current release may not support all simulator targets referenced in this document.

1.2 Accompanying Documentation

The Documentation Roadmap page describes the documentation included in this version of CodeWarrior Development Studio for StarCore DSP Architectures.

You can access Documentation Roadmap by:

- Clicking a shortcut link on the Desktop that the installer creates by default
- Opening `START_HERE.html` in `CWInstallDir\SC\Help` folder

1.3 Simulator System Requirements

This section lists the system requirement for installing simulators.

The following table lists the system requirements.

Table 1-1. Simulator System Requirement

Operating System	Hardware, Software
Windows	IBM®-x86-compatible host computer: 2 gigahertz with 4GB RAM
	32-bit and 64-bit x86 compatible workstations are supported
	Microsoft® Windows 7 Home Premium, Professional, Ultimate Operating System
	Microsoft® Windows Vista® (SP2) Home Basic, Home Premium, Business, Enterprise, Ultimate Operating System
	Windows Server 2012 R2
Linux	IBM®-x86-compatible host computer: 2 gigahertz with 4GB RAM
	32-bit and 64-bit x86 compatible workstations are supported
	Red Hat® Enterprise Linux (RHEL) 5 or Later

NOTE

For more information on installing simulator tools, refer to *Quick Start for StarCore DSPs* at the following location:

`<`*CWInstallDir*`>\sc`, where *CWInstallDir* is the StarCore installation folder.

Chapter 2
Accessing Simulators

This chapter explains ways to access the ISS and PACC simulators.

- From the command line using section runsim.
- From the CodeWarrior IDE using section CCSSIM.

The following table lists a summary of features of the ISS and PACC described in this manual.

Table 2-1. Simulator Features

Simulators	Speed	Cycle Accuracy
ISS	Fast	N/A
PACC	Medium	<95-105% cycle accurate

2.1 runsim

`runsim` is a command-line simulator used to perform various tasks.

Following tasks are performed by `runsim`, such as:

- Execute an application and display program output in the command window. This execution is much faster than interactive execution with the debugger, so is particularly appropriate for long execution runs.
- Direct an alternate device, such as a different simulator, to execute your application.
- Apply many options to application execution, such as outputting time information or callback messages, printing program flow, printing values of some registers, and performing a bus trace.

- Collect and output high-level profiling information, such as function size and the number of times functions are called.
- Collect and output low-level profiling information, such as the number of times an assembler instruction is executed. This task is not supported for sc3900plat_pacc device.

NOTE

By default, `runsim` stops at the `__crt0_end` label, which the C compiler automatically generates. If application is written in assembly language then the `__crt0_end` label needs to be entered manually.

NOTE

Also, the default entry point is the value specified by symbol `__crt0_start`, generated by the C compiler as well. If this symbol is not available then the ELF's entry point is used. Entry point overriding can be done by using command line option `-startlbl <start_label | start_addr>`. Whether the argument represents a label or a number, it is detected automatically.

The format for the `runsim` command is:

```
runsim <options> <file.eld>
```

where:

`<options>` are one or more of the `runsim` flags, and

`<file.eld>` is the `.eld` format file to be executed.

NOTE

Some of the command-line options may not be supported by a CodeWarrior build tool simulator model. For example, the command-line option `-nc` for multicore is only valid for the MSC8154ISS, MSC8151ISS, MSC8152ISS, MSC8156ISS, and MSC8157ISS simulators.

The following command runs the `file.eld` application on device, `<device_name>`, which is supported by `runsim`:

```
runsim <-d <device_name>> <file.eld>
```

The `-d` argument is used to select the device. The below table lists the various devices that you can specify with the `-d` argument in the `runsim` command. You can view the flags and devices supported by the `runsim` command using the help flag, `-h`:

```
runsim -h
```

Table 2-2. Supported Devices

Argument for -d Option	Device
`sc3850plat_iss`	SC3850 instruction accurate platform model
`sc3850plat_pacc`	SC3850 performance accurate platform model
`msc8157iss`	MSC8157 chip instruction accurate simulator
`msc8158iss`	MSC8158 chip instruction accurate simulator
`msc8156iss`	MSC8156 chip instruction accurate simulator
`msc8154iss`	MSC8154 chip instruction accurate simulator
`msc8256iss`	MSC8256 chip instruction accurate simulator
`msc8254iss`	MSC8254 chip instruction accurate simulator
`msc8252iss`	MSC8252 chip instruction accurate simulator
`msc8251iss`	MSC8251 chip instruction accurate simulator
`msc8151iss`	MSC8151 chip instruction accurate simulator
`msc8152iss`	MSC8152 chip instruction accurate simulator
`sc3850plat`	StarCore SC3850 unified iss and pacc model. Available in Gearshift Mode.
`sc3900plat_iss`	SC3900 instruction accurate platform model
`sc3900plat_pacc`	SC3900 performance accurate platform model
`b4860iss`	B4860 functional SoC model
`b4420iss`	B4420 functional SoC model

2.1.1 Using runsim with Simulator

This section describes the steps to run your application using runsim.

`runsim` is a simple, generic command-line simulator front end that can run target application programs. It provides standard host I/O capabilities, such as `printf()` and general file I/O handling.

Perform the following steps to run your application using runsim :

1. Make sure that the `SC\ccs\bin` folder is included in the `PATH` environment variable (Windows) or in the `PATH` and `LD_LIBRARY_PATH` environment variables (Unix/Linux).

NOTE

The `runsim` command-line simulator can be accessed from the following location:

`<`*CWInstallDir*`>\SC\ccs\bin`

where *CWInstallDir* is the CodeWarrior installation directory.

2. Open a command window.
   a. Select **Start > Run**.

      The **Run** dialog box appears.

   b. In the **Open** text box, enter `cmd`.
   c. Click OK.

      A command window appears, showing the default directory and the `>` prompt.

3. Run your application.
   a. Enter the command `cd`, followed by the path to the `<`*CWInstallDir*`>\SC\ccs\bin` folder, where *CWInstallDir* is the CodeWarrior installation directory
   b. Press the **Enter** key.

      The command window changes to the CodeWarrior Tools `SC\ccs\bin` directory.

   c. Enter the command `runsim`, followed by appropriate options and the name of the `.eld` file to be executed.
   d. Press the **Enter** key.

      The execution starts and the program results appear in the command window, below the `runsim` command.

Following are the examples of the `runsim` command:

- runsim executes delta.eld using sc3850 platform simulator, providing an output profiling table that compares function size and execution cycles.

```
runsim -d sc3850plat_pacc -p file_trace_size_cycle.rep delta.eld
```

2.1.2 Using CheckPoint Feature

The CheckPoint feature allows you to stop the simulator that is running and then save the simulator state. A checkpoint is created for saving the simulator state and you can restart the simulation later from this checkpoint.

You can switch between the ISS and PACC simulation models on the checkpoints. For example, you can start running `sc3850plat_iss`, create a checkpoint, and run the simulation again using `sc3850plat_pacc`.

The CheckPoint feature is supported only for the following single-core simulators:

- `sc3850plat_iss`
- `sc3850plat_pacc`

NOTE

The CheckPoint feature is currently supported only with `runsim`.

Following sections describe the usage, benefits and limitations of the CheckPoint feature:

- Usage
- Benefits
- Limitations

2.1.2.1 Usage

A checkpoint is set by putting the target into the debug mode. Then, all the registers and memory are dumped to the checkpointing file.

You can specify a checkpoint using the following three methods:

- By address-Use it for selecting only one checkpoint. A break point is put at the specified address.
- By cycle number-Use it to select one or more checkpoints. After the given cycle number, the core is put into the debug mode by setting the core to STEP mode and then flushing the pipe only after it exits any hardware loops.
- By time-Use it to select multiple check points. Similar to the By cycle number method, the core is put into the debug mode. You can have all the checkpoints in the command line.

You can resume from a checkpoint by recreating the original state. You can recreate the original state by reading the registers and memory location from the file. You can create a checkpoint with a device (`-d` option) and restore the checkpoint with another device that is compatible with the first one.

The below table lists the `runsim` flags used to activate the CheckPoint feature.

Table 2-3. Flags used to activate the CheckPoint feature

Flag	Description
`-checkpoint_file <fname>`	Sets the checkpoint file.
`-checkpoint_addr [hex]`	Specifies the program's check point by the PC address; that is, the StarCore core register, which points to the next Variable Length Execution Set (VLES) to be executed. A breakpoint will be put there.
`-checkpoint_count [dec]`	Specifies the number of hits at given `checkpoint_addr` before creating the checkpoint.
`-checkpoint_cycle [dec]`	Specifies the cycle number at which the program will be checkpointed. You can give multiple check point cycles.
`-checkpoint_cycle_inc [dec]`	Increments the first given checkpointing cycle number (by the number specified in decimals) to define the new checkpointing cycle.
`-checkpoint_time_start [dec]`	Enables the checkpoint after the first `checkpoint_time_start` minutes. The default is `0`, which specifies the start of the program.
`-checkpoint_time_min [dec]`	Takes a new checkpoint from every `checkpoint_time_min` minutes (specified in decimals) after `checkpoint_time_start` minutes from the start of the program.
`-checkpoint_overwrite`	Overwrites the previous checkpoint file with the last one. In the default behavior, the next file will have the counter appended to the end of its name (`file_0`, `file_1`).
`-checkpoint_stop_execution`	Exits after taking the first checkpoint.
`-checkpoint_restore`	Restores from the checkpoint.

NOTE

The state of the simulator is written to the file specified by the `checkpoint_file` parameter. If multiple checkpoints are set while the application is running, the overwriting policy is selected with the `checkpoint_overwrite` parameter. The default is no overwriting.

2.1.2.1.1 Examples

- The following command checkpoints the program at address, `0xdb52` for `titanium_iss`:

  ```
  runsim -checkpoint_file test -checkpoint_addr 0xdb52 -d titanium_iss test.eld
  ```

- The following command resumes the program from the last checkpoint made for `titanium_pacc`:

  ```
  runsim -checkpoint_file test -checkpoint_restore -d titanium_pacc
  ```

- The following command specifies the number of hits at address, `0xdb52` for `Sc3850issp`:

  ```
  runsim -checkpoint_file test -checkpoint_addr 0xdb52 -checkpoint_count 7 -d sc3850issp
  test.eld
  ```

- The following command specifies the cycle number `1500000` at which the program `sc3850issp` will be checkpointed:

  ```
  runsim -checkpoint_file test -checkpoint_cycle 1500000 -d sc3850issp test.eld.
  ```

- The following command increments the first given checkpointing cycle number `1000000` by `500000`:

  ```
  runsim -checkpoint_file test -checkpoint_cycle 1000000 -checkpoint_cycle_inc 500000 -d
  sc3850issp test.eld
  ```

2.1.2.2 Benefits

This section lists the benefits of using CheckPoint feature.

Following list describes the benefits:

- Speed up the performance analysis and profiling

  Some applications have a huge initialization part which makes performance analysis slower. Using the CheckPoint feature, you can run `plat_iss` faster until initialization part gets completed, create a check point, and then restart the application running with slower `plat_pacc`.

- Speed up debugging

  CheckPoint feature helps in speeding up debugging when an application fails after running for long. You can create the check point exactly before the problem case and then start running the simulator with trace, exactly before the problem case.

- Generate problem report

  External customers may not want to provide the binary `.eld` files due to some IP security issues. In such a case, they can get the file with the checkpoint status only, along with other input files that the application is using, and then run the simulator from the checkpoint file for the problem debugging.

2.1.2.3 Limitations

This section provides details on the limitations of CheckPoint feature.

Following is the list of limitations:

- It can be used only for SC platform simulators or SC core simulators.
- It has limited support for the targets: 8158, 8157, 8156, 8154, 8151, 8152, B4860, and B4420.
- It does not support additional `runsim` command-line parameters.

NOTE

It is not advisable to use non standard arguments on `runsim` command line when using parameters related to checkpointing. The name of the parameters related to checkpointing starts with `checkpoint_`. You can use `-acccfg` parameter to change `cfg.so` file for plat pacc.

- Caches get flushed before checkpointing and empty after restarting.
- DPU counters are reset after restarting from checkpointing.
- Valid result for any performance measurement is not guaranteed.
- The checkpointing file has to have the same relative path as the `.eld` file that was checkpointed to the files used while the application was running.

2.1.3 Using Hardware Port Feature

The Hardware Port feature allows memory mapping of some addresses into input / output files. If you have files formatted to one 32bit hexadecimal number per line, you can easily read(write) from(to) a variable by having it always pointing to the next value in the files.

The applications, using Hardware Port feature, run significantly faster than the same applications, using standard C input/output library functions.

NOTE

For 3850 based devices, entering debug mode may result in the Hardware Port to return a wrong value because of access to the files that is used to read the values from, resulting in a desynchronization with the input files.

Following sections describe the usage, benefits and limitations of the Hardware Port feature:

2.1.3.1 Usage

To activate Hardware Port feature, you need to use runsim flag.

The following `runsim` flag is used to activate the Hardware Port feature:

```
-smodel "<port_type> [-v] [endianness] <address> [config_file] <path> [non_recursive]"
```

The following table describes the `-smodel` parameters used to activate the Hardware Port feature.

Table 2-4. Parameters used to activate the Hardware Port feature

Parameter	Description
`<port_type>`	Differentiates whether the other parameters are declared for reading or writing to a port. The supported options are: • `Rx_port`, for reading • `Tx_port`, for writing. **NOTE:** Each line needs to be `12 Bytes` long, for example "`0x12345678\r\n`", in the files used by `Rx_port`.
`[-v]`	Displays the debugging information for Hardware Port.
`[endianness]`	Sets the type of endianness to be used, depending on your system. The supported options are: • big-endian • little-endian **NOTE:** The default value is big-endian.
`<address>`	Represents the physical address of the variable used for porting. For example, `0x10001338`. **NOTE:** The address can be upto 64 bits long.
`[config_file]`	Represents the name of the configuration file used by the application. It should remain same for all the directories. If `[config_file]` name is not provided then the `config` name will be used.
`<path>`	Specifies the entry point, where the Hardware Port feature starts looking for files to read from. The value specified should be an absolute path. **NOTE:** If you are using the writing port then `<path>` cannot point to a directory!
`[non_recursive]`	Represents an option used to specify that the path provided for `Rx_port` cannot be a recursive folder.

2.1.3.1.1 Examples

This section describes some examples on how to configure a Hardware Port.

- Configuring multiple files for sequential reading
- Configuring Hardware Port for reading

2.1.3.1.1.1 Configuring multiple files for sequential reading

To setup multiple files for reading, you must provide configuration files (named config or the one specified in the config_file parameter) in each folder of interest. These configuration files must contain directories and files of interest in their current directory on each line. Starting from the specified `<path>`, the program will recursively look in all directories of interest and creates the list of files based on the `config` files.

The following listing shows an example `config` file for `<path>` folder.

Listing 2-1. Example config file for <path> folder

```
folder1
file3

folder2

folder4
```

The following listing shows an example `config` file for folder1.

Listing 2-2. Example config file for folder1

```
File1.1
File1.2
```

The following listing shows an example `config` file for folder2.

Listing 2-3. Example config file for folder2

```
File2.1
File2.2
```

According to the above listings, the order of the configuration files will be:

```
file1.1, file1.2, file3, file 2.1, file2.2
```

NOTE

The paths, folder names and file names should not contain any white spaces.

If the configuration file has an entry that is not a valid file or a folder then the loading of files into the queue will stop there.

NOTE

Make sure if the `<path>` for the `Rx_port` is a folder it should not ended with \ or /.

2.1.3.1.1.2 Configuring Hardware Port for reading

To configure the Hardware Port for reading a file, perform the following:

1. Declare a global volatile 32bit variable, for example `data_addr`, within your application.
2. Read the variable address and ensure that it is not inside a cacheable segment in the MMU.
3. Convert your Virtual Address to the Physical Address, because this is the one you'll need to pass to `-smodel` later. For example, `0x10001338`.
4. Within your application reading file section, write the assignment, for example `*your_buffer = data_addr; [etc]`. Now everytime you read from `data_addr` you'll receive the next value from the file.
5. If you need to read from a sequence of files instead of just one, you will need to add configuration files to your folder as described in Configuring multiple files for sequential reading section.
6. To start your application with runsim simply add `-smodel` to the list of commands. For example, `runsim.exe -smodel "Rx_port 0x10001338 big-endian d:/../foldername" -d devicename testname.eld`.
7. Alternatively, if you use the CodeWarrior CCSSIM2 ISS connection, you will have to edit the CCS server start-up by appending `-smodel "Rx_port 0x10001338 big-endian d:/../foldername"` to the CCS executable path.

NOTE

CCS executable must be version XX.XX and above.

2.1.3.2 Benefits

This section lists the benefits of using the Hardware Port feature.

Following list describes the benefits:

- Speed up of file input and output

Some applications heavily use file I/O function `fscanf()` and `sscanf()`. By using this simple feature the speed is exponentially improved as the feature helps read files in binary mode and converts them internally.

- No crash

  If you encountered problems using `fscanf()` and `sscanf()` many times. This feature should help you with a fix.

- You can use both reading and writing ports at the same time as long as they don't use the same file

  Your application can use one `Rx_port` and one `Tx_port` independently. However, rules for opening the same file or non-existing file do apply.

- Supports both big endian and little endian input and output.

2.1.3.3 Limitations

This section provides details on the limitations of using the Hardware Port feature.

Following is the list of limitations:

- Used only for SC platform simulators or SC core simulators.
- Not supported for all platforms precedent to 3900.
- You need to manually write the list of files that need to be read from.
- Supports only one kind of format for the files - each line needs to be 12 Bytes long, for example: `0x12345678\r\n`.

NOTE

Do not add `\r` when writing the input file unless it's in binary mode.

2.2 CCSSIM

CCSSIM enables remote access to simulators through the CCS protocol. To run the simulator on a remote computer, execute `ccssim2` on the remote machine, using the command prompt.

To start the generic CCS Remote Connection executable, start `ccssim2` from `CWInstall/starcore_support/ccs/bin`. The simulator can interact with the hardware I/O through CCSSIM.

2.2.1 Running a Simulator Remotely

To run your application on the simulator:

1. Build your source code using the StarCore Build Tools (compiler, assembler, and linker).
2. Run `ccssim2` to execute the server. The ccssim2 server is located at the path `CWInstall/starcore_support/ccs/bin`.
3. Use CodeWarrior debugger to debug the application using ccssim through remote connection.

NOTE

For more information, Refer to *Targeting StarCore DSPs* manual.

Chapter 3
Simulator Models for SC3900 Architecture

This chapter explains all the simulator functions and models that are supported by the StarCore SC3900 architecture.

The simulator models supported in this release are:

- Single Core SC3900 Simulators
- B4860 System on Chip Simulators

3.1 Single Core SC3900 Simulators

This section explains how the single core SC3900 simulators are intended for developing part of application, while running on single SC3900 core.

Following simulator models are supported in this release:

- SC3900 Single Core Platform ISS
- SC3900 Single Core Platform PACC

NOTE

SC3900 Single Core Platform ISS and PACC simulators are functional models of SC3900 FVP subsystem. For more information, refer to SC3900 FVP Subsystem Reference Manual.

Both SC3900 and SC3900fp are supported for this release. Use the following parameters to select SC3900 simulator:

- With runsim: `runsim -imodel "sc3900rev1" -d sc3900plat_pacc <eld-file>`
- With CodeWarrior: `ccssim2 -imodel "sc3900rev1"`

Run `runsim -v` option to check the number of the revision used.

Currently, the following functions and components are supported by SC3900 simulators:

- Supported Simulator Functions
- Peripherals and Components

3.1.1 Supported Simulator Functions

The following table lists the functions supported by the PACC and ISS simulators.

Table 3-1. Functions supported by SC3900 Single Core Simulators

Function	PACC		ISS	
	Planned	Current Status	Planned	Current Status
Windows	Yes	Yes	Yes	Yes
Linux32	Yes	Yes	Yes	Yes
Linux64	Yes	Yes	Yes	Yes
Cycle_Aware	Yes	Not verified; PACC accuracy was verified only for hot cache (noL1) mode (Only CAS + Address queue + SRAM contention stalls are calculated)	No	No
Cache_Support	Yes	L1 only	No	No
MMU_Support	Yes	Not fully verified	Yes	Yes
Single-Core_Processor	Yes	Yes	Yes	Yes
Multi-core_Processor	No	No	No	No
Measure_Cache Hits/ Misses	Yes	Not supported	No	No
Queue_Holds	Yes	Yes	No	No
Instruction_Cache	Yes	L1 only	No	No
Data_Cache	Yes	L1 only	No	No
Memory_Translation	Yes	Yes	Yes	Yes
Memory_Protection	Yes	Not fully verified	Yes	Yes
DTU	Yes	Not supported	Yes	Not fully verified
Epic	Yes	Not fully verified	Yes	Not fully verified
Timers	Yes	Not fully verified	Yes	Not fully verified
CME	Yes	Functional model is supported	Yes	Functional model is supported
Debug API with CodeWarrior	Yes	Yes	Yes	Yes
FSL DBG API for breakpoint	Yes	Not supported	Yes	Not fully verified

Table continues on the next page...

Table 3-1. Functions supported by SC3900 Single Core Simulators (continued)

Function	PACC		ISS	
	Planned	Current Status	Planned	Current Status
FSL DBG API for cache control	Yes	Not supported	No	No
Hardware ports	Yes	Yes	Yes	Yes
Check Points	Yes	Not supported	Yes	Not supported
TLM2 Support	Yes	Yes	Yes	Yes

3.1.2 Peripherals and Components

The following table lists aspects that you may need while using the SC3900 Platform ISS and PACC simulators.

Table 3-2. SC3900 Single Core Simulator Aspects

Aspect	Description
Core Simulators	Following core simulators are supported: • ISS - Supports SC3900 functional accuracy and is used for SC3900 platform ISS device. • CAS - Provides clock accuracy on VLES and bus level. This simulator is functional and clock accurate. **NOTE:** CAS does not provide accuracy for entering to or exit from the debug mode.
CME	The CME model for this release is functional one. There are no debugging features. The current CME features are: • Memory mapped registers (read only with the below exceptions, if not supported it will return 0 on read instruction). **External CME Programming** - External CME programming is done using four registers: CME_CA CME_CC CME_CS (read/write) CME_CR (read) **External Query Channel** - External query programming is done using two registers: CME_QCC (contains query's controls) CME_QCA (contains query address) The query result is sampled in the CME_QU1 and CME_QU2 registers.

Table continues on the next page...

Table 3-2. SC3900 Single Core Simulator Aspects (continued)

Aspect	Description
	Debug Channel - Debug programming is done using two registers: CME_DCC (contains debug instruction controls) CME_DCA (contains debug address) The query result is sampled in the CME_DQU1 and CME_DQU2 registers. **CME_CTR register** • Block cache instructions are supported • DQUERY and PQUERY instructions are supported • Suspend/resume functionality is supported • In case of MMU error the CME is suspended. All channels can also be reset through CME_CTR and can reprogram CME again.
MMU	Full featured and verified with SC3900 subsystem tests.
EPIC	Full featured and partially verified.
Timer	Full featured and partially verified.
DTU	DTU is partially featured and partially verified. Currently, DTU is connected only with SC3900 Platform ISS simulator. The current DTU limitations are: • No debug exceptions. • No dynamic partition allocation. • No PC/Addr at stage S registers. • No Exp detector. • Only RZ0 counter is supported. • Always using the triad A configuration for both triads.
Connection with L2 caches (KIBO)	L2 cache is not provided with this release. The KIBO model is being simulated by "KIBO stub" which provides the ELink-TLM connectivity to L1 cache and SGB. The KIBO stub uses a write through policy (always cache miss) and writes/reads from the system memory implemented by runsim/CW. Supported links are: DLink,ILink,RLink,Clink (Clink is never returned by the KIBO stub as a response). Only single beat and single link (only Clink or only RLink) responses are supported.
Scalable Platform PACC device	The SC3900 platform PACC simulator is scalable and supports the following performance modes: • Simulate clocks of core and address queue and can be used for optimization of the core pipeline stalls and performance analysis. • Simulate clocks of core, address queue and L1 cache. • Simulate clocks of full system, including core, address queue, L1, L2, L3, DDR components.
Hardware breakpoint	Provides support for the following hardware breakpoints: • Instruction address breakpoint • Instruction address range breakpoint • Data address range breakpoint

3.2 B4860 System on Chip Simulators

B4860 System on Chip (SoC) ISS is a B4860 functional simulator. This simulator is intended for development of full chip applications.

For more information, refer *PSC9164 SOC Architecture Specification*.

NOTE

B4860 SoC ISS simulator is now supported on both Linux 64 bit and Windows OS.

Following features and components are supported by B4860 SoC ISS:

- Supported Simulator Functions
- Peripherals and Components
- B4860 ISS MAPLE B3 Support
- Configuring B4860 Simulator
- Running U-Boot and Linux on B4860 Simulator

3.2.1 Supported Simulator Functions

This section lists the functions supported by the B4860 instruction set simulator.

The below table provides the list of functions that are currently supported by B4860 ISS.

Table 3-3. Functions supported by B4860 Instruction Set Simulator

Function	B4860 ISS	
	Planned	Current Status
Windows	Yes	Yes
Linux32	Yes	Not Supported
Linux64	Yes	Yes
SC3900 Platform ISS	Yes	Yes
MPIC	Yes	Yes
MAPLE3	Yes	Yes
Virtual Interrupt (VI)	Yes	Yes
e6500 cores	No	Yes
CCM	Yes	Yes
CPC	Yes	Yes
BMAN	Yes	Yes
QMAN	Yes	Yes

Table continues on the next page...

Table 3-3. Functions supported by B4860 Instruction Set Simulator (continued)

Function	B4860 ISS	
	Planned	Current Status
FMAN	Yes	Yes
CAAM	Yes	Yes
I2C	Yes	Yes
PBL	Yes	Yes
Connection with CodeWarrior for StarCore and Power Architecture®	Yes	Yes
Traffic IO Support	Yes	Yes
Hardware memory mapped ports for fast IO with files	Yes	Yes
Check Points	Yes	Not Supported

3.2.2 Peripherals and Components

This section lists the aspects that you may need while using the B4860 instruction set simulator

The below table lists the aspects and their description.

Table 3-4. B4860 Instruction Set Simulator Aspects

Aspect	Description
e6500 Cores	• Functional Model only, No timing. • Debug Registers/Interrupts are not supported. • Performanced Management Registers/Interrupts/mtpmr/mfpmr are not supported. • Timers are approximate since functional model does not model time. • DCache is not modeled. DCBZ/DCBA clear memory. Rest of DCache instructions are nopped. • ICache is not modeled. ICache instructions are nopped. • Altivec Assist interrupt is not modeled. • Altivec mvidsplt instruction is not modeled. • Hardware Table Walk on ITLB misses is minimally testing. • Hardware Table Walk on Cache misses is not supported. • Minimal testing of LRAT and Hardware Table Walk. • SCCSBAR register does not mirror shifted CCSBAR.
CCM	• Secure boot not supported • Stashing not supported
CPC	• Write back cache and I/O stash are not supported. • ATQ priority arbitration (CSSA) is not supported.

Table continues on the next page...

Table 3-4. B4860 Instruction Set Simulator Aspects (continued)

Aspect	Description
	• Interrupts are not supported. • Decoration storage through ReadU, WriteU and Notify is not supported.
BMAN	No error interrupts.
QMAN	• CEETM is not supported. • CS tail drop rejections (part of congestion management) are not modeled. • Avoid Blocking FQD control bit is not supported. • Query WQ length command is not supported. • Query Congestion State and CGR Test Write commands are not supported. • Congestion state change notification (functional) interrupts are not supported. • Error interrupts are not modeled. • Debug interface is not modeled. • Performance monitor interface is not supported. • Dynamic debug interface is not supported. • QCSP0-9_EQCR_CI_CENA/CINH: PB field is not supported. • QMAN_HID_CFG: Only SRPP, SECO, IRPA, DFTD, and SFTD fields are supported.
MPIC	• Shared Message Signaling Interrupt is not supported. • Timers are not supported. • No behavioral difference between edge and level sensitive interrupts. Level behavior is not modeled.
I2C	• I2C device as Master Mode is supported, Slave mode is NOT supported. • Only Register accesses and EEPROM R/W supported
IFC	ECC encoding and decoding in NAND-FCM mode are not supported. Buffer control (BCTL) enable/disable support in NAND-FCM, NOR-FCM and GPCM mode are not supported. Address shift mode in NOR-FCM is not supported. Burst mode for multi-beat reads in NOR-FCM mode is not supported. Parity checking in GPCM mode is not supported. Address Data multiplexing shift in GPCM mode is not supported. Ready-Busy status indication for each bank is not supported. NAND Flash timeout error, ECC error, and interrupt generation are not supported. NAND Flash Page read completion status are not supported. NOR Flash command sequence timeout error and interrupt generation are not supported. GPCM timeout error and interrupt generation are not supported. Loop (Multi-Page) feature in NAND-FCM is not supported. IOBIST modes are not supported. Software reset functionality are not supported.
PBL	• PBL assumes that only IFC-NOR interface is used as RCW source, PBL source and as Boot location. • PBL Interface with IFC-NAND is not supported. • PBL Interface with I2C is not supported. • PBL Interface with SPI is not supported. • PBL Interface with SDHC is not supported. • PBL Hard coded RCWs is not supported.

Table continues on the next page...

Table 3-4. B4860 Instruction Set Simulator Aspects (continued)

Aspect	Description
Frame Manager	Frame Manager SWSIM is a software simulators package which currently includes a simulation module for P4080/ P3040/P5020/P1023/P2040 Frame Manager hardware block. For more information, refer *Frame Manager Software Simulator Release Notes*.
Virtual Interrupt	Virtual Interrupts are implemented, but not really tested with sc3900 interrupts: It gets interrupts from the program and transmits them to the 6 sc3900 cores.
MAPLE3 Support	Refer the B4860 ISS MAPLE B3 Support section for details.

3.2.3 B4860 ISS MAPLE B3 Support

This section lists the supported Maple Processing Elements or Maple components.

NOTE

By default the MAPLE mode is REV2. For REV1 support, add the following flag:

```
-imodel "-MAPLE3_REV1"
```

The base addresses of MAPLE is fixed as:

- MAPLE extended to 36 bit addressing and `0xF` prefix could be added in MSB 32 bits.
- MBus could be changed in `cfg` file in the `ccm.laws` field.
- SBus could be changed by CCSR base value change.
  a. MBus0 `0xFA0000000` (default `0xE0000000`)
  b. MBus1 `0xFA0400000` (default `0xE0400000`)
  c. MBus2 `0xFA0800000` (default `0xE0800000`)
  d. SBus0 `0xFfe800000` (default `0xFE800000`)
  e. SBus1 `0xFfe810000` (default `0x FE810000`)
  f. SBus2 `0xFfe820000` (default `0x FE820000`)

Following are the Maple Processing Elements or Maple components:

- Memory Map
- PSIF3 Registers
- eTVPE2
- eFTPE2
- DEPE2
- EQPE

- EQPE2
- PDPE2
- PDPE2Rev2
- PUPE2
- PUPE2Rev2
- DL2
- ULF2
- ULB2
- CGPE Registers
- TCPE Registers
- Buffer Descriptor (BD) Programming Model
- Flexible Interrupt Scheme
- CRPE

3.2.3.1 Memory Map

The memory model of Maple is represented as:

- Three MAPLE 3 elements are implemented including two LW and one W.
- SBus memory is implemented at address `0x[F]fe800000` and can be configured by CCSRBAR.
- MBus memory is implemented at address `0x[F]fa000000` including:
    - DRAM
    - eTVPE2 registers
    - DEPE2 registers
    - EQPE2 registers
    - PDPE2 registers
    - PUPE2 registers
    - eFTPE2 registers (both in L&W) (LTE WCDMA)

In MAPLE3 following registers are implemented in memory map:

- DL2 registers
- ULF2 registers
- ULB2 registers
- CGPE registers (W)
- TCPE registers (W)

3.2.3.2 PSIF3 Registers

Following features are implemented in PSIF3 registers:

- Four RISC cores are simulated, including registers on the SBus memory
- Internal DMA is partially simulated including MMU
- DTU is simulated
- PIC is simulated
- Scheduler is simulated

3.2.3.3 eTVPE2

Following features are implemented in eTVPE2 registers:

- Register's memory map is implemented with reset values
- Pipeline mode is not supported
- Passes the basic tests for 3GLTE and UMTS
- Some modes do not support. eTVPE2::UMTS::EDCH separated vector (Not implemented in model)
- Qualifies the SmartDSP OS eFVPE2 test and some tests vectors
- Qualifies SmartDSP OS Single core tests
- Full SmartDSP OS test on Multicore is not supported

NOTE

MAPLE3 support is not yet fully tested with SmartDSP OS multi-core support. Some of the tests may pass on reduced number of iterations.

3.2.3.4 eFTPE2

Following features are implemented for eFTPE2:

- Register's memory map is implemented with reset values
- Qualifies the SmartDSP OS eFTPE2 test and some test vectors

3.2.3.5 DEPE2

Following features are implemented for DEPE2:

- Register's memory map is implemented with reset values
- Failed on full SmartDSP OS test (iterations)
- Qualifies the SmartDSP OS DEPE2 test and some tests vectors

NOTE

MAPLE3 support is not yet fully tested with SmartDSP OS multi-core support. Some of the tests may pass on reduced number of iterations.

3.2.3.6 EQPE

Following features are implemented for EQPE:

- Register's memory map is implemented with reset values
- Testing is in progress

3.2.3.7 EQPE2

Following features are implemented for EQPE2:

- Register's memory map is implemented with reset values
- Testing is in progress
- Qualifies basic tests

3.2.3.8 PDPE2

Following are the new features implemented for PDPE2:

- Register's memory map is implemented with reset values
- Qualifies the SmartDSP OS PDPE2 test and some tests vectors

3.2.3.9 PDPE2Rev2

Following are the new features implemented for PDPE2Rev2:

- Register's memory map is implemented with reset values
- Qualifies basic tests

3.2.3.10 PUPE2

Following are the new features implemented for PUPE2:

- Register's memory map is implemented with reset values
- Testing is in progress

3.2.3.11 PUPE2Rev2

Following are the new features implemented for PUPE2Rev2:

- Register's memory map is implemented with reset values
- Testing is in progress
- Qualifies basic tests

3.2.3.12 DL2

Following are the new features implemented for DL2:

- Register's memory map is implemented with reset values
- Testing is in progress

3.2.3.13 ULF2

Following features are implemented for ULF2:

- Register's memory map is implemented with reset values
- Testing is in progress

3.2.3.14 ULB2

Following features are implemented for ULB2:

- Register's memory map is implemented with reset values
- Testing is in progress

3.2.3.15 CGPE Registers

Following features are implemented for CGPE:

- Register's memory map is implemented with reset values
- Testing is in progress

3.2.3.16 TCPE Registers

Following features are implemented for TCPE registers:

- Register's memory map is implemented with reset values
- Testing is in progress

3.2.3.17 Buffer Descriptor (BD) Programming Model

Following feature is implemented for BD Programming Model:

- For multiple master support, up to 8 High Priority BD rings and 8 Low Priority BD rings are there for each processing element

3.2.3.18 Flexible Interrupt Scheme

Following feature is implemented for Flexible interrupt scheme:

- Supports BD interrupts mapped from IRQ_242 and above

3.2.3.19 CRPE

Following features are implemented for CRPE:

- Currently only the CPRI-CRPE interactions are supported, full CPRI data path does not get simulated.
- Partially registers are implemented in order to perform WCDMA demo
- Only CPRI 0 registers are supported at the moment (although there are six CPRI modules)
- Qualifies SmartDSP OS Single core tests
- The CRPE put and get data to and from files in predefined format:
  - For UpLink up to 24 antenna files

    The data structure for ant 0 file for OVS 2 is:

    ```
    chip 0 phaze 0 <8I,8Q>

    chip 0 phaze 1 <8I,8Q>

    chip 1 phaze 0 <8I,8Q>

    chip 1 phaze 1 <8I,8Q>
    ```

    The data structure for ant 0 file for no OVS is:

    ```
    chip 0 phaze 0 <8I,8Q>

    chip 1 phaze 0 <8I,8Q>
    ```

  - For DownLink only in CPRI mode in CRPE-DL, upto16 antenna output files (or less), <16I,16Q>

    The data structure is:

    ```
    chip 0 <16I,16Q>

    chip 1 <16I,16Q>
    ```

NOTE

DL writes out to file every 2083 system clocks.

UL reads from file every 2083*16 system clocks.

CPRI Timer Interrupts feature is still not implemented where each 10ms CPRI should produce interrupts.

3.2.4 Configuring B4860 Simulator

This section discuss the configuration options supported by B4860 ISS.

B4860 instruction set simulator supports the following initialization and configuration files:

- `b4860iss.ini` - Required for StarCore components initialization. This file is loaded at startup.
- `b4860iss_sim_init_params.cfg` - Provides default or configurable simulator initialization parameters. This file is loaded at startup.

Following sections discuss various configuration options supported by B4860 ISS:

- Configure B4860 Simulator Options
- Configuration Files

3.2.4.1 Configure B4860 Simulator Options

This section provides details on configuring B4860 simulator options. It also has a table that lists the various options available.

The simulator can run with the following configuration options:

- `./runsim -imodel "option=value -option... " -smodel "option=value -option... " -d b4860iss -nc 1 test.eld`
- `./ccssim2 -imodel "option=value -option... " -smodel "option=value -option... " -port 41475`

where,

- `-imodel` - Required during the StarCore simulator initialization (Initialization Model).
- `-smodel` - Required after the initialization of the simulator, and before starting the execution of the simulator (Start Model).

NOTE

`Runsim` and `ccssim2` can run without -`imodel` and -`smodel` options. In this case the default values are used `./runsim -d b4860iss -nc 1 test.eld` and `./ccssim2 -port 41475`.

The below tables lists the various options available:

Table 3-5. Main Options

Name	imodel	smodel	Description
`sc_pa_maple_run=n1:n2:n3`	Yes	Yes	Enables the execution phase at the startup of StarCore, PowerArchitecture, and Maple cores. Default value is: 0:0:1.
`sc_pa_maple_ratios=n1:n2:n3`	Yes	Yes	Sets the execution phase cycle ratios for StarCore, PowerArchitecture, and Maple. This parameter can be used to configure the speed/ synchronization/execution tunings of the simulator. Zero means no execution cycles for the corresponding component. Default value is: 20:1:10.
`sc_pa_maple_sync_exec=n1:n2:n3`	Yes	Yes	Enables or disables synchronization of MAPLE execution with StarCore and PA cores execution. Currently only the StarCore cores are taken into account and therefore, MAPLE doesn't get cycles when all StarCore cores are stopped. Default value is: 1:0:1
`sim_tio_hub=tio_server_name:tio_server_port`	Yes	No	Enables the TIO support in B4860 simulator and configures TIO server name and port. The value is set in `b4860iss_sim_init_params.cfg` file.
`sim_pa_run_vcores=<enable_mask>`	Yes	No	Enables/disables (1/0) run at reset for a PA virtual cores (PA core thread0 or thread1). 10101010 enables run at reset for all main threads (thread0 of all PA cores). Default value is: 10000000. It enables run at reset for the 1'st virtual core (PA core0 / thread0).
`- MULTITHREAD`	Yes	No	Enables multithreading: 1 thread for StarCore, 1 thread for PA cores, and 1 thread for each MAPLE3 (A, B, C). Default value is: • Enabled, if the host PC is a multicore machine • Disabled, if the host PC is a singlecore machine

Table continues on the next page...

Table 3-5. Main Options (continued)

Name	imodel	smodel	Description
`- NOMULTITHREAD`	Yes	No	Disables multithreading. Default value is: • Disabled, if the host PC is a multicore machine • Enabled, if the host PC is a singlecore machine

Table 3-6. Advanced Options

Name	imodel	smodel	Description
`sim_log_level=<level>`	Yes	Yes	Sets the specified log level. Supported log levels are NONE=0, ERROR=1, WARNING=2, INFO=3, DEBUG=4, TRACE=5. The specified level must be a positive integer (Level >=0). Default value is: 1
`sim_log_file=<sim.log>`	Yes	Yes	Writes log to the specified log file.
`ipmodels_log_level=<level>`	Yes	Yes	Sets log level for `ipmodels` components. The specified level must be a positive integer (Level >=0). Default Value is: 0
`pa_sim_config_file=<test.cfg>`	Yes	Yes	Sets the configuration file loaded at startup. It can be used to load binary images, and write memory mapped registers.
`mapletrace_file <maple.trc>`	No	Yes	Specifies the maple trace file.
`mapletrace start/stop`	No	Yes	Starts/Stops maple ucode trace.
`- cpricfg <cpri_config.xml>`	Yes	No	Loads the specified CPRI configuration file.

3.2.4.2 Configuration Files

This section provides details on the various initialization parameters along with certain examples.

The following table lists the various initialization parameters that can be changed in the configuration file `b4860iss_sim_init_params.cfg`.

Table 3-7. Initialization Parameters

Name	Description
`sim.enable_fman_tio`	Enables FMAN TIO support. When this option is enabled you could use `fm_tio_inject` and `fm_tio_capture` applications to inject pcap files. Default value is: False
`sim.jit_run_quanta`	Sets the maximum number of instructions to be executed on the execution phase for PA cores. Default value is: 1000
`sim.clock_dpa`	Enables DPAA clock. Default value is: False
`sim.dpa_clock_divisor`	Configures the DPAA clock ratio. The value of `dpa_clock` is `sim.jit_run_quanta / sim.dpa_clock_divisor`. Default value is: 100
`fman0.log_filename`	Set log filename for FMAN0. Default value is: `fman.log`
`fman0.log_level`	Set log level for FMAN0. Default value is: 0
`sim.dpa_bman_log_levelsim.dpa_qman_log_levels im.dpa_caam_log_levelsim.dpa_fman0_log_level`	Sets log level for BMAN, QMAN, CAAM, FMAN wrapper. Default value is: 0 (for all)
`configunit.model_p5020`	Sets `configunit.model_p5020=1` for old images/use-cases (p5020) Default value is: 0
`ccm.laws`	Configures custom LAW entries. The values are set in the `b4860iss_sim_init_params.cfg` file.

3.2.4.2.1 Examples

Run the following commands, if you want to enable the execution only on StarCore:

```
./runsim -smodel "sc_pa_maple_run=0:0:0 sc_pa_maple_ratios=1:0:0" -d
b4860iss -nc 1 test.eld
```

```
./ccssim2 -smodel "sc_pa_maple_run=0:0:0 sc_pa_maple_ratios=1:0:0" -
port 41475
```

Run the following commands if you want to enable the execution only on StarCore and Maple:

```
./runsim -smodel "sc_pa_maple_run=0:0:1 sc_pa_maple_ratios=20:0:10" -d
b4860iss -nc 1 test.eld
```

```
./ccssim2 -smodel "sc_pa_maple_run=0:0:1 sc_pa_maple_ratios=20:0:10" -
port 41475
```

Run the following commands if you want to enable TIO support:

```
./runsim -imodel "sim_tio_hub=10.1.171.5:42476" -d b4860iss -nc 1
test.eld
```

```
./ccssim2 -imodel "sim_tio_hub=10.1.171.5:42476" -port 41475
```

Run the following commands if you want to change CCSR value:

```
./runsim -imodel "ccm.ccsrbarh=0x0" -d b4860iss -nc 1 test.eld
```

```
./ccssim2 -imodel "ccm.ccsrbarh=0xf ccm.ccsrbarl=0xfe000000" -port
41475
```

Run the following command if you want to change RCW (recommended for SmartDSP OS)

```
./ccssim2 -imodel "ccm.ccsrbarh=0xf" -smodel "pa_sim_config_file=./
rcw/test_rcw_sdos.cfg"
```

Run the following command if you want to run MAPLE3 in REV1 mode (since the default mode is REV2)

```
./ccssim2 -imodel "-MAPLE3_REV1"
```

3.2.5 Running U-Boot and Linux on B4860 Simulator

Follow the steps given in this section to run U-Boot and Linux on B4860 simulator.

For booting Linux with `ccssim2`, `ccs` and `tio_console` or with `runsim` and `tio_console`, refer to the `readme.txt`.

To access the `readme.txt` file, follow the steps given below:

1. Extract the archive files available at <`CWInstallDir>\SC\ccs\bin\linux64\sc_swsim_linux64.tgz`, where `<CWInstallDir>` is the path to your CodeWarrior installation.
2. Select `linux64\linux_ir0\readme.txt`.

   The file contains all the steps required to run U-Boot and Linux on B4860 Simulator.

Chapter 4
Using Traffic IO

This chapter explains the usage of Traffic IO using the net_demo SmartDSP OS utility project.

This chapter explains:

- Import net_demo Utility Project
- Configure Windows TCP/IP settings
- Execute net_demo Utility Project

4.1 Import net_demo Utility Project

This section lists the steps required to import the net_demo utility project.

NOTE

The procedure of importing the net_demo utility project assumes that you have the Windows Packet Capture Library (WinPcap) installed. For more details on WinPcap, visit http://www.winpcap.org/install/default.htm.

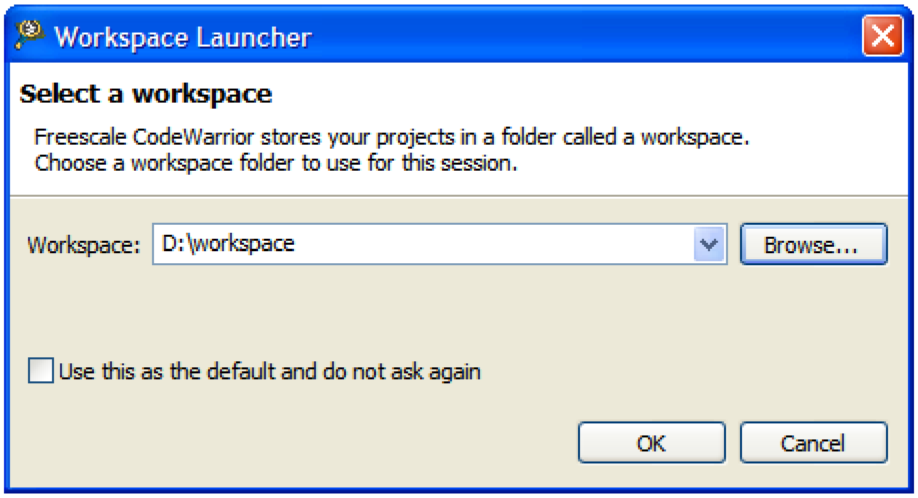
To import the **net_demo** utility project, perform these steps:

1. Select **Start** > **Programs** > **Freescale CodeWarrior** > **CodeWarrior for StarCore**< *number*> > **CodeWarrior**, where number is the version number of your product.

   The IDE launches and the WorkSpace Launcher dialog appears, asking you to select a workspace to use.

NOTE

You can select the **Use this as the default and do not ask again** checkbox to set default/selected path as a default location for storing all your projects.


Workspace Launcher

Select a workspace

Freescale CodeWarrior stores your projects in a folder called a workspace.
Choose a workspace folder to use for this session.

Workspace: D:\workspace

Figure 4-1. Workspace Launcher Dialog Box

2. Click **OK** to accept the default workspace. To use a different workspace, click **Browse** and specify the desired location.

   The IDE initializes and displays the **Workbench** window.

   NOTE

   The **Welcome** page is displayed when CodeWarrior is run for the first time. You can always return to this page by selecting **Help > Welcome** from the CodeWarrior IDE menu bar.

3. From the CodeWarrior IDE menu bar, select **File > Import**.

   The **Import** wizard appears.

4. Expand the **General** tree item.
5. Select **Existing Projects into Workspace** to import an existing SmartDSP OS demo application as shown below.

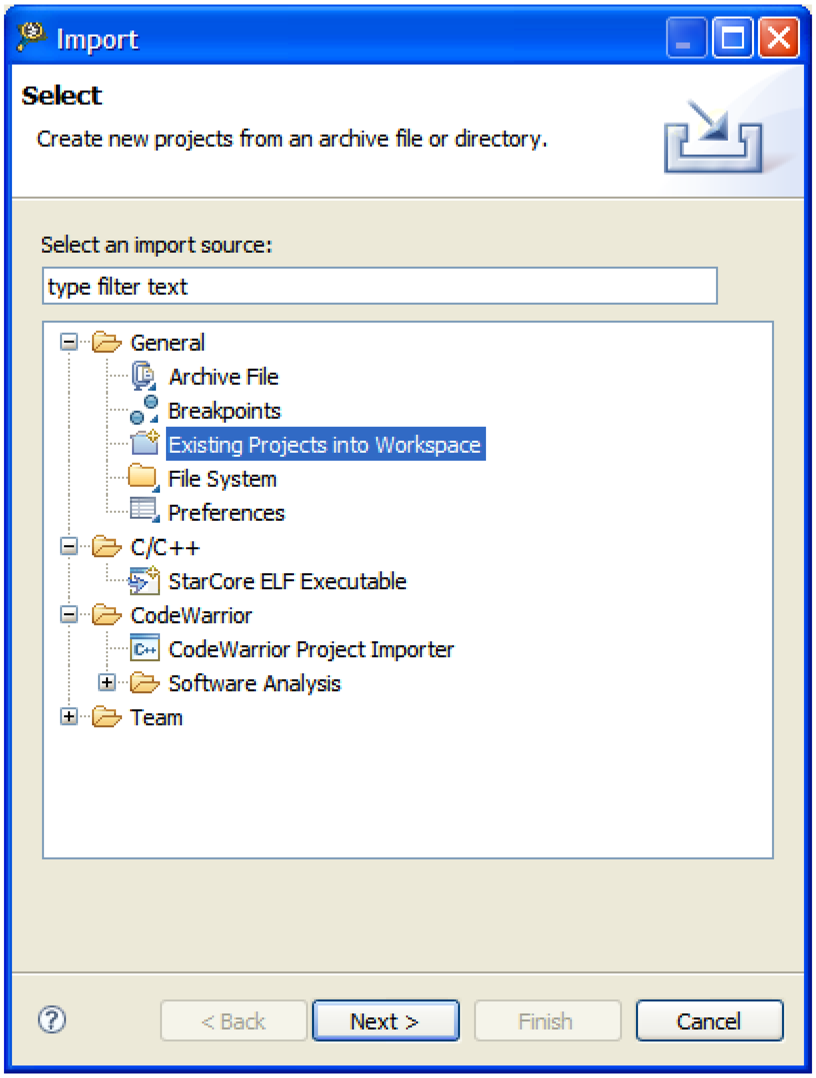


Figure 4-2. Import Wizard - Select Existing Projects into Workspace

6. Click **Next**.

   The **Import Projects** page appears.

7. Select the **Select root directory** option.

   The wizard enables the corresponding **Browse** button.

8. Click **Browse**.

   The **Browse For Folder** dialog box appears.

9. Use the dialog box to navigate to the **net_demo** utility project.

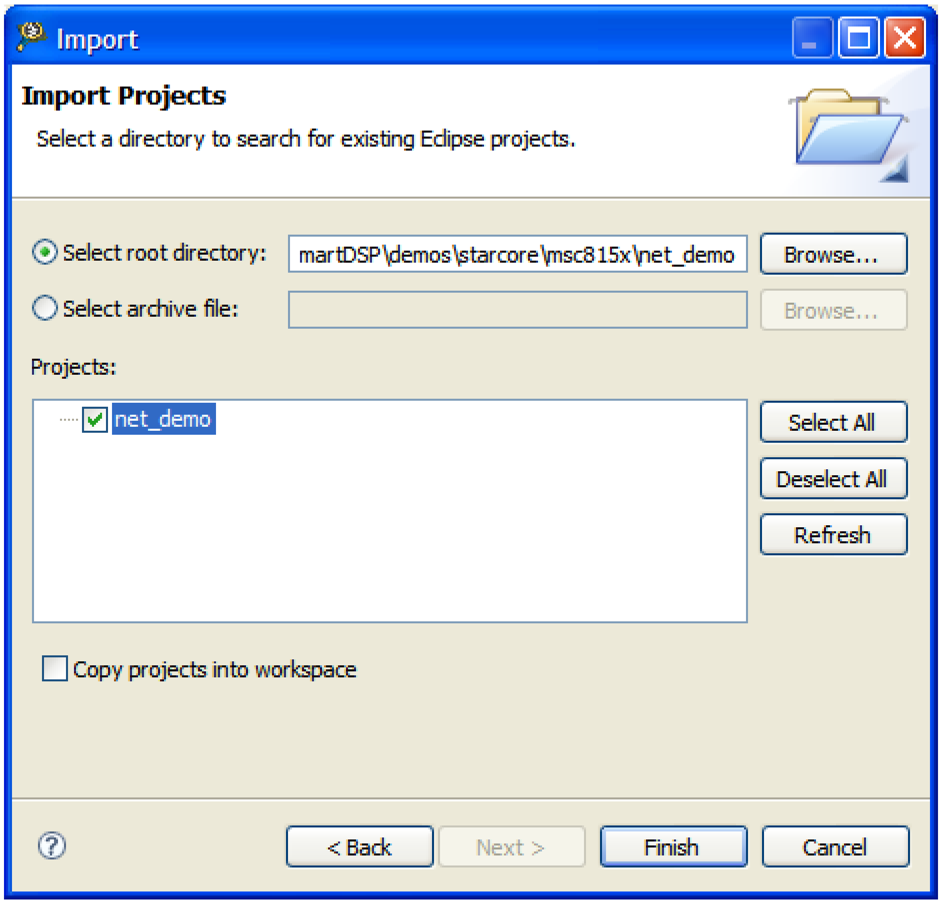
NOTE

The **net_demo** utility project is installed as a part of CodeWarrior installation and is available in the `<CWInstallDir>\SC\StarCore_Support \SmartDSP\demos\starcore`

`\msc815x\net_demo` folder, where`lt;CWInstallDir>` is the path to your CodeWarrior installation.

10. Click **OK**.

    The **Browse For Folder** dialog box closes. The path to the demo project appears in the **Select root directory** text box as shown below.



Import Projects

Select a directory to search for existing Eclipse projects.

Select root directory: martDSP\demos\starcore\msc815x\net_demo

Select archive file:

Projects:

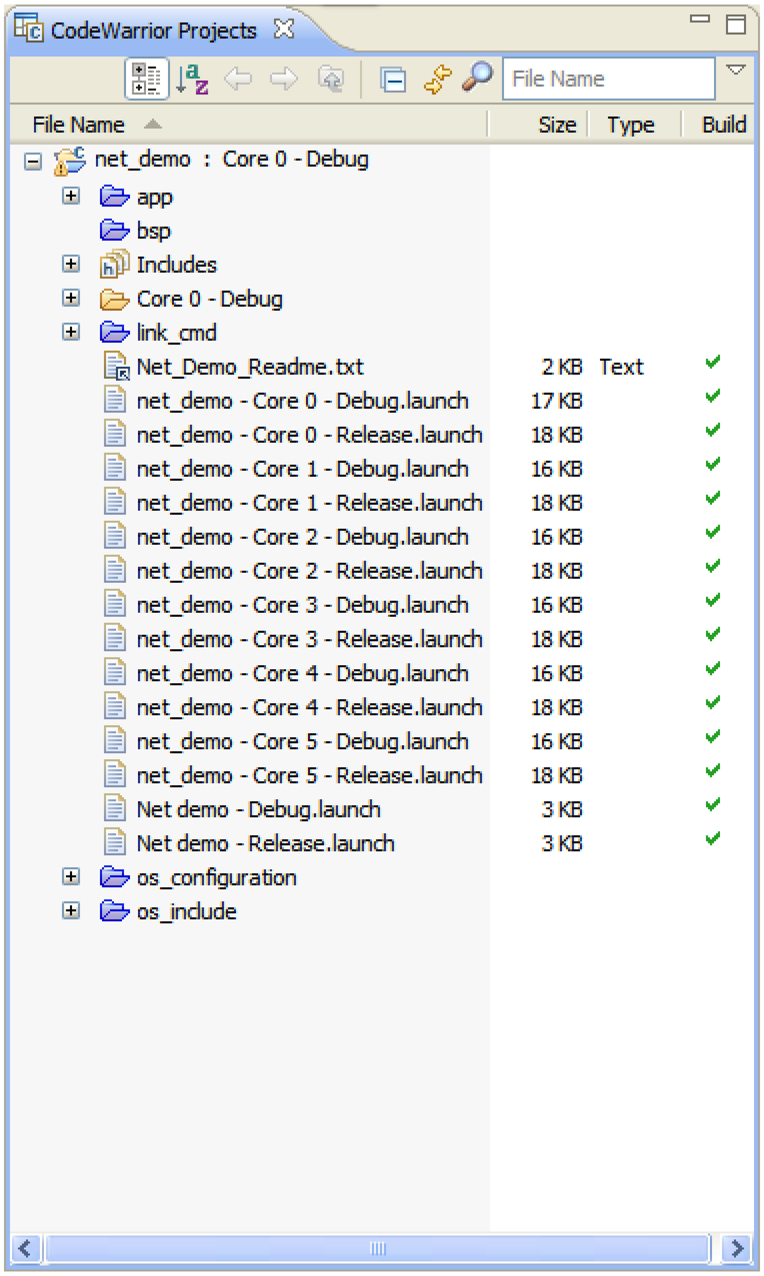
✓ net_demo

☐ Copy projects into workspace

Figure 4-3. Import Wizard - Import Existing Projects

11. Ensure that the **net_demo** utility project displayed in the **Projects** text box is checked.
12. Click **Finish**.

    The Import wizard closes and **C/C++** perspective appears. The CodeWarrior Projects view shows the **net_demo** utility project as displayed in the below figure.



File Name	Size	Type	Build
net_demo : Core 0 - Debug			
app			
bsp			
Includes			
Core 0 - Debug			
link_cmd			
Net_Demo_Readme.txt	2 KB	Text	✓
net_demo - Core 0 - Debug.launch	17 KB		✓
net_demo - Core 0 - Release.launch	18 KB		✓
net_demo - Core 1 - Debug.launch	16 KB		✓
net_demo - Core 1 - Release.launch	18 KB		✓
net_demo - Core 2 - Debug.launch	16 KB		✓
net_demo - Core 2 - Release.launch	18 KB		✓
net_demo - Core 3 - Debug.launch	16 KB		✓
net_demo - Core 3 - Release.launch	18 KB		✓
net_demo - Core 4 - Debug.launch	16 KB		✓
net_demo - Core 4 - Release.launch	18 KB		✓
net_demo - Core 5 - Debug.launch	16 KB		✓
net_demo - Core 5 - Release.launch	18 KB		✓
Net demo - Debug.launch	3 KB		✓
Net demo - Release.launch	3 KB		✓
os_configuration			
os_include			

Figure 4-4. CodeWarrior Projects-net_demo SmartDSP OS Utility Project

13. Select **Project > Build Project** from the CodeWarrior IDE menu bar to build the project.

4.2 Configure Windows TCP/IP settings

This section lists the steps required to configure TCP/IP settings.

To configure TCP/IP, follow these steps:

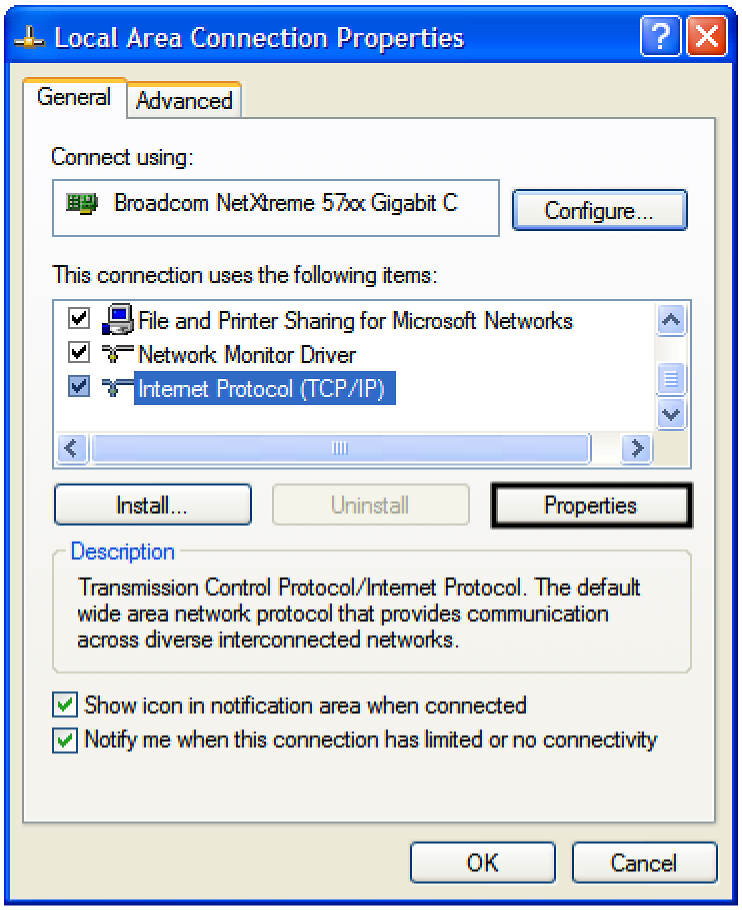
1. Select **Start > Control Panel**.

   The Control Panel window appears.

2. Select **Network and Internet Connections**.

   The **Network and Internet Connections** window appears.

3. Select **Network Connections**.
4. Right-click the network connection that you want to configure, select **Properties** from the context menu that appears.
5. The Local Area Connection Properties dialog box appears as shown below.



Local Area Connection Properties
General
Advanced
Connect using:
Broadcom NetXtreme 57xx Gigabit C
Configure...
This connection uses the following items:
File and Printer Sharing for Microsoft Networks
Network Monitor Driver
Internet Protocol (TCP/IP)
Install...
Uninstall
Properties
Description
Show icon in notification area when connected
Notify me when this connection has limited or no connectivity
OK
Cancel

Transmission Control Protocol/Internet Protocol. The default wide area network protocol that provides communication across diverse interconnected networks.

Figure 4-5. Local Area Connection Properties Dialog Box

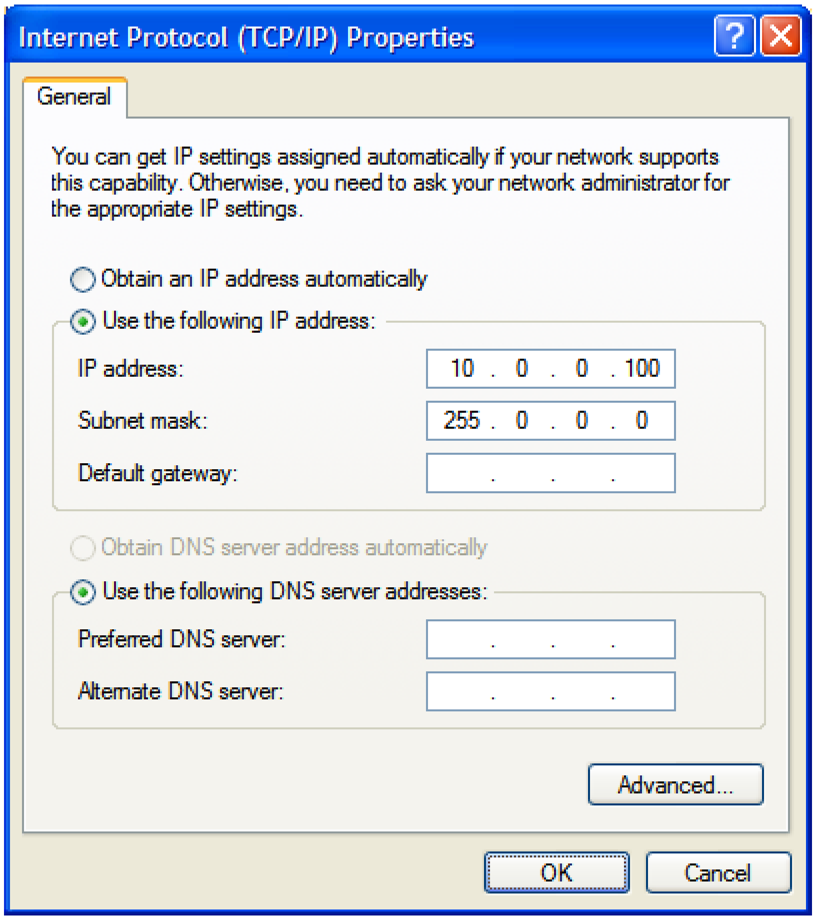
6. Select **Internet Protocol (TCP/IP)**, and then click **Properties**.

   The **Internet Protocol (TCP/IP) Properties** dialog box appears.

7. Select the **General tab**.
8. Select **Use the following IP address** option.
9. Specify `10.0.0.100` as IP address and `255.0.0.0` as Subnet mask.

NOTE

Do not specify any value in the **Preferred DNS server** and **Alternate DNS server** text boxes.



You can get IP settings assigned automatically if your network supports this capability. Otherwise, you need to ask your network administrator for the appropriate IP settings.

Obtain an IP address automatically [✓ Use the following IP address:]

IP address: 10 . 0 . 0 . 100

Subnet mask: 255 . 0 . 0 . 0

Default gateway: . . .

Obtain DNS server address automatically [✓ Use the following DNS server addresses:]

Preferred DNS server: . . .

Alternate DNS server: . . .

Figure 4-6. Internet Protocol (TCP/IP) Properties Dialog Box

10. Click **OK** to close the **Internet Properties (TCP/IP) Properties** dialog box.
11. Click **Close** to save and close the **Local Area Connection Properties** dialog box.

NOTE

This procedure will result in loss of Internet connectivity. To re-connect select the **Obtain an IP address automatically** option from the **Internet Protocol (TCP/IP) Properties** dialog box.

4.3 Execute net_demo Utility Project

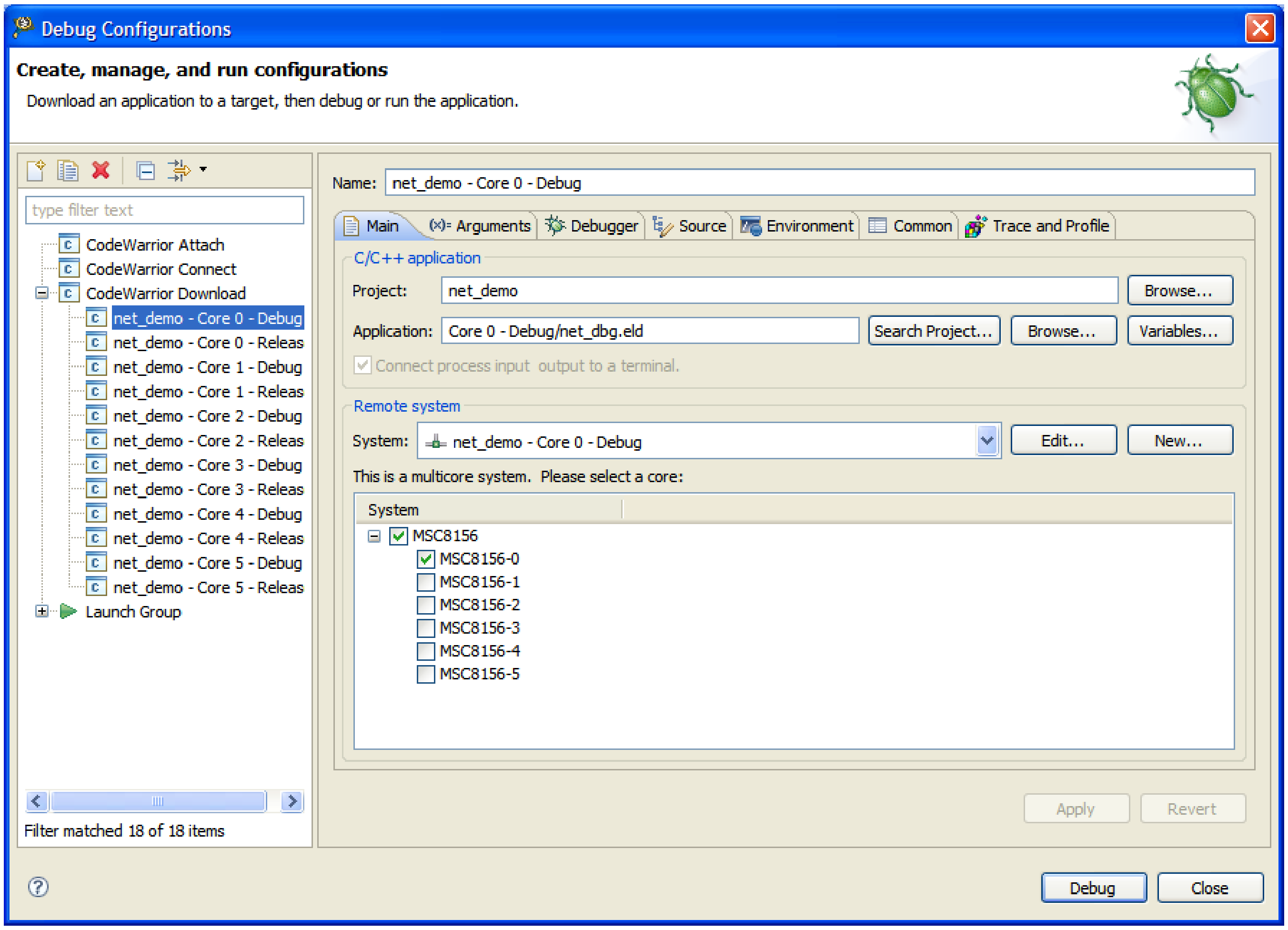
This section lists the steps required to execute the net_demo utility project.

To execute the **net_demo** utility project, perform the following steps:

1. Select the **net_demo** utility project in the **CodeWarrior Projects** view.
2. Select **Run > Debug Configurations**.

   The **Debug Configurations** dialog box appears as shown in the below figure.

3. Expand the **CodeWarrior Download** configuration.
4. From the expanded list, select the debug configuration that you want to modify. For example, select core 0.


Debug Configurations
Create, manage, and run configurations
Download an application to a target, then debug or run the application.
type filter text
CodeWarrior Attach
CodeWarrior Connect
CodeWarrior Download
net_demo - Core 0 - Debug
net_demo - Core 0 - Releas
net_demo - Core 1 - Debug
net_demo - Core 1 - Releas
net_demo - Core 2 - Debug
net_demo - Core 2 - Releas
net_demo - Core 3 - Debug
net_demo - Core 3 - Releas
net_demo - Core 4 - Debug
net_demo - Core 4 - Releas
net_demo - Core 5 - Debug
net_demo - Core 5 - Releas
Launch Group
Filter matched 18 of 18 items
Main
Arguments
Debugger
Source
Environment
Common
Trace and Profile
C/C++ application
Browse...
Search Project...
Variables...
Remote system
Edit...
New...
System
Apply
Revert
Debug
Close

Name: net_demo - Core 0 - Debug

Project: net_demo

Application: Core 0 - Debug/net_dbg.eld

[✓] Connect process input output to a terminal.

System: net_demo - Core 0 - Debug

This is a multicore system. Please select a core:

- [✓] MSC8156
  - [✓] MSC8156-0
  - [ ] MSC8156-1
  - [ ] MSC8156-2
  - [ ] MSC8156-3
  - [ ] MSC8156-4
  - [ ] MSC8156-5

Figure 4-7. Debug Configurations Dialog Box

5. Select the `net_demo - Core0 - Debug` remote system from the **System** dropdown list.
6. Click **Edit**.

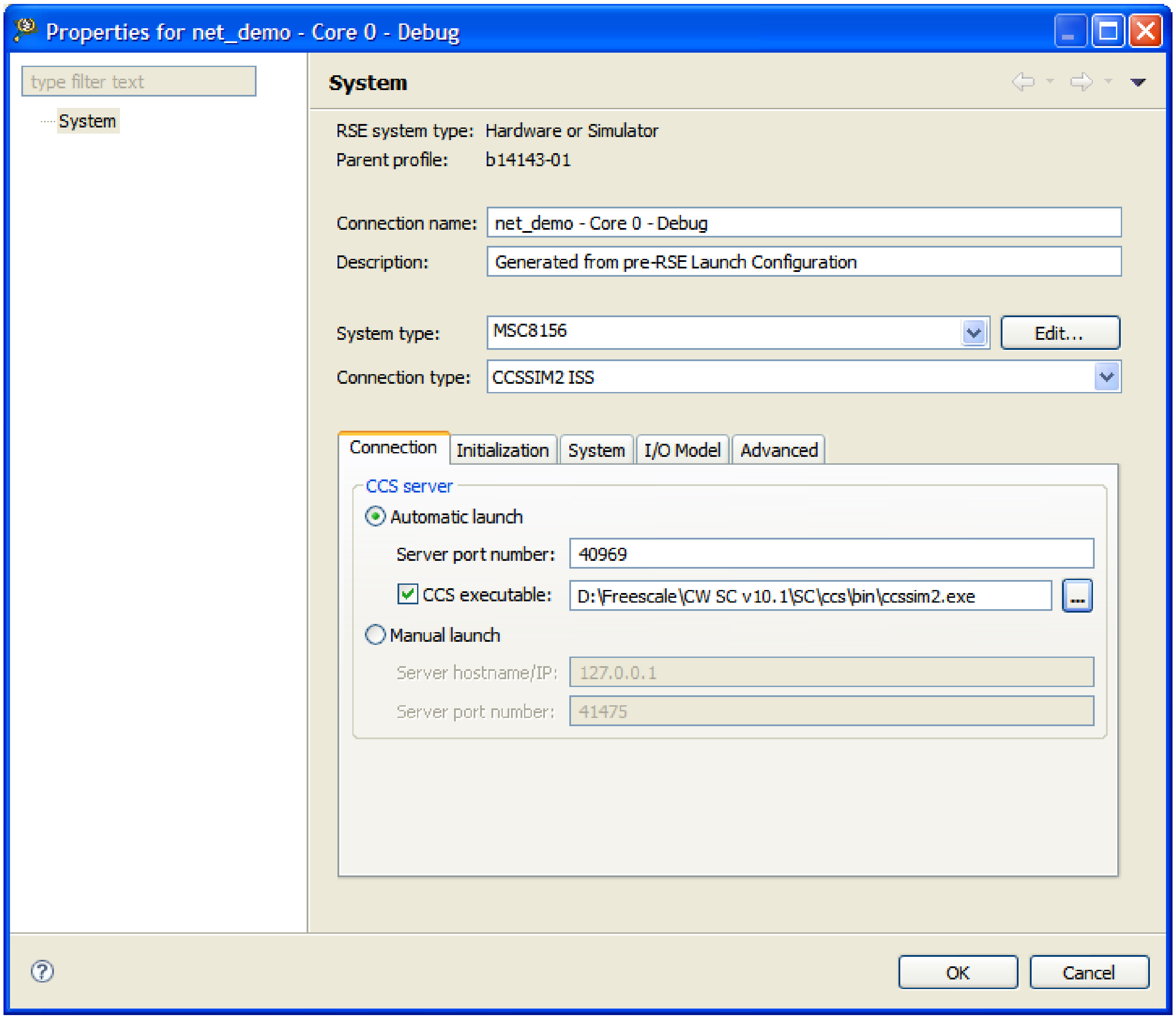
   The **Properties for net_demo** window appears as shown below.

7. Select **CSSIM2 ISS** from the **Connection type** drop down list.
8. Select the **Connection** tab.
9. Select the **Automatic launch** option.
10. Modify the **Server port number** to `40969`.

NOTE

If you use a custom port, modify the **Server port number** appropriately.

11. Check **CCS executable** to specify the server executable file path.



RSE system type: Hardware or Simulator
Parent profile: b14143-01
Connection name: net_demo - Core 0 - Debug
Description: Generated from pre-RSE Launch Configuration
System type: MSC8156
Connection type: CCSSIM2 ISS

CCS server
(● Automatic launch)
Server port number: 40969
[✓] CCS executable: D:\Freescale\CW SC v10.1\SC\ccs\bin\ccssim2.exe
○ Manual launch
Server hostname/IP: 127.0.0.1
Server port number: 41475

Figure 4-8. Properties for net_demo Window

12. Click **OK**.
13. Click **Apply** to save the changes.
14. Click **Close** to save and close the **Debug Configurations** dialog box.

NOTE

To change the network interface configuration modify the `net_demo.c` file in the **net_demo** utility project and rebuild the project. By default, the **net_demo** utility project uses `ucc1` slot.

15. Select **Start > Run**.

    The **Run** dialog box appears.

16. Specify **cmd.exe** in the **Open** text box to launch the command prompt window.
17. Navigate to the `<CWInstall>\SC\ccs\bin`. Where `<CWInstall>` is the path to your CodeWarrior installation. For example, in the command prompt type, `cd C:\Program Files\Freescale\CW SC v10.4\SC\ccs\bin`
18. Launch ccssim2 server using `-tio` argument. `ccssim2.exe -tio`
19. Launch `tio_bridge`. `tio_bridge.exe -ser ucc1 -flags update_ip_header_checksum`

NOTE

Do not close the command prompt window. The Traffic IO log appears in this window once the project is debugged successfully.

20. From the CodeWarrior IDE menu bar, select **Run > Debug Configurations**. CodeWarrior IDE uses the settings in the launch configuration to generate debugging information and initiate communications with the target board.

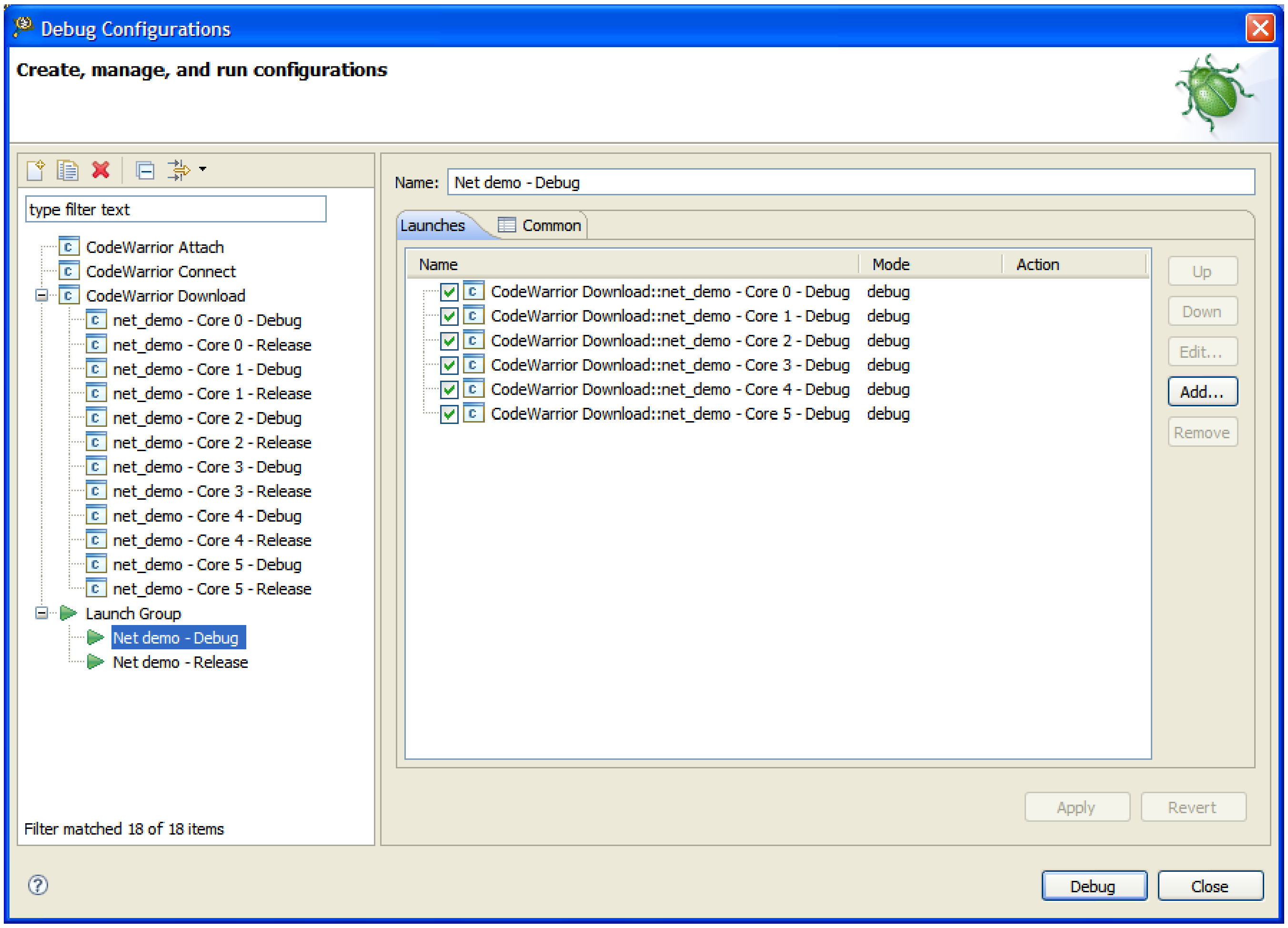
    The **Debug Configurations** dialog box appears. The left pane of this dialog box has a list of debug configurations that apply to the current application.

NOTE

For more information on how to use the debugger, refer to the *CodeWarrior Development Studio Common Features Guide*.

21. Expand the **Launch Group** configuration.
22. From the expanded list, select the launch group that you want to debug. For example, select `Net demo - Debug`.

    The below figure displays the **Debug Configurations** dialog box with the selected launch group.



Debug Configurations

Create, manage, and run configurations

type filter text

- CodeWarrior Attach
- CodeWarrior Connect
- CodeWarrior Download
  - net_demo - Core 0 - Debug
  - net_demo - Core 0 - Release
  - net_demo - Core 1 - Debug
  - net_demo - Core 1 - Release
  - net_demo - Core 2 - Debug
  - net_demo - Core 2 - Release
  - net_demo - Core 3 - Debug
  - net_demo - Core 3 - Release
  - net_demo - Core 4 - Debug
  - net_demo - Core 4 - Release
  - net_demo - Core 5 - Debug
  - net_demo - Core 5 - Release
- Launch Group
  - Net demo - Debug
  - Net demo - Release

Filter matched 18 of 18 items

Name: Net demo - Debug

Launches | Common

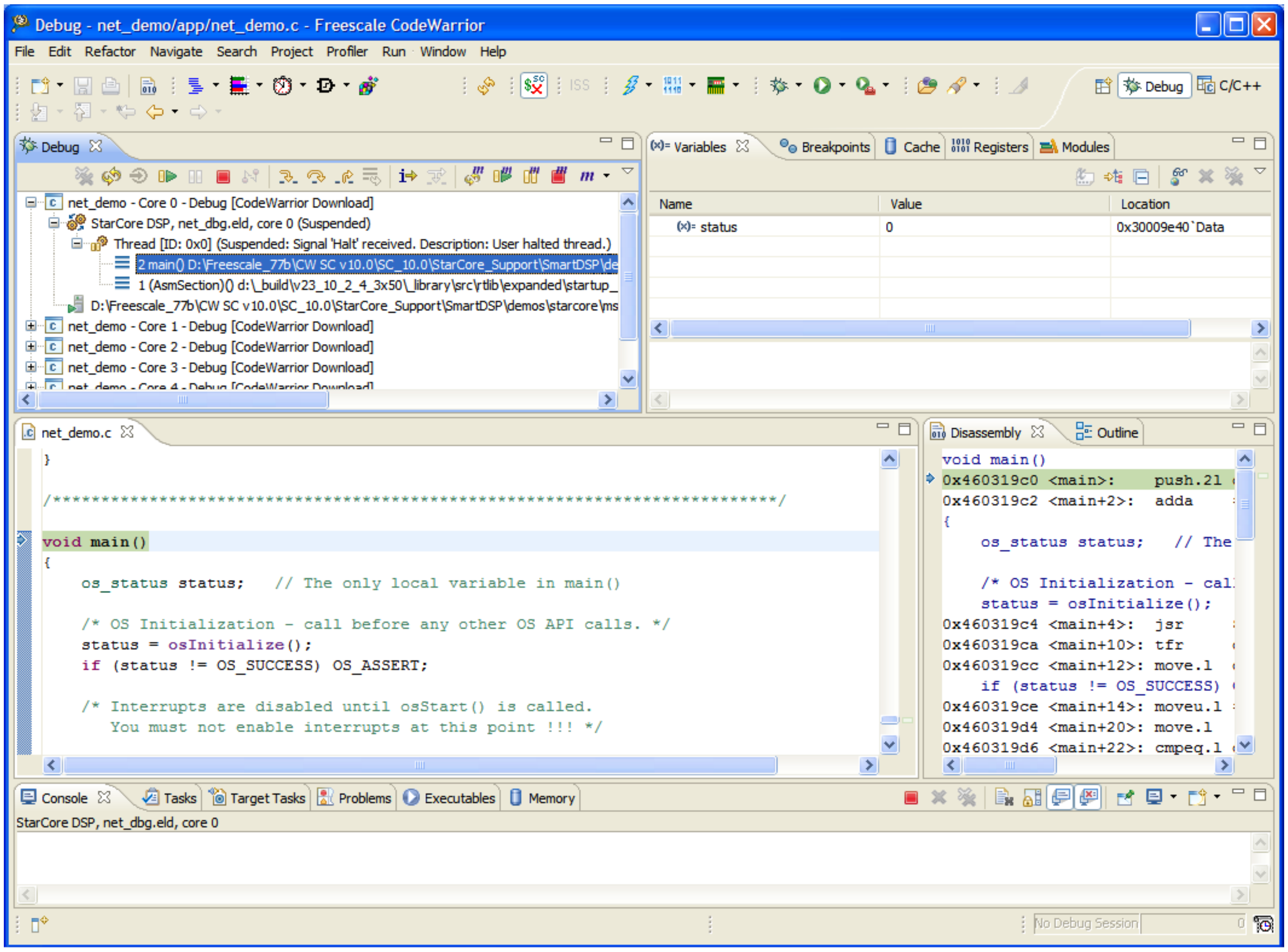
Name	Mode	Action
✓ CodeWarrior Download::net_demo - Core 0 - Debug	debug	
✓ CodeWarrior Download::net_demo - Core 1 - Debug	debug	
✓ CodeWarrior Download::net_demo - Core 2 - Debug	debug	
✓ CodeWarrior Download::net_demo - Core 3 - Debug	debug	
✓ CodeWarrior Download::net_demo - Core 4 - Debug	debug	
✓ CodeWarrior Download::net_demo - Core 5 - Debug	debug	

Figure 4-9. Debug Configurations Dialog Box

23. Select the cores that you want to debug.
24. Click **Debug**.

    The debugger downloads the selected cores and switches to the **Debug** perspective.

    Debugger halts execution at first statement of `main()`. The **Debug** view displays all the threads associated with the core.



Debug - net_demo/app/net_demo.c - Freescale CodeWarrior

File Edit Refactor Navigate Search Project Profiler Run Window Help

Debug

```
net_demo - Core 0 - Debug [CodeWarrior Download]
  StarCore DSP, net_dbg.eld, core 0 (Suspended)
    Thread [ID: 0x0] (Suspended: Signal 'Halt' received. Description: User halted thread.)
      2 main() D:\Freescale_77b\CW SC v10.0\SC_10.0\StarCore_Support\SmartDSP\de
      1 (AsmSection)() d:\_build\v23_10_2_4_3x50\_library\src\rtlib\expanded\startup_
  D:\Freescale_77b\CW SC v10.0\SC_10.0\StarCore_Support\SmartDSP\demos\starcore\ms
net_demo - Core 1 - Debug [CodeWarrior Download]
net_demo - Core 2 - Debug [CodeWarrior Download]
net_demo - Core 3 - Debug [CodeWarrior Download]
net_demo - Core 4 - Debug [CodeWarrior Download]
```

Variables | Breakpoints | Cache | Registers | Modules

Name	Value	Location
status	0	0x30009e40`Data

net_demo.c

```
}

/*****************************************************************************/

void main()
{
    os_status status;   // The only local variable in main()

    /* OS Initialization - call before any other OS API calls. */
    status = osInitialize();
    if (status != OS_SUCCESS) OS_ASSERT;

    /* Interrupts are disabled until osStart() is called.
       You must not enable interrupts at this point !!! */
```

Disassembly | Outline

```
void main()
0x460319c0 <main>:     push.21
0x460319c2 <main+2>:   adda
{
     os_status status;   // The
     /* OS Initialization - cal
     status = osInitialize();
0x460319c4 <main+4>:   jsr
0x460319ca <main+10>:  tfr
0x460319cc <main+12>:  move.l
     if (status != OS_SUCCESS)
0x460319ce <main+14>:  moveu.l
0x460319d4 <main+20>:  move.l
0x460319d6 <main+22>:  cmpeq.l
```

Console | Tasks | Target Tasks | Problems | Executables | Memory

StarCore DSP, net_dbg.eld, core 0

No Debug Session

Figure 4-10. Debug Perspective

25. Click **Multicore Resume** in the Debug view to resume all cores.
26. Select `main` on the call stack in the **Debug** view.
27. Click **Multicore Resume** again.
28. Switch to the command prompt window and wait until the following messages appear in the window:

```
Added new mac c2:c3:c4:c5:c6:cc serial=ucc1

Added new mac c2:c3:c4:c5:c6:c9 serial=ucc1

Added new mac c2:c3:c4:c5:c6:ca serial=ucc1

Added new mac c2:c3:c4:c5:c6:cb serial=ucc1

Added new mac c2:c3:c4:c5:c6:c8 serial=ucc1

Added new mac c2:c3:c4:c5:c6:c7 serial=ucc1
```

29. Launch a separate command window and ping all the IP addresses ranging from `10.0.0.1` to `10.0.0.6`.

The below listing displays the ping output for IP address `10.0.0.1`.

Listing 4-1. Ping IP address output

```
C:\>ping 10.0.0.1
Pinging 10.0.0.1 with 32 bytes of data:

Reply from 10.0.0.1: bytes=32 time<1ms TTL=128

Reply from 10.0.0.1: bytes=32 time<1ms TTL=128

Reply from 10.0.0.1: bytes=32 time<1ms TTL=128

Reply from 10.0.0.1: bytes=32 time<1ms TTL=128

Ping statistics for 10.0.0.1:

    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),

Approximate round trip times in milli-seconds:

    Minimum = 0ms, Maximum = 0ms, Average = 0ms
```

30. Select **Run > Multicore Terminate**.
31. The debugger terminates the active debug session. The threads associated with each core in the **Debug** view disappear.

Chapter 5
Tradeoff Analysis with Simulators

The CodeWarrior for Build Tools compiler is mostly used with compiler options in the command line, such as size-oriented compilation (`-Os` option) and cycle-oriented compilation. In size-oriented compilation, the size is minimized regardless of the performance of the code. A cycle-oriented compilation tries to minimize the cycles at runtime, without considering the source code size. The result is two executables with different outputs.

In real time, you may want to get good performance without sacrificing the source code size. Using a compilation configuration file, you may choose the functions that will use size-oriented compilation and the ones that will use a cycle-oriented compilation. This results in an executable with tradeoff between source code optimized for the size, and source code optimized for performance.

The choice of the compilation mode of each function can be done manually. You may choose a size-oriented compilation for control functions, and a cycle-oriented compilation for computation functions. However, you can also make use of the profiler information during a simulation. The profiler returns values on the size and consumed cycles of each function. The goal is to extract this information to automatically generate a configuration file that is stressed on the size of large functions and the cycles of time-consuming functions.

To extract the profiler information automatically for driving the compilation option, use the following tools:

- combine_dicho - this tool returns a configuration file, if profiler information is provided.
- tradeoff.pl - this tool is used to test different tradeoff strategies available in the `combine_dicho` tool.

5.1 combine_dicho

The `combine_dicho` tool automatically generates a configuration file output that is a tradeoff between source code size and execution speed with respect to the other two tested configurations. The goal of this application is to generate a compilation configuration that is cycle-oriented for functions which are time consuming, and size-oriented for functions that generate large source code.

The inputs to this application are:

- The Result Name. Two files are output: `<resname>.appli`, the configuration file that corresponds to the tradeoff, and `<resname>.opt_level`, a synthetic view of the optimization level used to compile each function. This last file is used by a recursive call of `combine_dicho`.
- The file with suffix, `_size_cycle.rep`, as generated by the profiler when the application is compiled with options that are cycle-oriented, and is executed.
- The file with suffix, `_size_cycle.rep`, as generated by the profiler when the application is compiled with options that are size-oriented, and is executed.
- The file with suffix, `.opt_level`, as created by a previous call to `combine_dicho`, that corresponds to the compilation of the cycle-oriented compilation. Use the word `none` if non-applicable.
- The file with suffix, `.opt_level`, as created by a previous call to `combine_dicho`, that corresponds to the compilation of the size-oriented compilation. Use the word `none` if non-applicable.

5.1.1 Example

This example shows two source files, `file1.c` and `file2.c`. The output will show the tradeoff between a `-O3` compilation (cycle-oriented), and a `-O3 -Os` compilation (size-oriented).

Given below are the steps:

1. Compile the application using cycle-oriented compilation. Use option `-Xllt -profile_level1` in order to generate profiling information of inlined functions:
   - `scc -O3 -Xllt -profile_level1 -o exec_perf.eld file1.c file2.c`
2. Run this application using the profiler, `runsim`. The generated file `prof_perf_size_cycle` will be used during the combination of cycle results and size results:
   - `runsim -p prof_perf exec_perf.eld`
3. Compile using size-oriented option, and run the application with the profiler:

- `scc -O3 -Os -Xllt -profile_level1 -o exec_size.eld file1.c file2.c`
- `runsim -p prof_size exec_size.eld`

4. Combine the profiling results to generate a `conf.appli` file:
   - `combine_dicho conf prof_perf_size_cycle.rep prof_size_size_cycle.repnone`
5. Compile the application with the new configuration. Do not specify the `-Os` option:
   - `scc -O3 -ma conf.appli -o exec_conf.eld file1.c file2.c`

5.2 tradeoff.pl

The objective of the tool is to return data (number of consumed cycles; size of the executable) so that you can choose the best tradeoff between speed and size that fits its constraints.

For example, you may want to obtain the best performances, with a source code size lower than a given threshold.

This tool runs an automatic process that:

- Compiles the application with cycle-oriented and size-oriented options.
- Runs the profiler to profile this application. The output file used by `combine_dicho` is `_size_cycle.rep`.
- Run the `combine_dicho` tool for the number of tries specified by the user. The output file is `_size_cycle.rep`. The application is compiled using the obtained configuration files. The source code size and the run time of the application are then stored.

For each tested configuration file, the source code size and run time is saved in a file named `profile/result_tradeoff_<target>`, where `<target>` is the target to compile in the makefile. Using the resulting data, you can choose the tradeoff configuration that best suits its needs.

5.2.1 Tradeoff Constraints

Compilation must be called through a makefile that contains:

- A target to clean previous compilation results; that is, `.obj`, `.eln` and `.eld` files. By default, this target is clean, but any other name can be specified as it will be described in the command line of the tool, `tradeoff.pl`.
- A directive to pass extra compilation options. It is useful to be able to specify `-Os` and `-ma` options.

Moreover, compilation is performed using gmake, that is make from GNU.

cygwin users should have the make package installed; '`gmake`' should be a symbolic link for `/usr/bin/make` as accomplished by running:

```
ln -s /usr/bin/make /usr/bin/gmake
```

Solaris users should have GNU make installed and available in PATH.

5.2.2 Tradeoff Usage

The following are the arguments that can be passed through `tradeoff.pl`.

- General options
  - `-h`: displays the tool's help.
  - `-depth <depth>`: recursive depth in the dichotomist tree. A depth of 0 returns 3 points, a depth of 1 returns 5 points, a depth of 2 returns 9 points, a depth of 3 returns 17 points in the series. Default value is 2.
- Options related with the makefile
  - `-f <makefile>`: the default name of the makefile is Makefile. The `-f` option allows you to call another makefile.
  - `-clean <clean>`: target in the makefile to clean for the compilation. Default value is `clean`.
  - `-cflags <cflags>`: flag in the makefile to pass additional options (like `-Os` or `-ma` options) to the compilation. Default value is `CFLAGS`.
  - `-options <options>`: Default compilation options to be passed to the makefile. Default value is `-O3 -Og`.
- Mandatory parameters
  - `<target>`: the target to compile in the makefile.
  - `<exe>`: name of the executable that is built by the makefile.
  - `<list of runtime arguments>`: list of the arguments to be passed when running the executable. `Tradeoff.pl` will run the following command:

    ```
    runsim -t -p profile/profileres <exe> <list of runtime arguments>.
    ```
- Results are saved in the file
  - `profile/result_tradeoff_<target>`

5.2.3 Example

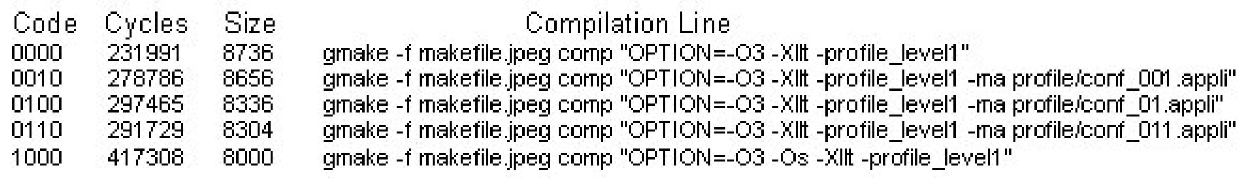
The example has one source file `jpeg.c`. The application takes no argument in the runtime. Follow the given steps to use `tradeoff.pl`:

1. Create a makefile (`makefile.jpeg`) to be able to compile the application. It contains a clean target (`myclean`), a target to compile the application (`comp`). You can pass additional options through the `OPTION` flag. The name of the executable is `bin/jpeg.eld`. Below is the makefile associated with the application:

Listing 5-1. Tradeoff example

```
CC = scc
CFLAGS = -mb ${OPTION}
comp:

        $(CC) $(CFLAGS) -o bin/jpeg.eld jpeg.c

myclean:
        rm -f jpeg.eln bin/jpeg.eld
```

2. To have the tradeoff in global optimization using `-O3` option, and to have a series computed with a depth of 1, run the following command: `tradeoff.pl -depth 1 -f makefile.jpeg -cflags OPTION\-clean myclean -options "-O3" comp bin/jpeg.eld`
3. The results can be read in the file `profile/result_tradeoff_comp` as shown in the below listing **Result of an Application** .



```
Code   Cycles   Size                          Compilation Line
0000   231991   8736    gmake -f makefile.jpeg comp "OPTION=-O3 -Xllt -profile_level1"
0010   278786   8656    gmake -f makefile.jpeg comp "OPTION=-O3 -Xllt -profile_level1 -ma profile/conf_001.appli"
0100   297465   8336    gmake -f makefile.jpeg comp "OPTION=-O3 -Xllt -profile_level1 -ma profile/conf_01.appli"
0110   291729   8304    gmake -f makefile.jpeg comp "OPTION=-O3 -Xllt -profile_level1 -ma profile/conf_011.appli"
1000   417308   8000    gmake -f makefile.jpeg comp "OPTION=-O3 -Os -Xllt -profile_level1"
```

Figure 5-1. Tradeoff result

In the tradeoff result as shown above, each line represents a trial.

- The first column is a binary code of one trial - the greater the source code, the better should be the size of the executable.
- The second column shows the number of cycles of the execution of this trial.
- The third column shows the size (in bytes) of the executable.
- Last column shows the command line used to compile the trial.

You can choose the strategy best suited to your requirement.

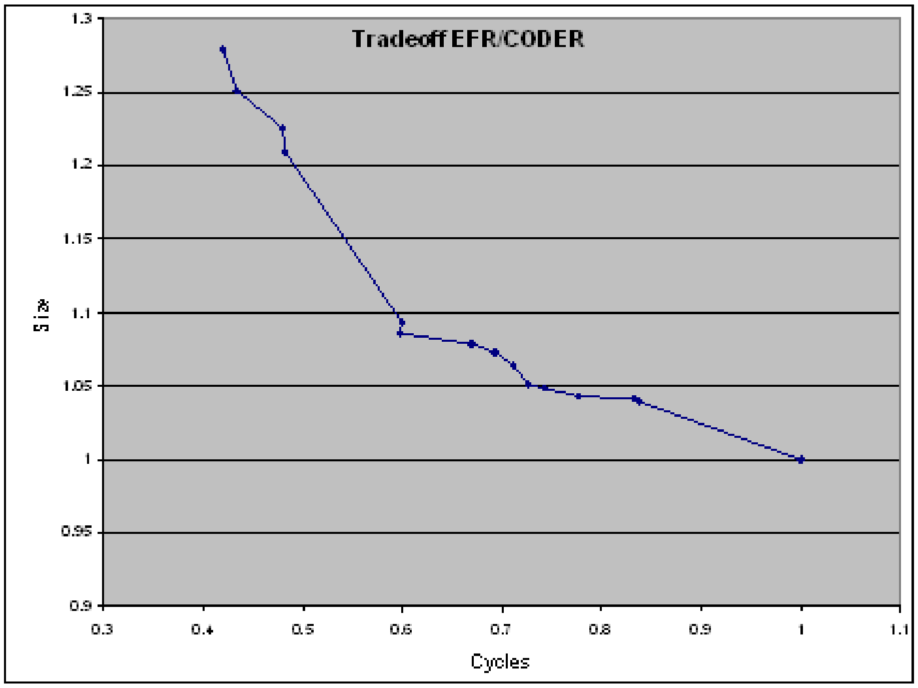
In the below listing, the results obtained on the coder of the EFR (the compilation line has been removed from the following trace) is displayed.

Listing 5-2. Result of an Application

```
Code          Cycles       Size                    Compilation line
00000000      3918826     47152

00001000      4051866     46112
```

```
00010000    4481293    45184

00010010    4505771    44576

00100000    5598560    40304

00100101    5573863    40016

00101000    6251150    39760

00101010    6470126    39552

01000000    6634867    39216

01010110    6786902    38752

01011000    6935248    38640

01011011    7257258    38448

01100000    7775553    38368

01101110    7812573    38336

01110000    7812573    38336

01111000    7812573    38336

10000000    9334324    36864
```

The below figure shows the normalized representation of the application result.



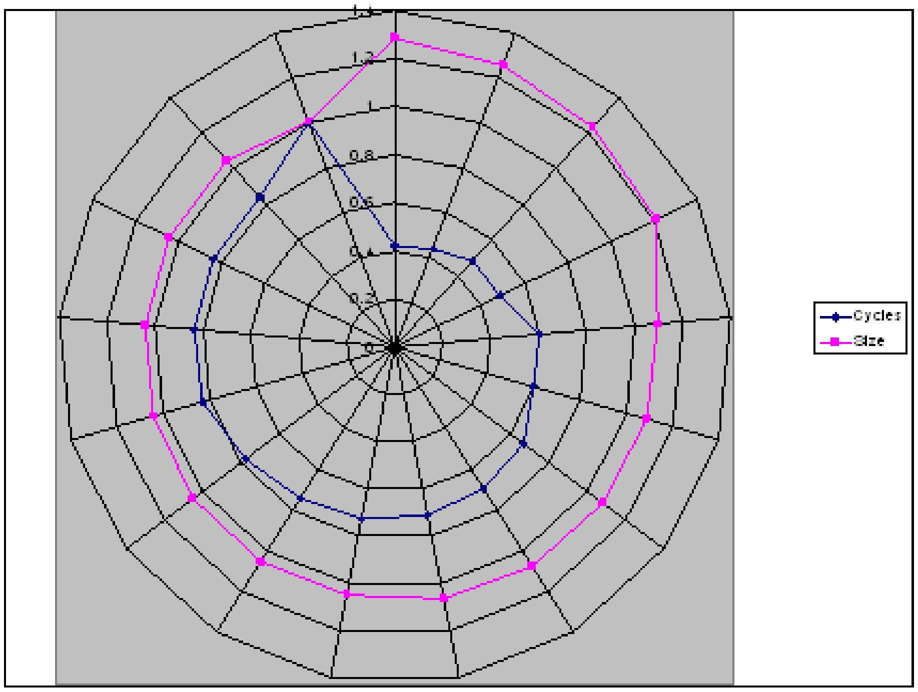


Figure 5-2. Normalized Representation

Chapter 6
Accuracy Benchmarks

The timing of the cache commands introduced in this release has not been fully verified.

This chapter describes:

- sc3850plat_pacc
- sc3850plat_pacc, DDR Memory

6.1 sc3850plat_pacc

The golden results for the sc3850plat_pacc simulator are received from an MSC8156 ADS board running at 1Ghz frequency. Same applications are linked to M2, M3, and DDR memory and the results are compared. All results are taken from dpu measures.

6.1.1 M2 Tests

Benchmark Test	Diff %
benchmark-C_OPT_telecom_autcor00_DATA1_19	0.17
benchmark-C_OPT_telecom_autcor00_DATA2_21	-0.08
benchmark-C_OPT_telecom_autcor00_DATA3_23	2.37
benchmark-C_OPT_telecom_conven00_DATA1_25	-0.04
benchmark-C_OPT_telecom_conven00_DATA2_27	0.04
benchmark-C_OPT_telecom_conven00_DATA3_29	0.04
benchmark-C_OPT_telecom_fbital00_DATA2_31	0.00
benchmark-C_OPT_telecom_fbital00_DATA3_33	0.00
benchmark-C_OPT_telecom_fbital00_DATA6_35	0.00
benchmark-C_OPT_telecom_fft00_DATA1_37	5.00

Table continues on the next page...

Benchmark Test	Diff %
benchmark-C_OPT_telecom_viterb00_DATA1_43	-9.49
benchmark-networking_routelookup_54	-0.03
benchmark-telecom_autcor00_DATA1_18	-0.02
benchmark-telecom_autcor00_DATA2_20	-0.02
benchmark-telecom_autcor00_DATA3_22	-2.11
benchmark-telecom_conven00_DATA1_24	-0.01
benchmark-telecom_conven00_DATA2_26	-0.01
benchmark-telecom_conven00_DATA3_28	-0.01
benchmark-telecom_fbital00_DATA2_30	0.00
benchmark-telecom_fbital00_DATA3_32	-0.02
benchmark-telecom_fbital00_DATA6_34	-0.00
benchmark-telecom_fft00_DATA1_36	6.1
benchmark-telecom_viterb00_DATA1_42	3.17
benchmark-telecom_viterb00_DATA2_44	3.17
benchmark-telecom_viterb00_DATA3_46	3.17
benchmark-telecom_viterb00_DATA4_48	3.17
benchmark-mac_hs-mac_hs_exec_3	-0.21
benchmark-pstone-auto_4	-0.32
benchmark-pstone-blit_5	-1.76
benchmark-pstone-compress_6	-1.8
benchmark-pstone-des_7	0.1
benchmark-pstone-dhry21_8	-2.1
benchmark-pstone-engine_9	-0.06
benchmark-pstone-eval2_10	-11.12
benchmark-pstone-g3fax_12	-0.02
benchmark-pstone-pocsag_14	-0.26
benchmark-pstone-summin_15	-0.19
benchmark-pstone-ucbqsort_16	0.3
benchmark-pstone-v42bis_17	-0.44
benchmark-sc_ffl-deblock_hor_chr_59	-0.1

6.1.2 M3 Tests

Benchmark Test	Diff %
benchmark-pstone-auto_4	0.38
benchmark-pstone-blit_5	1.87
benchmark-pstone-compress_6	-1.41

Table continues on the next page...

Benchmark Test	Diff %
benchmark-pstone-des_7	-0.39
benchmark-pstone-dhry21_8	5.40
benchmark-pstone-engine_9	0.23
benchmark-pstone-eval2_10	-0.71
benchmark-pstone-g3fax_12	-0.02
benchmark-pstone-pocsag_14	0.27
benchmark-pstone-summin_15	-1.02
benchmark-pstone-ucbqsort_16	0.37
benchmark-pstone-v42bis_17	0.08
benchmark-sc_ffl-find_r16_abs_max_55	6.09
benchmark-sc_ffl-find_r16_abs_max_pos_56	5.12
benchmark-sc_ffl-find_r16_max_57	5.43
benchmark-sc_ffl-find_r16_max_pos_58	5.11

6.2 sc3850plat_pacc, DDR Memory

This section lists the benchmarks for sc3850plat_pacc.

Benchmark	Diff %
benchmark-C_OPT_telecom_autcor00_DATA1_19	-3.24
benchmark-C_OPT_telecom_autcor00_DATA2_21	-1.06
benchmark-C_OPT_telecom_autcor00_DATA3_23	1.59
benchmark-C_OPT_telecom_conven00_DATA1_25	-0.29
benchmark-C_OPT_telecom_conven00_DATA2_27	-0.5
benchmark-C_OPT_telecom_conven00_DATA3_29	-0.29
benchmark-C_OPT_telecom_fbital00_DATA2_31	-0.04
benchmark-C_OPT_telecom_fbital00_DATA3_33	-0.29
benchmark-C_OPT_telecom_fbital00_DATA6_35	-0.04
benchmark-C_OPT_telecom_fft00_DATA1_37	4.54
benchmark-C_OPT_telecom_viterb00_DATA1_43	9.34
benchmark-networking_routelookup_54	0.01
benchmark-telecom_autcor00_DATA1_18	-2.5
benchmark-telecom_autcor00_DATA2_20	-0.23
benchmark-telecom_autcor00_DATA3_22	1.9
benchmark-telecom_conven00_DATA1_24	-0.16
benchmark-telecom_conven00_DATA2_26	-0.11
benchmark-telecom_conven00_DATA3_28	-0.19

Table continues on the next page...

Benchmark	Diff %
benchmark-telecom_fbital00_DATA2_30	-0.02
benchmark-telecom_fbital00_DATA3_32	-1.94
benchmark-telecom_fbital00_DATA6_34	-0.26
benchmark-telecom_fft00_DATA1_36	5.54
benchmark-telecom_viterb00_DATA1_42	3.08
benchmark-telecom_viterb00_DATA2_44	3.09
benchmark-telecom_viterb00_DATA3_46	3.11
benchmark-telecom_viterb00_DATA4_48	3.12
benchmark-mac_hs-mac_hs_exec_3	-0.81
benchmark-pstone-auto_4	-2.32
benchmark-pstone-blit_5	0.16
benchmark-pstone-compress_6	-3.86
benchmark-pstone-des_7	-0.55
benchmark-pstone-dhry21_8	-4.25
benchmark-pstone-engine_9	-1.34

Chapter 7
Speed Benchmarks

Speed benchmark tests were done on a computer with the following configuration:

Windows XP SP2, Intel® Pentium DualCore Processor, 2.4GHz, 1Gb RAM.

NOTE

The performance of various simulator models observed on Linux machine is lower than on Windows.

Table 7-1. Tests done on ISS and PACC Models

Test	src.eld	
model	sc3850plat_iss	sc3850plat_pacc
Windows (Kinstr/sec)	772.2	26.76

Networking Look-Up algorithm.

Route Look-Up is a distillation of the fundamental operation of IP datagram routers: receiving and forwarding IP datagrams, implementing an IP lookup mechanism based on a Patricia Tree (or trie).

Table 7-2. Networking Look-Up Tests

Test	benchmark-networking_routelookup_54/networking_routelookup.eld	
model	sc3850plat_iss	sc3850plat_pacc
Windows (Kinstr/sec)	1231.5	46.5

Telecom Bit Allocation Transform.

Bit Allocation benchmark tests the ability of the target processors to spread a stream of data over a series of buffers, which it then modulates and transmits on a telephone line in ADSL applications.

Table 7-3. Bit Allocation benchmark Tests

Test	benchmark-telecom_fbital00_DATA2_30/telecom_fbital00_DATA2.eld	
model	sc3850plat_iss	sc3850plat_pacc
Windows (Kinstr/sec)	1456.48	58.3

Index

How to Reach Us:

Home Page:
freescale.com

Web Support:
freescale.com/support

Document Number CWSCSIMUG
Revision 10.9.0, 06/2015