

# Improve Program Execution Speed Using Ethernet File I/O for StarCore DSPs

This application note presents a new debugger feature that allows users to perform real-time file I/O operations via an Ethernet connection between a MSC815xADS board and a CodeWarrior debugger client on the host side.

Traditionally, users rely on the standard file I/O libraries that accomplish the I/O operations via the JTAG communication channel. Due the limited speed of the JTAG interface, such operations take a very long time to be executed—especially if the program handles large blocks of data. To overcome this drawback, the Ethernet File I/O is the solution.

This document assumes the reader is familiar with the Eclipse IDE. It also assumes that the user understands the CodeWarrior debugger, plus the Ethernet and Internet protocols.

All of the results and screen captures are based on the CodeWarrior for StarCore DSPs v10.1.11 release. For later releases there might be minor GUI changes, but the operation of the tools should be similar.

## Contents

1	Ethernet File I/O Description .....	2
2	How NetFileIO Works .....	4
3	NetFileIO Setup .....	5
4	NetFileIO Examples .....	11
5	Appendix .....	17
6	Revision History .....	18

# 1 Ethernet File I/O Description

## 1.1 What Is It

The file I/O over the Ethernet capability (also known as NetFileIO), is a CodeWarrior tools feature that facilitates data transfer between a hardware target and CodeWarrior client.

### NOTE

Currently only two StarCore platforms are supported: MSC8156ADS and MSC8154ADS. These platforms also support the MSC8151, MSC8152, and MSC825x variants.

Compared to alternate connection interfaces, the transfer rate for download/upload data for NetFileIO is much larger:

- JTAG (no additional libraries are required): 0.5–7 KBPS
- HSST (two additional libraries are required): 22–300 KBPS
- NetFileIO (two additional libraries are required): 4–8 MBPS

The main advantage of using this interface over the “classic” JTAG one is the download/upload speed of the data transfer. By using NetFileIO, the application under test can execute in real time, while being under debugger control.

## 1.2 When to Use It

NetFileIO should be used in conjunction with applications that do not use SmartDSP OS (SDOS).

### NOTE

SDOS provides full support for the NET stack, which can be configured via the driver for the QUICC Engine Unified Ethernet controller (UEC). In this case, the developer should use the SDOS HEAT utility to support file I/O over Ethernet.

### NOTE

NetFileIO implements only minimal network stack functions. Only ARP is supported, along with some additional configurations that are enabled through definitions in the header files.

For bare board applications, there are no restrictions in using NetFileIO.

NetFileIO is best suited for applications which require high data transfer rates through the use of standard I/O operations.

Use NetFileIO for:

- Bare-board applications designed for testing the bit-exactness results of kernels. In this case, the program must first load test vectors stored on a remote host, execute the kernel, and then store the results back onto the host so that they can be checked against the reference test vectors.
- Profiling bare-board applications. NetFileIO can be used as an alternative to the HSST download method. For more detail about HSST, consult the application note AN4233. “HSST Setup for VTB Fast Download.”

## 1.3 Constraints

The current implementation of the NetFileIO libraries supports only a Gigabit Ethernet connection. The 10BaseT and 100BaseTX connections are not yet supported.

### NOTE

For lower-speed networks (for example, 10/100Mb) the NetFileIO libraries need to be modified and rebuilt according to the required physical layer requirements.

The ADS board must be configured via specific switches to use the RGMII settings. Refer to the *MSC8156ADS Hardware Getting Started Guide* for details. A configuration example appears later in this document.

The physical Ethernet connection between the host and target may use straight or cross-over cable.

## 1.4 Acronyms and Definitions

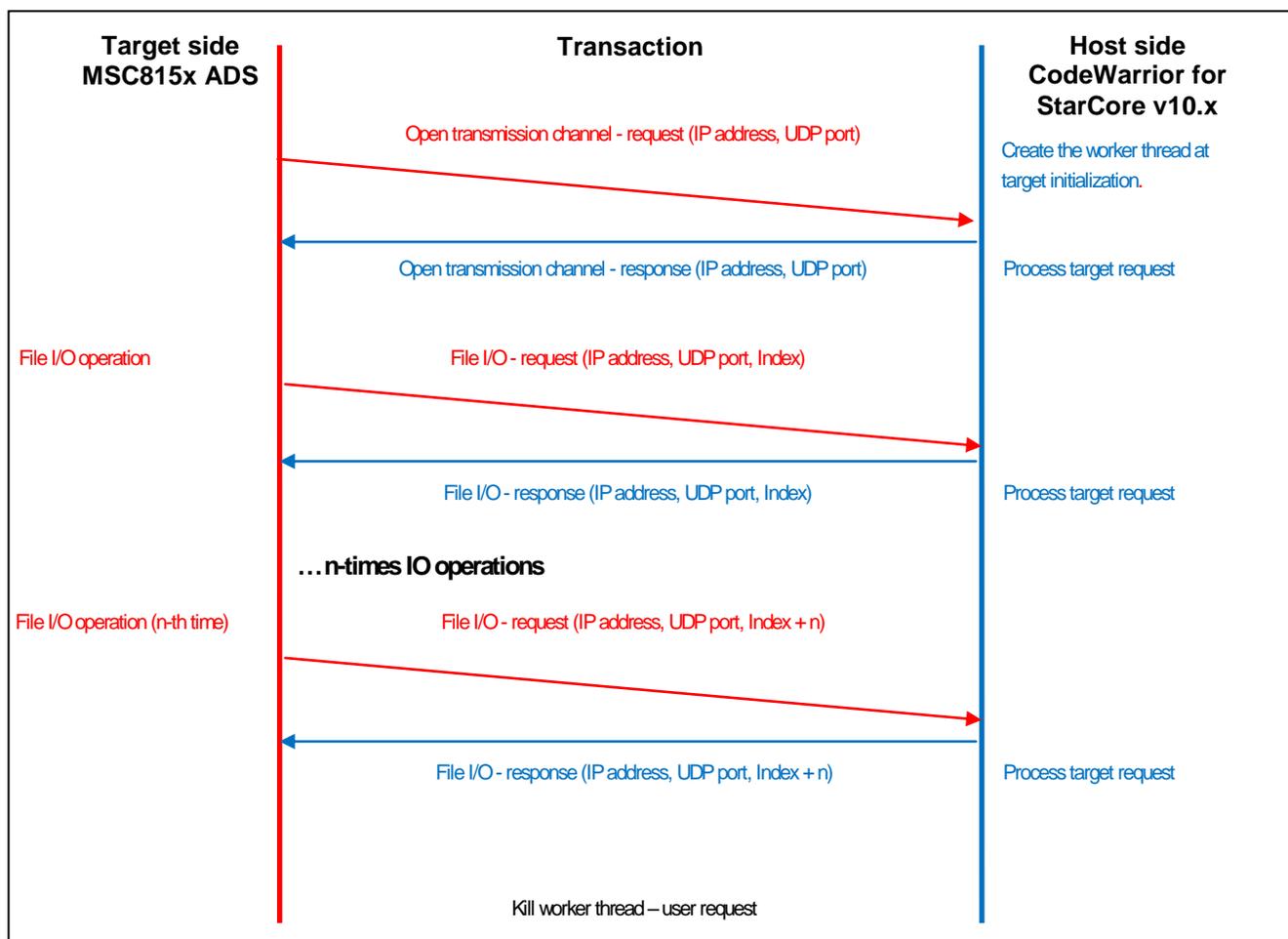
Table 1. Acronyms and definitions

Term	Definition
ARP	Address Resolution Protocol
GUI	Graphical User Interface
File IO	File Input Output
IDE	Integrated Development Environment
IP	Internet Protocol
DSP	Digital Signal Processor
NetFileIO	File Input Output over the Network
OS	Operating System
HEAT	Host Exchange over Asynchronous Transfer
HSST	High Speed Serial Transfer
UDP	User Datagram Protocol

## 2 How NetFileIO Works

NetFileIO implements a request-response protocol over the Ethernet connection. The target, the MSC815xADS board, sends a request and the CodeWarrior debugger on the host responds. See [Figure 1](#) for a schematic timeline that depicts the sequence of requests/responses.

Multiple communication channels can be opened between the target and the host. If the UDP transport protocol is used, then each of channels is defined by the MSC815xADS board's network IP address (one per target) and by the UDP port (one for each core).



**Figure 1. A Timeline of NetFileIO Transactions.**

NetFileIO implements a protocol that avoids data loss based on frame retransmission and a frame sequence number. For each new file I/O transaction, the frame's sequence number is incremented. If the host does not respond within a timeout period, the target resends the frame.

### NOTE

The timeout interval for a File I/O transaction is 1 second, and the maximum number of retries is 32.

The host decides, based on the sequence number, if the current frame is new or if it has been transmitted again and served already. If the same transaction is received, the CodeWarrior debugger ignores it.

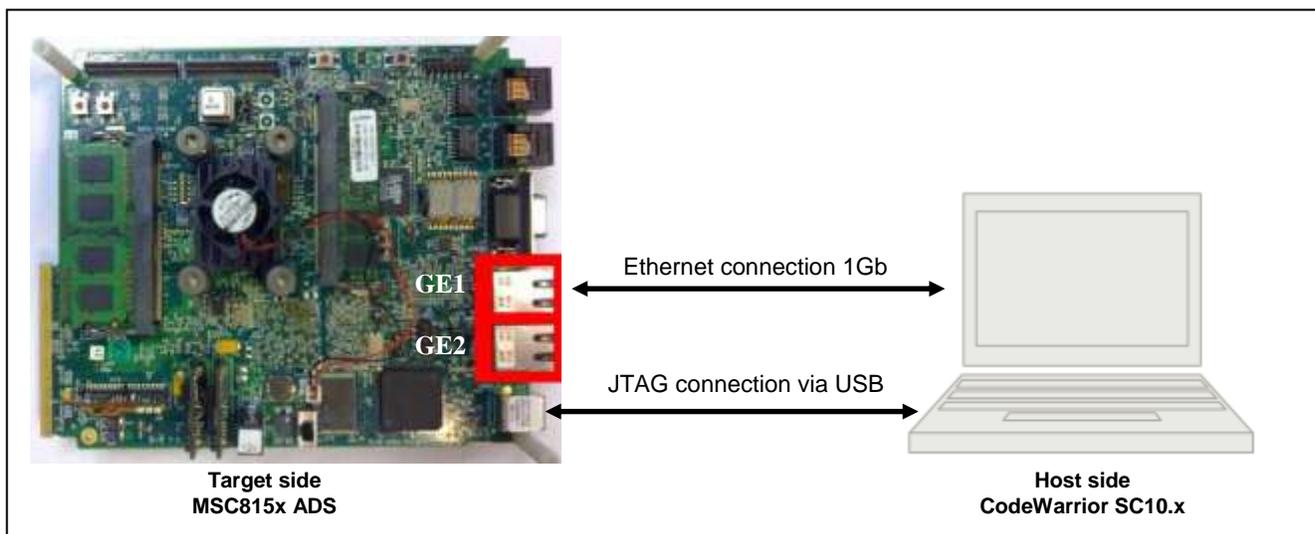
### 3 NetFileIO Setup

#### 3.1 Hardware Setup

The MSC815xADS board has two Gigabit Ethernet ports (GE1 and GE2) that can be configured independently for RGMII or SGMII interface mode. In order to work, NetFileIO requires the ADS board to be configured for RGMII mode.

**NOTE**

To ensure that NetFileIO works correctly, make sure that MSC8156ADS DIP Switches (SW) are set according to the *MSC8156ADS Hardware Getting Started Guide*. The default configuration presented in the chapter, Switch Default Settings, is appropriate for running the demos shown in this application note.



**Figure 2. Hardware Configuration – Generic View.**

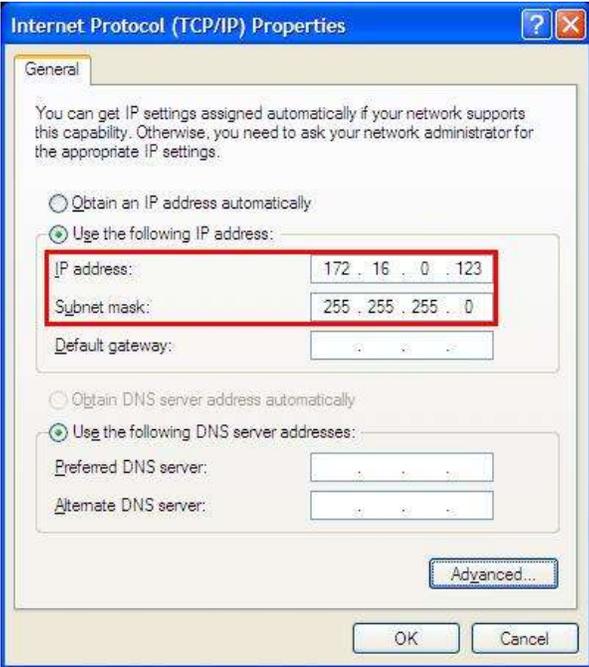
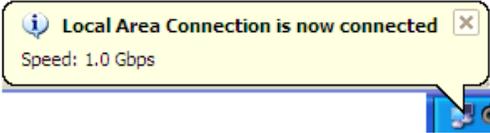
Figure 2 depicts a generic hardware setup that can be used for file I/O via Ethernet. The Ethernet connection is used only for file I/O. The program download and debug is accomplished through the JTAG interface.

**NOTE**

The host must have a Gigabit Ethernet card. Otherwise, NetFileIO will not work.

Table 2 shows the steps required to set up and configure the hardware properly.

**Table 2. Hardware Configuration Steps**

Step	Description
1	Power off the MSC8156 ADS board.
2	Make sure RGMII mode is enabled. <b>SW2.6: RGMII Port 2 IO (GE2) - OFF</b> Default: OFF (RGMII2 is active.) <b>SW2.7: RGMII Port 1 IO (GE1) - OFF</b> Default: OFF (RGMII1 is active.)
3	Connect a straight Ethernet cable between the target's GE1 port and the host PC's network card.
4	Connect the USB cable between the target's J13 port and the host PC's USB port. Alternatively, use the USB TAP or Ethernet TAP to connect the target with the host.
5	<p>On the target side, open the Local Area Connection properties and make sure the IP address and Subnet mask fields are configured.</p> <p>For all of the examples used in this application note, the host will be configured as below:</p> 
6	<p>Power on the ADS board and check that the network is up and running. If the setup was done correctly, on the right lower corner of the desktop the following message appears.</p> 

## 3.2 Software Setup

The software support for NetFileIO is divided in two parts:

- Library support – These libraries implement the Ethernet driver and Ethernet `syscall` function.
- Debugger IO Model – It provides support for network transfers.

### 3.2.1 Support Libraries

For NetFileIO support, there are two libraries that must be linked with the application:

- `ethernet-syscalld.elb` or `ethernet-syscallr.elb`. They are located in directory: `{CW install dir}\SC\StarCore_Support\EthernetIO\SyscallLib`
- `815x-ethernetd.elb` or `815x-ethernetr.elb`, located in the directory: `{CW install dir}\SC\StarCore_Support\EthernetIO\EthLib`

#### NOTE

In CodeWarrior there are two flavors of each library: one built with debug information (d suffix on the file name) for testing, while the second one is built for release (r suffix on the file name) for shipping code.

Figure 3 presents the software implementation of NetFileIO libraries.

The low-level Ethernet support for data and control is implemented by the Ethernet driver library. This library provides a socket API and contains the data structures and initialization code needed for Ethernet support. The structures are initialized at the start up based on the settings made in CodeWarrior debugger.

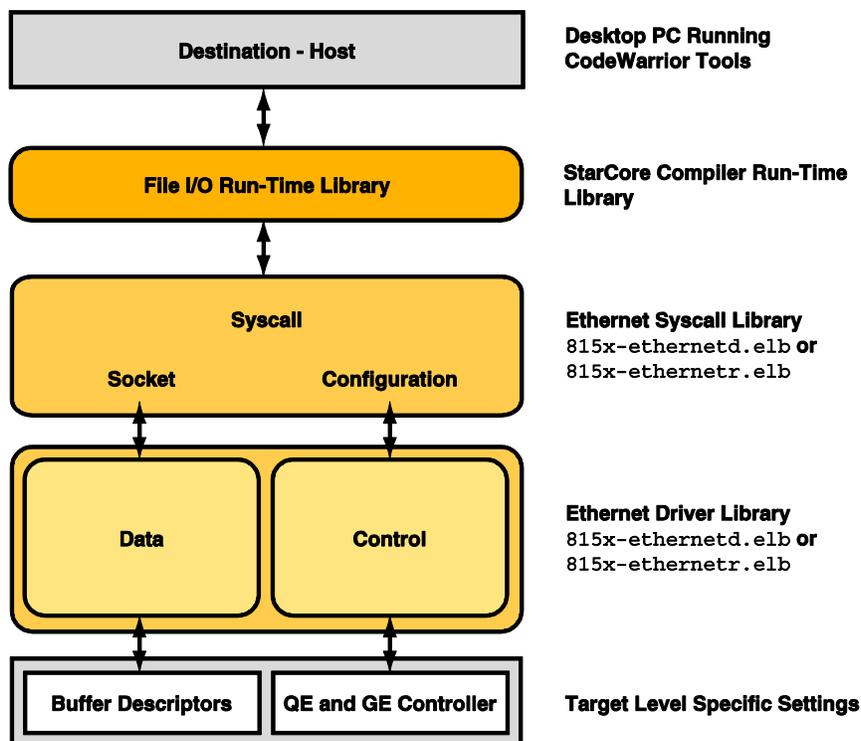
#### NOTE

The Ethernet driver library implements the following protocols: Ethernet, ARP, IP, UDP and Internet Control Message Protocol – Echo Reply.

The Ethernet driver uses 32 Buffer Descriptors (16 for transmission and 16 for receiving) and 16 Data Buffers for receiving Ethernet frames.

#### NOTE

The cores and the QUICC Engine module share these buffers.



**Figure 3. The Libraries That Implement NetFileIO.**

The high-level Ethernet support that connects the Ethernet driver with the StarCore compiler run-time library is implemented via the Ethernet Syscall library, and provides a `syscall` API. This library provides support for the following functions: `fwrite()`, `fread()`, `fopen()`, `fseek()`, `fclose()`, `unlink()`, `rename()`, `system()`, `mkdir()` and `rmdir()`.

### 3.2.2 Debugger I/O Model

The CodeWarrior debugger configures the Ethernet driver based on the options entered in the **I/O Model** tab of the **Network transfer** dialog window (Figure 4). Table 3 summarizes the purpose of the options presented in this dialog.

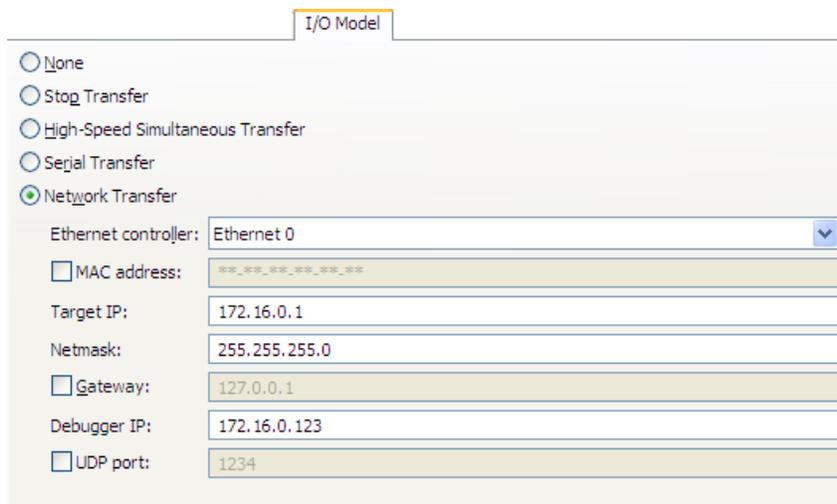


Figure 4. CodeWarrior Debugger Settings for the Network Configuration.

Table 3. CodeWarrior Debugger Network Configuration Options.

Field	Description	Type
Ethernet controller	Select the Ethernet controller used for hardware connection.	Mandatory
MAC address	MAC address assigned to the MSC815xADS board. If not set, then the CodeWarrior debugger uses the default address.	Optional
Target IP	The IP assigned by CodeWarrior debugger to the MSC815xADS board, via the structures exposed in Ethernet driver.	Mandatory
Netmask	The Netmask assigned by CodeWarrior debugger to the MSC815xADS board via the structures exposed in Ethernet driver.	Mandatory
Gateway	MSC815xADS board gateway address. This parameter is useful for board configuration when the target and host are not part of the same network.	Optional
Debugger IP	Debugger IP address must be the same as host network card's IP address. The DSP board uses this parameter as destination IP address for NetFileIO protocol.	Mandatory
UDP port	UDP port used by NetFileIO protocol. This parameter is core-specific.	Optional

---

**NOTE**

For a multicore environment, the CodeWarrior debugger increments the UDP port automatically for each of the cores.

**NOTE**

For a daisy-chained environment that uses multiple MSC815x DSPs, the CodeWarrior debugger increments the UDP port automatically for each of the cores and increments the MAC address and Target IP for each of the processors.

## 4 NetFileIO Examples

### 4.1 File I/O Over Ethernet

This simple example demonstrates how to set up the CodeWarrior debugger to use NetFileIO support. The example is based on a CodeWarrior stationary project, which is then modified to perform file I/O over the Ethernet connection. The required steps are:

1. Set up the hardware as it is explained in chapter 3.1.
2. Launch the CodeWarrior IDE and create a new stationary project. For this example, an ADS8156 board was selected.
3. Open the **Project Properties** window (Figure 5) and under the **Tool Settings** tab add the NetFileIO required libraries described in chapter 3.2.1.

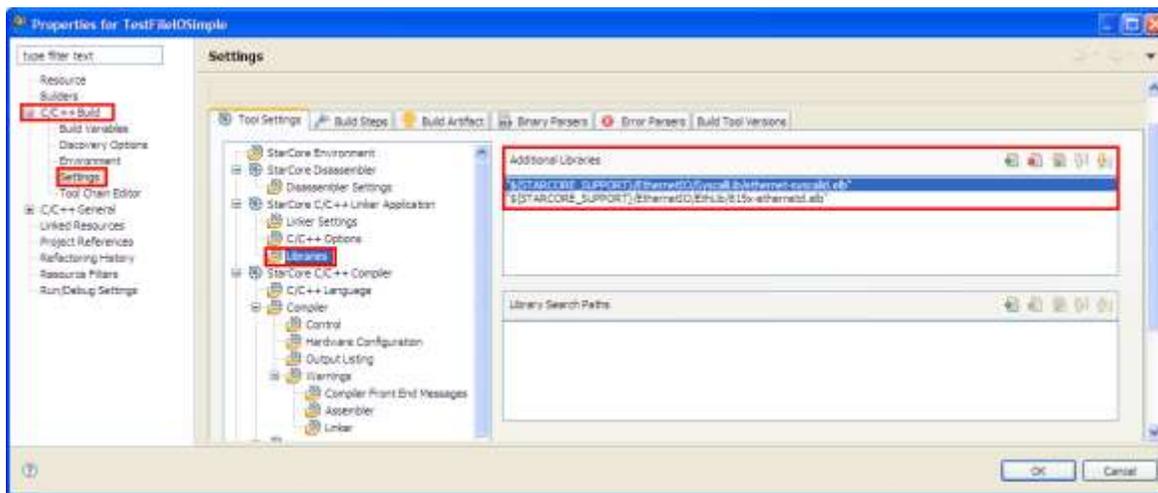


Figure 5. Adding the NetFileIO Libraries to the Project.

#### NOTE

For portability reasons, it is best to avoid using absolute paths in the project. Instead, use a path that is linked with a build system variable.

4. Copy and paste the code from this document's Appendix into `m8156_main.c`. This code opens and fills a file with a 64 MB data pattern.
5. Modify the project's `.appli` file to place the data into the private memory of each core. For simplicity, this application note places the data into the M2 memory:

```
module "m8156_main" [
    rom = M3__cacheable_wb_sys_shared_rom
```

```

bss = M2__cacheable_wb_sys_private_bss
data = M2__cacheable_wb_sys_private_data
]

```

6. Open the **Debug Configuration** window and choose the project's launch configuration. In the **Main** tab, edit the **Remote System** properties. In the Properties window that appears, select the **I/O Model** tab and modify the settings as show in [Figure 6](#).

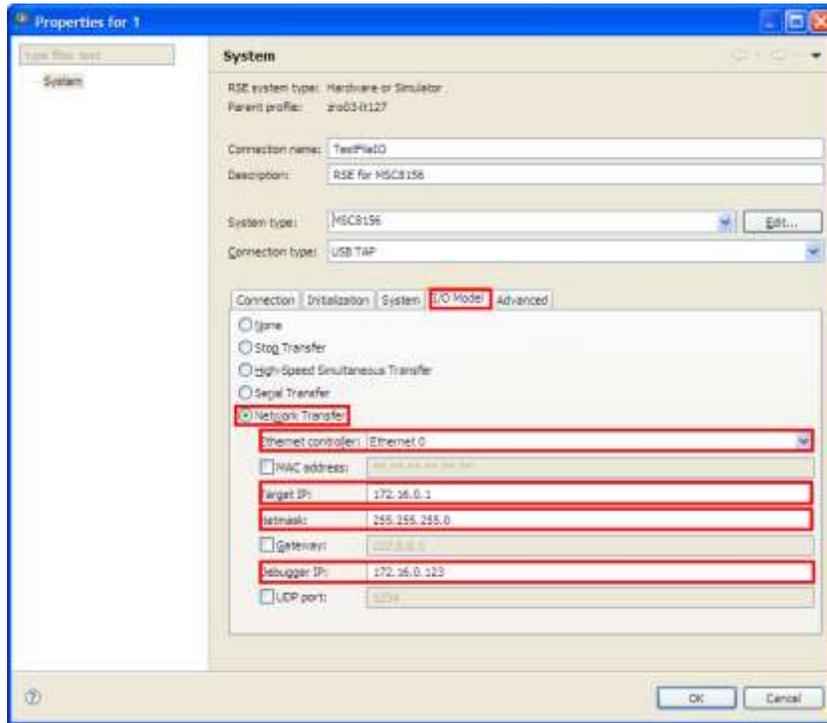
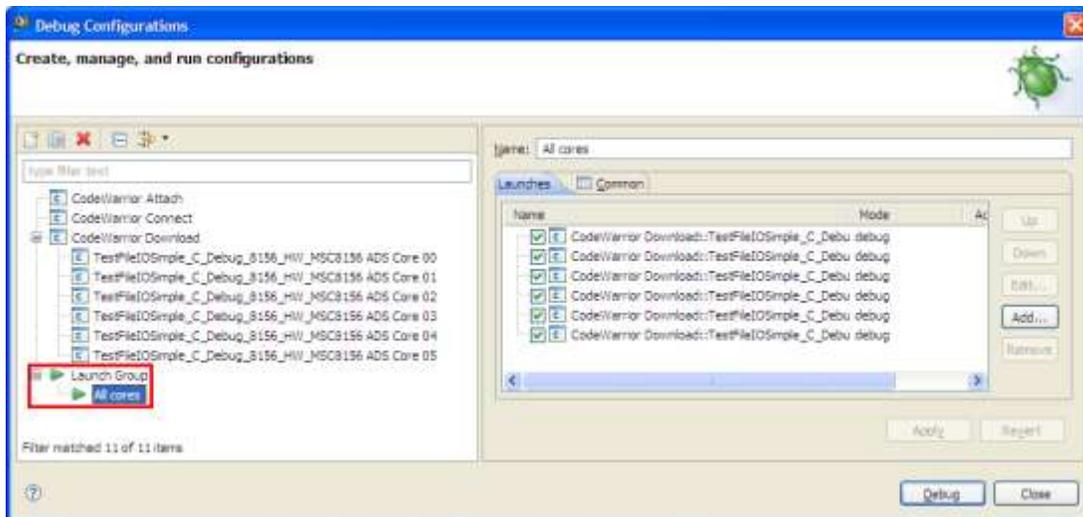


Figure 6. Modifying the Network Transfer Options for the Project.

- Go back to the **Debug Configuration** window and make a new launch group, All cores, that runs all six cores simultaneously (Figure 7).



**Figure 7. Making a Launch Group That Runs All Six Cores.**

- Download and execute the program.

If the program runs successfully, the following message is printed in the **Console** view for each core:

Starting FileIO test over Ethernet.

Written file size is 64.000 (MB)

Additionally, six files will be created into the project directory:

...\TestFileIOSimple\C\_Debug\_8156\_HW\Core1-6\_FileIO\_test.bin.

#### **NOTE**

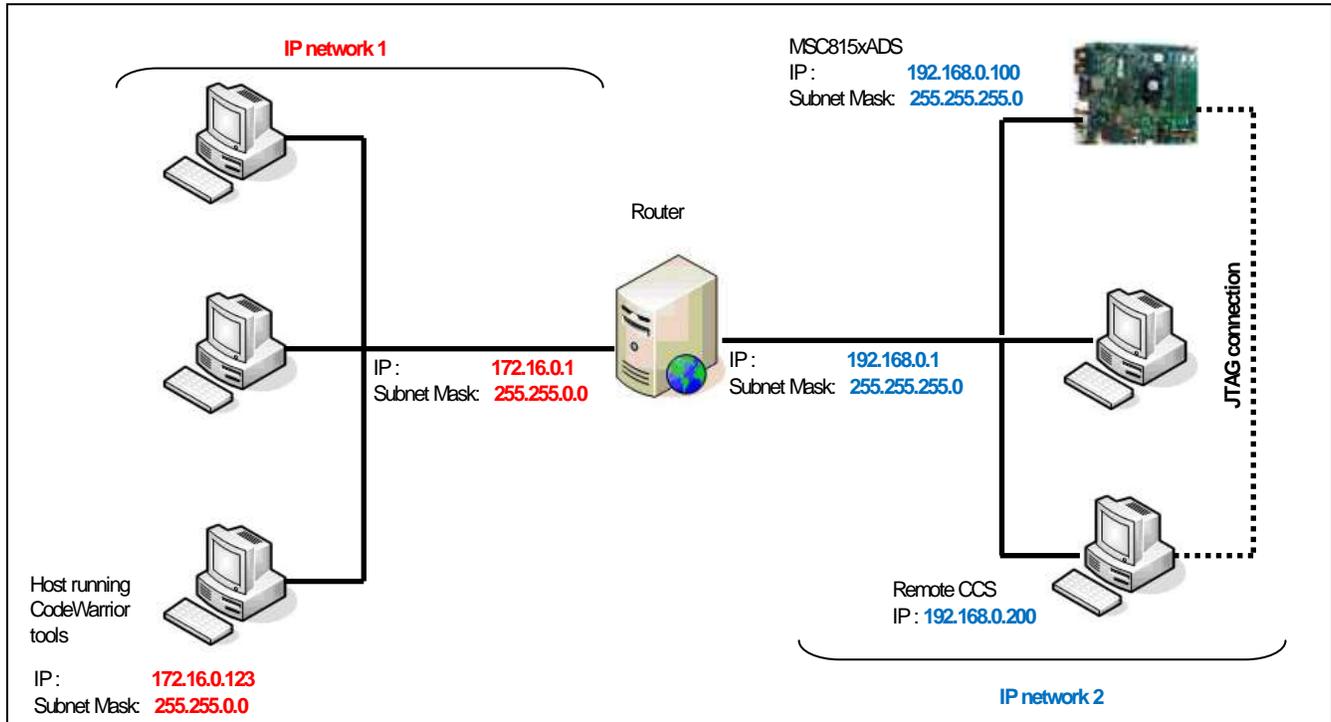
More complex examples related to NetFileIO can be found in:

{CW Install dir}\SC\StarCore\_Support\EthernetIO\

## 4.2 File I/O Over an IP Network

This example demonstrates the NetFileIO capabilities for situations where the MSC815xADS target does not reside on the same network as the host computer that executes the CodeWarrior debugger. This topology is common for testing farms where the targets are often connected in a different network than that of the hosts.

The classic approach for this arrangement is to use a remote CCS server to perform IO operations. Consider the following IP network topology (Figure 8):



**Figure 8. NetFileIO Operating Over an IP Network.**

The figure depicts the case where the host computer executing the CodeWarrior debugger (172.16.0.123) is connected to an MSC815xADS board that resides in a different network (192.168.0.100). The CodeWarrior debugger is connected to the board with a JTAG connection via a remote CCS (192.168.0.200)

In order to execute the NetFileIO protocol, the CodeWarrior debugger needs to know the gateway address (192.168.0.1) and to have an active CCS connection with the MSC815xADS board.

The software setup proceeds as follows:

1. Open the CCS remote server (192.168.0.200) and configure the connection. Assuming there is a USB TAP the configuration, these commands are:

```
>> delete all
>> config cc utap
```

A successful connection can be confirmed by issuing the following commands:

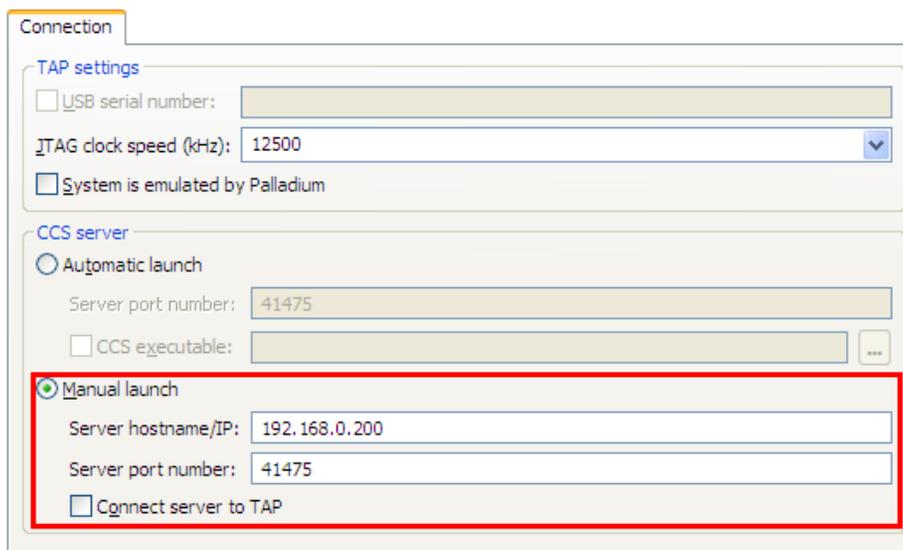
```
>> show cc
```

```
0: USB TAP (JTAG) (utap:) Loader software ver. {1.10}
```

```
Sending code to USB TAP - please wait
```

```
0: USB TAP (JTAG) (utap) CC software ver. {1.3}
```

2. Configure the CodeWarrior debugger (171.16.0.123) to connect remote to MSC815xADS board via the remote CCS (192.168.0.200). Open the **RSE properties** and set the **Connection** tab as below:



**Figure 9. Modifying the CCS Server Settings.**

### NOTE

This example uses a USB TAP connection. A similar setup can be used with an Ethernet TAP connection. For both connections, make sure the port number is set correctly. The example described here uses the default port.

- Now configure the I/O model that the CodeWarrior debugger (171.16.0.123) uses for the network. Open the **RSE properties** and set the options in the **I/O Model** tab as shown in [Figure 10](#).

The screenshot shows the 'I/O Model' configuration window. It has five radio button options: 'None', 'Stop Transfer', 'High-Speed Simultaneous Transfer', 'Serial Transfer', and 'Network Transfer'. The 'Network Transfer' option is selected. Below this, there are several input fields: 'Ethernet controller' is a dropdown menu showing 'Ethernet 0'; 'MAC address' is a text field with a placeholder '\*\*\*\*\_\*\*\_\*\*\_\*\*\_\*\*' and an unchecked checkbox; 'Target IP' is '192.168.0.100'; 'Netmask' is '255.255.255.0'; 'Gateway' is '192.168.0.1' with a checked checkbox; 'Debugger IP' is '172.16.0.123'; and 'UDP port' is '1234' with an unchecked checkbox. A red rectangle highlights the 'Network Transfer' section and its associated fields.

**Figure 10. Setting the I/O Model For the Network That the CodeWarrior Debugger Uses.**

- Download and execute the program. The output should be similar to that of the first example (4.1).

## 5 Appendix

The following code generates the large data files that the examples use to test the NetFileIO connections.

```

#include <stdio.h>
#include <stdlib.h>

#define SIZE_BUF          4096
#define NO_BUFFERS       4 * 4096
unsigned char buff[SIZE_BUF];

int main(void)
{
    FILE* pFile = NULL;
    int i = 0;
    char outFileName[32];

    for (i = 0; i < SIZE_BUF; i++)
        buff[i] = (unsigned char)i;

    printf("Starting FileIO test over Ethernet.\n");
    sprintf(outFileName, "Core%d_FileIO_test.bin", (((*(volatile unsigned
int*)0xffff06028)>>16) & 0xff));

    if (NULL == (pFile = fopen(outFileName, "w+b")))
    {
        printf("FileIO test[error]: cannot open file %s\n", outFileName);
        return 0;
    }

    for (i = 0; i < NO_BUFFERS; i++)
    {
        fwrite(buff, sizeof(unsigned char), SIZE_BUF, pFile);
    }

    fclose(pFile);
    printf("Written file size is %.3f(MB) \n", ((float)NO_BUFFERS * SIZE_BUF) / 1048576.0);

    return 1;
}

```

## 6 Revision History

Table 4 provides a revision history for this application note.

**Table 4. Revision History**

Rev. Number	Date	Substantive Change
0	05/02/11	Initial release.

**How to Reach Us:****Home Page:**

[www.freescale.com](http://www.freescale.com)

**Web Support:**

<http://www.freescale.com/support>

**USA/Europe or Locations Not Listed:**

Freescale Semiconductor  
Technical Information Center, EL516  
2100 East Elliot Road  
Tempe, Arizona 85284  
+1-800-521-6274 or +1-480-768-2130  
[www.freescale.com/support](http://www.freescale.com/support)

**Europe, Middle East, and Africa:**

Freescale Halbleiter Deutschland GmbH  
Technical Information Center  
Schatzbogen 7  
81829 Muenchen, Germany  
+44 1296 380 456 (English)  
+46 8 52200080 (English)  
+49 89 92103 559 (German)  
+33 1 69 35 48 48 (French)  
[www.freescale.com/support](http://www.freescale.com/support)

**Japan:**

Freescale Semiconductor Japan Ltd.  
Headquarters  
ARCO Tower 15F  
1-8-1, Shimo-Meguro, Meguro-ku,  
Tokyo 153-0064, Japan  
0120 191014 or +81 3 5437 9125  
[support.japan@freescale.com](mailto:support.japan@freescale.com)

**Asia/Pacific:**

Freescale Semiconductor China Ltd.  
Exchange Building 23F  
No. 118 Jianguo Road  
Chaoyang District  
Beijing 100022  
China  
+86 010 5879 8000  
[support.asia@freescale.com](mailto:support.asia@freescale.com)

**For Literature Requests Only:**

Freescale Semiconductor Literature Distribution  
Center  
1-800-441-2447 or 303-675-2140  
Fax: 303-675-2150  
[LDCForFreescaleSemiconductor@hibbertgroup.com](mailto:LDCForFreescaleSemiconductor@hibbertgroup.com)

Information in this document is provided solely to enable system and software implementers to use Freescale Semiconductor products. There are no express or implied copyright licenses granted hereunder to design or fabricate any integrated circuits or integrated circuits based on the information in this document.

Freescale Semiconductor reserves the right to make changes without further notice to any products herein. Freescale Semiconductor makes no warranty, representation or guarantee regarding the suitability of its products for any particular purpose, nor does Freescale Semiconductor assume any liability arising out of the application or use of any product or circuit, and specifically disclaims any and all liability, including without limitation consequential or incidental damages. "Typical" parameters that may be provided in Freescale Semiconductor data sheets and/or specifications can and do vary in different applications and actual performance may vary over time. All operating parameters, including "Typicals", must be validated for each customer application by customer's technical experts. Freescale Semiconductor does not convey any license under its patent rights nor the rights of others. Freescale Semiconductor products are not designed, intended, or authorized for use as components in systems intended for surgical implant into the body, or other applications intended to support or sustain life, or for any other application in which the failure of the Freescale Semiconductor product could create a situation where personal injury or death may occur. Should Buyer purchase or use Freescale Semiconductor products for any such unintended or unauthorized application, Buyer shall indemnify and hold Freescale Semiconductor and its officers, employees, subsidiaries, affiliates, and distributors harmless against all claims, costs, damages, and expenses, and reasonable attorney fees arising out of, directly or indirectly, any claim of personal injury or death associated with such unintended or unauthorized use, even if such claim alleges that Freescale Semiconductor was negligent regarding the design or manufacture of the part.

Freescale, the Freescale logo, CodeWarrior, and StarCore are trademarks of Freescale Semiconductor, Inc. Reg. U.S. Pat. & Tm. Off. QUICC Engine is a trademark of Freescale Semiconductor, Inc. All other product or service names are the property of their respective owners.

© 2011 Freescale Semiconductor, Inc.