Application Report
SPRAAQ5A–June 2008

Network Developer's Kit (NDK) Benchmark and Sizing

Srinivas Kurla

ABSTRACT

This document describes a procedure to benchmark the Network Developer's Kit (NDK) using Code Composer Studio™ software and some additional command-line tools. Source code is provided in a companion file to allow the results to be reproduced by the customer, as well as to adapt the benchmark procedure to a custom hardware and platform.

Also provided is a procedure that describes how to use the *Code Generation Tools XML Output Perl Utility Scripts* (CG-XML) package to accurately evaluate the size of a custom application including all the libraries used. In addition, benchmark results are provided for several supported development kits. These report CPU load based on different network packet size and packet transfer rate.

Source code is provided in a companion file to allow you to reproduce the results or adapt the benchmark procedure to your custom hardware and platform.

Project collateral and source code discussed in this application report can be downloaded from the following URL: http://www-s.ti.com/sc/techlit/spraaq5.zip.

Contents

List of Figures

1 Background

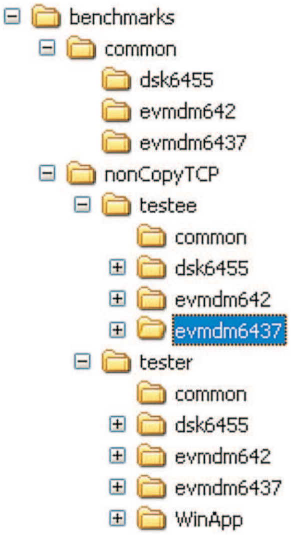
The NDK is a lightweight TCP/IP protocol stack designed to be used in conjunction with the C64x™ family of Texas Instruments DSPs.

This document is restricted to socket communications performance and does not cover specific tests for servers and custom applications. By isolating the NDK, the benchmarks described here allow a more accurate performance measurement of the customer's application software.

2 How it Works

The benchmark consists of two sets of application projects called *Tester* and *Testee*. The Testee project is designed to run on the DSP hardware to measure data throughput and CPU utilization. A Tester project is provided both for the DSP (using the NDK) and for the Windows application, using socket calls to drive the data to the Testee.

After installation, the structure of the two projects can be seen in the following illustration:



```
⊟ benchmarks
   ⊟ common
         dsk6455
         evmdm642
         evmdm6437
   ⊟ nonCopyTCP
      ⊟ testee
            common
         ⊞ dsk6455
         ⊞ evmdm642
         ⊞ evmdm6437
      ⊟ tester
            common
         ⊞ dsk6455
         ⊞ evmdm642
         ⊞ evmdm6437
         ⊞ WinApp
```

The *common* folder contains the source files responsible for hardware initialization of each board. These are shared among all the projects.

The *nonCopyTCP* folder contains the *Tester* and the *Testee* sets of projects. Each set is comprised of projects created for each supported hardware platform and performs different roles.

- Tester contains free running tasks that control the benchmark procedure by starting up the flow of data and switching between the send and receive procedures.
- Testee contains daemon servers that perform measurements of both send and receive throughput, as well calculating the CPU load need to sustain the throughput.

2.1 *Tester Actions*

Tester contains a sendclient() function that runs after the network startup. It performs a TCP connection to the dtask_tcp_receive_srv() server daemon (located in the Testee application) through port 1001. After the connection is accepted, sendclient() sends the frame size to the Testee and subsequently starts sending a stream of data frames at a defined frame rate.

Communication between the Tester and Testee applications is shown in Figure 1.

The process is repeated for a sequence of pre-defined values of frame sizes and frame rates with a fixed number of frames. All these parameters are defined in the common header file nonCopyTCP.h, together with the IP addresses of the two boards and the output format configuration.

```
#ifdef _64P_
#include <bcache.h>
//  Setup Cache sizes for C64P applications, see DSP/BIOS BCACHE module
//  for more information
BCACHE_Size bcacheSize = { BCACHE_L1_32K,  BCACHE_L1_32K, BCACHE_L2_256K};
#endif
#ifdef  CHIP_DM642
#include <csl.h>
#include <csl_cache.h>
//  Setup Cache size of DM642 device using Chip Support Library (CSL)
CACHE_L2Mode l2CacheSize =  CACHE_128KCACHE;
#endif

#define TESTER_IPADDR_STRING "146.252.161.147"  // Tester IP address is only
                                                // used on DSP tester application
#define TESTEE_IPADDR_STRING "146.252.161.171"

//#define BENCH_CSV                             // Output data in CSV format

#define BENCH_FRAMES     5000                   // Number of frames

int frameSize[] =  {8192, 16384, 32768, 65536}; // Array of frame Size

int frameRate[] =  {0, 10, 30};                 // Array of frame rate in milliseconds (ms)
                                                // resolution; 0 equals Maximum throughput
```

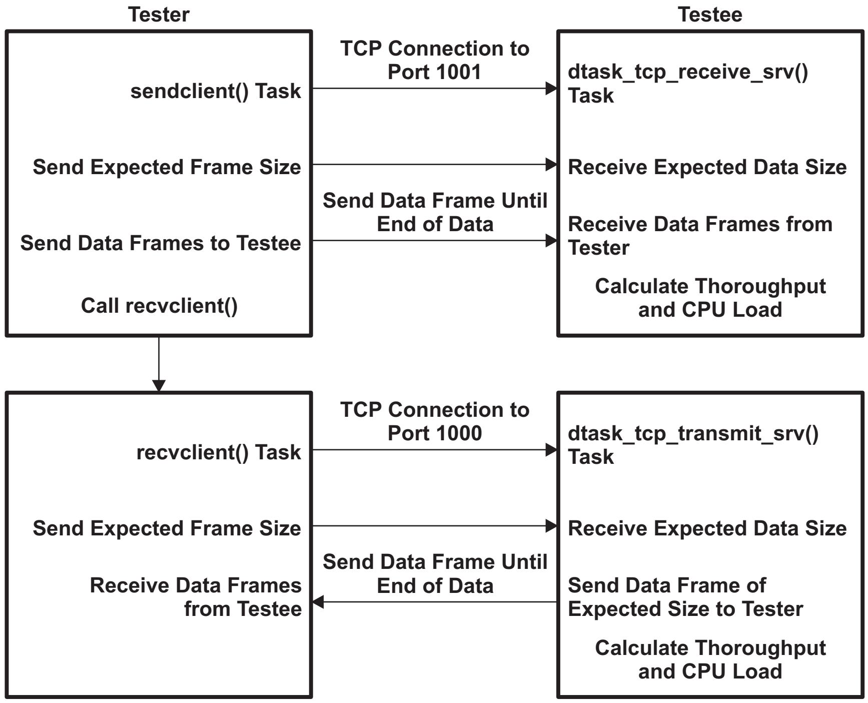

Tester
Testee
sendclient() Task
TCP Connection to Port 1001
dtask_tcp_receive_srv() Task
Send Expected Frame Size
Receive Expected Data Size
Send Data Frame Until End of Data
Send Data Frames to Testee
Receive Data Frames from Tester
Call recvclient()
Calculate Thoroughput and CPU Load
recvclient() Task
TCP Connection to Port 1000
dtask_tcp_transmit_srv() Task
Send Expected Frame Size
Receive Expected Data Size
Send Data Frame Until End of Data
Receive Data Frames from Testee
Send Data Frame of Expected Size to Tester
Calculate Thoroughput and CPU Load

Figure 1. Tester-Testee Communication

The cache size configuration is used by the Testee which is contained in the same file; by default these are set to the maximum sizes for L1 and L2 cache. The DSP/BIOS™ BCACHE module is used to setup the cache for C64x+™ devices. For C64x devices (e.g. DM642), the Chip Support Library (CSL) is used to setup the cache.

Note: The TESTER_IPADDR_STRING is only used on a DSP-side Tester application. This setting is ignored for the Windows Tester application.

After sending all the data, the sendclient() function creates a receiving task for the recvclient() function and terminates. This new task also performs a TCP connection to a server daemon. However, this time it connects to the dtask_tcp_transmit_srv() through port 1000. After the connection is accepted, it again sends the frame size to the Testee and prepares itself to receive the data stream. After receiving all data frames, it terminates its task.

2.2 Testee Actions

Testee contains the two daemon servers that are spawned whenever data arrives at TCP ports 1000 and 1001, which are completely controlled by Tester. At the end of each daemon server, the Testee application calculates both the measured throughput and the CPU load (using the THRLOAD API calls) and displays the results in a Code Composer Studio stdout window.

Output being displayed can be output in comma-separated value (CSV) format for post-processing, by defining BENCH_CSV in the common header file. By default, the #define BENCH_CSV line is commented out.

2.3 Considerations

To minimize the influences of other concurrent applications and routines, all services and servers are disabled with the exception of the DHCP service. Although by default the benchmark programs use static IP addresses, the DHCP service is enabled in order to provide support needed by your network configuration. However, if DHCP is used, its effect on the benchmark is negligible unless there are major changes in network structure.

Also, to minimize the effect of CPU activity and to reach the maximum possible throughput without losing reliability, the chosen communications protocol is non-copy (or zero-copy) TCP. This disables any CPU-intensive memory copy operations in the receiving sockets. (NDK does not implement zero-copy transfers in send sockets.)

Despite the fact that cache sizes can be adjusted in the Testee side, the Tester side is set to its maximum available cache size and optimal memory configuration to avoid any influences in the measurements.

3 Requirements and Installation

In order to execute the benchmarks and accurately reproduce the results obtained, the procedure and equipment described in the following subsections was used.

3.1 Equipment Required

To avoid any external influences and allow reproducibility of the results, the setup to run NDK benchmarks consists of:

- Two boards of the same type (DSK6455, evaluation module (EVM) DM642, and so on), one running the Tester and other running the Testee application.
- An Ethernet crossover patch cable (Belkin A3X126-03-YLM-M or similar) or an Ethernet switch (Linksys EZXS55W or similar).
- Two PCs running Code Composer Studio to build, load, and run the Tester and Testee applications. A single PC can be used with an IEEE Standard 1149.1-1990, IEEE Standard Test Access Port and Boundary-Scan Architecture (JTAG) Splitter/Expander (Spectrum Digital #701204 or similar) to load and run the applications on two DSP boards.

Note: A Windows® PC Tester application is provided for single-board users. However, using this PC Windows application may not produce ideal performance values, since Windows isn't a real-time operating system (OS). Single-board users need only one PC to run the Windows Tester application and to use Code Composer Studio to build, load, and run the DSP Testee application.

3.2 Software Requirements

- Code Composer Studio v 3.3 or greater
- NDK v1.92 or greater installed anywhere on your system. (The examples assume it is installed at C:\CCStudio_v3.3\ndk_1_92) available at: https://www-a.ti.com/downloads/sds_support/targetcontent/NDK/index.html
- Board support library (BSL) and network support drivers package for the board you are using.

Note: If you are using a pre-production board, you may have to make changes in the Code Composer Studio project settings to use different libraries. See Section 4.2.1 for additional details.

- Application example provided along with this application note. This can be downloaded from http://www-s.ti.com/sc/techlit/spraaq5.zip.
- Thrload module (which is part of the RF Modules package) installed anywhere on your system. (The examples assume it is installed at C:\CCStudio_v3.3\rfmodules.) This module allows you to obtain task execution time and CPU load information at run-time.
- CG-XML scripts package. (The examples assume it is installed at C:\CCStudio_v3.3\cg_xml.) This package contains a collection of Perl scripts that do post processing on linker and OFD XML output generated by CodeGen Tools, which include footprint data, call graphs, etc.

The two last modules are available, respectively, by following the *RF Modules* and *Code Generation Tools XML Output Perl Utility Scripts* links from the following webpage (requires my.ti.com registration): https://www-a.ti.com/downloads/sds_support/applications_packages/index.htm.

For additional installation instructions and usage information please refer to the package documentation.

To re-build the Microsoft® Windows Tester application (for single-board users), Microsoft Visual Studio C++ 2005 Express Edition must be installed on your computer. Download and install the following components (available for free) from the Microsoft Web Site (http://msdn.microsoft.com/vstudio/express/):

- Microsoft Visual C++ 2005 Express Edition
- Microsoft Visual C++ 2005 Express Service Pack 1
- Microsoft Platform SDK for Visual C++ 2005 Express (Microsoft Windows Server 2003 R2 Platform SDK)

Make sure to set the environment variables required for Visual Studio C++.

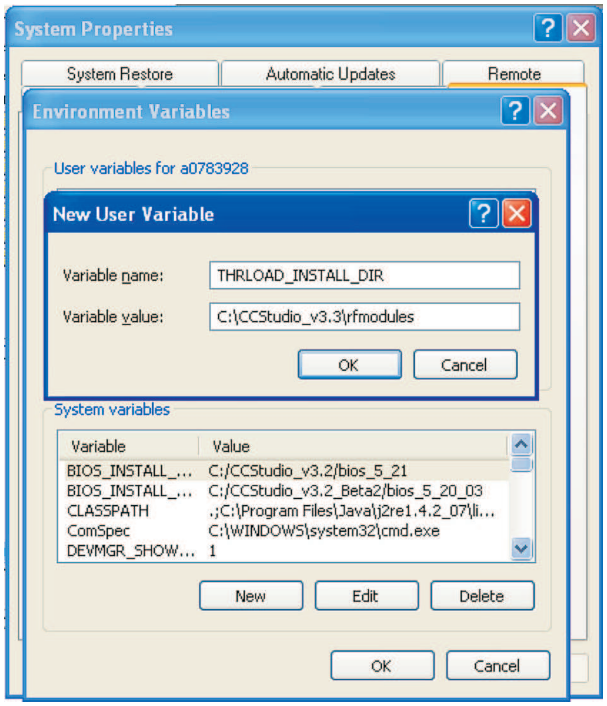
Run the following batch script at the DOS prompt before building the examples:

```
C:\Program Files\Microsoft Visual Studio 8\Common7\Tools\vsvars32.bat
```

3.3 Installing the Software

After installing the RF Modules and CG-XML Scripts software, you need to create an environment variable called THRLOAD_INSTALL_DIR pointing to the RF modules installation folder. The Code Composer Studio applications provided rely on this environment variable to find the appropriate files to re-build the applications.

For example, if you installed this package at C:\CCStudio_v3.3\rfmodules, then you should create a new environment variable and set its value using the Windows Control Panel. (From the Windows Start menu, choose Control Panel. Double-click the System icon. In the Advanced tab, click *Environment Variables*. In the System variables box, click New. (See the following figure.)



Variable name: THRLOAD_INSTALL_DIR

Variable value: C:\CCStudio_v3.3\rfmodules

Variable	Value
BIOS_INSTALL_...	C:/CCStudio_v3.2/bios_5_21
BIOS_INSTALL_...	C:/CCStudio_v3.2_Beta2/bios_5_20_03
CLASSPATH	.;C:\Program Files\Java\j2re1.4.2_07\li...
ComSpec	C:\WINDOWS\system32\cmd.exe
DEVMGR_SHOW...	1

Also, make sure the environment variable NDK_INSTALL_DIR is set and correctly points to your NDK installation. The procedure to verify this is very similar as previously mentioned.

To install the benchmark software package, simply unzip the file spraaq5.zip to the folder where NDK files are located. For example, %NDK_INSTALL_DIR%\packages\. This creates a benchmark folder in the %NDK_INSTALL_DIR%\packages\ti\ndk directory.

4 Performance Data

This section describes how to build, load, and run the examples provided with this application note to gather performance data. To obtain the best results, these examples should be run on two DSPs connected via a crossover cable. A Windows application (Tester) is also provided for cases where only one DSP board is available.

Note: Performance data gathered using the Windows PC application may not be ideal, since Windows is not a real-time OS and the application depends on various other factors running on the system. Therefore, results gathered using the PC are only for informational purposes.

4.1 Procedure to Execute the Benchmarks

1. Connect both DSP boards to either an Ethernet switch or an Ethernet crossover cable. If you are using a single DSP board, connect the Windows PC to the Ethernet switch that the DSP board is connected to.
   Optionally, you can connect the Windows PC to the DSP directly using the Ethernet crossover cable and configuring a static TCP/IP address (in the Network Connections under the Windows Control Panel) for the DSP board to communicate to via TCP. The Windows PC needs to be configured to obtain an IP address to be able to communicate to the DSP board via TCP.
2. Open Code Composer Studio.
3. First, open the project nonCopyTCPTestee.pjt. Remember that this is the project that performs the benchmark and displays the results. If you are using a single DSP board, follow the procedure in Section 4.1.1 on how to build and run the Tester application for Windows.

4. Build (F7), load, and run the nonCopyTCPTestee.out file located in the bin directory of the project file. You should see the following information displayed on the Code Composer Studio stdout window:

   ```
   Non-Copy TCP Testee Benchmark Using MAC Address: 00-0e-99-02-90-7a Network Added:
   If-1:146.252.161.171 Link Status: 100Mb/s Full Duplex on PHY 0
   ```

5. Second, open the project nonCopyTCPTester.pjt. Remember that this is the project that will act as the data server.
6. 6. Build (F7), load, and run the nonCopyTCPTester.out file located in the bin directory of the project file. You should see the following information displayed on the Code Composer Studio stdout window:

   ```
   Non Copy TCP Tester Benchmark Using MAC Address: 00-0e-99-02-71-57 Network Added:
   If-1:146.252.161.147 Sending Task started Link Status: 100Mb/s Full Duplex on PHY 1
   ```

7. As the benchmark runs, you start to see the actions from the Testee and the Tester based on the settings in the nonCopyTCP.h file. The stdout window of the two projects shows the following:

Testee results (informational only):

```
Non-Copy TCP Testee Benchmark
Using MAC Address: 00-0e-99-02-90-7a
Network Added: If-1:146.252.161.171
Link Status: 100Mb/s Full Duplex on PHY 0
Link Status: No Link on PHY 0
Link Status: 100Mb/s Full Duplex on PHY 0
DSP Operating at 594 MHz with 256k L2 Cache

Received packets of 8192 bytes for a total of 40960000 bytes
for a throughput of 93992 kb/s at a CPU load of 25%

Received packets of 8192 bytes for a total of 40960000 bytes
for a throughput of 6552 kb/s at a CPU load of 2%

...

Received packets of 65536 bytes for a total of 327680000 bytes
for a throughput of 37136 kb/s at a CPU load of 11%

Received packets of 65536 bytes for a total of 327680000 bytes
for a throughput of 15384 kb/s at a CPU load of 5%

Transmitted 5000 frames of 8192 bytes for a total of 40960000
at a rate of 93170 kb/s with a CPU load of 36%

Transmitted 5000 frames of 8192 bytes for a total of 40960000
at a rate of 6556 kb/s with a CPU load of 3%

...

Transmitted 5000 frames of 65536 bytes for a total of 327680000
at a rate of 34473 kb/s with a CPU load of 15%

Transmitted 5000 frames of 65536 bytes for a total of 327680000
at a rate of 14889 kb/s with a CPU load of 7%
```

Note: Output for Testee may vary, especially if the Windows application is being used as a Tester.

Tester results (informational only):

```
Non Copy TCP Tester Benchmark
Using MAC Address: 00-0e-99-02-70-04
Network Added: If-1:146.252.161.147
    Sending Task started
    Sending 5000 frames of 8192 at maximum frame rate
Link Status: 100Mb/s Full Duplex on PHY 1
    Sending 5000 frames of 8192 at 10 ms frame rate
    Sending 5000 frames of 8192 at 30 ms frame rate
...
    Sending 5000 frames of 65536 at 10 ms frame rate
    Sending 5000 frames of 65536 at 30 ms frame rate
    Send Task terminated
    Receive Task started
    Requesting 8192 bytes frames at maximum frame rate...
     ...received 5000 frames
    Requesting 8192 bytes frames at 10 ms frame rate...
     ...received 5000 frames
...
    Requesting 65536 bytes frames at 10 ms frame rate...
     ...received 5000 frames
    Requesting 65536 bytes frames at 30 ms frame rate...
     ...received 5000 frames
    Receive Task terminated
```

You can export the data by right-clicking on the Code Composer Studio stdout window and selecting *Save Output to File* and selecting the *Enable file output* option or by simply copying and pasting the information into a text file.

4.1.1 Building the Windows Application

The example provided with this application note contains a Windows application to run on a single-DSP board configuration.

See Section 3 for the software and hardware needed to re-build the Windows Tester application. Also, make sure the environment variables required by Visual Studio C++ have been set. The ti/ndk/benchmarks/nonCopyTCP/tester/winApp directory contains the source file tester.c, needed to create the application.

Open a Windows command prompt window in the winApp directory. At the command line, type:

```
cl tester.c wsock32.lib
```

Note: To access the Microsoft Visual C++ compiler and associated libraries, the environment variable must be set correctly. Refer to the Visual Studio Installation Guide for further information on setup. The winsock32 libraries are provided as part of the Windows Server SDK Platform package.

Compiling the file tester.c generates a Windows executable (tester.exe), which can be executed in conjunction with the DSP-side Testee application. Be sure the DSP-side Testee application is running before executing the Windows application. Also be sure that both the Windows PC and the board are connected via an Ethernet crossover cable or an Ethernet switch.

4.2 Using the Throughput and CPU Load Benchmarks on Custom Hardware

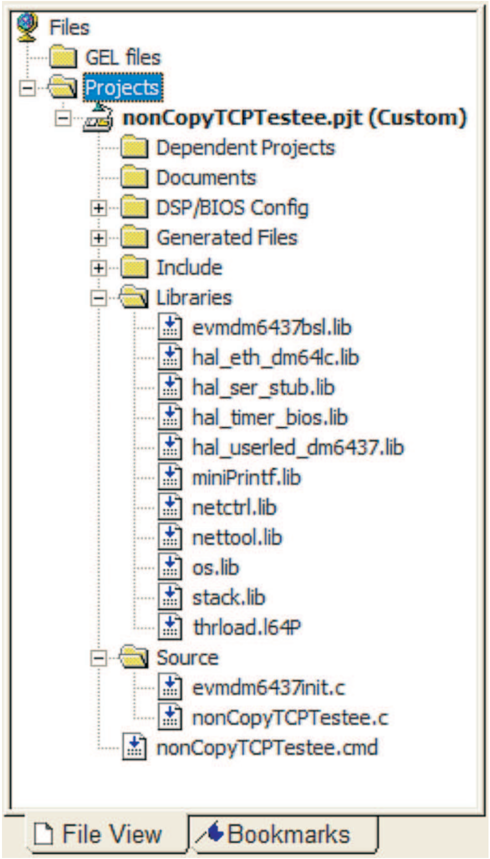
To use the performance benchmarks on custom hardware or on a different development board (either pre-production or future boards), either integrate the new hardware abstraction layer (HAL) and BSL into the projects supplied, or add the source files to an existing project. Both methods are shown in the next sections.

4.2.1 Integrating Other HAL or BSL Files in Supplied Projects

First, download and install the BSL, if required. For pre-production and future boards, also download the Network Support Package. If you have a custom board and have modified the NDK HAL drivers, make sure you have their exact location in your system.

It is a good idea to make a backup copy of the benchmarks folder before making any changes.

Open the project (either Testee or Tester) that corresponds to the DSP family of your hardware board (C64x or C64x+) and locate the HAL library files in the project tree, as shown in the following figure.



```
Files
  GEL files
  Projects
    nonCopyTCPTestee.pjt (Custom)
      Dependent Projects
      Documents
      DSP/BIOS Config
      Generated Files
      Include
      Libraries
        evmdm6437bsl.lib
        hal_eth_dm64lc.lib
        hal_ser_stub.lib
        hal_timer_bios.lib
        hal_userled_dm6437.lib
        miniPrintf.lib
        netctrl.lib
        nettool.lib
        os.lib
        stack.lib
        thrload.l64P
      Source
        evmdm6437init.c
        nonCopyTCPTestee.c
      nonCopyTCPTestee.cmd
```

Remove the NDK HAL files (starting with hal_) and replace them with the new ones. To add the new libraries, right-click on the project name and select *Add Files to Project*. Go to the folder where the HAL libraries are located and add them to your project.

If you are using a development board from a different vendor, replace the BSL file with one provided by the board manufacturer. The BSL file for the DM6437 EVM is evmdm6437bsl.lib in the previous picture. You may need to remove this file completely for proper operation when using a custom hardware board.

The hardware initialization source file (evmdm6437init.c in the previous picture) may need to be removed or modified, since it may contain calls to BSL functions. For details, please refer to the BSL documentation.

4.2.2 Using Benchmark Files in a Custom Application

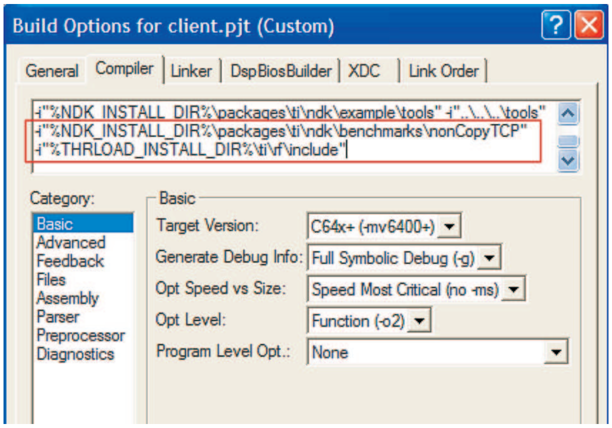
To include the performance benchmarks in a custom application, two steps are required:

1. Add two files to your existing project:
   - The main benchmark source file: nonCopyTCPTestee.c or nonCopyTCPTester.c.
   - The thrload library file: thrload.l64 for C64x DSPs or thrload.l64P for C64x+ DSPs, located in the RF modules installation folder %THRLOAD_INSTALL_DIR%\ti\rf\lib.

2. Copy the header files nonCopyTCP.h and thrload.h to the project folder, or include their folders in the compiler search path as shown here. (Choose Project -> Build Options and go to the Compiler tab to find the Directories section.)

```
-I"%NDK_INSTALL_DIR%\packages\ti\ndk\benchmarks\nonCopyTCP"
-I"%THRLOAD_INSTALL_DIR%\ti\rf\include"
```

Your project should look something like this:



```
-i"%NDK_INSTALL_DIR%\packages\ti\ndk\example\tools" -i"..\..\..\tools"
-i"%NDK_INSTALL_DIR%\packages\ti\ndk\benchmarks\nonCopyTCP"
-i"%THRLOAD_INSTALL_DIR%\ti\rf\include"
```

Basic

Target Version: C64x+ (-mv6400+)

Generate Debug Info: Full Symbolic Debug (-g)

Opt Speed vs Size: Speed Most Critical (no -ms)

Opt Level: Function (-o2)

Program Level Opt.: None

Please keep in mind that the benchmark source file (nonCopyTCPTestee.c or nonCopyTCPTester.c) contains all of the network initialization routines (similar to the client.c source file) and has the DHCP service enabled in case your network requires it. If you are including these files in your project, you will need to remove the network initialization from your application files and manually add any additional services required by your application.

If you are using one of the supplied benchmarking projects, it is a good idea to start with one that most closely matches the DSP family of your hardware board (C64x or C64x+). You should also make sure the memory configurations are correctly set in the DSP/BIOS configuration file. For example, if you are using TMS320C6454, the most suitable benchmark project is the one created for DSK6455, but you must adjust the IRAM memory size accordingly.

4.3 *Performance Considerations*

There are some cache and buffer size considerations to maximize performance using the NDK on your custom application.

4.3.1 Cache and Cache Size

Cache has many considerations, but specific to the NDK, you'll need to ensure that the stack buffer packets are aligned on a cache line boundary. An example is provided in the linker command file (*.cmd) provided with the benchmark examples of this application note, where the buffers are aligned on an L2 cache line (128-byte) boundary.

```
SECTIONS {

    .far:NDK_PACKETMEM {} align = 128 > DDR2
    .far:NDK_MMBUFFER {} align = 128 > DDR2
    .far:NDK_OBJMEM {} align = 128 > DDR2
}
```

Allocating large cache sizes can significantly improve performance, so try to maximize cache size whenever possible. For more information regarding Cache on TI devices, see the *TMS320C64x+ DSP Cache User's Guide* (SPRU862) or *TMS320C6000 DSP Cache User's Guide* (SPRU656) for your particular device. Refer to section 6 for example-specific data.

4.3.2 Network Buffer Sizes

The buffer limit for TCP and UDP packet is configurable, and should be maximized. Both the Tester and Testee benchmark example applications illustrate the network configuration function on the application.

```
    // TCP Transmit buffer size
    rc = 8192;
    CfgAddEntry( hCfg, CFGTAG_IP, CFGITEM_IP_SOCKTCPTXBUF,
                 CFG_ADDMODE_UNIQUE, sizeof(uint), (UINT8 *)&rc, 0 );

    // TCP Receive buffer size (copy mode)
    rc = 8192;
    CfgAddEntry( hCfg, CFGTAG_IP, CFGITEM_IP_SOCKTCPRXBUF,
                 CFG_ADDMODE_UNIQUE, sizeof(uint), (UINT8 *)&rc, 0 );

    // TCP Receive limit (non-copy mode)
    rc = 8192;
    CfgAddEntry( hCfg, CFGTAG_IP, CFGITEM_IP_SOCKTCPRXLIMIT,
                 CFG_ADDMODE_UNIQUE, sizeof(uint), (UINT8 *)&rc, 0 );

    // UDP Receive limit
    rc = 8192;
    CfgAddEntry( hCfg, CFGTAG_IP, CFGITEM_IP_SOCKUDPRXLIMIT,
                 CFG_ADDMODE_UNIQUE, sizeof(uint), (UINT8 *)&rc, 0 );
```

Though it may not always be possible due to memory limitations, setting the limit size equal to the maximum expected packet size minimizes the work required by the network stack to reassemble the packets, and allows for better performance. Make sure the limit size includes the size of the network packet headers.

5 Sizing

This section describes how to gather size data for applications and libraries provided in the NDK. Since the NDK is comprised of several library files, gathering size information from them is a useful way to gather more detailed information about memory usage.

5.1 Gathering Size Data From Applications and Libraries

To perform memory size benchmarks in your system, you must have installed the Code Generation Tools XML Output Perl Utility Scripts (CG-XML) and a Perl interpreter. This package contains several Perl utilities that will be used to gather size data from libraries (.LIB) or output (.OUT) files. Please refer to the CG-XML documentation for more details on this tool.

For example, to gather size information from the NDK library stack.lib compiled for the C64x+ family, open a DOS command window and change to the library folder as follows:

```
cd C:\CCStudio_v3.3\ndk_1_92\packages\ti\ndk\lib\c64plus
```

In sequence, use the ofd6x utility to generate an XML debug information file from the library and concatenate it to the sectti.pl utility from the CG-XML package by issuing the following command:

```
C:\CCStudio_v3.3\C6000\cgtools\bin\ofd6x.exe -x stack.lib | perl
C:\CCStudio_v3.3\cg_xml\ofd\seccti.pl
```

You should see output similar to the following:

```
Reading from stdin ...
====================================================================
REPORT FOR LIBRARY: stack.lib
====================================================================


************************************************************
REPORT FOR FILE: raw.o64P
************************************************************
                          Name : Size (dec)  Size (hex)  Type
------------------------------ : ----------  ----------  ----
                         .text :        640  0x00000280  CODE
                          .far :         16  0x00000010  UDATA
                        .const :         40  0x00000028  DATA

************************************************************
REPORT FOR FILE: ipin.o64P
************************************************************
                          Name : Size (dec)  Size (hex)  Type
------------------------------ : ----------  ----------  ----
                         .text :       1568  0x00000620  CODE
                          .bss :          8  0x00000008  UDATA
                        .cinit :         28  0x0000001c  DATA
                        .const :         72  0x00000048  DATA
. . .

************************************************************
REPORT FOR FILE: ppp.o64P
************************************************************
                          Name : Size (dec)  Size (hex)  Type
------------------------------ : ----------  ----------  ----
                         .text :       2432  0x00000980  CODE
                       .switch :         24  0x00000018  DATA
                        .const :        271  0x0000010f  DATA

------------------------------------------------------------
Totals by section type
------------------------------------------------------------
  Uninitialized Data :       1444  0x000005a4
    Initialized Data :       4717  0x0000126d
                Code :      68800  0x00010cc0
```

In a similar manner, gathering size information from your entire application is a useful way to gain an overall idea about memory usage.

As another example, to gather size information from the NDK example project client.pjt compiled for the DSK6455, open a DOS command window and move to the Project\bin folder:

```
cd C:\CCStudio_v3.3\ndk_1_92\packages\ti\ndk\example\network\client\dsk6455\bin
```

Similarly, use the ofd6x utility to generate an XML debug information file from the output file and concatenate it to the sectti.pl utility from the CG-XML package by issuing the following command:

```
C:\CCStudio_v3.3\C6000\cgtools\bin\ofd6x.exe -x client.out | perl
C:\CCStudio_v3.3\cg_xml\ofd\seccti.pl
```

You should see output similar to the following:

```
Reading from stdin ...
************************************************************
REPORT FOR FILE: client.out
************************************************************
                Name : Size (dec)  Size (hex)  Type   Load Addr   Run Addr
-------------------- : ----------  ----------  ----   ----------  ----------
                .clk :          8  0x00000008  UDATA  0x00862738  0x00862738
                .prd :         32  0x00000020  UDATA  0x008c44fc  0x008c44fc
                .swi :         88  0x00000058  UDATA  0x008c4458  0x008c4458
                .tsk :        208  0x000000d0  UDATA  0x008bef20  0x008bef20
                .idl :         32  0x00000020  UDATA  0x008c44dc  0x008c44dc
                .bss :       1391  0x0000056f  UDATA  0x008c2be0  0x008c2be0
            .hwi_vec :        512  0x00000200  CODE   0x008c2400  0x008c2400
                .far :     403238  0x00062726  UDATA  0x00800000  0x00800000
                .mem :          4  0x00000004  UDATA  0x008be25c  0x008be25c
               .bios :      21408  0x000053a0  CODE   0x008b42e0  0x008b42e0
. . .
              .trace :        512  0x00000200  N/A    0x008c3da8  0x008c3da8
              .stack :       5120  0x00001400  UDATA  0x008c0000  0x008c0000
                .hst :         44  0x0000002c  UDATA  0x008c44b0  0x008c44b0
                .log :         48  0x00000030  DATA   0x008c3150  0x008c3150
                .pip :        200  0x000000c8  UDATA  0x008c4320  0x008c4320
                .sts :        112  0x00000070  DATA   0x008c43e8  0x008c43e8
          .IRAM$heap :     131072  0x00020000  N/A    0x0088e200  0x0088e200

------------------------------------------------------------
Totals by section type
------------------------------------------------------------
  Uninitialized Data :     421681  0x00066f31
    Initialized Data :      45267  0x0000b0d3
                Code :     205568  0x00032300
```

Note that the sectti.pl utility works for both libraries and output files.

A companion batch file is provided to automatically perform sizing measurements of all the NDK libraries and the HAL libraries for a chosen hardware platform. The output is redirected to a series of text files that contains the size information for each library. In order to use it, issue the following command:

```
C:\CCStudio_v3.3\ndk_1_92\packages\ti\ndk\benchmarks\sizing > codesize <platform> <cg_xml_path>
<ccs_path>
```

Parameters are as follows:

- <platform> can be any of the currently supported platforms: EVMDM642, DSK6455 or EVMDM6437.
- <cg_xml_path> is the path to the installation directory of the CG XML package. Its usage is optional and must be set if it is installed in a directory other than the default C:\CCStudio_v3.3\cg_xml.
- <ccs_path> is the installation directory of Code Composer Studio. Its usage is optional and must be set if it is installed in a directory other than the default C:\CCStudio_v3.3.

Note: Due to limitations of the command-line environment, avoid installing tools in folders with spaces, like C:\Program Files\cg_xml.

Help is available if you run the batch file with no parameters.

6 Benchmark and Sizing Results

Sizing and performance benchmark results are provided in the companion spraaq5.zip file for convenience. The data used to generate the tables and graphs in the subsections that follow can be found in the ti/ndk/benchmark/csv directory. The ti/ndk/benchmark/sizing/<board> directories include the NDK sizing information for the different libraries.

The benchmark setup consisted of two boards of each platform connected through an Ethernet crossover cable. Code Composer Studio 3.3.34, DSP/BIOS 5.31.02, and TI Code Generation Tools 6.0.8 were used. The performance results used the NDK v1.94 release. The performance data gathering used various TCP data packet sizes (8 KB, 16 KB, 32 KB and 64 KB) at various data rates (maximum, 10 ms, and 30 ms per packet transfer rates). All data and code was placed in external memory for all platforms.

Information for the following platforms and configuration is provided:

- DM642 Evaluation Module (EVM) rev 3. 720 MHz using L2 cache sizes of 256 KB, 128 KB and 64 KB
- TMS320C6455 DSK. 1 GHz using L2 cache sizes of 256 KB, 128 KB, 64 KB, and 32 KB
- TMS320DM6437 Evaluation Module (EVM). 594 MHz using L2 cache sizes of 128 KB, 64 KB, and 32 KB

The following sections show throughput vs. CPU load graphically for each of the platforms. For the complete set of data values, see the spraaq5.zip file.

To obtain more accurate and up-to-date information, you are encouraged to build and run the provided application on a TI DSP device.

The following sections evaluate Benchmark performance results of 194_NDK on DM642, DSK6455, DM6437 boards. Network Interface Management Unit (NIMU) is introduced in 194_NDK. Results show performance of 194_NDK with NIMU architecture and with LL architecture. For complete information regarding NIMU architecture, see the *TMS320C6000 Network Developer's Kit (NDK) Software User's Guide* (SPRU523) and the *TMS320C6000 Network Developer's Kit (NDK) Software Programmer's User's Guide* (SPRU524).

6.1 TMS320DM642 EVM Performance

6.1.1 With NIMU Architecture

The following table shows maximum receive and transmit throughput and CPU load data for a DM642 EVM running at 720 MHz using a frame size of 8192 and various L2 cache settings.

L2 Cache Size	Operation	Measured Throughput (Kbits/sec)	CPU Load (±1%)
64K	Receive	94,595	62%
128K	Receive	94,595	28%
256K	Receive	94,595	28%
64K	Transmit	93.971	50%
128K	Transmit	93,998	32%
256K	Transmit	93,891	27%

Figure 2 shows TCP transmit and receive throughput vs. CPU load for the TMS320DM642 EVM running at 720 MHz with 64 KB, 128 KB and 256 KB L2 cache, using an 8 KB frame size.

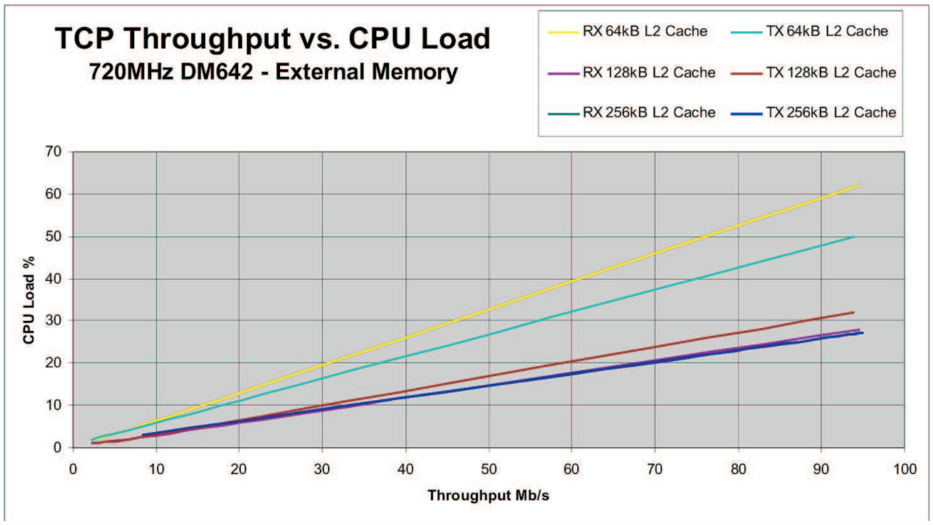


Figure 2. 720MHz TMS320DM642 TCP Throughput vs. CPU Load for Various L2 Cache Sizes

Figure 3 shows TCP transmit and receive throughput vs. CPU load for the TMS320DM642 running at 720 MHz with 256 KB L2 cache at varying frame sizes.

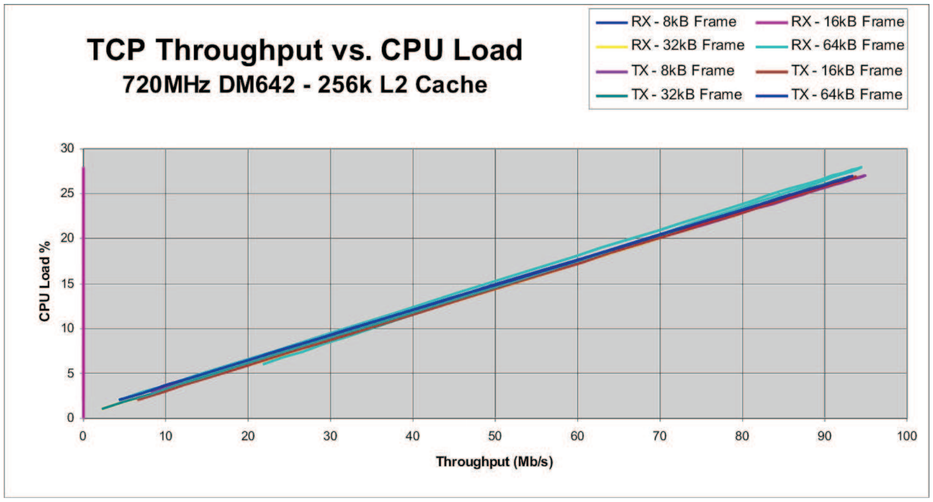


Figure 3. 720 MHz TMS320DM642 TCP Throughput vs. CPU Load for Various Frame Sizes

6.1.2 With LL Architecture

The following table shows maximum receive and transmit throughput and CPU load data for a DM642 EVM running at 720 MHz using a frame size of 8192 and various L2 cache settings.

L2 Cache Size	Operation	Measured Throughput (Kbits/sec)	CPU Load (±1%)
64K	Receive	94,595	71%
128K	Receive	94,595	30%
256K	Receive	93,756	29%
64K	Transmit	93.864	67%
128K	Transmit	93,783	32%
256K	Transmit	93,998	36%

Figure 4 shows TCP transmit and receive throughput vs. CPU load for the TMS320DM642 EVM running at 720 MHz with 64 KB, 128 KB and 256 KB L2 cache, using an 8 KB frame size.

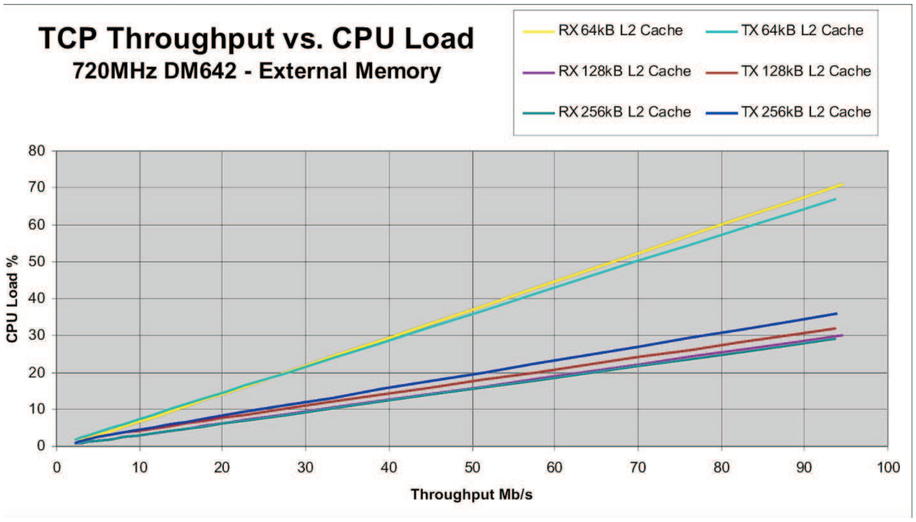


Figure 4. 720 MHz TMS320DM642 TCP Throughput vs. CPU Load for Various L2 Cache Sizes

Figure 5 shows TCP transmit and receive throughput vs. CPU load for the TMS320DM642 running at 720 MHz with 256 KB L2 cache at varying frame sizes

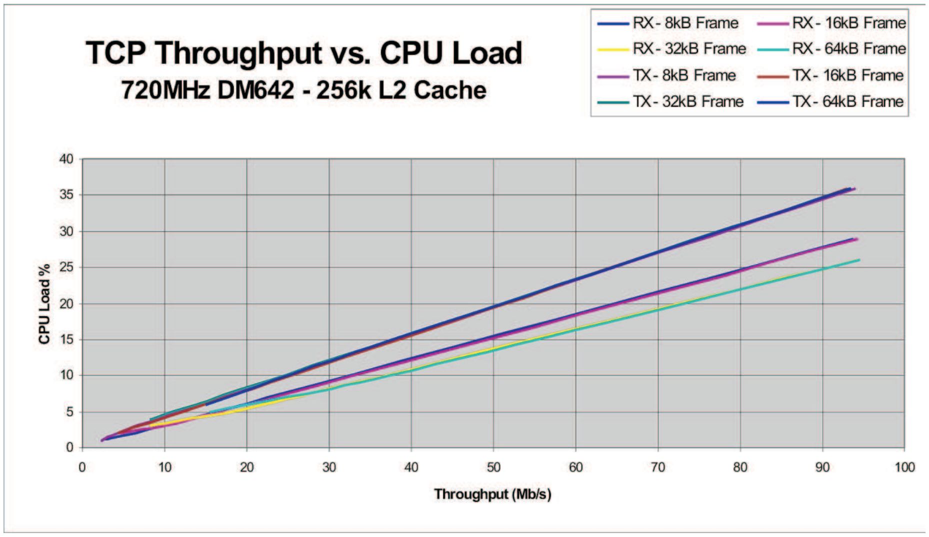


Figure 5. 720 MHz TMS320DM642 TCP Throughput vs. CPU Load for Various Frame Sizes

6.2 TMS320C6455 DSK Performance

6.2.1 With NIMU Architecture

The following table shows maximum receive and transmit throughput and CPU load data for a DSK6455 running at 1 GHz using a frame size of 8192 and various L2 cache settings.

L2 Cache Size	Operation	Measured Throughput (Kbits/sec)	CPU Load (±1%)
32K	Receive	94,595	37%
64K	Receive	94,595	24%
128K	Receive	94,595	17%
256K	Receive	93,703	18%
32K	Transmit	94,650	30%
64K	Transmit	94,677	22%
128K	Transmit	94,677	17%
256K	Transmit	94,705	14%

Figure 6 shows TCP transmit and receive throughput vs. CPU load for the TMS320C6455 running at 1 GHz with 32 KB, 64 KB, 128 KB and 256 KB L2 cache, using an 8 KB frame size.

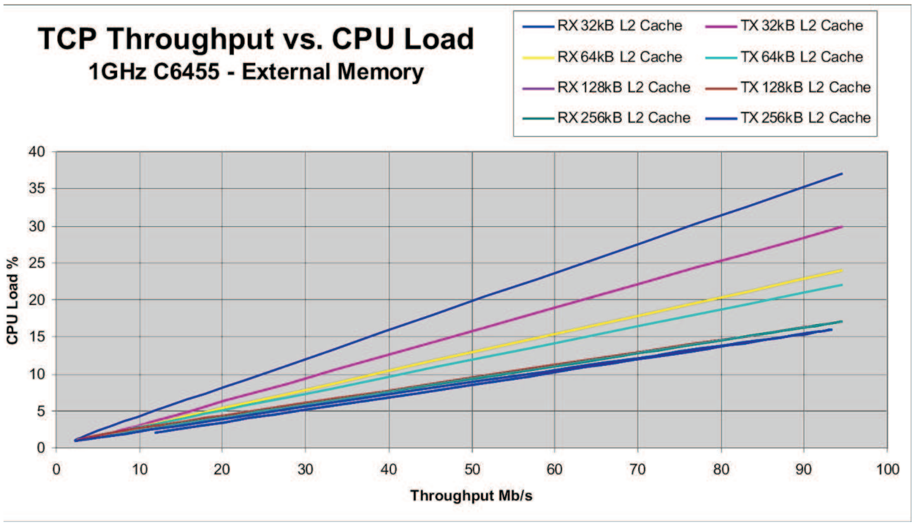


Figure 6. 1GHz TMS320C6455 TCP Throughput vs. CPU Load for Various L2 Cache Sizes

Figure 7 shows TCP transmit and receive throughput vs. CPU load for the TMS320C6455 running at 1 GHz with 256 KB L2 cache at varying frame sizes.

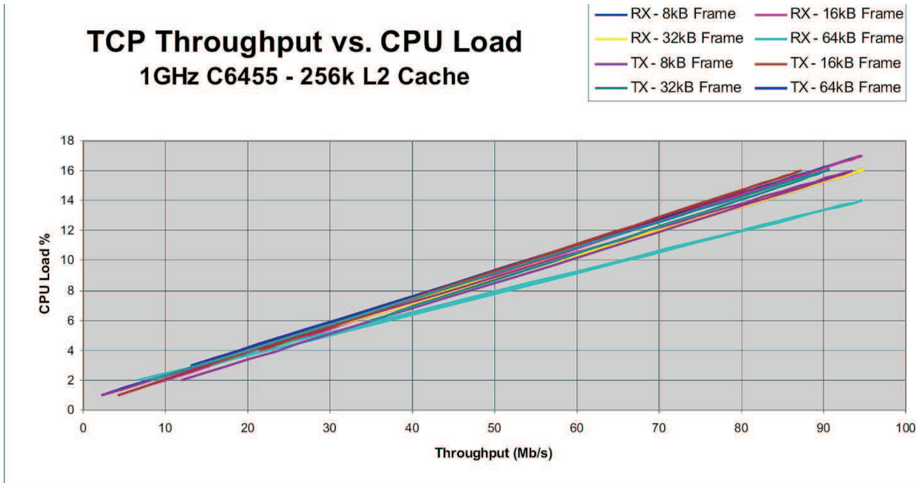


Figure 7. 1GHz TMS320C6455 TCP Throughput vs. CPU Load for Various Frame Sizes

6.2.2 With LL Architecture

The following table shows maximum receive and transmit throughput and CPU load data for a DSK6455 running at 1 GHz using a frame size of 8192 and various L2 cache settings.

L2 Cache Size	Operation	Measured Throughput (Kbits/sec)	CPU Load (±1%)
32K	Receive	94,595	27%
64K	Receive	94,595	18%
128K	Receive	94,595	16%
256K	Receive	94,541	16%
32K	Transmit	94,677	27%
64K	Transmit	94,650	25%
128K	Transmit	94,759	21%
256K	Transmit	94,623	21%

Figure 8 shows TCP transmit and receive throughput vs. CPU load for the TMS320C6455 running at 1 GHz with 32 KB, 64 KB, 128 KB and 256 KB L2 cache, using an 8 KB frame size.

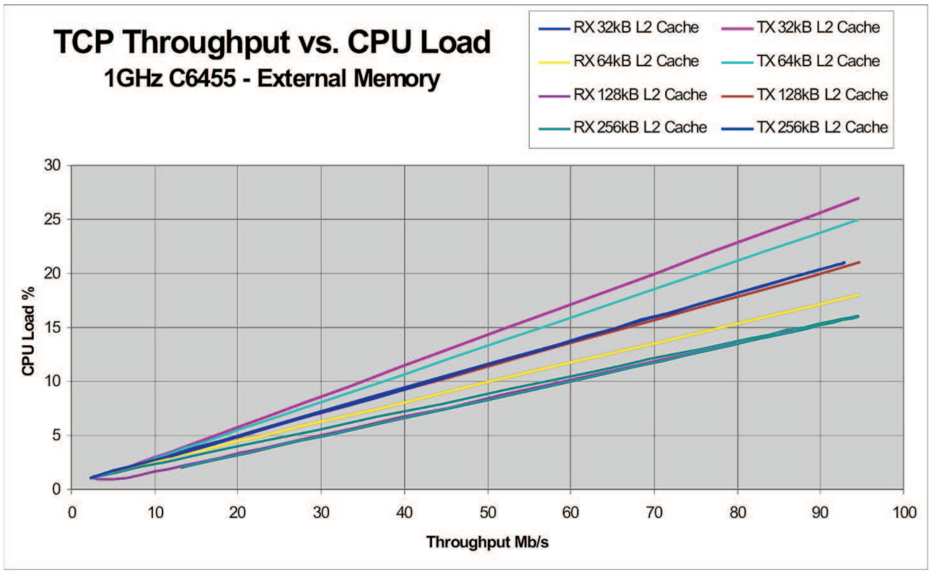


Figure 8. 1GHz TMS320C6455 TCP Throughput vs. CPU Load for Various L2 Cache Sizes

Figure 9 shows TCP transmit and receive throughput vs. CPU load for the TMS320C6455 running at 1 GHz with 256 KB L2 cache at varying frame sizes.

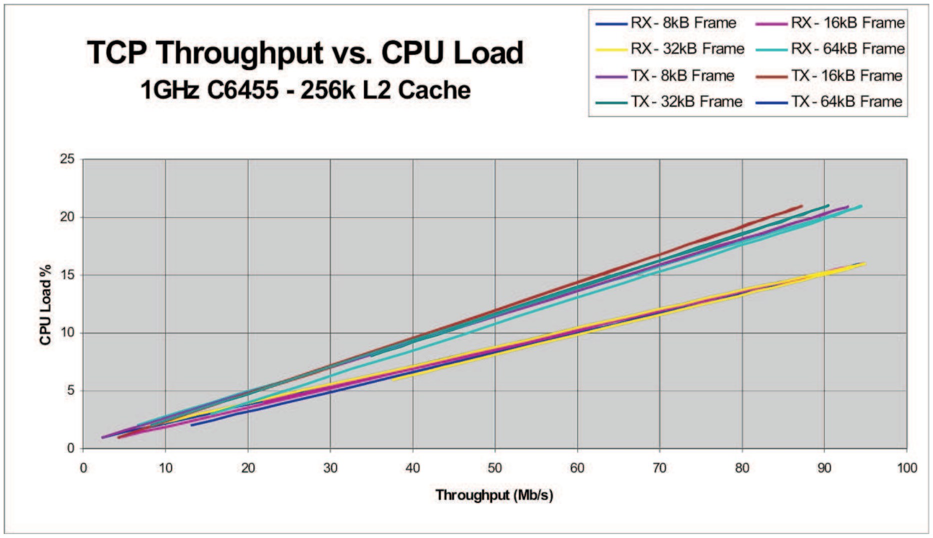


Figure 9. 1GHz TMS320C6455 TCP Throughput vs. CPU Load for Various Frame Sizes

6.3 TMS320DM6437 EVM Performance

This section does not show results of NIMU. Following data is only using LL architecture. The following table shows maximum receive and transmit throughput and CPU load data for a DM6437 EVM running at 594 MHz using a frame size of 8192 and various L2 cache settings.

L2 Cache Size	Operation	Measured Throughput (Kbits/sec)	CPU Load (±1%)
32K	Receive	87,381	54%
64K	Receive	80,471	27%
128K	Receive	91,581	29%
32K	Transmit	81,350	57%
64K	Transmit	87,732	43%
128K	Transmit	91,941	37%

Figure 10 shows TCP transmit and receive throughput vs. CPU load for the TMS320DM6437 EVM running at 594 MHz with 32 KB, 64 KB and 128 KB L2 cache, using an 8 KB frame size.

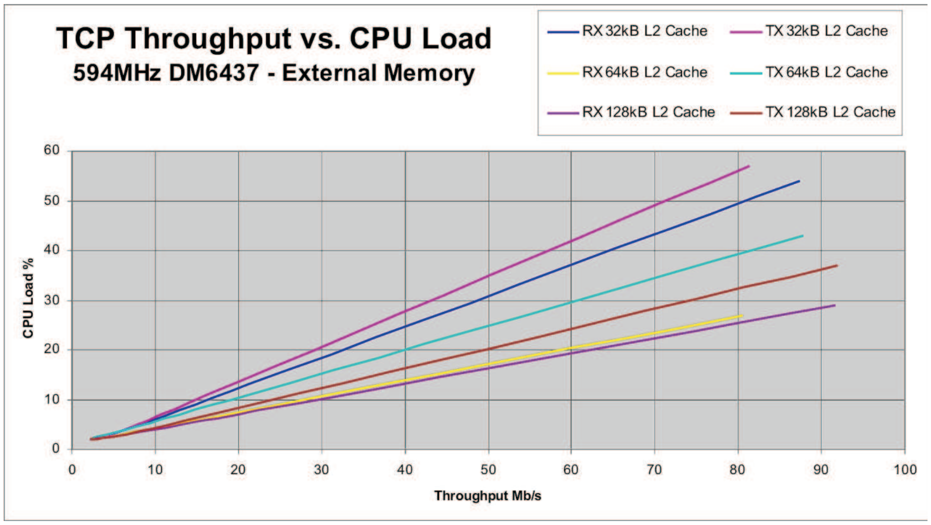


Figure 10. 594 MHz TMS320DM6437 TCP Throughput vs. CPU Load for Various L2 Cache Sizes

Figure 11 shows TCP transmit and receive throughput vs. CPU load for the TMS320C6437 EVM at 594 MHz with 128 KB L2 cache at varying frame sizes.

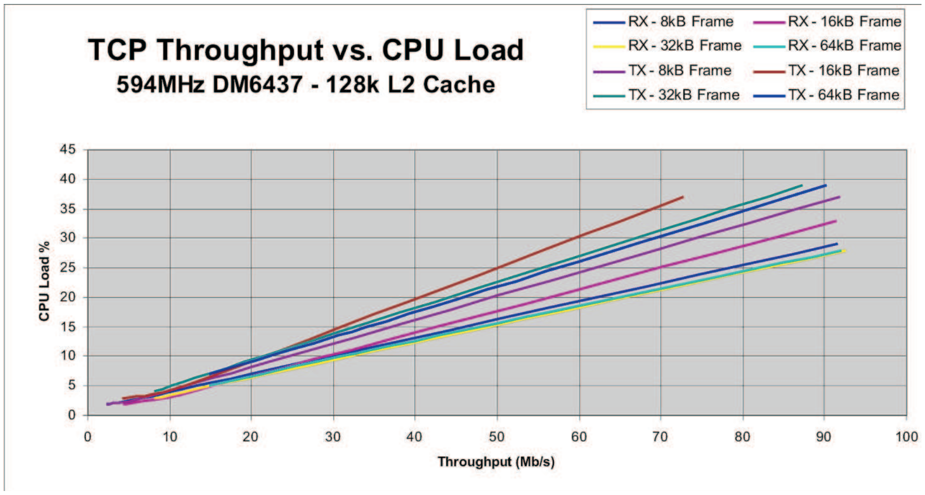


Figure 11. 594 MHz TMS320DM6437 TCP Throughput vs. CPU Load for Various Frame Sizes

7 Conclusion

The benchmark suite presented in this document is a foundational set to measure performance and gather sizing data. It enables you to carry out data size evaluations and performance even on customized systems. Also, the frame size and rate settings provide the flexibility needed to closely simulate a custom environment where applications are in place.

As the performance data in the previous section shows, a decrease in cache size can significantly affect performance—but not throughput—particularly with cache sizes smaller than 64 KB. Note that the DM642 device will not be able to achieve optimal maximum throughput with L2 cache sizes smaller than 64 KB due to MIPS limitations. The performance data for different buffer sizes shows that, in general, the performance impact is minimal (± 1-2% of CPU load). These differences may become more significant as the device's CPU speed decreases.

Additionally, though no relevant data is presented with this application report, the alignment of NDK static buffers also has a small effect on CPU performance (± 1-2% of CPU load), as described in Section 4.3.1. This may become more significant as both cache sizes and CPU speed decrease.

To optimize performance, it is recommended that L2 caches be as large as possible with a lower limit of 64 KB. At higher CPU speeds, frame buffer sizes and buffer alignments don't have a significant impact on performance. However, at lower CPU speeds, frame buffer sizes and buffer alignments may require fine tuning, since their impact on performance becomes more significant.

Introduction of NIMU architecture in 194_NDK does not show any performance degradation. The results show that throughput and CPU is similar to LL architecture.

8 References

- *TMS320C6000 Network Developer's Kit (NDK) Software User's Guide* (SPRU523)
- *TMS320C6000 Network Developer's Kit (NDK) Software Programmer's Reference Guide* (SPRU524)
- *TMS320C6000 DSP/BIOS Application Programming Interface (API) Reference Guide* (SPRU403)
- *TMS320C64x+ DSP Cache User's Guide* (SPRU862)
- *TMS320C6000 DSP Cache User's Guide* (SPRU656)

IMPORTANT NOTICE

Following are URLs where you can obtain information on other Texas Instruments products and application solutions:

Products		**Applications**	
Amplifiers	amplifier.ti.com	Audio	www.ti.com/audio
Data Converters	dataconverter.ti.com	Automotive	www.ti.com/automotive
DSP	dsp.ti.com	Broadband	www.ti.com/broadband
Clocks and Timers	www.ti.com/clocks	Digital Control	www.ti.com/digitalcontrol
Interface	interface.ti.com	Medical	www.ti.com/medical
Logic	logic.ti.com	Military	www.ti.com/military
Power Mgmt	power.ti.com	Optical Networking	www.ti.com/opticalnetwork
Microcontrollers	microcontroller.ti.com	Security	www.ti.com/security
RFID	www.ti-rfid.com	Telephony	www.ti.com/telephony
RF/IF and ZigBee® Solutions	www.ti.com/lprf	Video & Imaging	www.ti.com/video
		Wireless	www.ti.com/wireless

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