Micriμm

Empowering Embedded Systems

μC/TCP-IP
and
The ADSP-BF537 Processor

Application Note
AN-3530

www.Micrium.com

Table of Contents

1.00 Introduction

µC/TCP-IP is a compact, reliable, high-performance TCP/IP protocol stack. The result of many man-years of development, **µC/TCP-IP** exhibits the quality that is typical of Micrium software. It contains over 100,000 lines of the cleanest, most consistent ANSI C source code that you will ever find in a TCP/IP stack, and it is designed to be used with nearly any CPU, RTOS, and NIC (Network Interface Controller). Therefore, **µC/TCP-IP** enables the rapid configuration of required network options, minimizing your time to market.

This document explains how to use **µC/TCP-IP** with the Analog Device's ADSP-BF537 processor. The files included with this document were tested on ADSP-BF537 EZ-KIT LITE platform using Visual DSP++ 5.0.

The application example, provided with this document, uses **µC/OS-II** v2.86 and **µC/TCP-IP** v1.91. It's intended to demonstrate the use of **µC/TCP-IP** through:

- **Simple "ping" Echo Request** from a computer which is connected to the same network as your target board.

- **Test TCP or TTCP** which is a tool that performs TCP/IP or UDP/IP performance tests. TTCP is a commandline sockets-based benchmarking tool for measuring performance between two systems. It was originally developed in 1984 by Mike Muuss and Terry Slattery for the BSD operating system. The original TTCP and sources are in the public domain, and copies are available from many anonymous FTP sites.

1.01 Provided Files

µC/TCP-IP, like Micrium's other software modules, is provided in source code form, and it is compiled along with your application. **µC/TCP-IP**'s source code consists of a combination of device-independent files and files that adapt the software to specific processors, operating systems, and NICs.

`AN-3530.zip`, the file that includes this document, furnishes all these files including **µC/OS-II** and **µC/TCP-IP** source files.

Once `AN-3530.zip` is decompressed, you obtain the folders briefly described below. The content of **µC/TCP-IP** folders is described in more detail in the **µC/TCP-IP** User's Manual.

`\Micrium\Software\uCOS-II\Source`

This folder contains the processor-independent code for **µC/OS-II**.

`\Micrium\Software\uCOS-II\Ports\ADSP-BF537\VDSP50`

This folder contains the **µC/OS-II** port files for the ADSP-BF537 processor. These files are thoroughly described in the Application Note AN-1530, which is available from http://www.micrium.com/analog

`\Micrium\AppNotes`

This directory contains the current document.

```
AN-3530.pdf
```

`\Micrium\Software\uC-TCPIP\Source`

µC/TCP-IP's device-independent files should be located in this folder.

`\Micrium\Software\uC-TCPIP\NIC\ETHER`

This folder should contain the files for your NIC driver. This application example was tested on the ADSP-BF537 EZ-KIT LITE which provides an Ethernet interface that contains a SMSC LAN83C185 device. This folder contains two subfolders:

`\Micrium\Software\uC-TCPIP\NIC\ETHER\LAN83C185`

```
net_nic_def.h
net_nic.h
net_nic.c
```

`\Micrium\Software\uC-TCPIP\NIC\ETHER\PHY\LAN83C185`

```
net_phy.h
net_phy.c
```

`\Micrium\Software\uC-TCPIP\IF\`

µC/TCP-IP's network interface code is placed in this folder. A subfolder containing code corresponding to the specific interface being used, such as Ethernet, should be located within this folder.

`\Micrium\Software\uC-TCPIP\OS\uCOS-II`

This directory holds the files that serve as the interface between **µC/TCP-IP** and **µC/OS-II**.

```
net_os.c
net_os.h
net_os_al.h
net_os_al_c.c
```

`\Micrium\Software\uC-TCPIP\Doc`

This directory contains **µC/TCP-IP** documentation files.

`\Micrium\Software\uC-LIB\`

The files in this folder contain Micrium's implementations of the standard library routines used by **µC/TCP-IP**. These routines are provided to simplify third-party certification of your products.

`\Micrium\Software\uC-CPU\`

This folder contains **cpu_def.h**, which declares #define constants for CPU alignment, endianness, and other generic CPU properties. It also contains the following subfolder:

`\Micrium\Software\uC-CPU\ADSP-BF537\VDSP50`

This subfolder contains **cpu.h** and **cpu_a.asm**. *cpu.h* file defines the Micriµm portable data types for 8-, 16-, and 32-bit signed and unsigned integers (such as CPU_INT16U, a 16-bit unsigned integer) for the ADSP-BF537 processor with the VDSP 5.0 toolchain. These allow code to be independent of processor and compiler word size definitions.

`\Micrium\Software\EvalBorads\ADI\ADSP-BF537_EZKIT_Lite\VDSP50\BSP`

This directory contains the following files:

```
bsp.h
bsp.c
net_bsp.h
net_bsp.c
```

`net_bsp.h` and `net_bsp.c` contain specific code to the NIC (Network Interface Controller) used and other functions that are dependent of the hardware.

`bsp.h` and `bsp.c` hide the hardware details from the application code, and allow you to initialize and setup the needed services like timers, leds and push-button switches.

`\Micrium\Software\EvalBorads\ADI\ADSP-BF537_EZKIT_Lite\VDSP50\uC-TCP-IP\EX1`

This is the example application folder which contains the following files:

Application files	VDSP++ generated files

`app_c.c`
`app_cfg.h`
`net_cfg.h`
`os_cfg.h`
`includes.h`
	`Ex1.djp`
	`Ex1.ldf`
	`Ex1.mak`
	`Ex1.pcf`
	`Ex1_basiccrt.s`
	`Ex1_heaptab.c`
	`Ex1_cplbtab.c`

\Micrium\Software\uC-TTCP\Doc
This directory contains the µC/TTCP documentation files.

\Micrium\Software\uC-TTCP\Source
This directory contains the µC/TTCP source files (`ttcp.c` and `ttcp.h`).

\Micrium\Software\uC-TTCP\OS\uCOS-II
This directory contains `ttcp_os.c` file. This file is a specific RTOS (**µC/OS-II**) for µC/TTCP.

\Micrium\Software\uC-TTCP\Cfg\Template
This directory contains `ttcp_cfg.h` file which is a configuration file for µC/TTCP.

\Micrium\Software\uC-TTCP\pcattcp
For our development, Micrium has selected the `PCAUSA` port of TTCP to Windows Sockets for the TTCP module running on the Windows host. It is called **pcattcp**. It is not part of the official release, but it's provided within this application example.

This directory contains the following files:

```
PCATTCP.chm
PCATTCP.exe
RELEASE.TXT
sourcesv2.zip
ttcpzip.exe
```

• `PCATTCP.chm`	is the pcattcp html file.
• `PCATTCP.exe`	is the tool itself.
• `RELEASE.TXT`	contains release informations for the pcattcp versions.
• `sourcesv2.zip`	is winzip file containing the pcattcp sources.
• `ttcpzip.exe`	a self-extract zip file containing the 4 files above (the download result).

You will find more information about pcattcp at http://www.pcausa.com/Utilities/pcattcp.htm

1.02 Configuring µC/TCP-IP

net_cfg.h, which should be placed in your project's Application folder, is **µC/TCP-IP**'s configuration file. net_cfg.h contains constants that represent configurable parameters, such as the number of timers, the number of buffers for packet reception and transmission, and the number of sockets that your application can open. These constants are described in the **µC/TCP-IP** User's Manual.

1.03 BSP Files

AN-3530.zip includes *two BSP files*: bsp.c, and bsp.h. These files declare the functions that setup the ADSP-BF537 EZ Kit Lite evaluation platform.

AN-3530.zip includes *two Network BSP files*: net_bsp.c, and net_bsp.h. These files declare the functions that your NIC driver will use to access your hardware. The files assume that you are using the SMSC LAN83C185 device.

1.04 Configuring the Network

This example application assumes that the ADSP-BF537 EZ Kit Lite board is connected to a network containing a computer from where a "ping" Echo Request is sent or from where the **pcattcp** program is executed in order to complete TTCP performance tests.

A static IP address of 192.168.1.13 is assigned to the board, this IP address is defined in app_cfg.h file.

As shown in the Figure 1.1, we have used a peer to peer configuration for this example application.

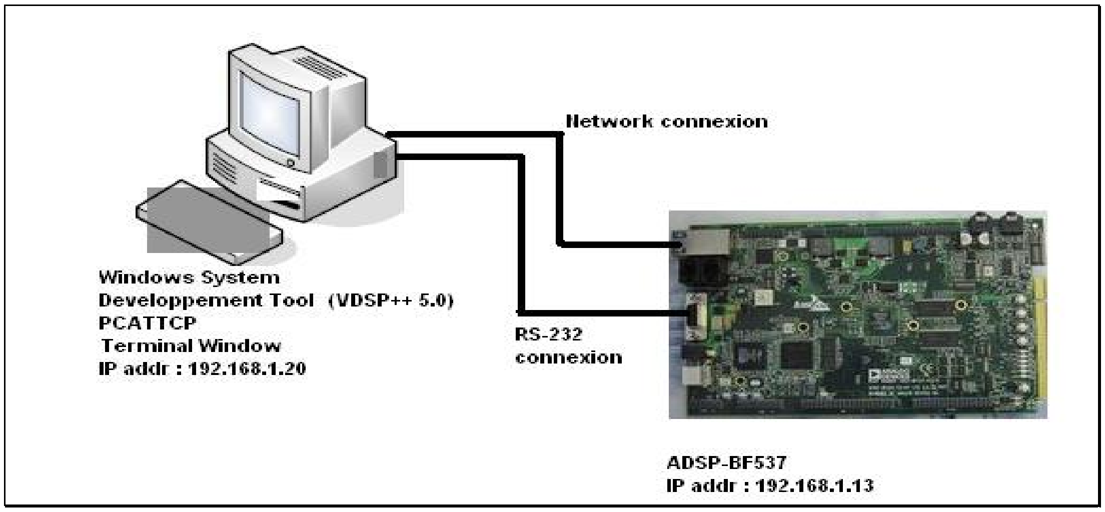


Figure 1.1 Network configuration

2.00 µC/TTCP and pcattcp

µC/TTCP is compliant with the other TTCP tools available in the public domain. It was written for target systems running µC/OS-II and µC/TCP-IP.

This section describes the µC/TTCP and the pcattcp usage.

2.01 µC/TTCP

As soon as the application is launched on the target, the Terminal Window running on the Development system will display the application status and a command line. The user controls µC/TTCP by entering parameters on the command line.

The command usage is displayed if the user makes an error entering the command line or by pressing ENTER at the “>” command prompt. Here is the command usage:

```
Usage: -t [-options] host
       -r [-options]
Common options: \r\n");
    -l ##   length of buffers read from or written to network (default 8192)
    -u      use UDP instead of TCP
    -p ##   port number to send to or listen at (default 5001)
    -s -t:  source a pattern to network
    -r:     sink (discard) all data from network
    -d      set SO_DEBUG socket option (not supported)
    -b ##   set socket buffer size (not supported)
    -f X    format for rate: k,K = kilo{bit,byte}; m,M = mega; g,G = giga
Options specific to -t:
    -n ##   number of source buffers written to network (default 2048)
    -D      don't buffer TCP writes (sets TCP_NODELAY socket option)
Options specific to -r:
    -B      for -s, only output full blocks as specified by -l
    -T      "touch": access each byte as it is read
```

As an example, to start the TTCP TCP Receive Test start µC/TTCP with the "-r" option.

```
>-r
```

The TCP server will start and then wait until a remote TTCP client makes a connection attempt.

Sinkmode (-s)

The simplest and most popular TTCP mode of operation is called "sinkmode". In this mode of operation the TTCP transmitter sends a fabricated data pattern and the TTCP receiver simply sinks (discards) any data that it receives.

This is the default TTCP mode of operation.

Standard Streams

Alternatively, TTCP can use what is called "standard streams" or "standard I/O". With µC/TTCP, the Serial port is used as the standard I/O.

Standard stream has only been implemented in Receiver mode. The code for Transmitter mode still needs to be developed.

Defaults

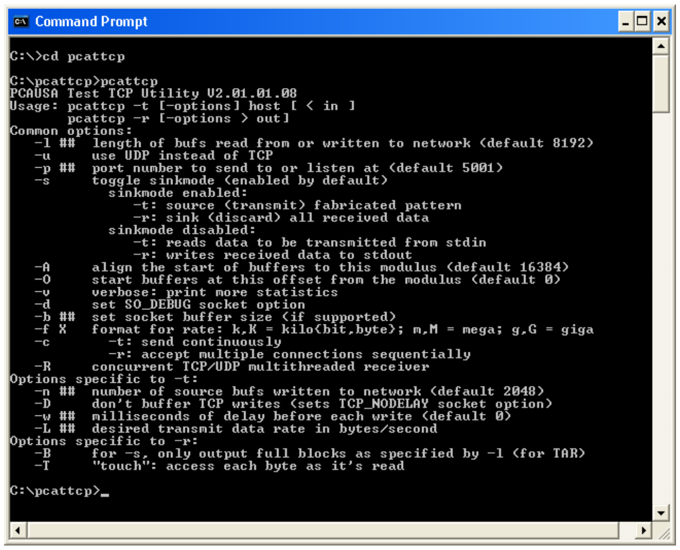
The µC/TTCP defaults for the command parameters are:

Parameter	Value	Command line parameter
Buffer Length	8192	-l
Number of buffers used	2048	-n
Transport layer protocol	TCP	-u
Layer 4 port number	5001	-p
Receiver/Transmitter	Receiver mode	-r or –t
Sink mode	Enabled	-s
Block read	Disabled	-B
Output format	'Kilobits per second'	-f
Received data processing	Disabled	-T
Socket options	Not supported	-d
Buffer TCP writes	Not supported	-D

2.02 pcattcp

Follow the steps below to run pcattcp :

- Copy the content of ttcpzip.exe to a directory (C:\pcattcp for example).
- Open a DOS prompt window
- Change directory to the directory now containing pcattcp.exe
- From the DOS prompt, start pcattcp: C:\pcattcp>pcattcp



```
C:\>cd pcattcp

C:\pcattcp>pcattcp
PCAUSA Test TCP Utility V2.01.01.08
Usage: pcattcp -t [-options] host [ < in ]
       pcattcp -r [-options > out]
Common options:
   -l ##  length of bufs read from or written to network (default 8192)
   -u     use UDP instead of TCP
   -p ##  port number to send to or listen at (default 5001)
   -s     toggle sinkmode (enabled by default)
            sinkmode enabled:
               -t: source (transmit) fabricated pattern
               -r: sink (discard) all received data
            sinkmode disabled:
               -t: reads data to be transmitted from stdin
               -r: writes received data to stdout
   -A     align the start of buffers to this modulus (default 16384)
   -O     start buffers at this offset from the modulus (default 0)
   -v     verbose: print more statistics
   -d     set SO_DEBUG socket option
   -b ##  set socket buffer size (if supported)
   -f X   format for rate: k,K = kilo{bit,byte}; m,M = mega; g,G = giga
   -c       -t: send continuously
            -r: accept multiple connections sequentially
   -R     concurrent TCP/UDP multithreaded receiver
Options specific to -t:
   -n ##  number of source bufs written to network (default 2048)
   -D     don't buffer TCP writes (sets TCP_NODELAY socket option)
   -w ##  milliseconds of delay before each write (default 0)
   -L ##  desired transmit data rate in bytes/second
Options specific to -r:
   -B     for -s, only output full blocks as specified by -l (for TAR)
   -T     "touch": access each byte as it's read

C:\pcattcp>_
```

Without any arguments, pcattcp will output the command usage. For more detailed information about the pcattcp commands, please refer to the html help file by double clicking on PCATTCP.chm. This will automatically open a help file.

3.00 An Example Application

app_c.c is a simple example application that can be used to test that **µC/TCP-IP** is running properly on your ADSP-BF537 system. app_c.c creates one task (in addition to the idle and statistic tasks created by **µC/OS-II**) that initializes **µC/TCP-IP, µC/TTCP** and then blinks a LED.

3.01 app_c.c

App_BootTask(), the single task created in app_c.c, as well as App_InitTCPIP(), the function that App_BootTask() calls to initialize **µC/TCP-IP**, are described below in Listing 3-2 and Listing 3-3, respectively.

```
int  main (void)
{

    CPU_INT08U  os_err;


    BSP_Init();                                              /* Initialize BSP.             */
    printf(("\n\n\n"));
    printf(("Initialize OS...\n"));
    OSInit();                                                /* Initialize OS.              */

                                                             /* Create start task.          */
    OSTaskCreateExt( App_BootTask,
                    (void *)0,
                    (OS_STK *)&App_BootTaskStk[APP_OS_CFG_BOOT_TASK_STK_SIZE - 1],
                     APP_OS_CFG_BOOT_TASK_PRIO,
                     APP_OS_CFG_BOOT_TASK_PRIO,
                    (OS_STK *)&App_BootTaskStk[0],
                     APP_OS_CFG_BOOT_TASK_STK_SIZE,
                    (void *)0,
                     OS_TASK_OPT_STK_CHK | OS_TASK_OPT_STK_CLR);

#if (OS_TASK_NAME_SIZE >= 16)                                /* Give a name to tasks.       */

    OSTaskNameSet(OS_TASK_IDLE_PRIO, "Idle", &os_err);
    OSTaskNameSet(APP_OS_CFG_BOOT_TASK_PRIO,"Boot Task", &os_err);

#endif

    printf("Start OS...\n");
    OSStart();                                               /* Start OS.                   */
}
```

Listing 3-2, `App_BootTask()`

```
static  void  App_BootTask (void  *p_arg)
{

    CPU_INT08U i;


    (void)p_arg;
    printf("Initialize OS timer...\n");
    BSP_CoreTmrInit();                                                                /* (1) */

#if (OS_TASK_STAT_EN > 0)
    printf("Initialize OS statistic task...\n");
    OSStatInit();                                                                     /* (2) */
#endif

    printf("Application resources initialization...\n");
    App_Init();                                                                       /* (3) */

    App_InitTCPIP();                                                                  /* (4) */

#if (APP_TTCP_ENABLED == DEF_ENABLED)

    APP_TRACE_DEBUG(("\n******************************************************************"));
    APP_TRACE_DEBUG(("\n*                                                                *"));
    APP_TRACE_DEBUG(("\n*      Micrium uC/TCP-IP TTCP Performance measurement         *"));
    fflush((FILE *)0);
    APP_TRACE_DEBUG(("\n*            ADSP-BF537 EZKIT Lite Platform                   *"));
    APP_TRACE_DEBUG(("\n*                                                                *"));
    APP_TRACE_DEBUG(("\n******************************************************************"));
    APP_TRACE_DEBUG(("\n"));
    fflush((FILE *)0);
    TTCP_Init();                                                                      /* (5) */

#endif

    for (i =1 ; i < 7; i++) {                                                         /* (6) */
        BSP_ClrLED(i);
    }

    while (DEF_YES) {                                                                 /* (7) */

         BSP_ToggleLED(2);
         OSTimeDly(150);
    }
}
```

L3-2(1) `App_BootTask()` calls `BSP_CoreTmrInit()` (declared in `bsp.c` file) to initialize the core timer which will provide the ticks source for **µC/OS-II**.

L3-2(2) `OSStatInit()` is called to initialize **µC/OS-II**'s statistic task. This call only occurs if you have enabled the statistic task by setting `OS_TASK_STAT_EN` to `1` in os_cfg.h file. The statistic task determines what percentage of the CPU is being used by your tasks. The statistic task also performs stack checking for any of your tasks that were created with `OSTaskCreateExt()` with the stack checking option set.

L3-2(3) `App_Init()` initializes LEDs on ADSP-BF537 EZ-KIT Lite, and installs the appropriate ISR for each interrupt level group.

L3-2(4) `App_InitTCPIP()`, which is described in Listing 3-3, is called to perform required initializations for **µC/TCP-IP**.

L3-2(5) Initialize TTCP application.

L3-2(6) Clear all LEDs on ADSP-BF537 EZ-KIT Lite evaluation platform.

L3-2(7) The body of this task toggles LED#2 and then delays for 150 ticks. Once the body of the task is executing, you should be able to 'ping' the board or to run TTCP performance tests.

Listing 3-3, `App_InitTCPIP()`

```
#define  APP_CFG_IP_ADDR_STR_THIS_HOST          "192.168.1.13"
#define  APP_CFG_IP_ADDR_STR_NET_MASK           "255.255.255.0"

static  NET_IP_ADDR  App_IP_Addr;
static  NET_IP_ADDR  App_IP_Mask;

static  void  App_InitTCPIP (void)
{
     NET_ERR  err;


     APP_TRACE_DEBUG(("Initialize TCP/IP stack...\n"));

     err = Net_Init();                                                                    /* (1)
*/

     if (err != NET_ERR_NONE) {
         APP_TRACE_DEBUG(("Net_Init() failed: error #%d, line #%d.\n", err, __LINE__));
         while (DEF_YES) {
            ;
         }
     }


     App_IP_Addr        = NetASCII_Str_to_IP(APP_CFG_IP_ADDR_STR_THIS_HOST, &err);      /* (2)
*/
     App_IP_Mask        = NetASCII_Str_to_IP(APP_CFG_IP_ADDR_STR_NET_MASK,  &err);
     NetIP_CfgAddrThisHost(App_IP_Addr, App_IP_Mask);                                     /* (3)
*/

     APP_TRACE_DEBUG(("    IP address   = %s\n", APP_CFG_IP_ADDR_STR_THIS_HOST));
     APP_TRACE_DEBUG(("    IP mask      = %s\n", APP_CFG_IP_ADDR_STR_NET_MASK));

}
```

L3-3(1) `Net_Init()` is called to initialize **µC/TCP-IP**. More information on `Net_Init()` can be found in the **µC/TCP-IP** User's Manual.

L3-3(2) A static IP address of `192.168.1.13` is assigned to the board. If you wish to use another address, you should modify the value of `APP_CFG_IP_ADDR_STR_THIS_HOST` declared in `app_cfg.h` file. The address that is set using this call is the address that you should use to 'ping' the board.

L3-3(3) Configure the board accordingly to the chosen IP address and Network Mask.

3.02 TTCP results

In this section, we present the TTCP results obtained in the following conditions:

• ADSP-BF537 Core Frequency	525 MHz.
• ADSP-BF537 System Frequency	131 MHz.
• TTCP Buffer size	**8192 bytes** for TCP and **1432 bytes** for UDP.
• TTCP number of buffer used	2048
• Instruction cache	enabled
• Data cache	enabled for banks A and B

- pcattcp running on a PC with the following caracteristics:

  Intel Core 2 CPU 2.40 GHz
  2.00 GB of RAM

  .

As you can see, the TTCP UDP transmit and receive test uses a buffer length of 1432 bytes because µC/TCP-IP does not presently support transmission fragmentation, µC/TTCP limits the buffer size to 1432 bytes.

TCP Transmit Test

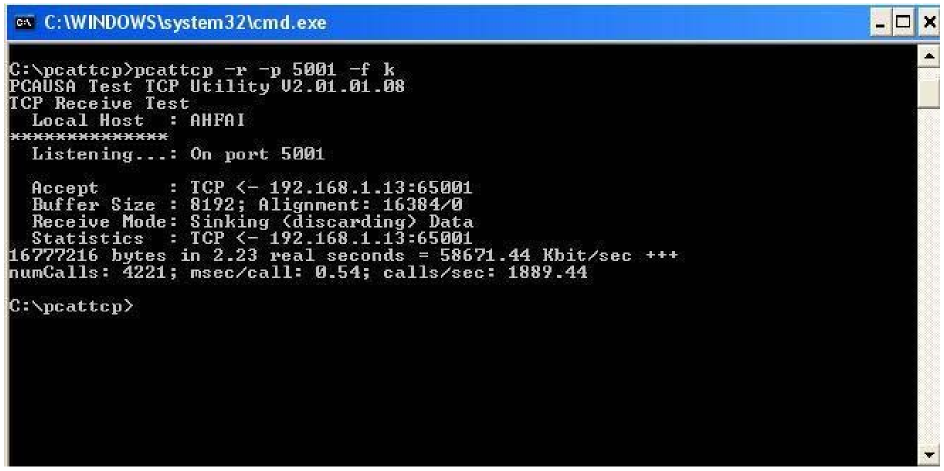
Target

To start the TTCP TCP Transmit Test start µC/TTCP with the "-t" option followed by the dotted IP address of the remote TTCP client.

```
>-t 192.168.1.20

ttcp-t: BufLen=8192, NumBuf=2048, port=5001, NumConns=1 tcp  -> 192.168.1.20
TTCP_RemoteIP = 192.168.1.20, TTCP_RemoteAddr = c0a80114
ttcp-t: Client socket 9 opened
Connecting to addr: 1401a8c0, port: 8913
ttcp-t: Client socket 9 connected
ttcp-t: Client socket 9 closed

ttcp-t: 16777216 bytes in 2.235 real sec = 7330.65 KB/sec (60052676.760 bps)
ttcp-t: 4207 I/O calls, msec/call = 0.531, calls/sec = 1882.326
```

Windows host TTCP session

The Windows host running pcattcp must initiate the Receiver part of this test. Figure 3.1 shows the command used:


C:\WINDOWS\system32\cmd.exe

```
C:\pcattcp>pcattcp -r -p 5001 -f k
PCAUSA Test TCP Utility V2.01.01.08
TCP Receive Test
  Local Host  : AHFAI
**************
  Listening...: On port 5001

  Accept      : TCP <- 192.168.1.13:65001
  Buffer Size : 8192; Alignment: 16384/0
  Receive Mode: Sinking (discarding) Data
  Statistics  : TCP <- 192.168.1.13:65001
16777216 bytes in 2.23 real seconds = 58671.44 Kbit/sec +++
numCalls: 4221; msec/call: 0.54; calls/sec: 1889.44

C:\pcattcp>
```

Figure 3.1 TCP Receiver part.

TCP Receive Test

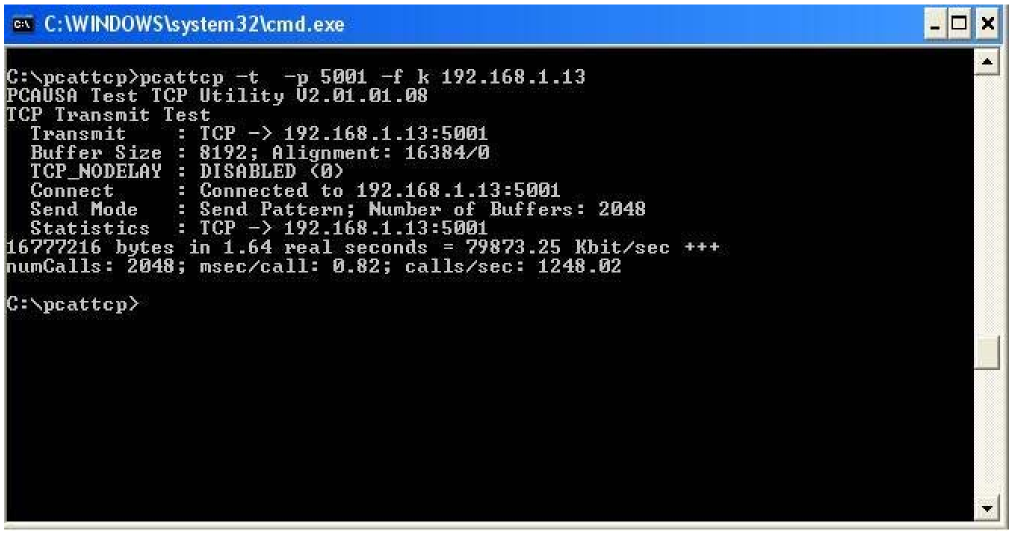
Target

To start the TTCP TCP Receive Test start µC/TTCP with the "-r" option. The TCP receiver will start and then wait until a remote TTCP client makes a connection attempt.

```
>-r

ttcp-r: BufLen=8192, NumBuf=2048, port=5001, NumConns=1  tcp
TTCP_RemoteIP = 0.0.0.0, TTCP_RemoteAddr = 0
ttcp-r: Listening socket opened = 9 for PORT: 5001
ttcp-r: Waiting for 1 client to request connection.

TTCP_RemoteAddLen = 16, NET_SOCK_ADDR_SIZE=16.
ttcp-r: Server socket 8 active
ttcp-r: Client socket 8 closed
ttcp-r: Listen socket 9 closed

ttcp-r: 16777216 bytes in 1.645 real sec = 9959.88 KB/sec (81591328.440 bps)
ttcp-r: 11487 I/O calls, msec/call = 0.143, calls/sec = 6982.979
```

Windows host TTCP session

The Windows host running pcattcp must initiate the TCP Transmitter part of this test. Figure 3.2 shows the command used:



```
C:\pcattcp>pcattcp -t  -p 5001 -f k 192.168.1.13
PCAUSA Test TCP Utility V2.01.01.08
TCP Transmit Test
  Transmit     : TCP -> 192.168.1.13:5001
  Buffer Size  : 8192; Alignment: 16384/0
  TCP_NODELAY  : DISABLED (0)
  Connect      : Connected to 192.168.1.13:5001
  Send Mode    : Send Pattern; Number of Buffers: 2048
  Statistics   : TCP -> 192.168.1.13:5001
16777216 bytes in 1.64 real seconds = 79873.25 Kbit/sec +++
numCalls: 2048; msec/call: 0.82; calls/sec: 1248.02

C:\pcattcp>
```

Figure 3.2 TCP Transmitter part.

UDP Transmit Test

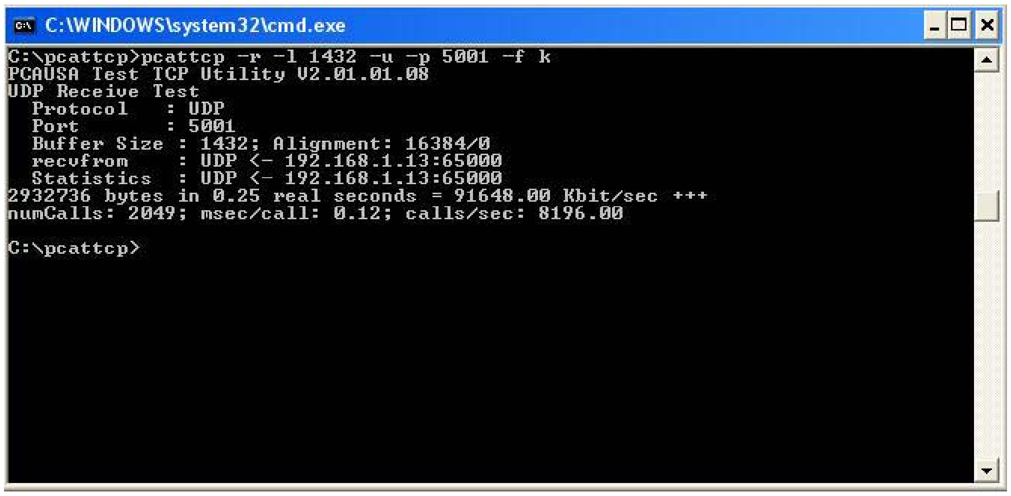
Target

To start the TTCP UDP Transmit Test start µC/TTCP with the "-t" option followed by the "-u " option followed finally by the dotted IP address of the remote TTCP client.

```
>-t -u 192.168.1.20

ttcp-t: BufLen=1432, NumBuf=2048, port=5001, NumConns=1  udp -> 192.168.1.20
TTCP_RemoteIP = 192.168.1.20, TTCP_RemoteAddr = c0a80114
ttcp-t: Client socket 9 opened
Connecting to addr: 1401a8c0, port: 8913
ttcp-t: Client socket 9 connected
ttcp-t: Client socket 9 closed

ttcp-t: 2932736 bytes in 0.259 real sec = 11057.92 KB/sec (90586440.464 bps)
ttcp-t: 2048 I/O calls, msec/call = 0.126, calls/sec = 7907.336
```

Windows host TTCP session

The Windows host running pcattcp must initiate the UDP Receiver part of this test. Figure 3.3 shows the command used:



```
C:\pcattcp>pcattcp -r -l 1432 -u -p 5001 -f k
PCAUSA Test TCP Utility V2.01.01.08
UDP Receive Test
  Protocol    : UDP
  Port        : 5001
  Buffer Size : 1432; Alignment: 16384/0
  recvfrom    : UDP <- 192.168.1.13:65000
  Statistics  : UDP <- 192.168.1.13:65000
2932736 bytes in 0.25 real seconds = 91648.00 Kbit/sec +++
numCalls: 2049; msec/call: 0.12; calls/sec: 8196.00

C:\pcattcp>
```

Figure 3.3 UDP Receiver part.

UDP Receive Test

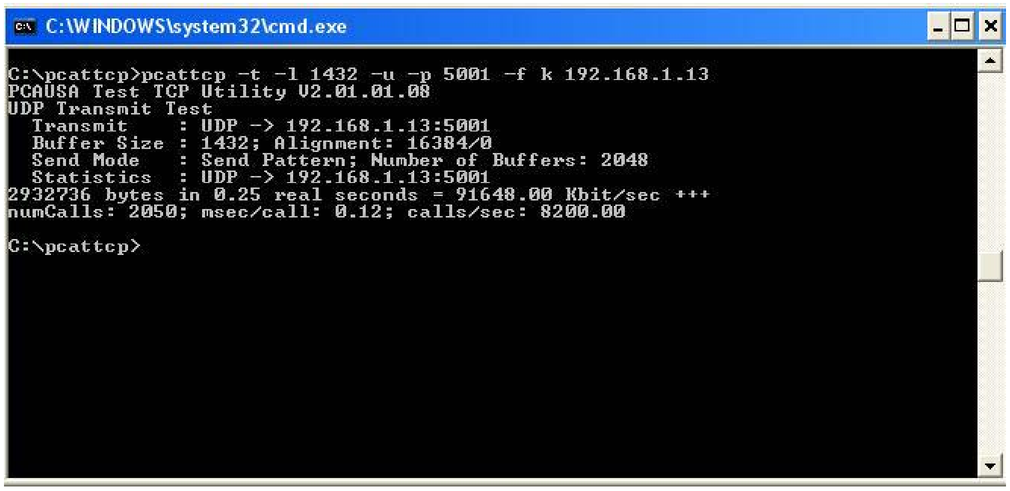
Target

To start the TTCP UDP Receive Test start µC/TTCP with the "-r" option followed by the "-u " option. The UDP receiver will start and then wait until a remote TTCP client makes a connection attempt.

```
>-r -u

ttcp-r: BufLen=1432, NumBuf=2048, port=5001, NumConns=1  udp
TTCP_RemoteIP = 0.0.0.0, TTCP_RemoteAddr = 0
ttcp-r: UDP socket 9 opened
ttcp-r: Waiting for client to send UDP datagrams.
ttcp-r: Client socket 9 closed

ttcp-r: 2932736 bytes in 0.258 real sec = 11100.78 KB/sec (90937552.584 bps)
ttcp-r: 2048 I/O calls, msec/call = 0.125, calls/sec = 7937.985
```

Windows host TTCP session

The Windows host running pcattcp must initiate the UDP Transmitter part of this test. Figure 3.4 shows the command used:



```
C:\pcattcp>pcattcp -t -l 1432 -u -p 5001 -f k 192.168.1.13
PCAUSA Test TCP Utility V2.01.01.08
UDP Transmit Test
  Transmit    : UDP -> 192.168.1.13:5001
  Buffer Size : 1432; Alignment: 16384/0
  Send Mode   : Send Pattern; Number of Buffers: 2048
  Statistics  : UDP -> 192.168.1.13:5001
2932736 bytes in 0.25 real seconds = 91648.00 Kbit/sec +++
numCalls: 2050; msec/call: 0.12; calls/sec: 8200.00

C:\pcattcp>
```

Figure 3.4 UDP Transmitter part.

Acknowledgements

We would like to thank the following people for their support in making the **`LAN83C185`** device's driver for the ADSP-BF537.

Analog Devices Inc:

Mr.	***Deep B.***
Mr.	***Lukasiak, Tomasz.***

References

µC/OS-II, The Real-Time Kernel, 2nd Edition
Jean J. Labrosse
R&D Technical Books, 2002
ISBN 1-57820-103-9

Contacts

Analog Devices Inc. One Technology Way, P.O. Box 9106 Norwood, MA 02062-9106 U.S.A. +1 781.329.4700 +1 781.461.3113 (FAX). WEB: http://www.analog.com	**CMP Books, Inc.** 6600 Silacci Way Gilroy, CA 95020 USA Phone Orders: 1-800-500-6875 or 1-408-848-3854 Fax Orders: 1-408-848-5784 e-mail: rushorders@cmpbooks.com WEB: http://www.cmpbooks.com
Micriµm 949 Crestview Circle Weston, FL 33327 USA 954-217-2036 954-217-2037 (FAX) e-mail: Jean.Labrosse@Micrium.com WEB: www.Micrium.com	