

SmartFusion: Running Webserver, TFTP on lwIP TCP/IP Stack

Table of Contents

Introduction	1
Introduction to lwIP TCP/IP Stack	2
Detailed Description of Porting the lwIP with and without RTOS to SmartFusion	2
Configuring the lwIP TCP/IP Stack	6
lwIP API Reference	7
Detailed Description of the Design Examples	7
Conclusion	18
Appendix A – Design Files	19

Introduction

SmartFusion™ intelligent mixed signal FPGAs integrate FPGA technology with the hardened ARM® Cortex™-M3 based microcontroller subsystem (MSS) and programmable high-performance analog blocks built on a low power flash semiconductor process. The MSS consists of hardened blocks, such as a 100 MHz ARM Cortex-M3 processor, peripheral DMA (PDMA), embedded nonvolatile memory (eNVM), embedded SRAM (eSRAM), embedded FlashROM (eFROM), external memory controller (EMC), watchdog timer, I²C, SPI, 10/100 Ethernet controller, real-time counter (RTC), GPIO block, fabric interface controller (FIC), in-application programming (IAP), and system registers. The programmable analog block contains the analog compute engine (ACE) and analog front-end (AFE), consisting of ADCs, DACs, active bipolar prescalers (ABPS), comparators, current monitors, and temperature monitors.

Ethernet MAC in SmartFusion is a high-speed media access control (MAC) Ethernet controller with the following features:

- Carrier sense multiple access with collision detection (CSMA/CD) algorithms defined by the IEEE 802.3 standard.
- Complies with the low-pin-count reduced media independent interface (RMII™) specifications.
- In-built DMA controller to move data between external RAM and TX/RX FIFOs.

Refer to the [SmartFusion Microcontroller Subsystem User's Guide](#) for more details on 10/100 Ethernet MAC interface.

The lightweight IP (lwIP) stack is an open-source implementation of the TCP/IP stack specially designed for embedded systems. lwIP provides networking capability to SmartFusion-based embedded systems. This application note describes the porting of the lwIP TCP/IP stack with and without RTOS to SmartFusion. This application note also demonstrates lwIP applications such as a webserver that serves web pages from SPI Flash with FatFs file system and TFTP server on SmartFusion Evaluation Kit Board and SmartFusion Development Kit Board.

Introduction to lwIP TCP/IP Stack

lwIP is an implementation of the light weight TCP/IP stack. lwIP was developed by [Adam Dunkels](#) at the [Swedish Institute of Computer Science \(SICS\)](#). lwIP stack is more suitable for the embedded systems because of small data and code size requirements. lwIP can be used with OS or without OS. The core of lwIP consists of the actual implementations of the IP, ICMP, UDP, and TCP protocols, as well as the support functions such as buffer and memory management.

More details about the design and implementation can be found at following location:

www.sics.se/~adam/lwip/doc/lwip.pdf

lwIP is freely available (under a BSD-style license) in C source code format and can be downloaded from the following location:

download.savannah.gnu.org/releases/lwip/

Detailed Description of Porting the lwIP with and without RTOS to SmartFusion

Figure 1 shows the typical network application architecture on SmartFusion using lwIP stack.

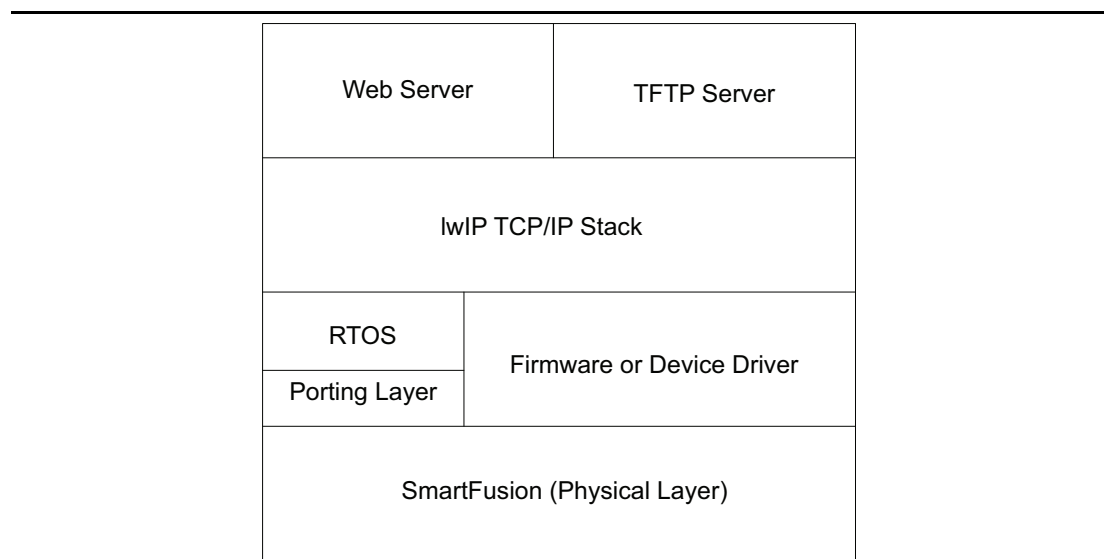


Figure 1 • Typical Network Application Architecture on SmartFusion Using lwIP Stack

The interface to the physical layer is done through the network interface (NetIF) data structure of the lwIP stack. The entries in the structure are:

```
struct netif {
    /** pointer to next in linked list */
    struct netif *next;

    /** IP address configuration in network byte order */
    struct ip_addr ip_addr;
    struct ip_addr netmask;
    struct ip_addr gw;

    /** This function is called by the network device driver
     * to pass a packet up the TCP/IP stack. */
    err_t (* input)(struct pbuf *p, struct netif *inp);
    /** This function is called by the IP module when it wants
     * to send a packet on the interface. This function typically
     * first resolves the hardware address, then sends the packet. */
}
```

```
err_t (* output)(struct netif *netif, struct pbuf *p,  
    struct ip_addr *ipaddr);  
/** This function is called by the ARP module when it wants  
 * to send a packet on the interface. This function outputs  
 * the pbuf as-is on the link medium. */  
err_t (* linkoutput)(struct netif *netif, struct pbuf *p);  
...  
...  
...  
}
```

The SmartFusion Ethernet MAC driver APIs are used in the low level API that are assigned to the input and output function pointers of the netif structure. For sending the data to the Ethernet MAC, the following example is implemented:

```
static err_t low_level_output(struct netif *netif, struct pbuf *p)  
{  
    ...  
    ...  
    ...  
  
    if( !MSS_MAC_tx_packet( out_buffer, p->tot_len, MSS_MAC_BLOCKING) )  
    {  
        printf("Failed Sending Data to Eth len =%d\n\r", p->tot_len);  
        return( ~ERR_OK);  
    }  
    ...  
    ...  
    ...  
}
```

Following is the example implementation for receiving the data from the Ethernet MAC and passing the received data to lwIP stack.

```
static struct pbuf *low_level_input(struct netif *netif)  
{  
    ...  
    ...  
    ...  
  
    len = MSS_MAC_rx_packet( s_rxBuff, 4096, MSS_MAC_NONBLOCKING);  
    ...  
    ...  
    ...  
}
```

```
void ethernetif_input( void * pvParameters )  
{  
  
    #if (NO_SYS == 0)  
        for( ;; ) {  
    #endif  
        do  
        {  
            /* move received packet into a new pbuf */  
            p = low_level_input( xNetIf );  
  
        } while( p == NULL );  
        ...  
        ...  
        ...  
    }  
}
```

You need to create a task for 'ethernetif_input' function to process the received data on the Ethernet MAC if the OS is used. If the OS is not used, you need to call the 'ethernetif_input' function in main loop.

You need to update the netif structure for sending the data to Ethernet MAC.

- netif->output = etharp_output; /*ethernetif_output;*/
- netif->linkoutput = low_level_output;

The lwIP with network interface can be run as a group of interdependent tasks in multi threaded system with RTOS or in a single 'while loop' in non multi threaded system. Its main loop basically checks for the following:

1. Incoming data on the network interface
2. Periodic time out for methods depends on the timers such as delayed acknowledgement

This application note also describes the usage of the lwIP stack with OS and without the OS on SmartFusion Evaluation Kit Board and SmartFusion Development Kit Board for webserver, TFTP, and FatFs file system applications. This application note is provided with two different design examples:

1. Method 1 demonstrates the usage of lwIP stack with FreeRTOS and a basic webserver application on SmartFusion Evaluation Kit Board and SmartFusion Development Kit Board.
2. Method 2 demonstrates the usage of lwIP stack RAW APIs, TFTP application, FatFs file system on SPI flash and a webserver getting the web pages from SPI Flash using the FatFs file system on SmartFusion Evaluation Kit Board and SmartFusion Development Kit Board. Using this method, you can directly load the updated the web pages from the Host PC to the SPI Flash using TFTP.

Using lwIP with RTOS Support Layer

Socket API or NetConn API of lwIP can be used along with RTOS. This method needs to have the RTOS support for the inter task communication mechanism. lwIP with RTOS will internally take care of periodic calls required and hence the application need not call the periodic calls of lwIP stack. There are basically three different tasks working in parallel to meet the application requirements. They are:

1. Network interface task: Monitors the Ethernet MAC hardware for the incoming packets and sends the packet to the network layer if the packet is either IP or ARP packet. Following is the sample code for this task. This task is implemented in ethernetif.c file of the demo package:

```
void ethernetif_input( void * pvParameters )
{
    ...
    #if (NO_SYS == 0)
        for( ;; ) {
    #endif
        ethernetif = xNetIf->state;
        do
        {
            /* move received packet into a new pbuf */
            p = low_level_input( xNetIf );

            } while( p == NULL );

        /* points to packet payload, which starts with an Ethernet header */
        ethhdr = p->payload;

        switch( htons( ethhdr->type ) )
        {
            /* IP packet? */
            case ETHTYPE_IP:
                /* update ARP table */
                etharp_ip_input( xNetIf, p );

                /* skip Ethernet header */
                pbuf_header( p, (s16_t)-sizeof(struct eth_hdr) );

                /* pass to network layer */
                if ( xNetIf->input( p, xNetIf ) != ERR_OK)
                {
                    pbuf_free(p);
                }
            }
        }
    #endif
}
```

```

p = NULL;
}
break;
case ETHTYPE_ARP:
    /* pass p to ARP module */
    etharp_arp_input( xNetIf, ethernetif->ethaddr, p );
break;
...
...
...

}

#if (NO_SYS == 0)
}
#endif
}

```

Figure 2 shows the flow chart of the lwIP Stack running on SmartFusion with FreeRTOS.

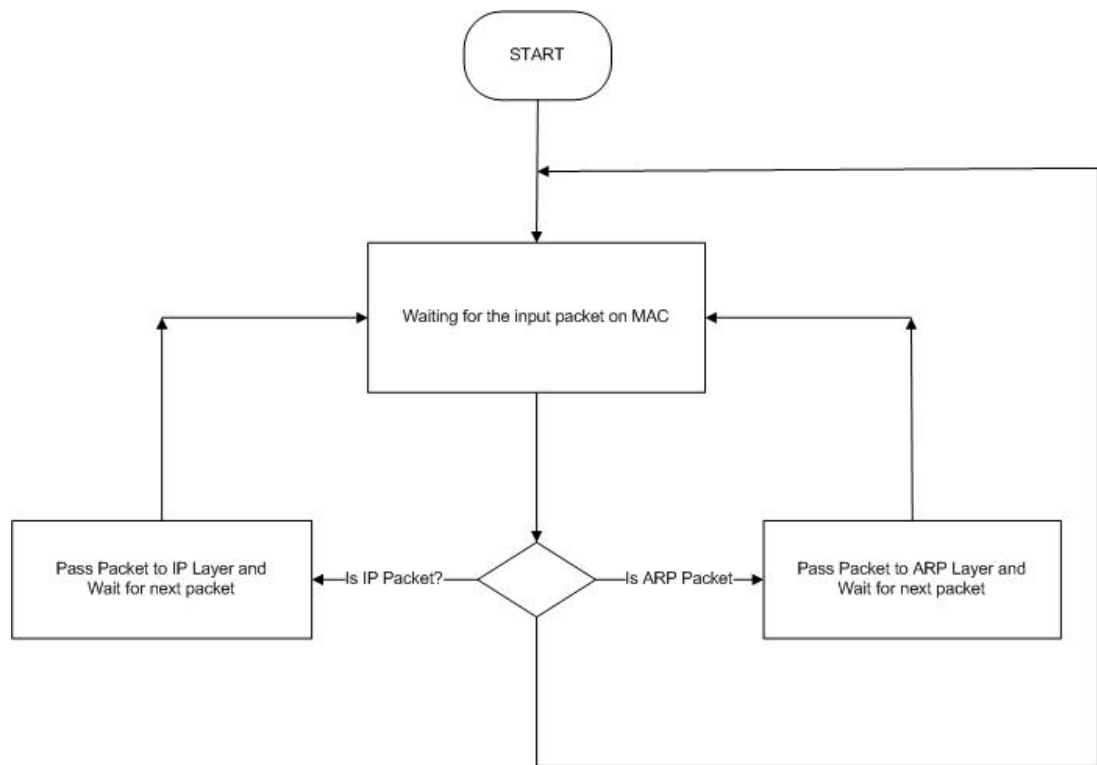


Figure 2 • Flow Chart of the lwIP Stack Running on SmartFusion with FreeRTOS

2. TCP/IP Thread: This is the main lwIP thread. This thread has exclusive access to lwIP core functions. Other threads (e.g application layer threads) communicate with this thread using message boxes. It also starts all the timers to make sure they are running in the right thread context. This thread processes the packet and provides the data buffer to the callback function of the application. This thread also takes the application data if application wants to send data over Ethernet, processes the data and calls the low level transmit function registered with the stack. Refer the code located in the file tcpip.c that is part of the lwIP stack for more details.
3. Application task: This task handles all the application layer activities. This task will register the call back functions with the TCP/IP stack and gets the data through the call back functions. This task is responsible for parsing the packet as per the application requirements, forming the output data

packet if required, and sending the packet to the TCP/IP stack. The examples of this task are HTTP, FTP, and, TFTP etc.

Following is the sample code for the application task with Netconn API:

```
portTASK_FUNCTION( vBasicWEBServer, pvParameters )
{
    struct netconn *pxHTTPListener, *pxNewConnection;

    /* Create a new tcp connection handle */
    pxHTTPListener = netconn_new( NETCONN_TCP );
    netconn_bind(pxHTTPListener, NULL, webHTTP_PORT );
    netconn_listen( pxHTTPListener );
    /* Loop forever */
    for(;;)
    {
        ...
        ...
        for( ;; )
        {
            /* Wait for a first connection. */
            pxNewConnection = netconn_accept(pxHTTPListener);

            if(pxNewConnection != NULL)
            {
                /* Parses the HTML Request and send the data to the TCP/IP stack */
                prvweb_ParseHTMLRequest(pxNewConnection);
            } /* end if new connection */
        }
    } /* end infinite loop */
}
```

Using lwIP without RTOS

lwIP provides the configurable option `RAW_API` to use the lwIP stack without any OS support. In this case it is the application code responsibility to queue and de-queue the requests coming to the stack and any synchronization issues. In this mode all the requests and timer-based API calls are to be handled in the main 'while loop' by the application code.

If you are not using the OS then you need to make sure the following functions are called periodically, these functions are called based on the time count of the GPT:

1. `tcp_timer()`
2. `etharp_timer()`

The lwIP main loop without OS sample code as follows:

```
do
{
    /* Period TCP/IP stack calls based on Timer timeout */
    PERIODIC(tcp_timer, TCP_TMR_INTERVAL, tcp_tmr());
    PERIODIC(arp_timer, ARP_TMR_INTERVAL, etharp_tmr());

    /* Poll for the input packet and pass the packet to TCP/IP stack*/
    ethernetif_input(NULL);
}while( 1 );
```

Configuring the lwIP TCP/IP Stack

lwIP TCP/IP stack can be configured for the required protocols, with and without OS, for various memory sizes based on the memory available to execute etc. All the configurable options are present with default values in the file `opt.h` file. You need to edit these options in `lwipopts.h` files according to your requirement.

For example, option for OS and without OS:

`NO_OS` this has to be set to 1 if you are not using the OS otherwise you need to set it to 0.

lwIP API Reference

lwIP provides the three types of the APIs to the TCP/IP applications:

- Raw API: Can be used without the OS. It is the direct interface to the lwIP stack.
- Netconn API: Has to be used with the OS. Provides more abstraction and thread level synchronization. These APIs are called sequential API.
- Socket API: Has to be used with the OS. Provides more abstraction and thread level synchronization. These APIs are similar to BSD socket API.

RAW API

This API is used with the systems not using the OS. The raw TCP/IP interface allows the application program to integrate better with the TCP/IP code. Program execution is event based by having callback functions being called from within the TCP/IP code. The TCP/IP code and the application program both run in the same thread. The raw TCP/IP interface is not only faster in terms of code execution time but is also less memory intensive.

Following is the reference to the RAW API in the lwIP package:

`doc/rawapi.txt`

Sequential API

Following is the sequence of the APIs to be used along with RTOS port of lwIP:

1. `netconn_new(NETCONN_TCP);`
2. `netconn_bind(pxHTTPListener, NULL, webHTTP_PORT);`
3. `netconn_listen(pxHTTPListener);`
4. `netconn_accept(pxHTTPListener);`
5. `netconn_recv(pxNetCon);`
6. `netconn_write(pxNetCon, webHTTP_OK, (u16_t) strlen(webHTTP_OK), NETCONN_COPY);`

Socket API

Following is the sequence of the APIs to be used along with RTOS port of lwIP:

1. `socket()` - get socket descriptor
2. `bind()` - associate socket with a port on the local machine
3. `connect()` - connect to a remote machine
4. `listen()` - wait for someone to connect to the local machine
5. `accept()` - acknowledge remote connection attempt and create a new descriptor
6. `send()` - send data to remote machine
7. `recv()` - receive data from remote machine
8. `close()` - close opened socket

Detailed Description of the Design Examples

This application note provides following example designs to demonstrate the usage of lwIP TCP/IP stack on SmartFusion Evaluation Kit Board and SmartFusion Development Kit Board.

- lwIP-based Webserver with FreeRTOS
- lwIP-based TFTP server, Webserver and FatFs File system without OS using RAW lwIP API

Webserver based on lwIP stack with FreeRTOS

This example design demonstrates the webserver application on lwIP using the FreeRTOS for the OS functionalities. In this method, sequential API of lwIP stack (Netconn API) is used for the webserver application.

Figure 3 shows the architecture of webserver application based on lwIP with FreeRTOS.

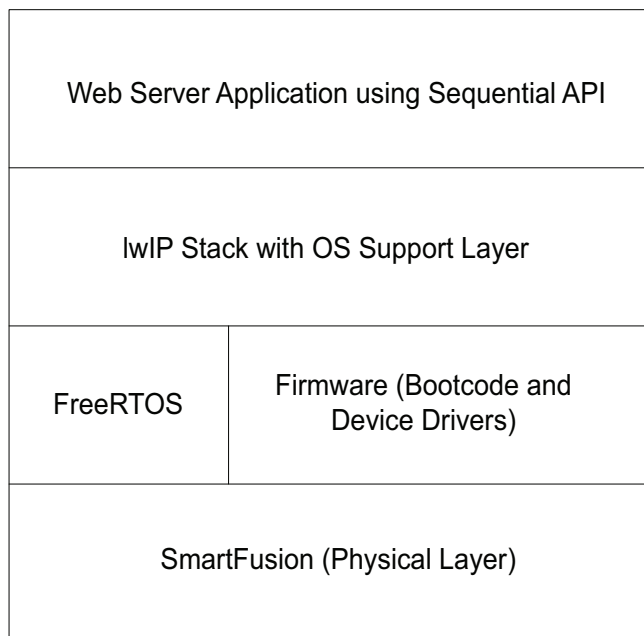


Figure 3 • Architecture of Webserver Application based on lwIP with FreeRTOS

Running the SoftConsole IDE Project in Debug Mode

Figure 4 shows the directory structure of webserver application with lwIP and FreeRTOS.

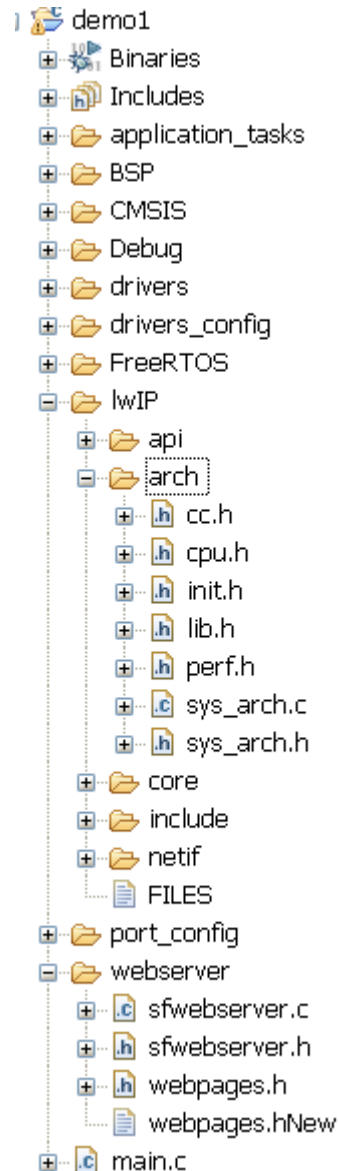


Figure 4 • Directory Structure of Webserver Application with lwIP and FreeRTOS

Following are the some of the important files in the porting of lwIP stack on to SmartFusion with FreeRTOS:

- Sys_arch.c and sys_arch.h files implement the OS support layer for the lwIP stack to use the OS features. These files implements the wrapper function calls to manage the tasks, semaphores and message queues etc.
- Sfwebserver.c and sfwebserver.h files implement the SmartFusion webserver based on lwIP using the NetConn sequential API
- Ethernetif.c files implement the low level HW access wrappers for the transmitting and receiving the Ethernet packets on the MAC layer.

Program the SmartFusion Evaluation Kit Board or SmartFusion Development Kit Board with the generated/provided *.pdb file (refer to "Appendix A – Design Files" on page 19) using FlashPro, and then power cycle the board.

Invoke the SoftConsole IDE, browse for the SoftConsole project folder (refer to "Appendix A – Design Files" on page 19) and launch the debugger. Start a HyperTerminal session with the baud rate set to 57600, 8 data bits, 1 stop bit, no parity, and no flow control. If your computer does not have HyperTerminal program, use any free serial terminal emulation program like PuTTY or Tera Term. Refer to the [Configuring Serial Terminal Emulation Programs tutorial](#) for configuring the HyperTerminal, Tera Term, or PuTTY.

This design example can run in both static IP and Dynamic IP mode. If static IP mode (SmartFusion is directly connected to the development PC without the open network) is required then comment the line #define OPEN_IP in "..apps_layer/web_task.c" file as shown in Figure 5.

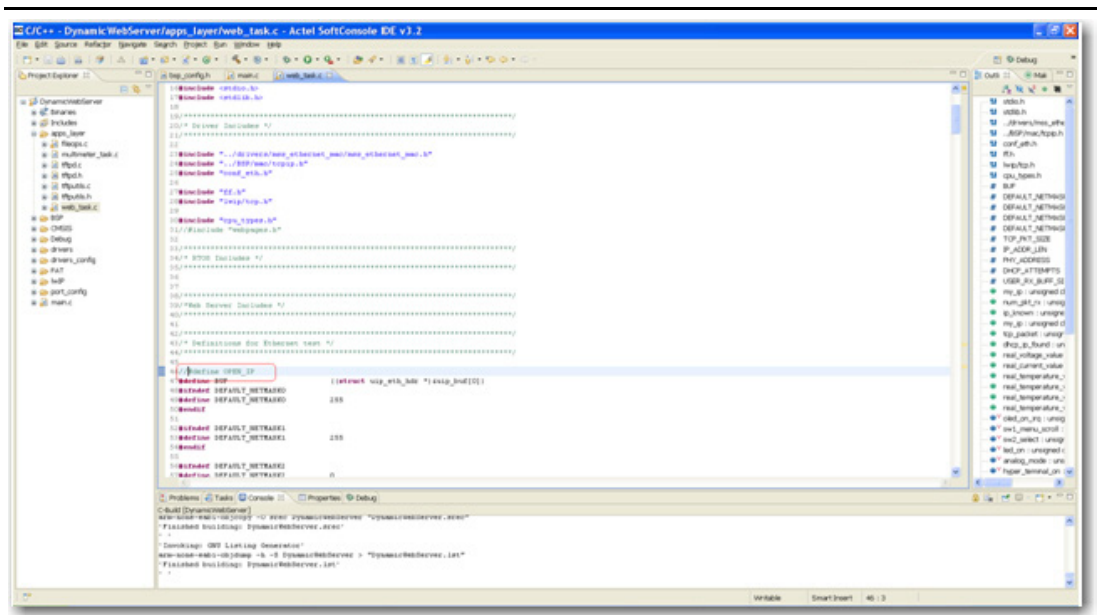


Figure 5 • Static and Dynamic IP Setting for MAC

For static IP mod: If the device is connected in static ip mode, IP address will be 192.168.0.14. Webpage can be accessed by typing <http://192.168.0.14> in internet explorer.

Only for Static IP mode, change the Host PC TCP/IP settings as shown in Figure 6.

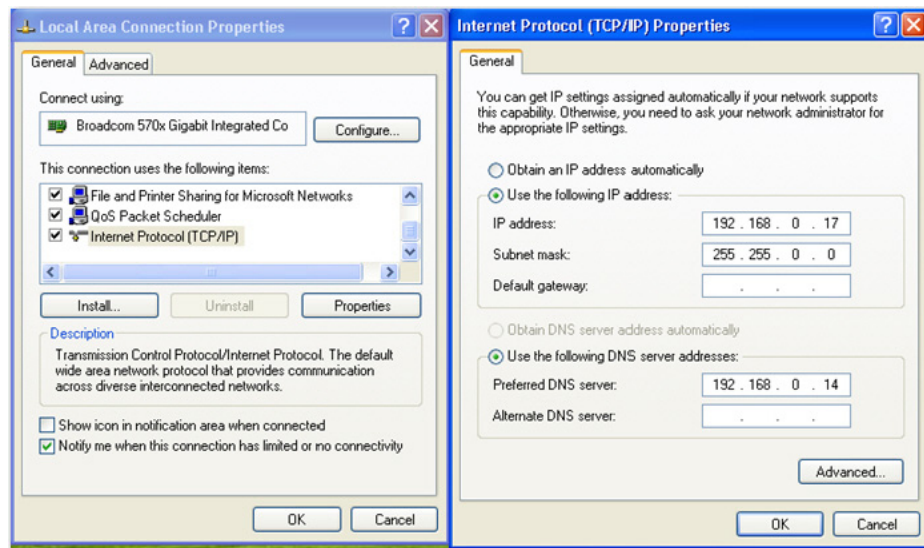


Figure 6 • Changing the IP Address Setting for the Static IP Mode in Host PC

Once these settings are verified, then compile, load and run the design using the SoftConsole. Follow the help provided in the Terminal emulation program. The IP address of the Kit board will be displayed on the Hyperterminal. Use this IP address to access the SmartFusion Webserver. Figure 7 shows the Hyperterminal Help menu for lwIP with FreeRTOS Demo.

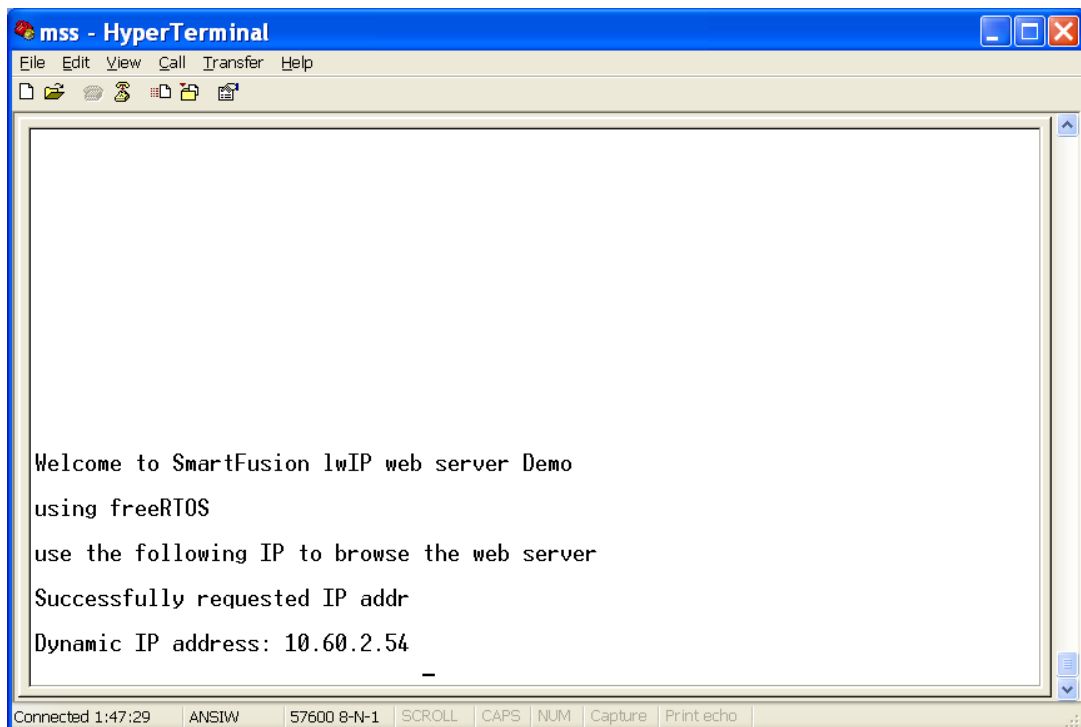


Figure 7 • Hyperterminal Help Menu for lwIP with FreeRTOS Demo

Figure 8 shows the SmartFusion Webserver with lwIP and FreeRTOS.

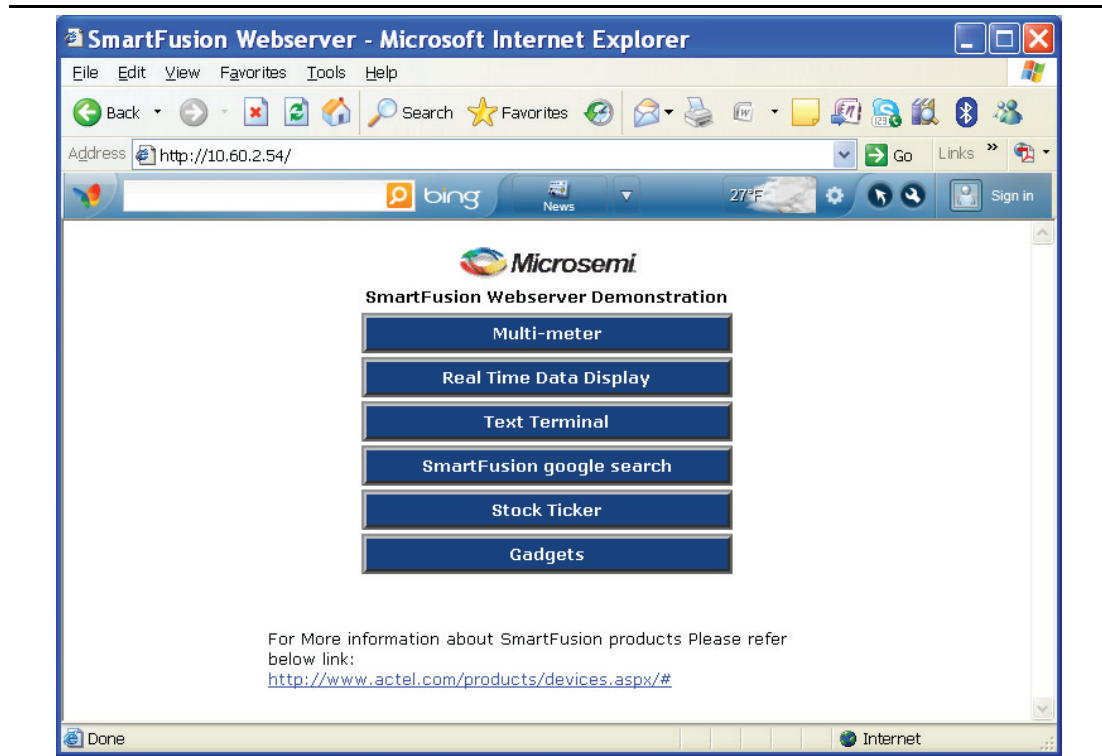


Figure 8 • SmartFusion Webserver with lwIP and FreeRTOS

TFTP server, FatFs file system on SPI Flash and Webserver based on lwIP Stack

This method uses the RAW API of the lwIP stack. Periodic TCP/IP stack calls and polling for the input packets are handled in a single main while loop.

This method demonstrates TFTP and webserver applications using the web pages stored at SPI Flash of the SmartFusion Evaluation Kit Board and SmartFusion Development Kit Board. FatFs file system is used to store the web pages in SPI Flash. TFTP application on lwIP is used to get the web pages from Ethernet and store in SPI Flash. These web pages are used by the SmartFusion webserver application on lwIP stack.

Figure 9 shows the architecture of webserver and TFTP server application based on lwIP and FatFs on SPI Flash.

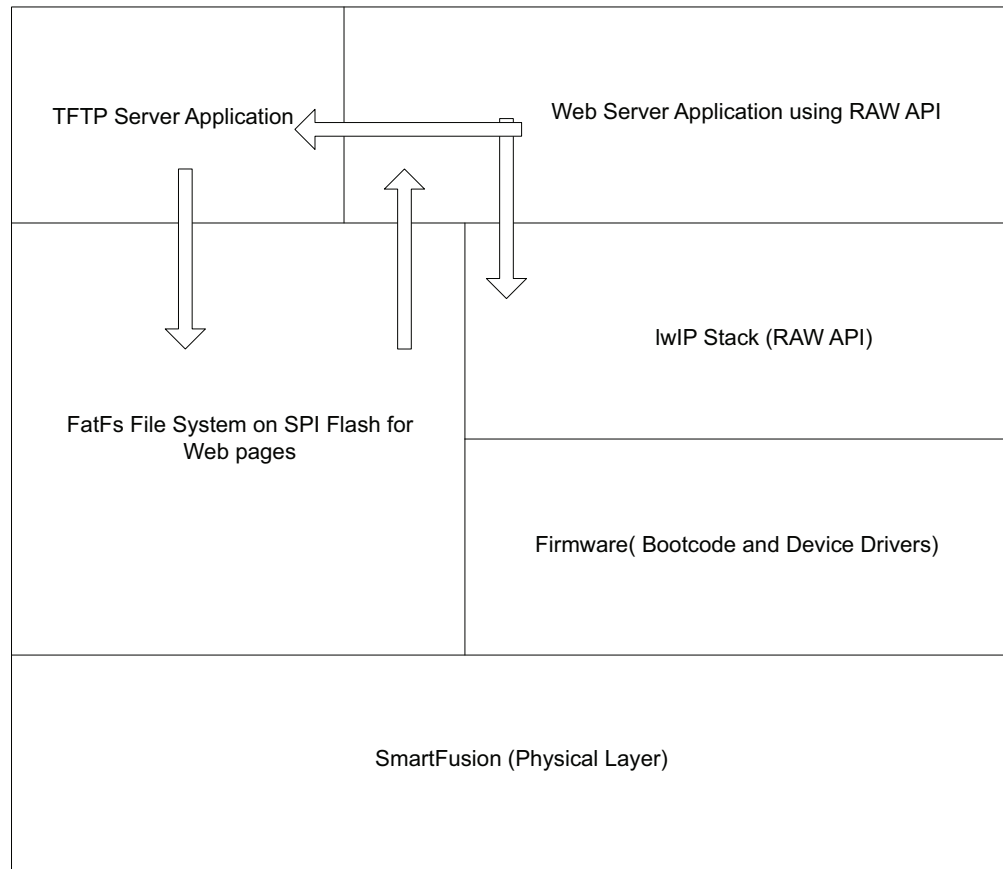


Figure 9 • Architecture of Webserver and TFTP Server Application based on lwIP and FatFs on SPI Flash

Running the SoftConsole IDE project in debug mode

Figure 10 shows the directory structure for webserver application with RAW lwIP API and FatFs.

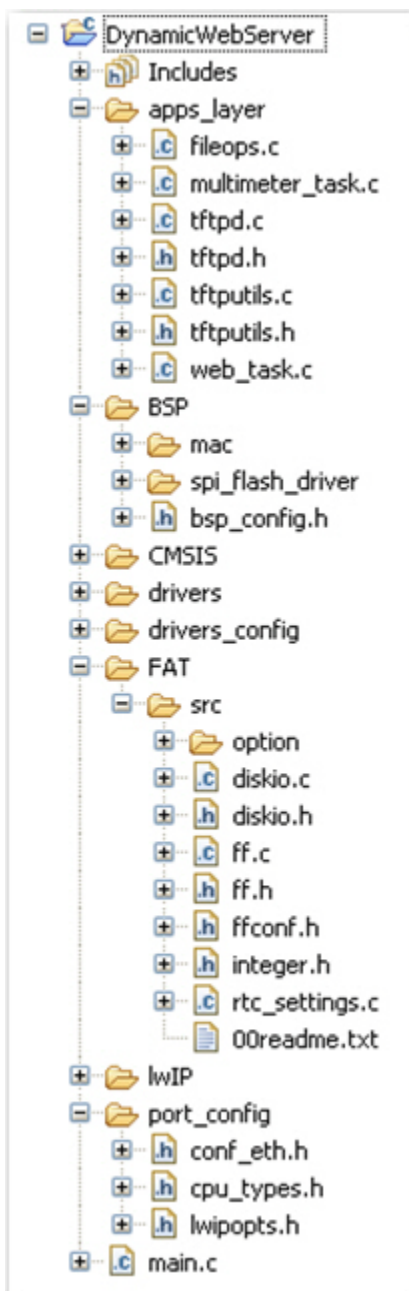


Figure 10 • Directory Structure for Webserver Application with RAW lwIP API and FatFs

The softconsole package contains the following components:

1. Apps_layer
 - tftpd.c & tftpd.h files correspond to the TFTP protocol on lwIP stack. These files implement and explain the usage of the UDP protocol of the lwIP TCP/IP stack.
 - web_task.c file has functions that correspond to the HTTP webserver on lwIP stack. This file does initialization of MAC, gets an IP address to the board and shows the implementation of using the TCP layer of lwIP stack.

- fileops.c: This file implements some of the basic functions that correspond to the usage of FAT File System on SPI Flash like opening a file, copying/moving a file, listing the FS entries (files & directories), status of the FS like how much space is used, number of files and directories etc.
 - multimeter_task.c: This file implements the ACE functionality to get the latest current, voltage, and temperature.
2. FAT File System
- Rtc_settings.c: Implements the RTC time functionality. As the RTC in SmartFusion is a simple counter, we need some logic to derive the actual calendar time from this counter. This file implements the calendar time from the simple RTC counter.
 - Diskio.c & diskio.h: This is the hardware interface layer for the file system. These files implement the SPI Flash access calls for all the HW functionality required by the FAT File system to store the Files & Directories on the SPI Flash.
3. lwIP Stack
- ..\lwIP\netif\ethernet.c & ethernet.h files implement the basic initialization of the lwIP stack and adding our MAC interface as the HW Ethernet interface for the lwIP stack.
 - ..\lwIP\netif\ethernetif.c file implements the glue layer for the TX and RX packets of the lwIP to the Ethernet MAC driver.
 - ..\portconfig\lwipopts.h file is the configuration interface for the lwIP stack.

Before compiling and loading the design check the following settings according to the board in use for SPI Flash interface settings.

Open the “bsp_config.h” and modify the settings based on the board. If you are using A2F-EVAL-KIT modify settings as shown in [Figure 11](#).

```
18/* Configuration for the SPI Flash */
19#define SPI_FLASH_ON_SF_DEV_KIT 0
20#define SPI_FLASH_ON_SF_EVAL_KIT 1
```

Figure 11 • bsp_config.h Settings for A2F-EVAL-KIT

If you are using A2F500-DEV-KIT, modify settings as shown in [Figure 12](#).

```
18/* Configuration for the SPI Flash */
19#define SPI_FLASH_ON_SF_DEV_KIT 1
20#define SPI_FLASH_ON_SF_EVAL_KIT 0
```

Figure 12 • bsp_config.h Settings for A2F500-DEV-KIT

Once these settings are verified then compile, load, and run the design using the SoftConsole. Follow the help provided in the Terminal emulation program. The IP address of the Kit will be displayed on the Hyperterminal. Use this IP address for both TFTP server and Webserver.

Figure 13 shows the welcome message on HyperTerminal and for initial settings.

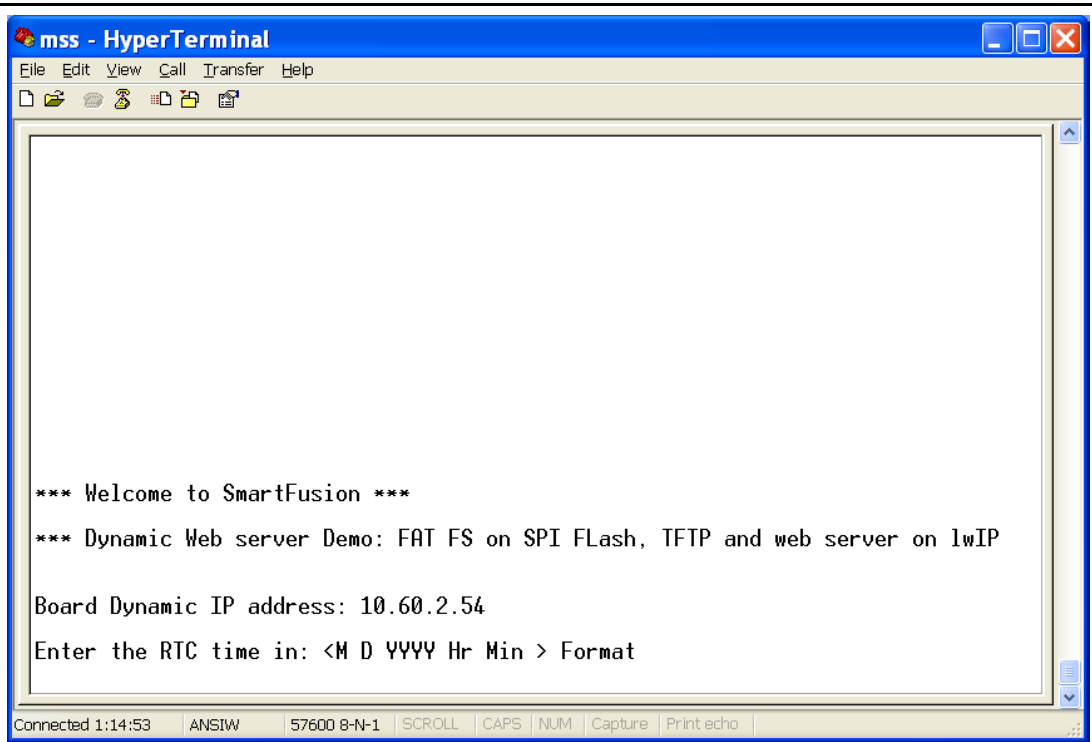


Figure 13 • Hyperterminal Menu for the SmartFusion demo with RAW lwIP API and FatFs

Enter the time as shown in Figure 14.

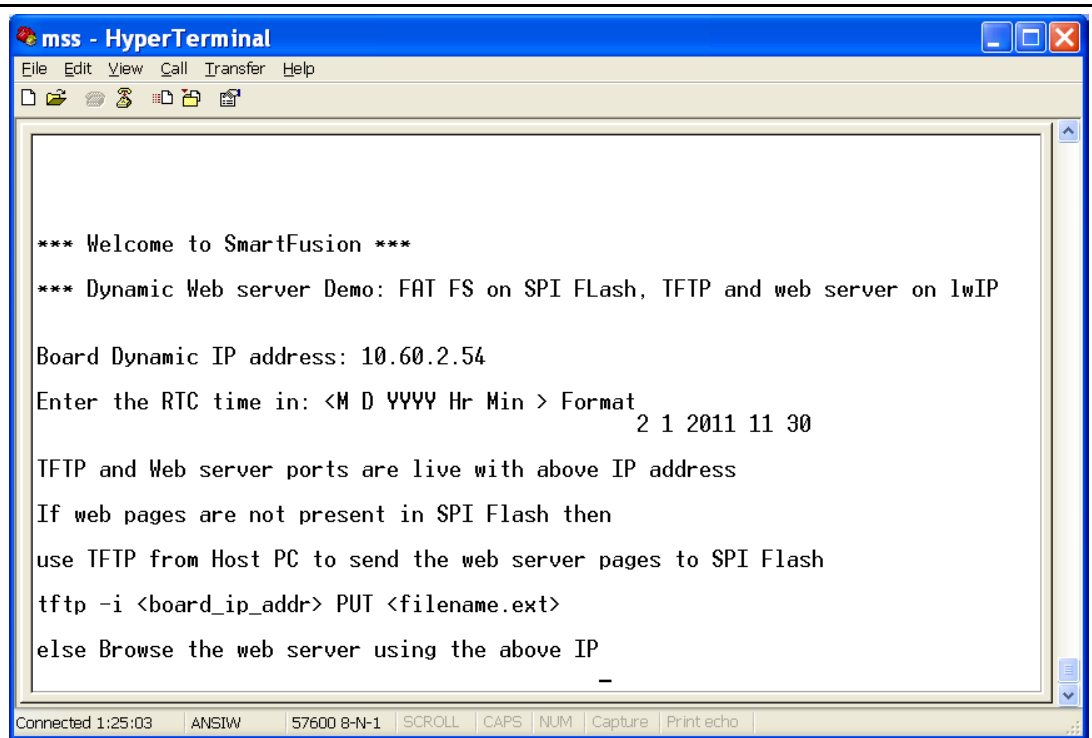
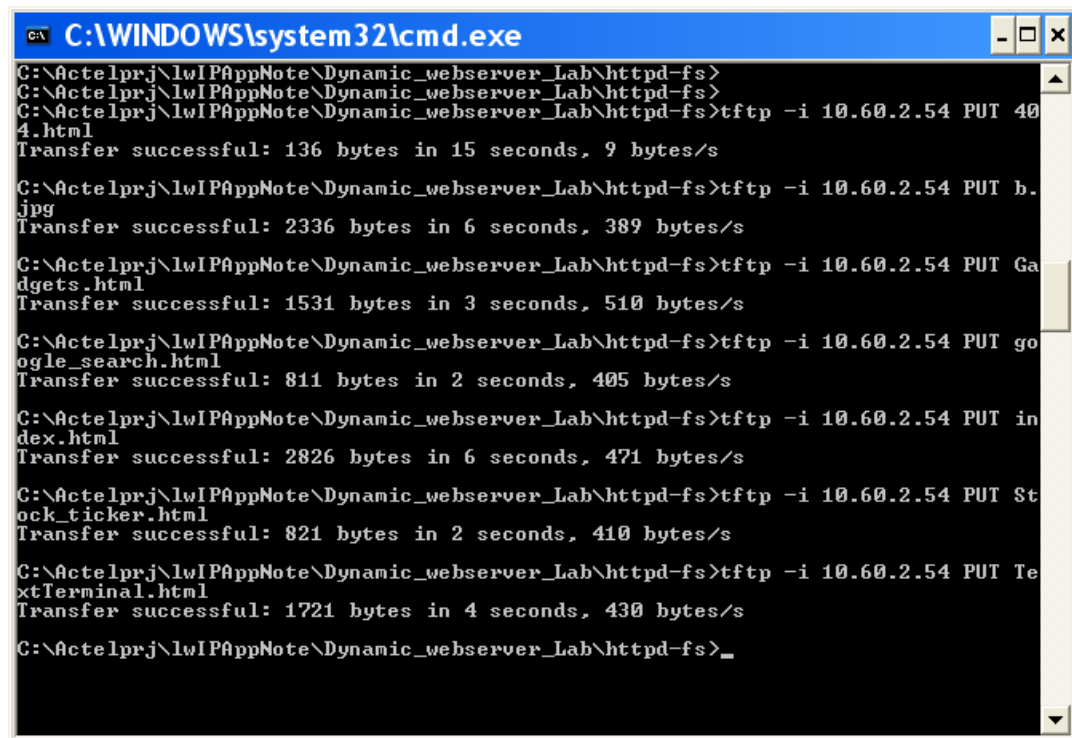


Figure 14 • Setting the Time for the RTC for FatFs File System

When this menu pops-up, send the HTML webserver files from Host PC to the SPI Flash using the tftp command as shown in Figure 14 on page 16. Figure 15 shows various tftp commands to send the entire webserver from PC to target.



```

C:\WINDOWS\system32\cmd.exe
C:\Actelprj\lwIPAppNote\Dynamic_webserver_Lab\httpd-fs>
C:\Actelprj\lwIPAppNote\Dynamic_webserver_Lab\httpd-fs>
C:\Actelprj\lwIPAppNote\Dynamic_webserver_Lab\httpd-fs>tftp -i 10.60.2.54 PUT 40
4.html
Transfer successful: 136 bytes in 15 seconds, 9 bytes/s
C:\Actelprj\lwIPAppNote\Dynamic_webserver_Lab\httpd-fs>tftp -i 10.60.2.54 PUT b.
jpg
Transfer successful: 2336 bytes in 6 seconds, 389 bytes/s
C:\Actelprj\lwIPAppNote\Dynamic_webserver_Lab\httpd-fs>tftp -i 10.60.2.54 PUT Ga
dgets.html
Transfer successful: 1531 bytes in 3 seconds, 510 bytes/s
C:\Actelprj\lwIPAppNote\Dynamic_webserver_Lab\httpd-fs>tftp -i 10.60.2.54 PUT go
ogle_search.html
Transfer successful: 811 bytes in 2 seconds, 405 bytes/s
C:\Actelprj\lwIPAppNote\Dynamic_webserver_Lab\httpd-fs>tftp -i 10.60.2.54 PUT in
dex.html
Transfer successful: 2826 bytes in 6 seconds, 471 bytes/s
C:\Actelprj\lwIPAppNote\Dynamic_webserver_Lab\httpd-fs>tftp -i 10.60.2.54 PUT St
ock_ticker.html
Transfer successful: 821 bytes in 2 seconds, 410 bytes/s
C:\Actelprj\lwIPAppNote\Dynamic_webserver_Lab\httpd-fs>tftp -i 10.60.2.54 PUT Te
xtTerminal.html
Transfer successful: 1721 bytes in 4 seconds, 430 bytes/s
C:\Actelprj\lwIPAppNote\Dynamic_webserver_Lab\httpd-fs>_

```

Figure 15 • TFTP Commands to send the Webserver files from Host PC to SPI Flash

After sending the webserver files using the TFTP successfully, access the webserver using the board IP address from Host PC. All the files sent in the previous step that are stored in SPI Flash will be served for the web client requests by using the lwIP RAW API and SPI Flash File system.

Figure 16 shows the webserver after successful request of the client.

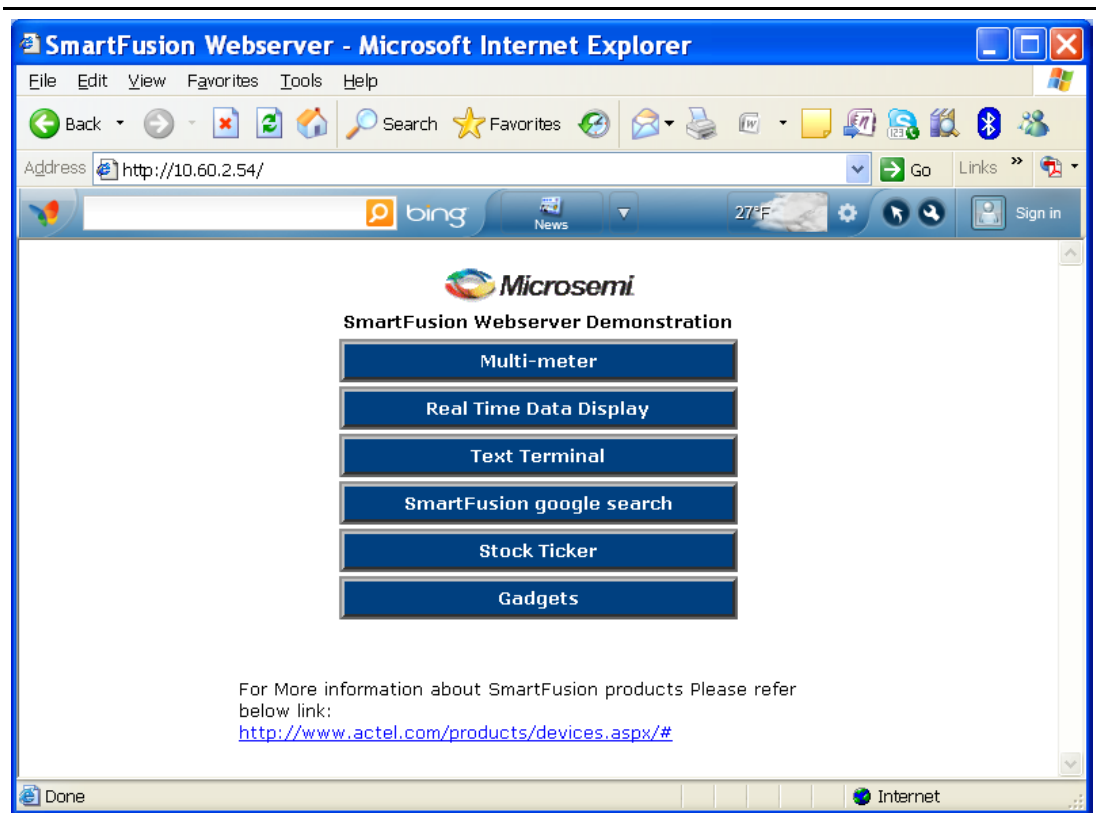


Figure 16 • SmartFusion Webserver with RAW lwIP API

Conclusion

This application note describes the porting of the lwIP TCP/IP stack with and without RTOS to SmartFusion. This application note also demonstrates lwIP applications such as Webserver that serves web pages from SPI Flash with FatFs file system and TFTP Server on SmartFusion Evaluation Kit Board and SmartFusion Development Kit Board.

Appendix A – Design Files

Design files are available on the Microsemi SoC Product Groups website for download:

www.actel.com/download/rsc/?f=A2F_AC365_DF

The design file consists of Libero IDE Verilog and VHDL projects, SoftConsole software project in debug mode, and programming file (*.PDB) for A2F500-DEV-KIT and A2F-EVAL-KIT. Refer to the Readme.txt file included in the design file for directory structure and description.



Microsemi Corporate Headquarters
2381 Morse Avenue, Irvine, CA 92614
Phone: 949-221-7100 · Fax: 949-756-0308
www.microsemi.com

Microsemi Corporation (NASDAQ: MSCC) offers the industry's most comprehensive portfolio of semiconductor technology. Committed to solving the most critical system challenges, Microsemi's products include high-performance, high-reliability analog and RF devices, mixed signal integrated circuits, FPGAs and customizable SoCs, and complete subsystems. Microsemi serves leading system manufacturers around the world in the defense, security, aerospace, enterprise, commercial, and industrial markets. Learn more at www.microsemi.com.

© 2011 Microsemi Corporation. All rights reserved. Microsemi and the Microsemi logo are trademarks of Microsemi Corporation. All other trademarks and service marks are the property of their respective owners.