

SmartFusion cSoC: 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 cSoC	2
Configuring the lwIP TCP/IP Stack	7
lwIP API Reference	7
Detailed Description of the Design Examples	8
Release Mode	19
Conclusion	19
Appendix A – Design Files	20
List of Changes	20

Introduction

The SmartFusion[®] customizable system-on-chip (cSoC) FPGA devices integrate FPGA technology with the hardened ARM[®] Cortex[®]-M3 processor 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, the Philips Inter-Integrated Circuit (I²C), serial peripheral interface (SPI), 10/100 Ethernet controller, real-time counter (RTC), general purpose input/output (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 the SmartFusion cSoC is a high-speed media access control (MAC) Ethernet controller with the following features:

- Carrier sense multiple access with the 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 the external RAM and TX/RX FIFOs.

For more information about 10/100 Ethernet MAC interface, refer to the [SmartFusion Microcontroller Subsystem User's Guide](#).

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

Introduction to lwIP TCP/IP Stack

The lwIP is an implementation of the light weight TCP/IP stack. It was developed by [Adam Dunkels](#) at the [Swedish Institute of Computer Science \(SICS\)](#). The lwIP stack is more suitable for the embedded systems because of small data and code size requirements. It can be used with or without OS. The core of the 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.

For more information on the design and implementation, refer to the following location:

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

The lwIP is 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 cSoC

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

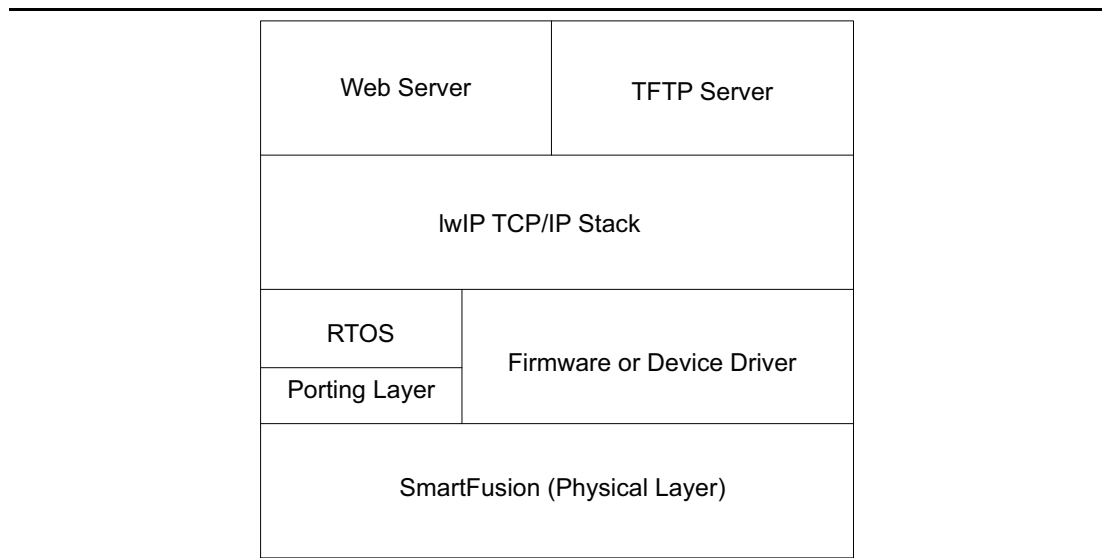


Figure 1 • Typical Network Application Architecture on SmartFusion cSoC Using lwIP Stack

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 cSoC 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 the 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 the 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 acknowledgment

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

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

Using lwIP with RTOS Support Layer

The Socket API or NetConn API of the lwIP can be used along with the RTOS. This method needs to have the RTOS support for the inter task communication mechanism. The lwIP with RTOS internally takes care of periodic calls required and hence the application does not need to call the periodic calls of the 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 an IP or ARP packet. Following is the sample code for the task that is implemented in the 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 );
                }
            }
        }
```

```

        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 the SmartFusion cSoC with the FreeRTOS.

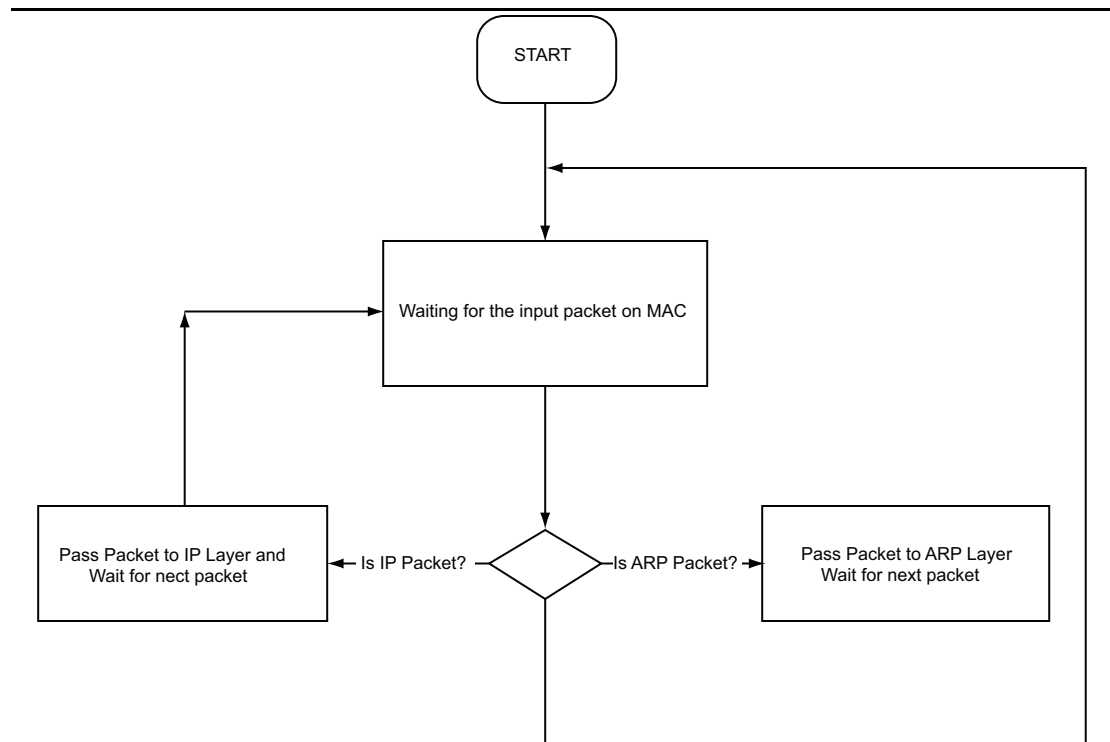


Figure 2 • FlowChart of the lwIP Stack Running on SmartFusion cSoC with FreeRTOS

2. TCPIP Thread: This is the main lwIP thread. It has exclusive access to the lwIP core functions. Other threads (for example, application layer threads) communicate with this thread using the message boxes. It also starts all the timers to make sure that 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 the application wants to send the data over the 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 information.

3. Application Task: This task handles all the application layer activities. This task registers 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 and so on.

Following is the sample code for the application task with the 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

The lwIP provides the configurable option *RAW_API* to use the lwIP stack without any OS support. In this case, it is the responsibility of the application code 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 that the following functions are called periodically. These functions are called based on the time count of the general purpose timer (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

The 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 and so on. All the configurable options are present with the default values in the file `opt.h` file. You need to edit these options in the `lwipopts.h` file according to the requirement.

For example, option for OS and without OS:

`NO_OS` need to set to 1, if you are not using the OS. If you are using the OS, set `NO_OS` to 0.

lwIP API Reference

The lwIP provides 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: Need to be used with the OS. Provides more abstraction and thread level synchronization. These APIs are called sequential API.
- Socket API: Need 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 that are not using the OS. The raw TCP/IP interface allows the application program to integrate better with the TCP/IP code. The program execution is an event based on 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 the 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 the 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 the following example designs to demonstrate the usage of lwIP TCP/IP stack on the SmartFusion Development Kit Board.

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

lwIP-based Webserver with FreeRTOS

This example design demonstrates the webserver application on the 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 the lwIP with FreeRTOS.

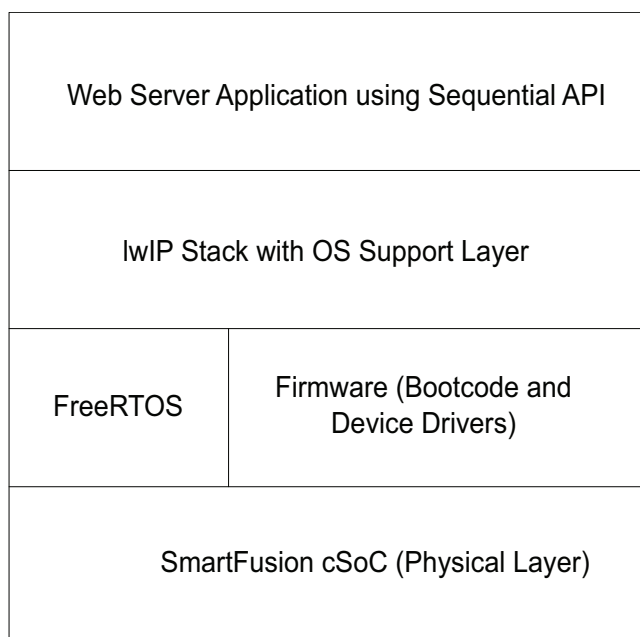


Figure 3 • Architecture of Webserver Application Based on lwIP with FreeRTOS

Running the SoftConsole IDE Project in Debug Mode

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

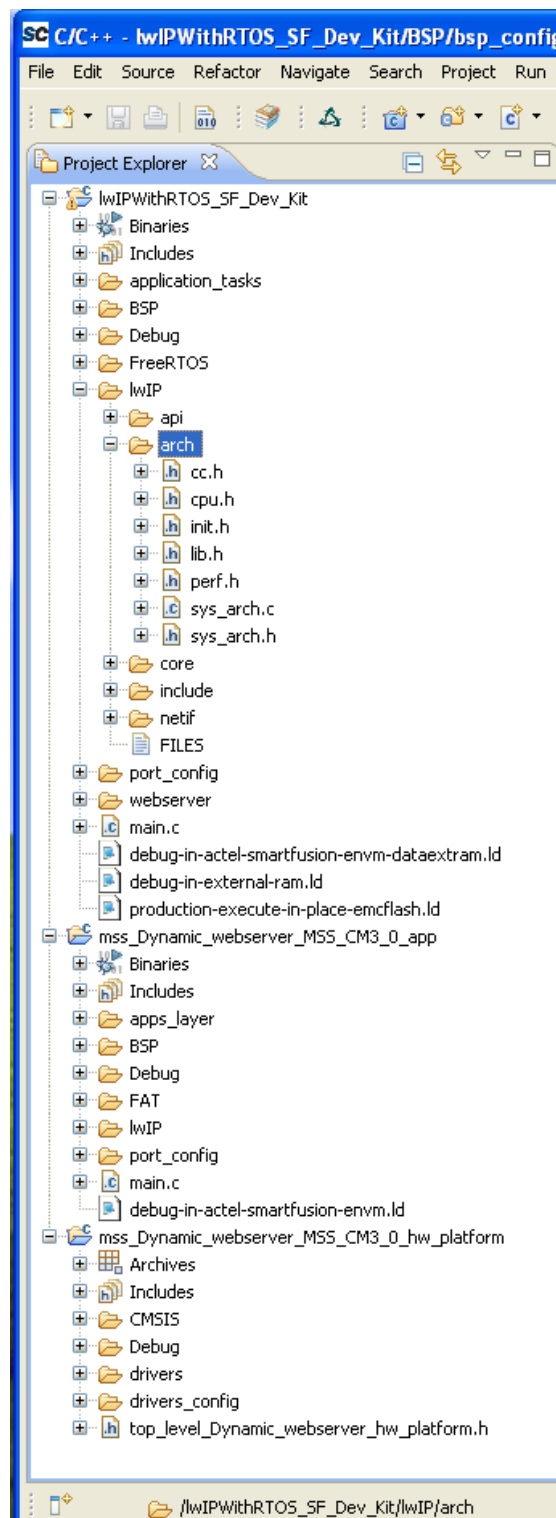


Figure 4 • Directory Structure of Webserver Application with lwIP and FreeRTOS

Following are some of the important files in porting the lwIP stack on to the SmartFusion cSoC 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 implement the wrapper function calls to manage the tasks, semaphores, and message queues and so on.
- Sfwebserver.c and sfwebserver.h files implement the SmartFusion Webserver based on the lwIP using the NetConn sequential API.
- Ethernetif.c files implement the low level HW access wrappers for transmitting and receiving the Ethernet packets on the MAC layer.

Board Settings

The design example works on the SmartFusion Development Kit Board with default board settings. Refer to the following user's guides for default board settings:

- [SmartFusion Development Kit User's Guide](#)

Program the Design

Program the SmartFusion Development Kit Board with the generated or provided *.STAPL file (refer to "[Appendix A – Design Files](#)" on [page 20](#)) using FlashPro and then power cycle the board.

Running the Application

Invoke the SoftConsole IDE, browse for the SoftConsole project folder (refer to "[Appendix A – Design Files](#)" on [page 20](#)) 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 PC does not have the 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 cSoC is directly connected to the development PC without the open network) is required, then comment the line `#define OPEN_IP` in “`..\application_tasks\web_task.c`” file as shown in Figure 5.

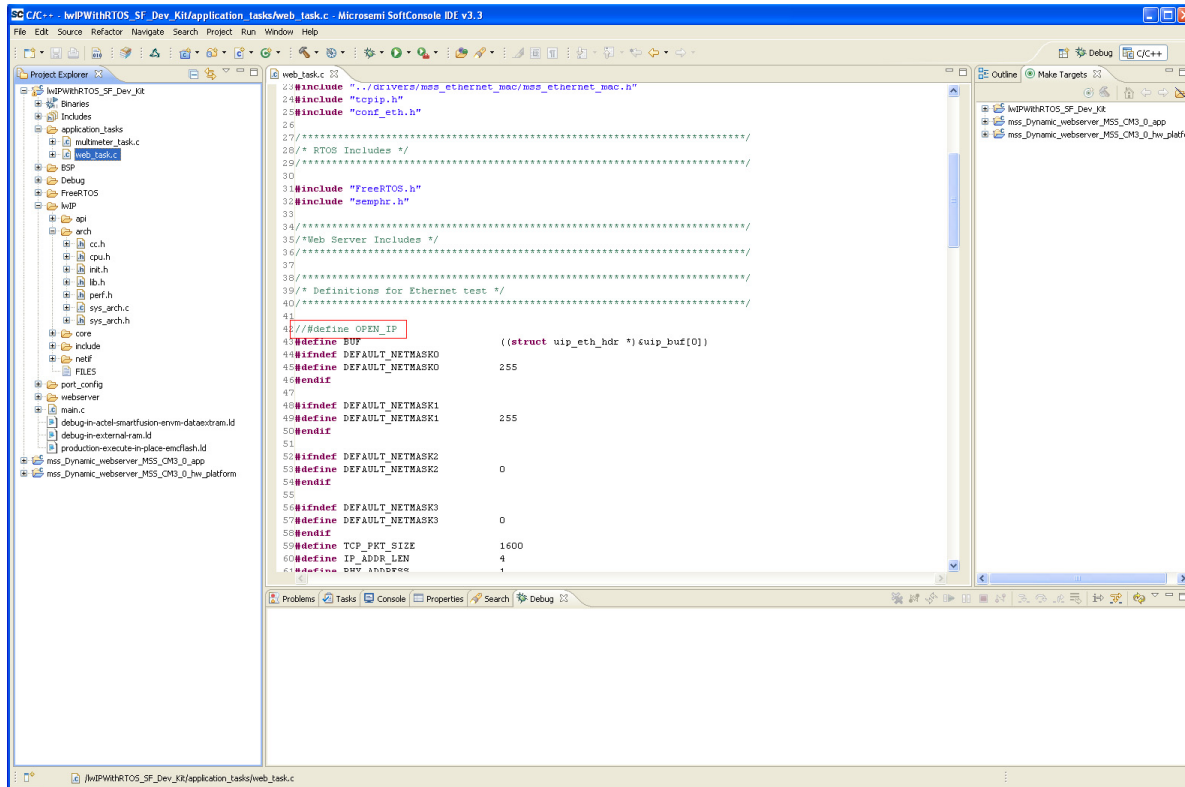


Figure 5 • Static and Dynamic IP Setting for MAC

For Static IP mode: If the device is connected in Static IP mode, the IP address is 192.168.0.14. You can access the web page at <http://192.168.0.14>.

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

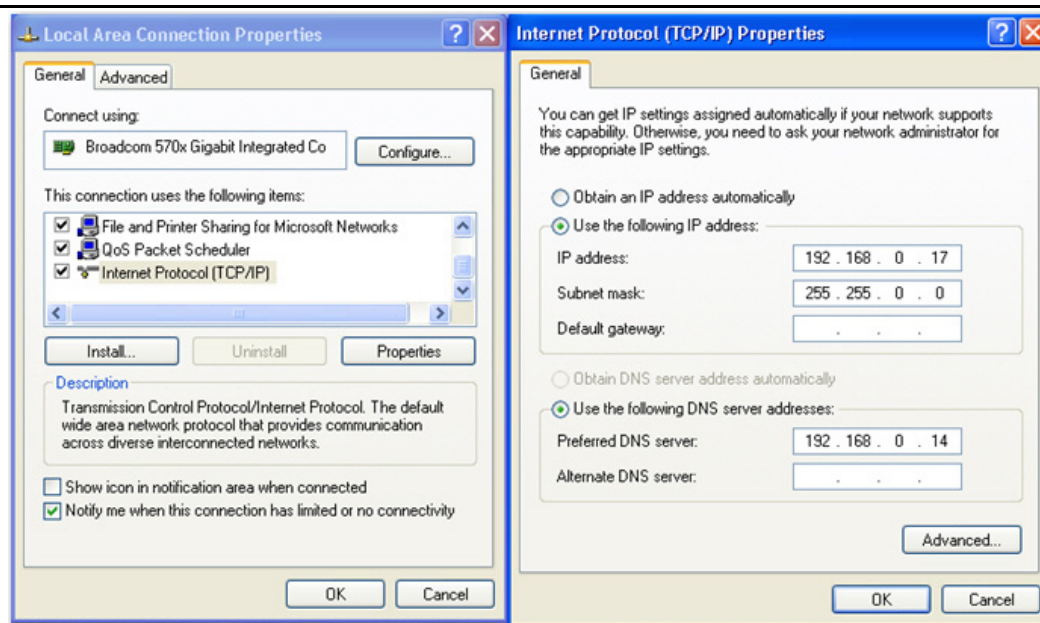


Figure 6 • Changing the IP Address Setting for the Static IP Mode in Host PC

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 is displayed on the Hyperterminal. Use this IP address to access the SmartFusion Webserver. Figure 7 shows the Hyperterminal Help menu for the lwIP with FreeRTOS Demo.

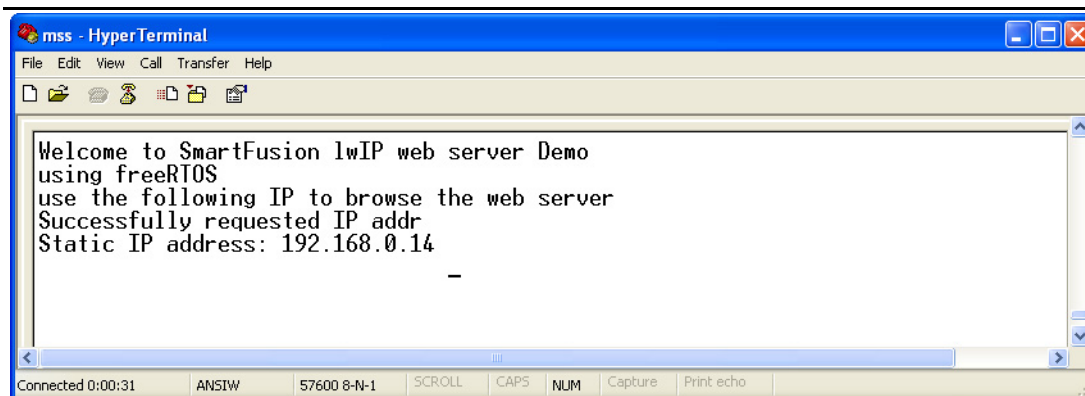


Figure 7 • Hyperterminal Help Menu for lwIP with FreeRTOS Demo

Figure 8 shows the SmartFusion Webserver with the lwIP and FreeRTOS.

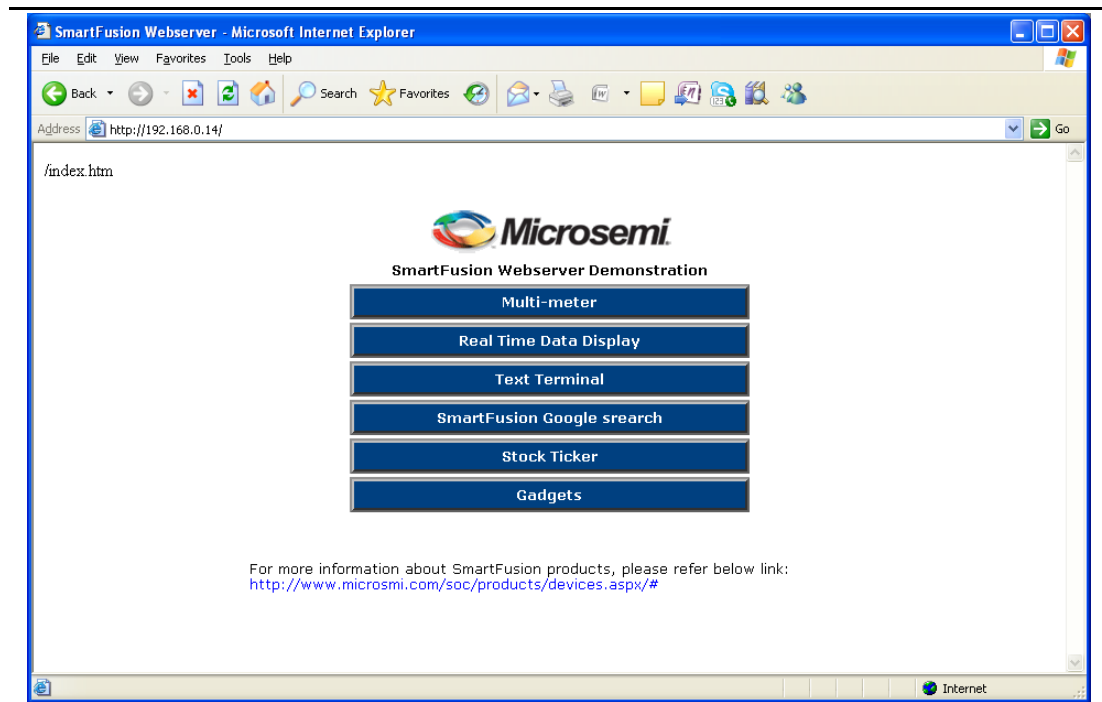


Figure 8 • SmartFusion Webserver with lwIP and FreeRTOS

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

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

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

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

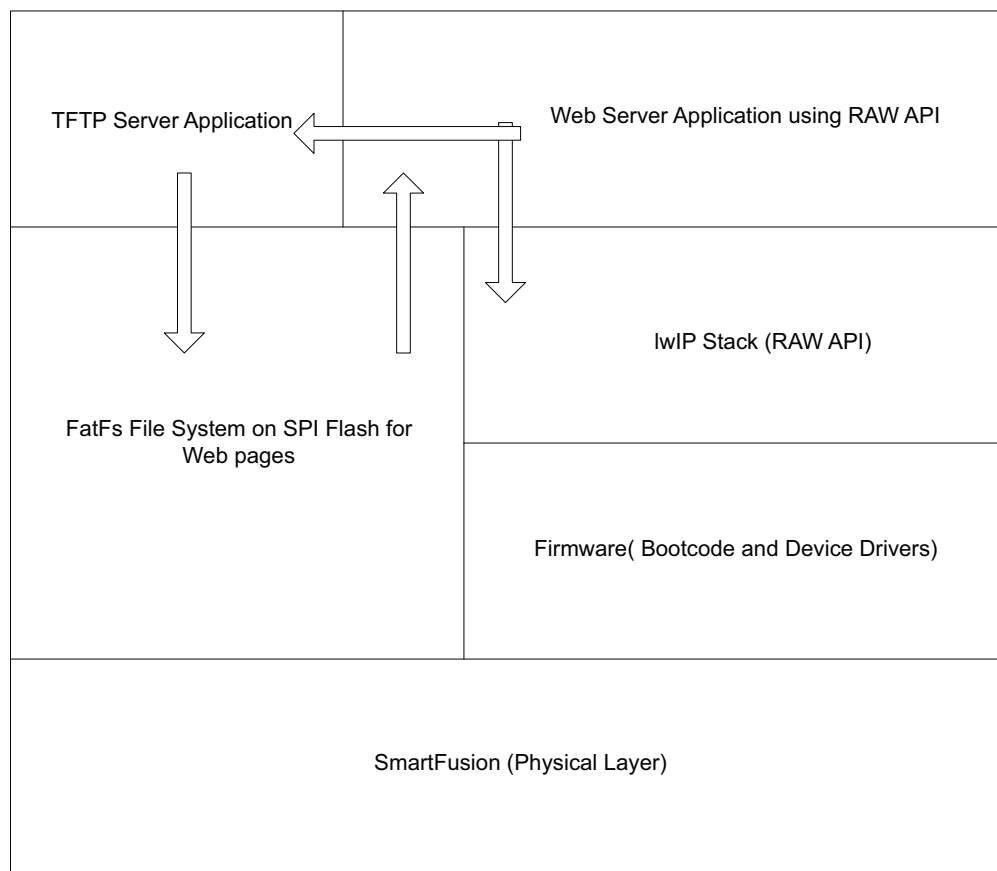


Figure 9 • Architecture of Webserver and TFTP Server Application Based on lwIP and FatFs on SPI Flash

Running the SoftConsole IDE Project in Debug Mode

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

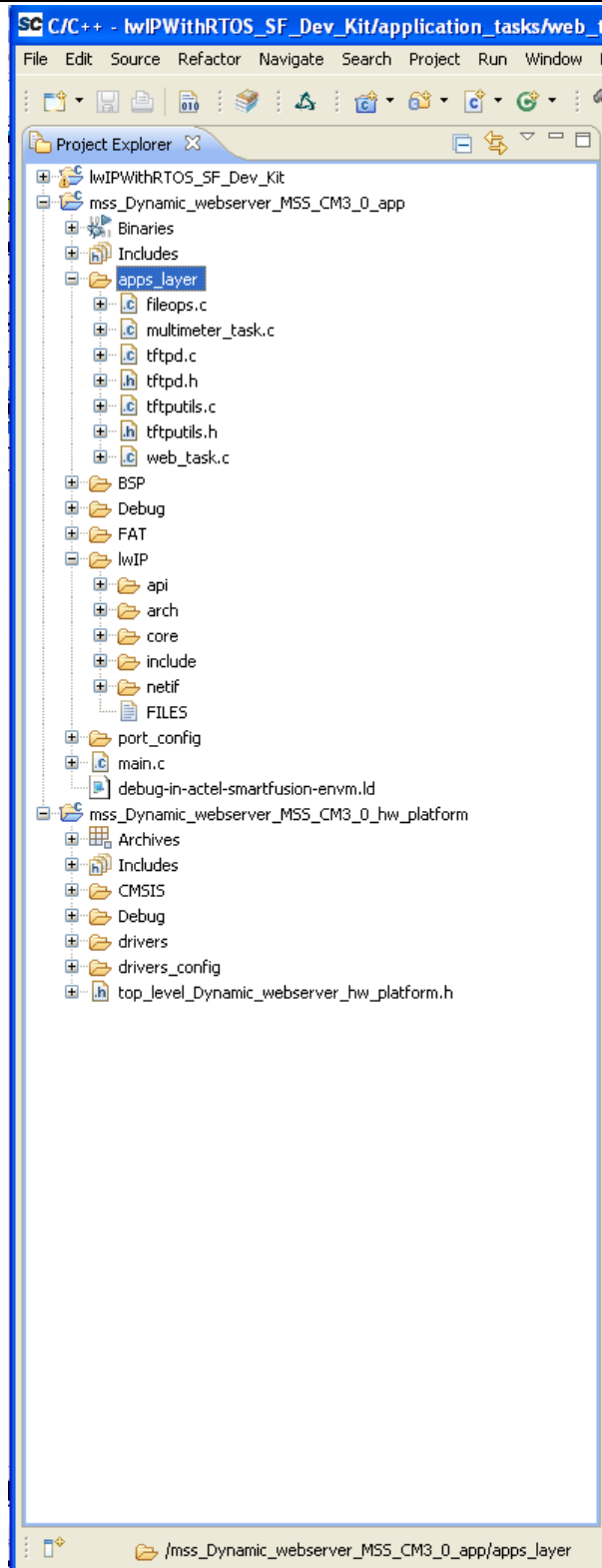


Figure 10 • Directory Structure for Webserver Application with RAW lwIP API and FatFs

The SoftConsole package contains the following components:

1. Apps_layer
 - tftpd.c and tftpd.h files corresponding to the TFTP protocol on the 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 corresponding to the HTTP webserver on the lwIP stack. This file initializes MAC, gets an IP address to the board and shows the implementation of using the TCP layer of the lwIP stack.
 - fileops.c: This file implements some of the basic functions corresponding to the usage of FAT File system on SPI Flash like opening a file, copying or moving a file, listing the FS entries (files and directories), status of the FS like how much space is used, number of files and directories and so on.
 - 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 cSoC is a simple counter, you 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 and 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 and directories on the SPI Flash.
3. lwIP Stack
 - ..\lwIP\netif\ethernet.c and ethernet.h files implement the basic initialization of the lwIP stack and adding the 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 the SPI Flash interface settings.

If you are using A2F500-DEV-KIT, modify the settings as shown in [Figure 11](#).

```
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 11 • 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 is displayed on Hyperterminal. Use this IP address for both TFTP server and webserver.

Figure 12 shows the welcome message on HyperTerminal and for initial settings.

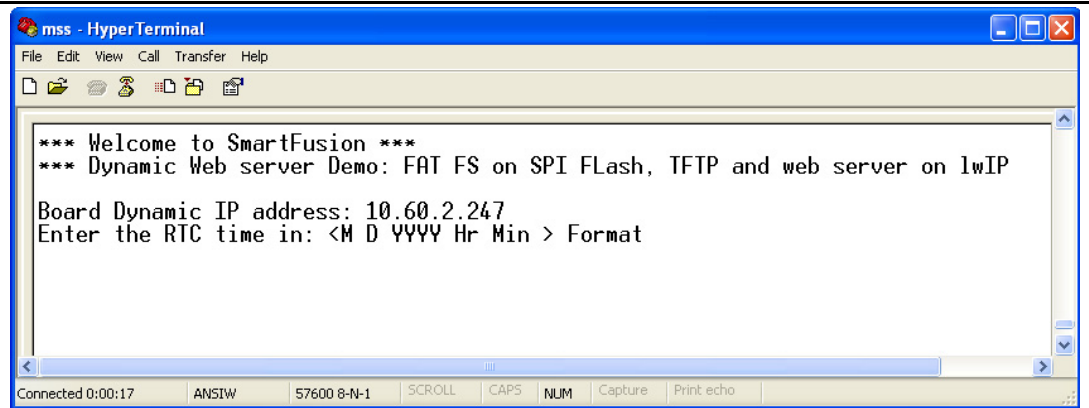


Figure 12 • Hyperterminal Menu for the SmartFusion Demo with RAW lwIP API and FatFs

Enter the time as shown in Figure 13.

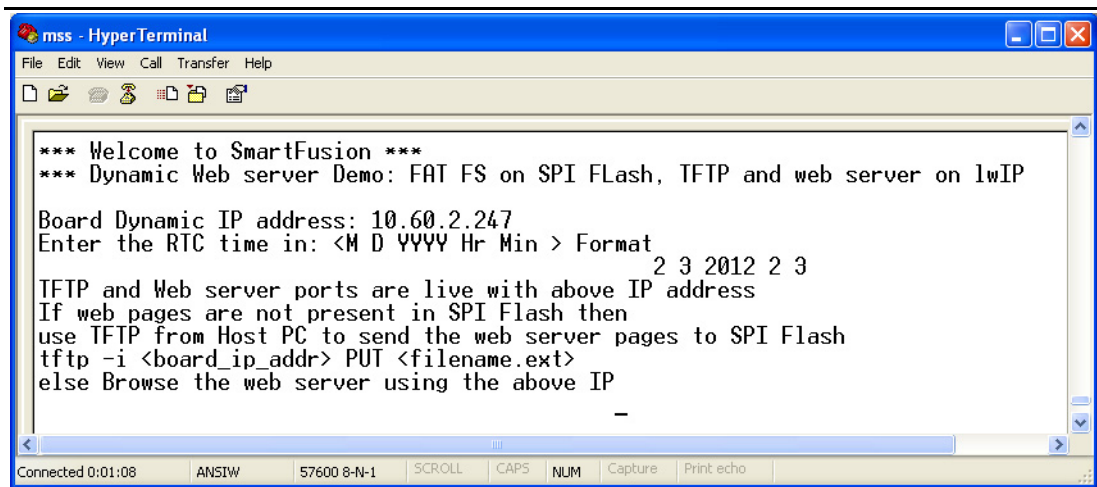
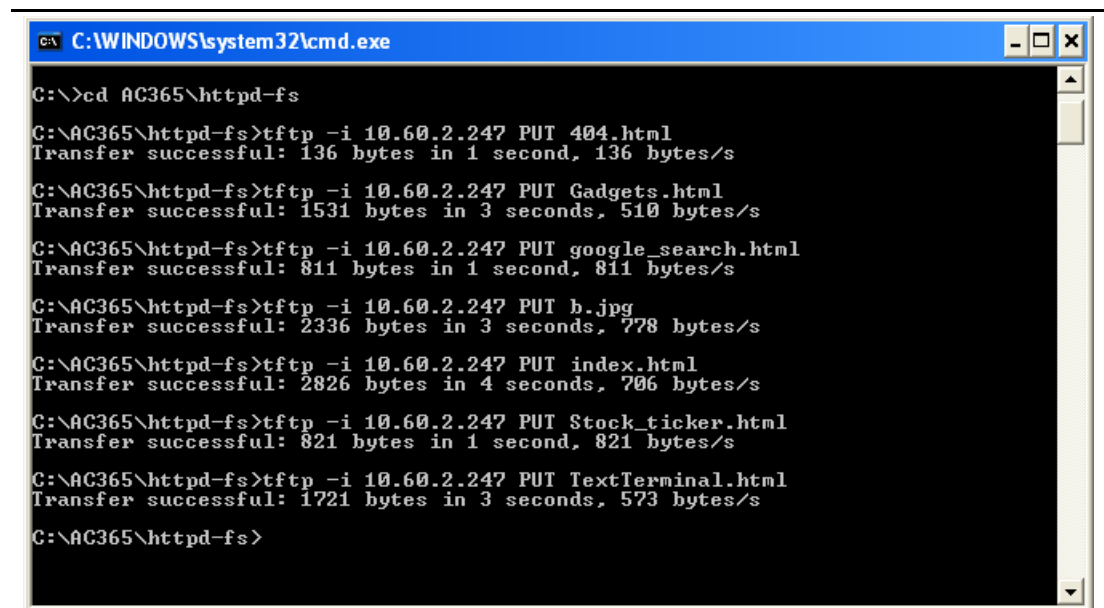


Figure 13 • Setting the Time for the RTC for FatFs File System

When this menu pops-up, send the HTML webserver files from the Host PC to the SPI Flash using the TFTP command as shown in [Figure 13 on page 17](#). [Figure 14](#) shows the various TFTP commands to send the entire webserver from the PC to the target system.



```
C:\WINDOWS\system32\cmd.exe

C:\>cd AC365\httpd-fs
C:\AC365\httpd-fs>tftp -i 10.60.2.247 PUT 404.html
Transfer successful: 136 bytes in 1 second, 136 bytes/s
C:\AC365\httpd-fs>tftp -i 10.60.2.247 PUT Gadgets.html
Transfer successful: 1531 bytes in 3 seconds, 510 bytes/s
C:\AC365\httpd-fs>tftp -i 10.60.2.247 PUT google_search.html
Transfer successful: 811 bytes in 1 second, 811 bytes/s
C:\AC365\httpd-fs>tftp -i 10.60.2.247 PUT b.jpg
Transfer successful: 2336 bytes in 3 seconds, 778 bytes/s
C:\AC365\httpd-fs>tftp -i 10.60.2.247 PUT index.html
Transfer successful: 2826 bytes in 4 seconds, 706 bytes/s
C:\AC365\httpd-fs>tftp -i 10.60.2.247 PUT Stock_ticker.html
Transfer successful: 821 bytes in 1 second, 821 bytes/s
C:\AC365\httpd-fs>tftp -i 10.60.2.247 PUT TextTerminal.html
Transfer successful: 1721 bytes in 3 seconds, 573 bytes/s
C:\AC365\httpd-fs>
```

Figure 14 • 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 the Host PC. All the files sent in the previous step that are stored in the SPI Flash are served for the web client requests by using the lwIP RAW API and SPI Flash file system.

Figure 15 shows the webserver after the successful request of the client.

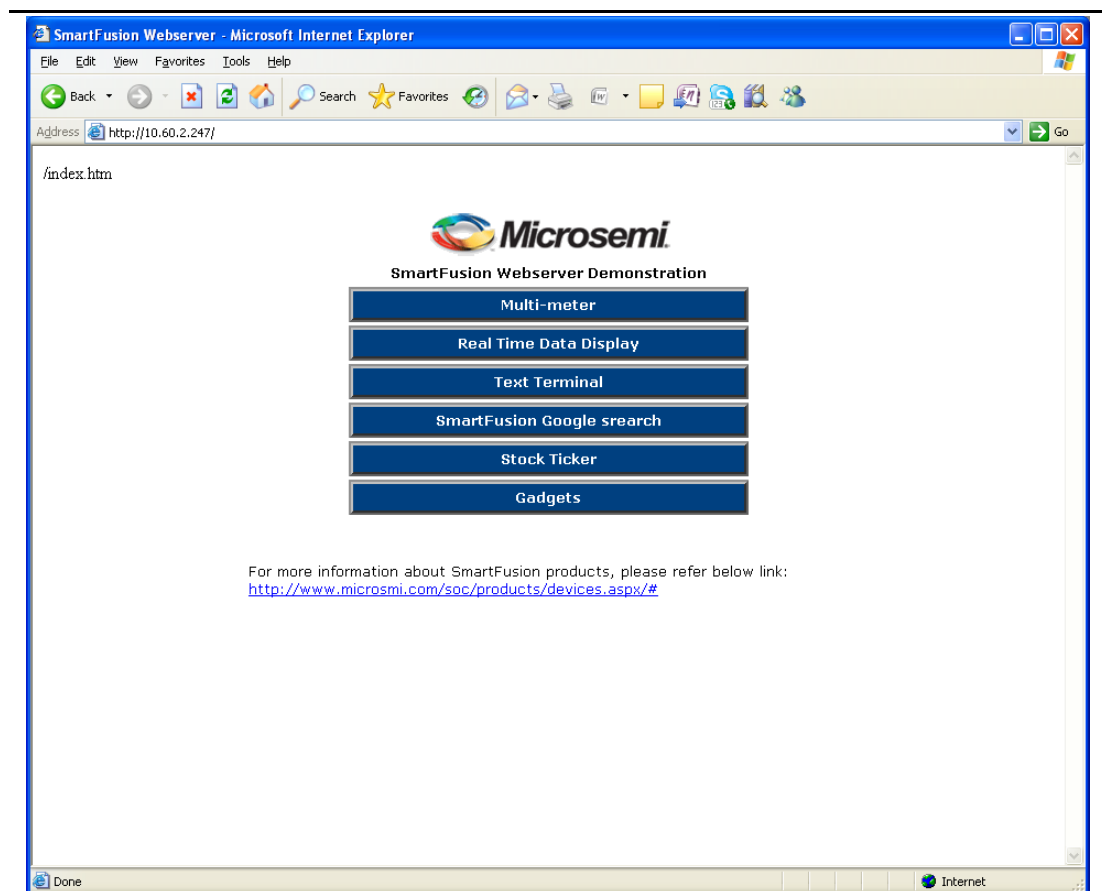


Figure 15 • SmartFusion Webserver with RAW lwIP API

Release Mode

The release mode programming file (STAPL) is also provided. Refer to the Readme.txt file included in the programming zip file for more information.

For more information on building an application in the release mode, refer to the [Building Executable Image in the Release Mode and Loading into eNVM Tutorial](#).

Conclusion

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

Appendix A – Design Files

Download the design files from the Microsemi SoC Products Group Website:

www.microsemi.com/soc/download/rsc/?f=A2F_AC365_DF.rar.

The design file consists of Libero System-on-Chip (SoC) projects and programming file (*.stp) for A2F500. Refer to the Readme.txt file included in the design file for directory structure and description.

You can download the programming files (*.stp) in release mode from the Microsemi SoC Products Group website: www.microsemi.com/soc/download/rsc/?f=A2F_AC365_PF.rar.

The programming zip file consists of STAPL programming files (*.stp) for A2F500 and a Readme.txt file.

List of Changes

The following table lists critical changes that were made in each revision of the document.

Revision*	Changes	Page
Revision 3 (November 2014)	Removed all the instances of "SmartFusion Evaluation Kit Board" and "EVAL-KIT" (SAR 62387).	N/A
Revision 2 (January 2013)	Added "Board Settings" section and modified "Detailed Description of the Design Examples" section (SAR 43469).	8
Revision 1 (February 2012)	Changed Figure 4 (SAR 36036).	9
	Changed Figure 5 (SAR 36036).	11
	Changed Figure 7 (SAR 36036).	12
	Changed Figure 8 (SAR 36036).	13
	Changed Figure 10 (SAR 36036).	15
	Changed Figure 12 (SAR 36036).	17
	Changed Figure 13 (SAR 36036).	17
	Changed Figure 14 (SAR 36036).	18
	Changed Figure 15 (SAR 36036).	19
	Added "Release Mode" (SAR 36036).	19
	Updated "Appendix A – Design Files" (SAR 36036).	20

Note: *The revision number is located in the part number after the hyphen. The part number is displayed at the bottom of the last page of the document. The digits following the slash indicate the month and year of publication.



Microsemi®

Microsemi Corporate Headquarters
One Enterprise, Aliso Viejo CA 92656 USA
Within the USA: +1 (800) 713-4113
Outside the USA: +1 (949) 380-6100
Sales: +1 (949) 380-6136
Fax: +1 (949) 215-4996
E-mail: sales.support@microsemi.com

Microsemi Corporation (Nasdaq: MSCC) offers a comprehensive portfolio of semiconductor and system solutions for communications, defense and security, aerospace, and industrial markets. Products include high-performance and radiation-hardened analog mixed-signal integrated circuits, FPGAs, SoCs, and ASICs; power management products; timing and synchronization devices and precise time solutions, setting the world's standard for time; voice processing devices; RF solutions; discrete components; security technologies and scalable anti-tamper products; Power-over-Ethernet ICs and midspans; as well as custom design capabilities and services. Microsemi is headquartered in Aliso Viejo, Calif. and has approximately 3,400 employees globally. Learn more at www.microsemi.com.

© 2014 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.