AN3120

Embedded Web Server Application Using SAM E54

Introduction

An embedded web server is a microcontroller-based server that can communicate over HTTP or HTTPS allowing access to users over the network, providing a means to control and monitor devices connected to it. For example, in a power plant, the power output and temperature can be monitored through sensors connected to the MCU (with embedded web server). It can also be controlled as needed, by adjusting the input and cooling system actuators that are connected to the MCU.

The following are merits of an embedded web server:

- Cost effective
- Offline monitoring
- Accessible from anywhere and anytime

This document describes a basic web server implementation using three LwIP APIs and the implementation of an advanced web server application on the SAM E54 Xplained Pro Evaluation Board which acquires real time sensor data and events. This data can be accessed and configured using a web page, therefore providing access to monitor and control on-board features. This document also provides an associated software package containing application code for implementation of an embedded web server. The example code for a basic web page implementation is provided as an example in Atmel START.

This document will mainly focus on TCP server connectivity, rather than an HTTP web server. The features are explained by demonstrating with a simple HTTP web server. The HTML, CSS and JavaScript files required for web page design are not explained in detail in this document.

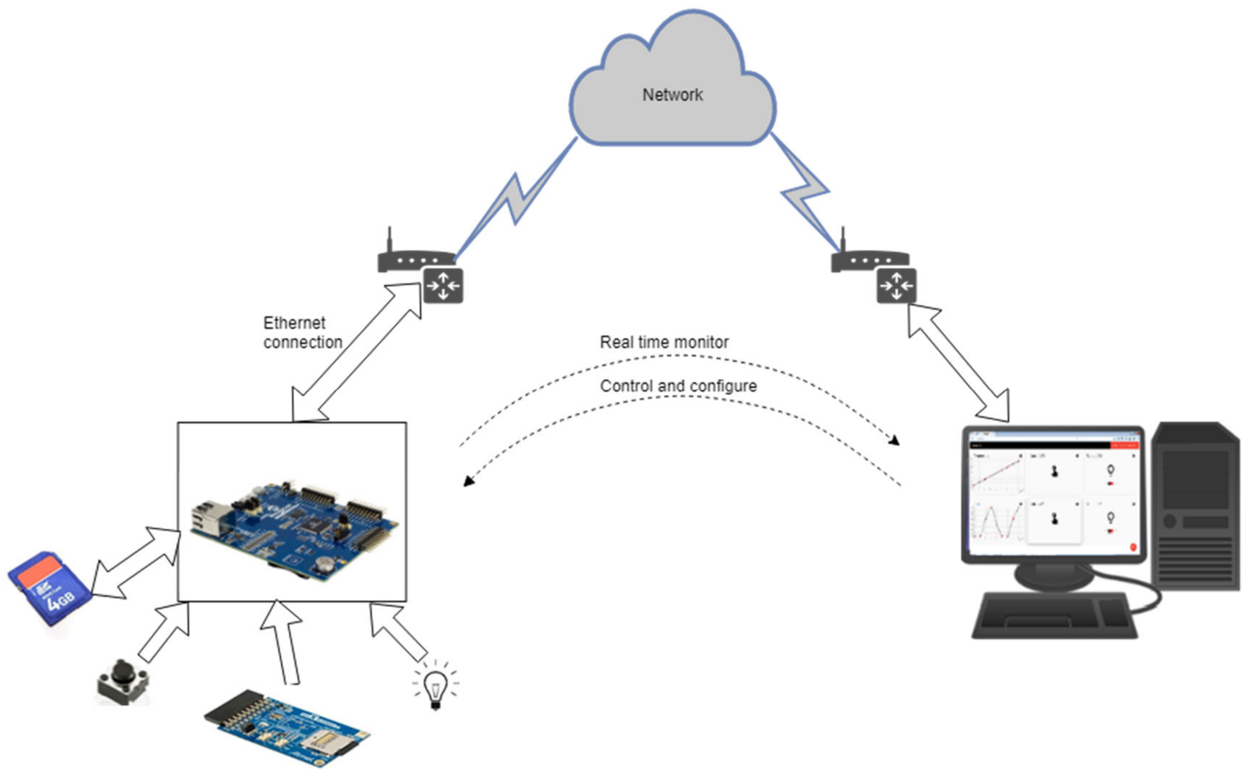
Table of Contents

1. Application Overview

A basic web server requires the implementation of an Ethernet interface and an HTTP server. The following figure shows the application overview which connects the microcontroller to the network through the ethernet, which in turn uses a GMAC peripheral for communication over the network. A PC is used to access all the required data from the microcontroller, which can be used for real time monitoring and controlling of the devices connected to the microcontroller. This application demonstration uses LwIP v1.4.0 and FreeRTOS v8.2.3 to create a basic and an advanced web server.

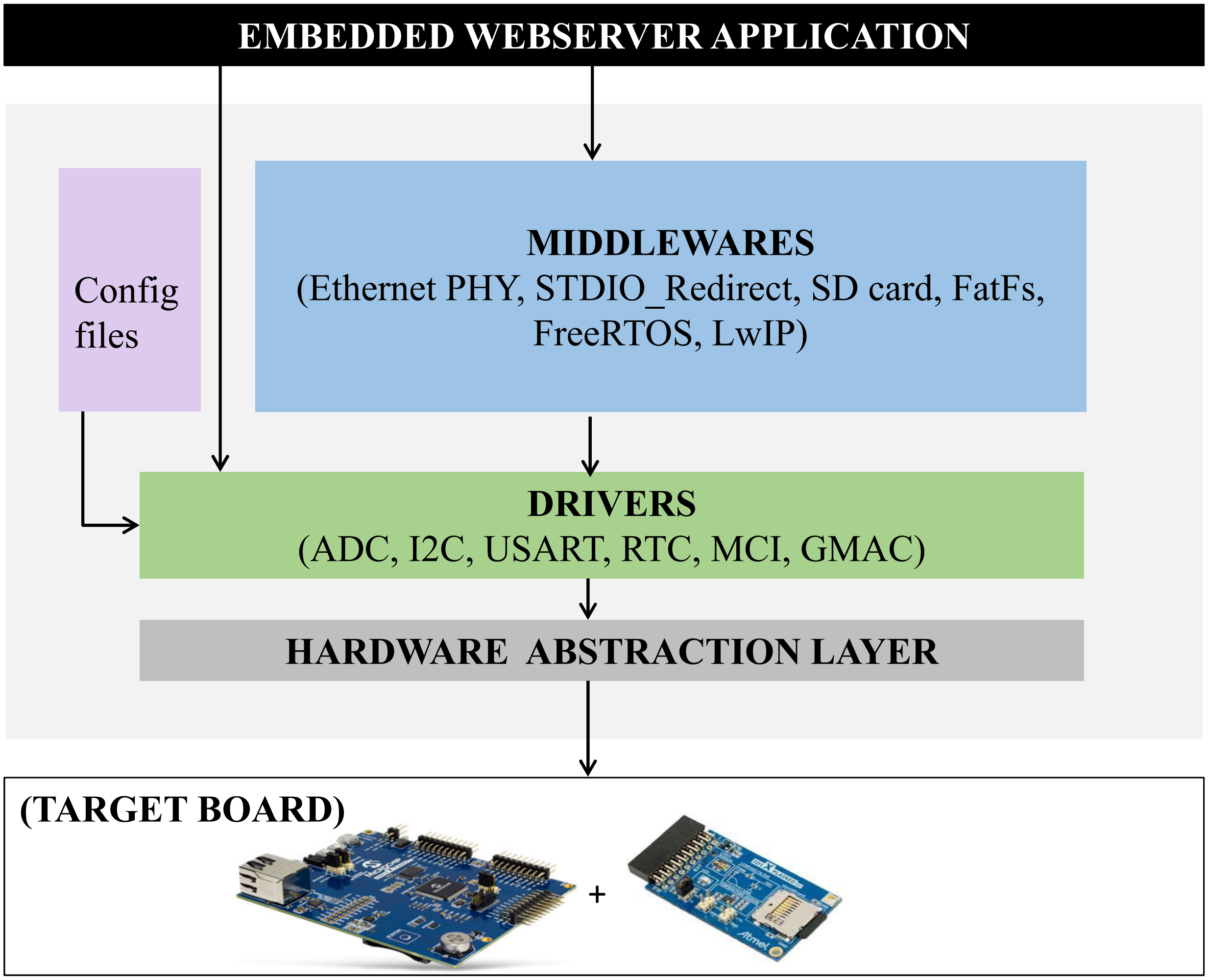
The basic web server implementation gives the user a brief idea of the three LwIP API's: Raw API, Netconn API, Socket API. In the advanced implementation, an embedded web server with the help of a Netconn API will send data, such as temperature, light, and button status to the web page. The application also logs the data in the SD card. LED control and alarm configuration can also be done if required by the user.

Figure 1-1. Application Overview



The software and hardware overview of drivers, and middleware used for the advanced web server application is shown in the following figure.

Figure 1-2. Software and Hardware Components Overview



2. LwIP

Light Weight Internet Protocol (LwIP) is a small independent implementation of the TCP/IP protocol suite. The main features of LwIP are provided in the following table.

Table 2-1. Features of LwIP

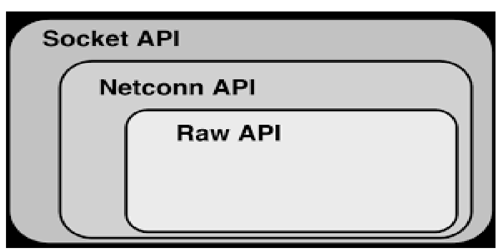
Features of LwIP	Description
IP	Internet Protocol, IPv4 and IPv6. Logical Address assigned by the software for network connectivity with other protocols **(Note 1)**.
TCP	Connection oriented protocol for reliable data communication **(Note 1)**.
DHCP	Network Management Protocol to assign IP address for a node from a DHCP server.
ICMP	Error reporting protocol to diagnose network connectivity.
IGMP	Multicast management.
UDP	Connection less communication model with minimum of protocol mechanism.
DNS	Dynamic name resolution of URL.
PPPoE	Network protocol for encapsulating PPP frames inside ethernet frames.

Note:

1. This feature is used in the demonstration application.

LwIP provides three application program interfaces (APIs) for programs to use for communication with the TCP/IP as shown in the image below:

Figure 2-1. LwIP APIs



- **Raw API** – It is the core API of LwIP. This API aims at providing the best performances while using a minimal code size. It handles asynchronous events using callbacks.
- **Netconn API** - It is designed to make the stack easier to use (compared to the event-driven raw API) while still preserving zero-copy functionality. To use this API, an operating system is needed as this API requires the use of threads. All packet processing (input and output) in the core of the stack is done inside a dedicated thread (the tcp_thread). Application threads using the Netconn API communicate with this core thread using message boxes and semaphores.
- **Socket API** - It is an Inter-processing Communication (IPC) programming interface originally provided as part of the Berkeley's UNIX operating system. It is an abstract representation (handle) for the local endpoint of a network communication path. These are represented as a file descriptor (file handle) in the Unix philosophy that provides a common interface for input and output. The main advantage of Socket API over other APIs is that it is compatible with other TCP/IP stacks as well.

The critical aspect of using LwIP stack is to configure it as per the application. We must configure the LwIP to use either of Socket API, Netconn API, or Raw API and setup the connection as per application needs.

2.1 LwIP Configuration

The configurations for the three LwIP APIs are provided in the table below:

Table 2-2. LwIP Configuration (Available in Config\lwipopts.h)

Options	Macros to be Configured		
	NO_SYS	LWIP_NETCONN	LWIP_SOCKET and LWIP_COMPAT_SOCKETS
Raw API	1	0	0
Netconn API	0	1	0
Socket API	0	1	1

Note: These APIs are applies to LwIP 1.4.0.

2.2 Static and Dynamic IP Configuration

LwIP can be configured to use in static and dynamic IP configuration. The user must use the LwIP configuration mentioned in the table below based on the IP configuration used.

Table 2-3. Prerequisites to be Made

Options	Value for Static IP	Value for DHCP	Description
CONF_TCPIP_STACK_INTERFACE_0_STATIC_IP (Config/lwip_macif_config.h)	1	0	Static IP configuration is enabled
CONF_TCPIP_STACK_INTERFACE_0_DHCP (Config/lwip_macif_config.h)	0	1	Dynamic configuration is enabled
LWIP_DHCP (Config/lwipopts.h)	0	1	Enable DHCP module

In Dynamic Host Configuration Protocol (DHCP), the device is automatically assigned an IP address by the DHCP server. The following code must be included in the `tcpip_init_done()` callback function for the DHCP.

```
//netif_set_up(&TCPIP_STACK_INTERFACE_0_desc); //enable this for static IP

/* DHCP mode. */
if (ERR_OK != dhcp_start(&TCPIP_STACK_INTERFACE_0_desc)) {
    LWIP_ASSERT("ERR_OK != dhcp_start", 0);
}
```

In static IP, to communicate with the web server, the IP address of the PC is configured to be in the same network as that of the server. In this demonstration application, the IP address of the embedded web server is configured as 192.168.1.100. The netmask address configured in the program is 255.255.255.0.

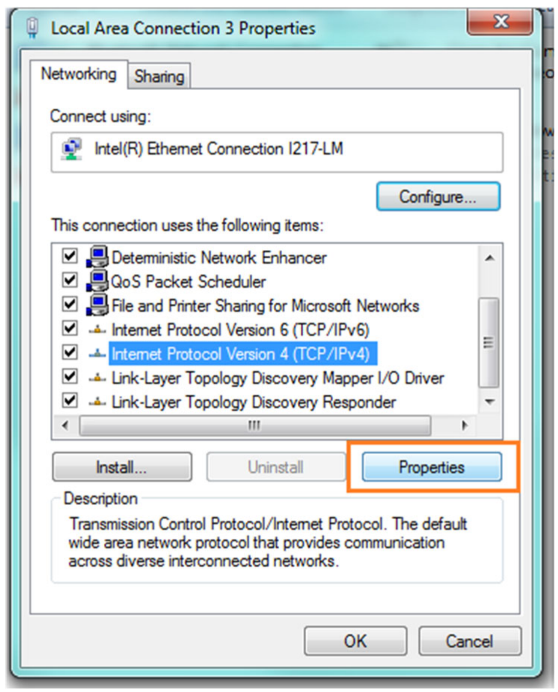
Therefore, all IP addresses having 192.168.1.xx are on the same subnet as the server. The following code must be enabled in the `tcpip_init_done()` callback function for enabling the static IP.

```
netif_set_up(&TCPIP_STACK_INTERFACE_0_desc);  /* enable this for static IP*/
```

To configure static IP on a PC installed with Windows 7, follow these steps:

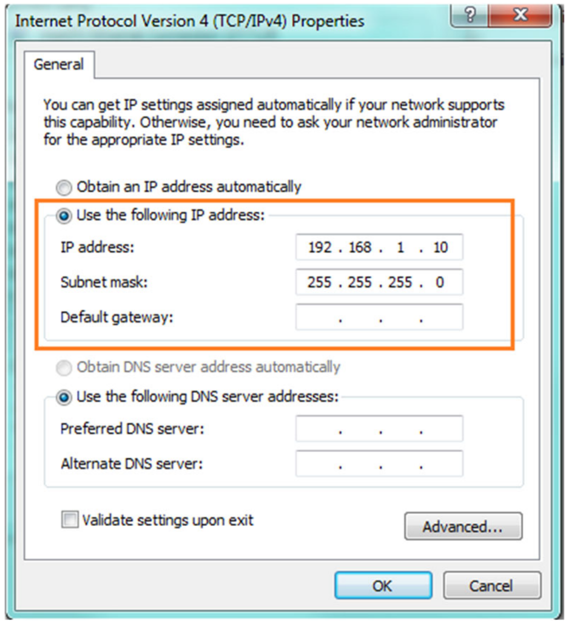
- Open Network and Sharing Center.
- Click **Change adapter setting** in the left pane.
- Right-click on **Local Area connection** and then select **properties**.
- Select **Internet Protocol Version 4** and then click **Properties**.

Figure 2-2. Lan Properties Tab



- Configure the IP address as a subnet mask, and then click **OK**.

Figure 2-3. TCP/IPv4 Properties Tab



3. Basic Web Server Implementations

This section describes how to realize a simple basic web server application using all three LwIP APIs. It helps the user to get familiar with LwIP APIs, their usage, and provides the details of each API implementation.

The output of the basic web server implementation will be a static web page displaying text. The IP address assigned is printed in the console. Whenever a user enters this IP address in the browser, the browser sends a connection request to the server. Once the connection is established, the file to be displayed is requested. On receiving this request, the server will send the data to be displayed.

Note: The example code for the basic web server implementation is available as part of Atmel START, which helps the user to understand three LwIP APIs. The example names are LwIP raw API example, LwIP netconn API example, and LwIP socket API example for raw, netconn and socket API implementation. These examples can be accessed from the following location: https://start.atmel.com/, under the BROWSE EXAMPLES option.

3.1 Raw API

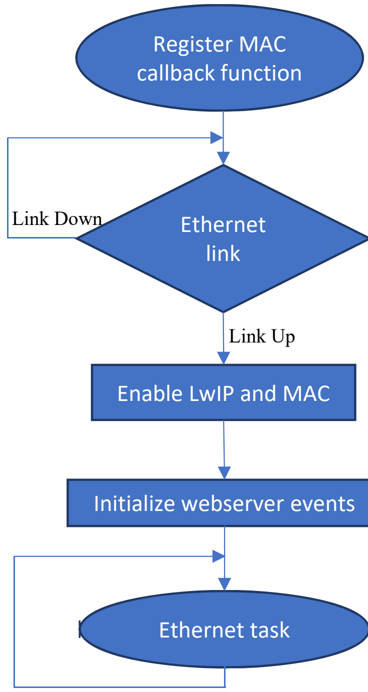
Raw API is a direct interface which uses the lowest level of LwIP programming. It is an event driven API designed to be used without an operating system that implements zero copy and receive. Raw APIs are implemented as a set of callback functions, which are then invoked by the LwIP core when activity related to that application occurs. This application demonstrates a web server implementation that displays a text message on the web page. Refer to the Application Note, AT04055: Using the lwIP Network Stack, for additional information on the TCP Raw API functions.

Table 3-1. TCP Raw API Functions

Function	API Name	API Description
TCP connection setup	`tcp_new()`	Creates a new TCP PCB
	`tcp_bind()`	Bind PCB to local IP address or port
	`tcp_listen()`	Make PCB listen for incoming connections
	`tcp_accept()`	Set callback used for new incoming connections
Sending TCP data	`tcp_write()`	Queue data for transmission
Receiving TCP data	`tcp_recv()`	Set callback for incoming data
Application polling	`tcp_poll()`	Set application poll callback
Closing connections and aborting connections	`tcp_close()`	Close the connection
	`tcp_abort()`	Abort the connection

The main function initializes the microcontroller, drivers, middleware, and performs the following steps as shown in the figure below:

Figure 3-1. Application Flow for Basic Web Server Using Raw API




```
int main(void)
{
    int32_t ret;

    atmel_start_init();
    systick_enable();

    printf("\r\nRaw API implementation\r\n");
    mac_async_register_callback(&COMMUNICATION_IO, MAC_ASYNC_RECEIVE_CB,
(FUNC_PTR)mac_receive_cb);

    eth_ipstack_init();
    do {
        ret = ethernet_phy_get_link_status(&ETHERNET_PHY_0_desc, &link_up);
        if (ret == ERR_NONE && link_up) {
            break;
        }
    } while (true);
    printf("Ethernet link up\n");
    TCPIP_STACK_INTERFACE_0_init((u8_t *)MAC_ADDRESS);
    #if CONF_TCPIP_STACK_INTERFACE_0_DHCP
    /* DHCP mode. */
    if (ERR_OK != dhcp_start(&TCPIP_STACK_INTERFACE_0_desc)) {
        LWIP_ASSERT("ERR_OK != dhcp_start", 0);
    }
    #else
    netif_set_up(&TCPIP_STACK_INTERFACE_0_desc); //enable this for Static IP
    #endif

    /*Handles web server events*/
    lwip_raw_api_init();

    while (true) {
        if (recv_flag) {
            recv_flag = false;
            ethernetif_mac_input(&TCPIP_STACK_INTERFACE_0_desc);
        }
        /* LWIP timers - ARP, DHCP, TCP, etc. */
        sys_check_timeouts();
```



```
        /* Print IP address info */
        if (link_up && TCPIP_STACK_INTERFACE_0_desc.ip_addr.addr) {
            link_up = false;
            print_ipaddress();
        }
    }
}
```

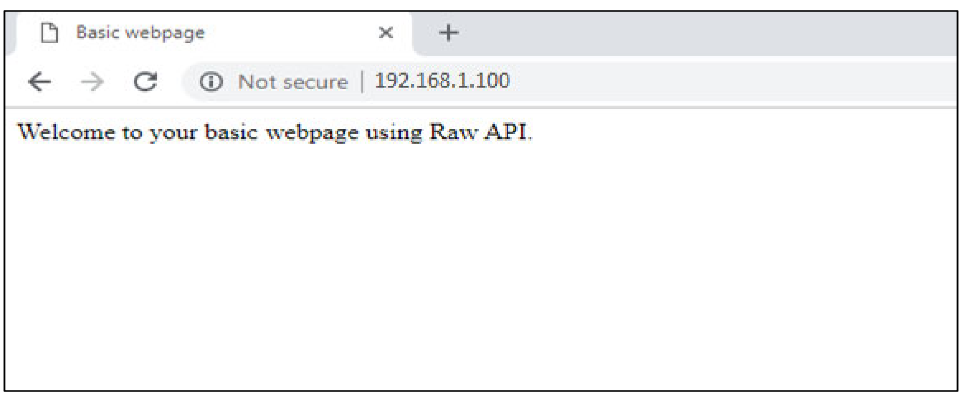
The actual TCP server initialization is made from the `main()` function by calling the `lwip_raw_api_init()` function. This function instantiates a new TCP protocol control block (PCB) and is bound to any IP address and port 80. The PCB listens for the incoming connection of the HTTP port 80.

```
void lwip_raw_api_init(void)
{
    struct tcp_pcb *pcb;

    pcb = tcp_new();
    tcp_bind(pcb, IP_ADDR_ANY, HTTP_PORT);
    pcb = tcp_listen(pcb);
    if (pcb != NULL) {
        tcp_accept(pcb, http_accept);
    }
}
```

Note: For additional information on `lwip_raw_api_init()`, refer to the Application Note, AT04055: Using the lwIP Network Stack, section 5.1.1, `httpd_init()`. However, `httpd_init()` is renamed as `lwip_raw_api_init()` in this document.

The server home page can be accessed using http://192.168.1.100 if a static IP is used. If a dynamic IP is used, the corresponding IP address (`TCPIP_STACK_INTERFACE_0_desc > ip`) is displayed in the console, and can be browsed for the result, as shown in the figure below.

Figure 3-2. Basic Web Page Using Raw API



3.2 Netconn API

Netconn API is a sequential API built on top of the Raw API. It is easier to use than Raw API at the expense of lower performances and increased memory footprint. The following section demonstrates how to develop a server that can serve several requests at the same time using the LwIP Netconn API.

A Netconn API-based program typically uses the following threads:

- Tcpip-thread: LwIP core thread which uses the Raw API.

- GMAC: net_if driver thread in charge of passing the Ethernet frame from the GMAC IP to the tcpip-thread.
- One or more user application threads performing the open, read, write, and close operations on Netconn connections.

The above threads communicate using message passing, which is fully handled by the Netconn API. Refer to the Application Note, AT04055: Using the lwIP Network Stack, for additional information on the TCP Netconn API functions. The application flow for a basic TCP connection using the Netconn API is shown in the figure below.

Figure 3-3. Basic TCP Connection Flow Using Netconn API

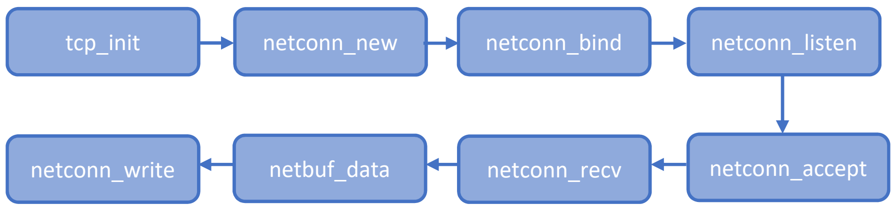


Table 3-2. TCP Netconn API Functions

API function	Description
`netconn_new()`	Creates a new Netconn connection structure
`netconn_bind()`	Binds a Netconn structure to a local IP address or port number
`netconn_listen()`	Sets a TCP Netconn connection into listen mode
`netconn_accept()`	Accepts an incoming connection on a listening TCP Netconn connection
`netconn_connect()`	Connects to a remote TCP host using IP address and port number
`netconn_recv()`	Receives data from a Netconn connection
`netbuf_data()`	Points to the data in first netbuf
`netconn_write()`	Send data on a connected TCP Netconn connection
`netconn_close()`	Closes a TCP Netconn connection without deleting it
`netconn_delete()`	Deletes an existing Netconn connection

The `main()` in `basic_main.c` calls `basic_netconn()` for implementing the basic web server application. It performs the following initializations:

- Create led task and ethernet task
- Start the FreeRTOS scheduler

```
void basic_netconn()
{
    /* Create LED task */
    task_led_create();

    /* Create task for Ethernet */
    if (xTaskCreate(netconn_basic_ethernet, "Ethernet_basic", TASK_ETHERNETBASIC_STACK_SIZE,
NULL, (TASK_ETHERNETBASIC_STACK_PRIORITY-1), &xCreatedEthernetBasicTask) != pdPASS) {
        while (1);
    }

    /* Start FreeRTOS scheduler */
```

```
    vTaskStartScheduler();
}
```

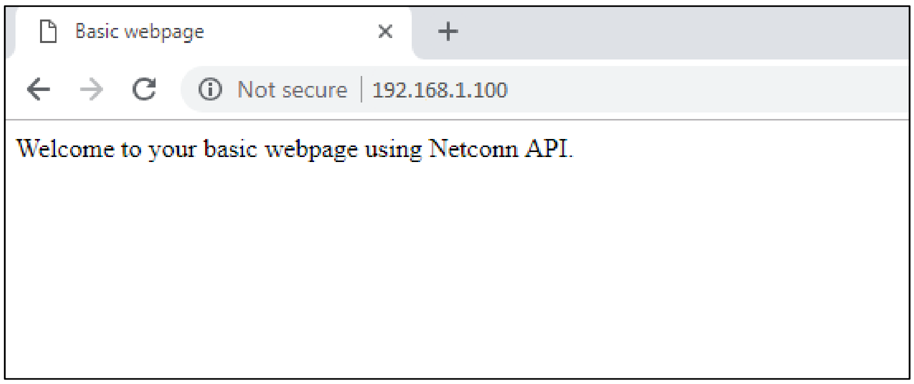
Inside the netconn ethernet task, the `tcpip_init()` function is called to initialize the LwIP stack. The `sys_sem_wait()` function is used to block the progress until the stack is initialized. A new connection structure is created using `netconn_new()` and `netconn_bind()` binds the connection to port 80 on any IP address. `netconn_listen()` listens for any incoming connection requests.

```
void netconn_basic_ethernet(void *p)
{

    struct netbuf *inbuf;
    char *rq;
    unsigned portSHORT len;
    int conn_check;
    sys_sem_t sem;
    err_t err_sem;
    err_sem = sys_sem_new(&sem, 0); /* Create a new semaphore. */
    tcpip_init(tcpip_init_done, &sem);
    sys_sem_wait(&sem);      /* Block until the lwIP stack is initialized. */
    sys_sem_free(&sem);      /* Free the semaphore. */

    print_ipaddress();

    struct netconn *conn_1, *newconn_1;
    /* create a connection structure */
    conn_1 = netconn_new(NETCONN_TCP);
    /* bind the connection to port on any IP address */
    conn_check = netconn_bind(conn_1, NULL, HTTP_PORT);
    while( conn_check!= ERR_OK)
    {
        LWIP_DEBUGF(LWIP_DBG_ON, ("Bind error=%d\n",conn_check));
        goto conn_close;
    }
    /* tell the connection to listen for incoming connection requests */
    netconn_listen(conn_1);
    for( ;; ){
         conn_check = netconn_accept(conn_1, &newconn_1);
         while(conn_check != ERR_OK){
             LWIP_DEBUGF(LWIP_DBG_ON, ("Connection accept error=%d\n",conn_check));
             goto conn_close;
         }
         if(newconn_1 != NULL){
             conn_check = netconn_recv( newconn_1, &inbuf);
            while( conn_check != ERR_OK){
                LWIP_DEBUGF(LWIP_DBG_ON, ("Receive error=%d\n",conn_check));
                goto conn_close;
            }
            if( inbuf != NULL ){
                /* Get the pointer to the data in the first netbuf
                fragment which we hope contains the request. */
                netbuf_data(inbuf,( void * ) &rq, &len);
                /* Check if the request was an HTTP "GET /\r\n". */
                if(( NULL != rq)&& ( !strncmp( rq, "GET", 3 ) )){
                    /* Send the header. */
                    conn_check = netconn_write(newconn_1, http_html_hdr,
sizeof(http_html_hdr), NETCONN_NOCOPY);
                    if( conn_check != ERR_OK){
                    LWIP_DEBUGF(LWIP_DBG_ON, ("Write error=%d\n",conn_check));
                    goto conn_close;
                    }
                    /* Send the actual web page. */
                    conn_check = netconn_write(newconn_1, netconn_webpage,
sizeof(netconn_webpage), NETCONN_NOCOPY);
                    if( conn_check != ERR_OK){
                    LWIP_DEBUGF(LWIP_DBG_ON, ("Write error=%d\n",conn_check));
                    goto conn_close;
                }
            }
            netbuf_delete(inbuf);
            }
conn_close:    /* Close the connection. */
            netconn_close(newconn_1);
```

```
            netconn_delete(newconn_1);
        }
    }
}
```

In case of error when a connection is being established, the corresponding error code will be displayed as a debug message. The corresponding IP address is displayed in the console which is known from the structure `TCPIP_STACK_INTERFACE_0_desc`.

Figure 3-4. Basic Web Page Using Netconn API



3.3 Socket API

LwIP Socket API are mainly used for programming distributed applications on the Internet. Refer to the Application Note, AT04055: Using the lwIP Network Stack, for additional information on the TCP Socket API functions. The application flow of the TCP connection using a Socket API is as shown in the figure below.

Figure 3-5. TCP Connection Flow Using Socket API

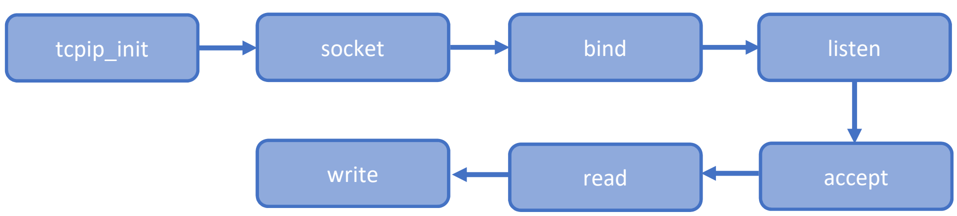


Table 3-3. Socket API Functions

API function	Description
`socket()`	Specify the type of communication protocol and it returns a socket descriptor.
`setsockopt()`	Set options associated with a socket.
`bind()`	Assigns a local protocol address to a socket.
`listen()`	Converts an unconnected socket into a passive socket, indicating that the kernel should accept incoming connection requests.
`accept()`	Returns a new socket descriptor for a client connection in the connection waiting queue.

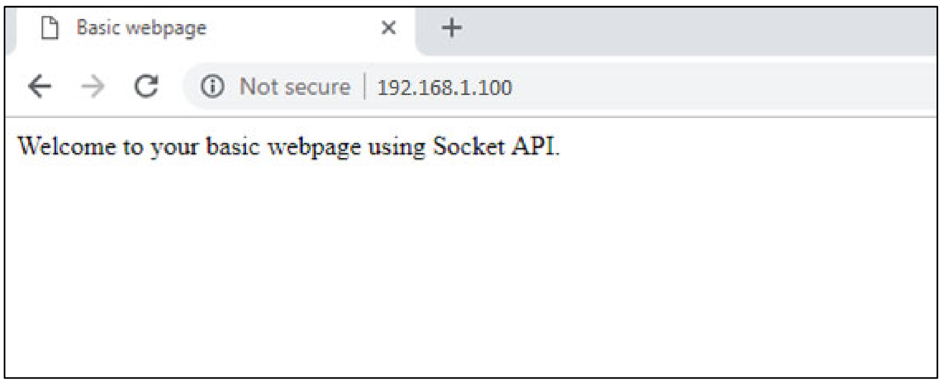
..........continued	
API function	**Description**
`read()` and `write()`	Used to communicate with a socket as long as it is connected.
`send()`	Similar to `write()`, but allows to specify some options.
`recv()`	Similar to `read()` but allows to specify some options to control how the data is received.
`close()`	Closes a socket and terminates a TCP socket.

The main function calls `basic_socket()` after initializing the microcontroller. `basic_socket()` is same as `basic_netconn()` explained for the Netconn API, except the `socket_basic_ethernet()`. In the socket webserver task, LwIP stack initialization process is the same as Netconn API. The difference lies in the APIs being called. The `AF_INET` macro is used to define the address family. `htonl` and `htons` are APIs used to convert the long and short data into big endian format, irrespective of whether the system is little endian or big endian. The IP address is displayed in the console in the same way as mentioned in the Netconn API.

```
void socket_basic_ethernet(void *p)
{
    struct sockaddr_in address;
    int s_create, new_socket;
    int addrlen = sizeof(address);
    int opt = 1;
    int socket_check;

    sys_sem_t sem;
    err_t err_sem;
    err_sem = sys_sem_new(&sem, 0); /* Create a new semaphore. */
    tcpip_init(tcpip_init_done, &sem);
    sys_sem_wait(&sem);     /* Block until the lwIP stack is initialized. */
    sys_sem_free(&sem);     /* Free the semaphore. */

    print_ipaddress();

    /*Create a socket*/
    s_create = socket(AF_INET, 1, 0);

    setsockopt(s_create, SOL_SOCKET, SO_REUSEADDR | SO_REUSEPORT,&opt, sizeof(opt));

    address.sin_family = AF_INET;
    address.sin_addr.s_addr = htonl(IPADDR_ANY);
    address.sin_port = htons( HTTP_PORT );

    /* bind the connection to port */
    socket_check = bind(s_create, (struct sockaddr *)&address,sizeof(address));
    if(socket_check<0){
        LWIP_DEBUGF(LWIP_DBG_ON, ("Bind error=%d\n",socket_check));
        goto socket_close;
    }
    /* tell the connection to listen for incoming connection requests */
    listen(s_create, 3);
    for(;;){
        new_socket = accept(s_create, (struct sockaddr *)&address,(socklen_t*)&addrlen);
        if(new_socket<=0){
            LWIP_DEBUGF(LWIP_DBG_ON, ("Connection error=%d\n",new_socket));
            goto socket_close;
        }
        socket_check = read( new_socket ,buffer, 1024);
        if(socket_check<=0){
            LWIP_DEBUGF(LWIP_DBG_ON, ("Read error=%d\n",socket_check));
            goto socket_close;
        }
        /* Check if the request was an HTTP "GET /\r\n". */
        if( !strncmp( buffer, "GET", 3 )){
```

```
            socket_check = write(new_socket , http_html_hdr , strlen(http_html_hdr));
            if(socket_check<=0){
                LWIP_DEBUGF(LWIP_DBG_ON, ("Write error=%d\n",socket_check));
                goto socket_close;
            }
            /*Send the actual webpage*/
            socket_check = write(new_socket , socket_webpage , strlen(socket_webpage));
            if(socket_check<=0){
                LWIP_DEBUGF(LWIP_DBG_ON, ("Write error=%d\n",socket_check));
                goto socket_close;
            }
        }
        /*Close connection*/
        socket_close:
        close(new_socket);
    }
}
```

Figure 3-6. Basic Web Page Using Socket API



4. Advanced Web Server Application

A control panel application running on an embedded web server is used to configure and manage the system settings. The SAM E54 Xplained Pro acts as the embedded web server. Any client (web browser) can connect to this server, and control or monitor the application. Implementation of such web server mainly requires managing the LwIP stack and processing the incoming connections and requests, refer to Section 4.2. Implementation.

The user interface for this application is provided by means of a dynamic web page (dynamic web page implies that the web page is updated automatically based on application status). The web page allows monitoring temperature and light sensor status, controlling the on-board LED and setting alarm if required. The dynamic web page files corresponding to the demonstration application (HTML, CSS, JavaScript) are stored in an SD card, refer to Section 4.2. Implementation.

Note: The application provided is expected to work when the SD card mounted contains all the required files (as described in the Section 4.2. Implementation) for the web page. The application is tested for Windows 7 with Google Chrome v73.0.3683.86, Firefox Quantum v66.0.2, Internet Explorer v11.0.9600.19230.

4.1 Application Setup

4.1.1 Required Components

Hardware prerequisites:

- SAM E54 Xplained Pro evaluation kit
- I/O1 Xplained Pro extension kit
- SD card
- Ethernet cable (RJ45)
- USB cable

Software prerequisites:

- Atmel START
- Atmel Studio 7 (v7.0.1931)
- SAME54_DFP (v1.0.87)

4.1.2 Setup Details

Hardware setup details are as follows:

- The I/O1 Xplained Pro extension must be connected to the EXT1 extension header of the SAM E54 Xplained Pro evaluation board.
- Connect the Ethernet port of the PC to the SAM E54 Xplained Pro board using a network cable.
- Insert the SD card.
- Power up the SAM E54 Xplained Pro board by connecting the USB cable to the DEBUG USB header.

Software setup details are as follows:

Table 4-1. Required Atmel START Software Modules

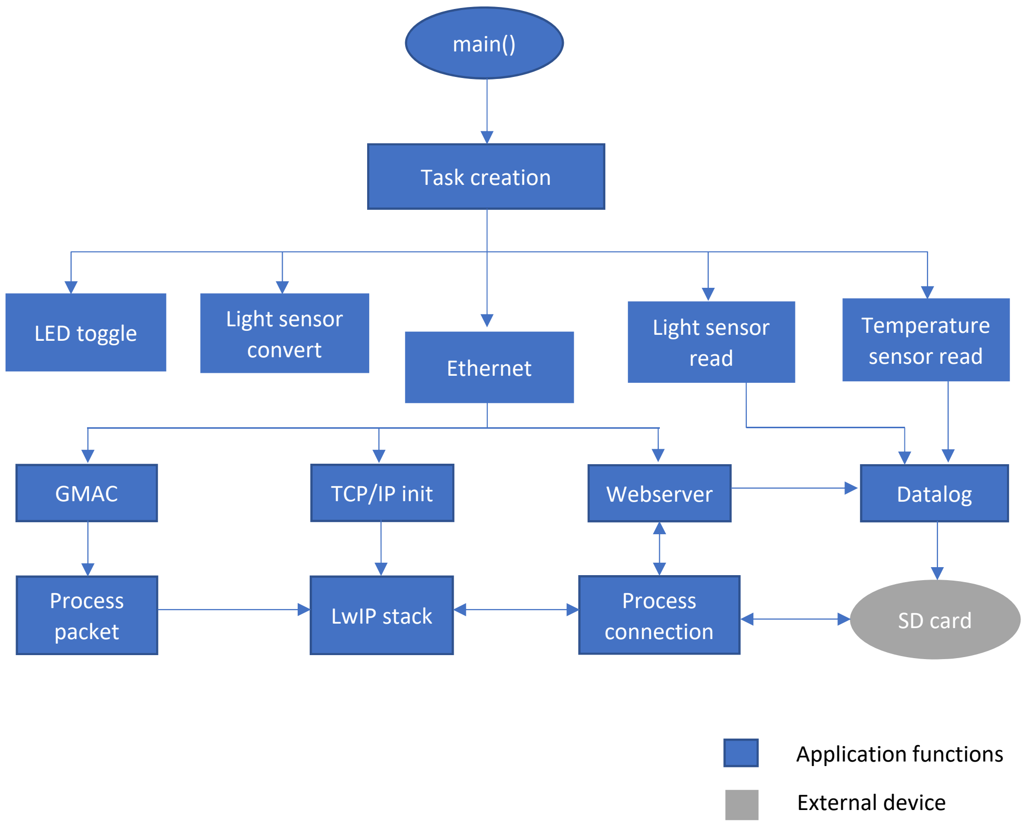
Modules	Description
Ethernet PHY	Driver required to implement physical layer functions.
TCP/IP Stack Interface	Interfaces the ethernet driver to TCP/IP stack. The LwIP module used is of version 1.4.0.
FreeRTOS v8.2.3	Offers tasks, scheduling and inter task communication for the application.
FatFs	A generic FAT file system module for small embedded systems.
STDIO Redirect	Provides means to redirect standard input/output to HAL I/O.
ADC, I^2C	Drivers needed for the sensors used in the application.

Note: If a *Verifying Flash…Failed* issue while programming the `.elf` file, use the `.hex` file or `.bin` file for device programming on the SAM E54 Xplained Pro board.

4.2 Implementation

The application flow for the advanced web server is shown in the figure below:

Figure 4-1. Application Flow



Application Flow Overview

The following steps are involved in developing the application:

- Microcontroller initialization
- Application task creation

There are five different tasks being created as shown in the following table.

Table 4-2. Application Tasks

Tasks	Description
LED task	Led blinking with an interval of 500 ms
Light sensor convert	ADC conversion of TEMT6000 light sensor output
Light sensor read	ADC channel read and the output is converted to percentage within the light sensor range
Temperature sensor read	I^2C read of the AT30TE758 temperature sensor data
Ethernet task	Creates a TCP/IP platform for communication with the web page

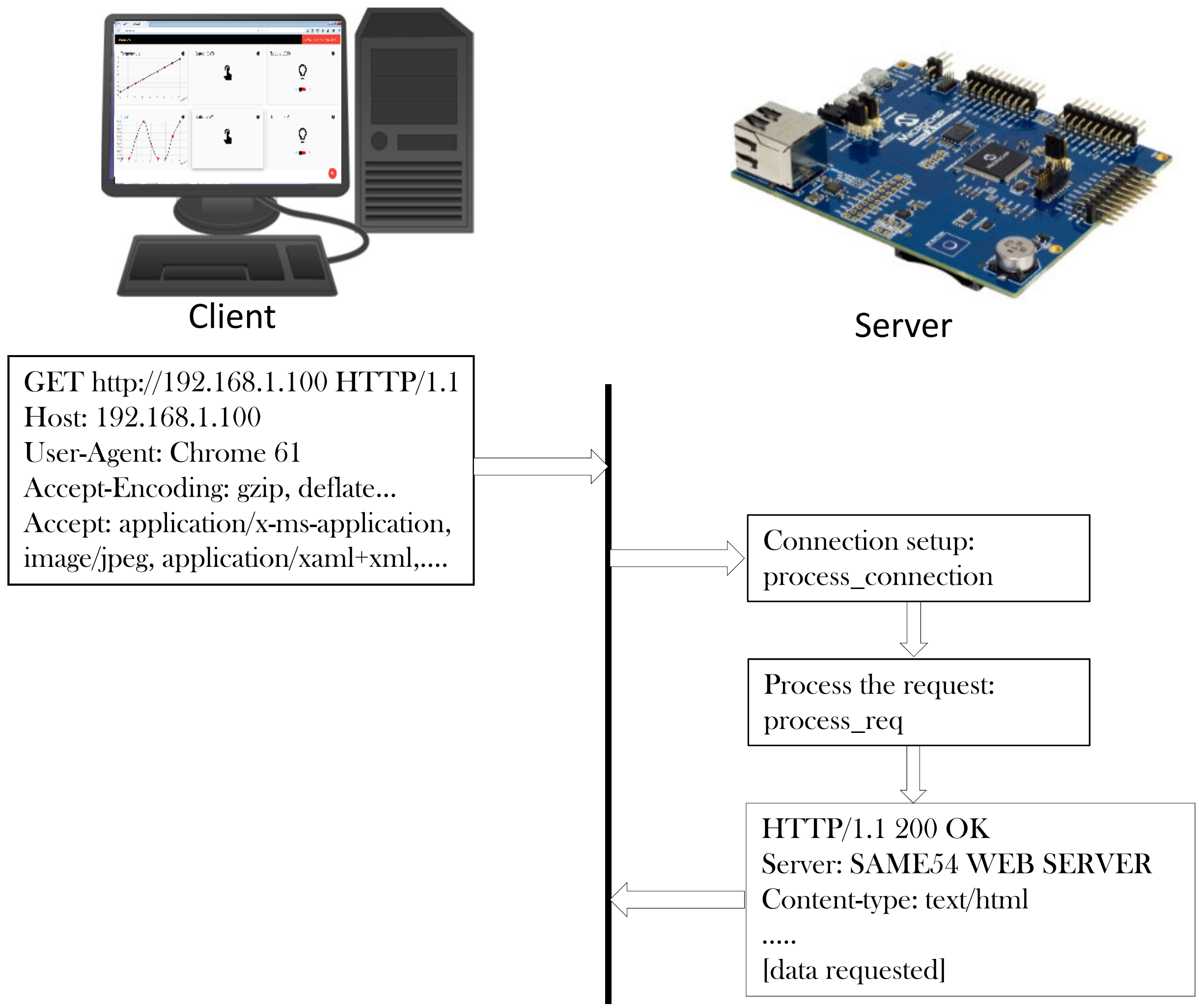
- **TCP/IP connection setup:**

The `task_ethernet` task performs the following functions:

- Initializes the LwIP module and the ethernet buffer transfers the processed packets to the LwIP stack.
- Creates a new thread `vBasicWEBServer` using `sys_thread_new`. This task checks for incoming connection and processes it. The TCP connection is setup in the same procedure as discussed in the basic web server using Netconn API, but the difference lies in how the received requests are treated.

- **Client- Server communication:**
`process_connection()` function is used for communication between the client and the server. On a request from the web browser, the server reads the data from the SD card, sending it to the web browser, and writes the data to the SD card for various commands from the web browser. The communication flow between client and server is shown in the figure below.

Figure 4-2. Client Server Communication Flow



The `process_req()` function parses the request and searches for the first occurrence of a space character or `\t or \r or \n`. It is assumed that the sensor data will not have these tokens. The string contains the path which is again searched for '?' (user defined symbol), if found the request is said to be a fetch or set request otherwise, it is taken as a file request.

- The request obtained can be classified as a fetch or set request, if the request obtained is: *GET /?get_sensor_data HTTP/1.1\r\n* .
- The request obtained can be classified as a file request, if the request obtained is: *GET /logo.png HTTP/1.1\r\n*.

The following figure shows the functions calls executed inside the `process_req()` function.

Figure 4-3. Function Call Graph

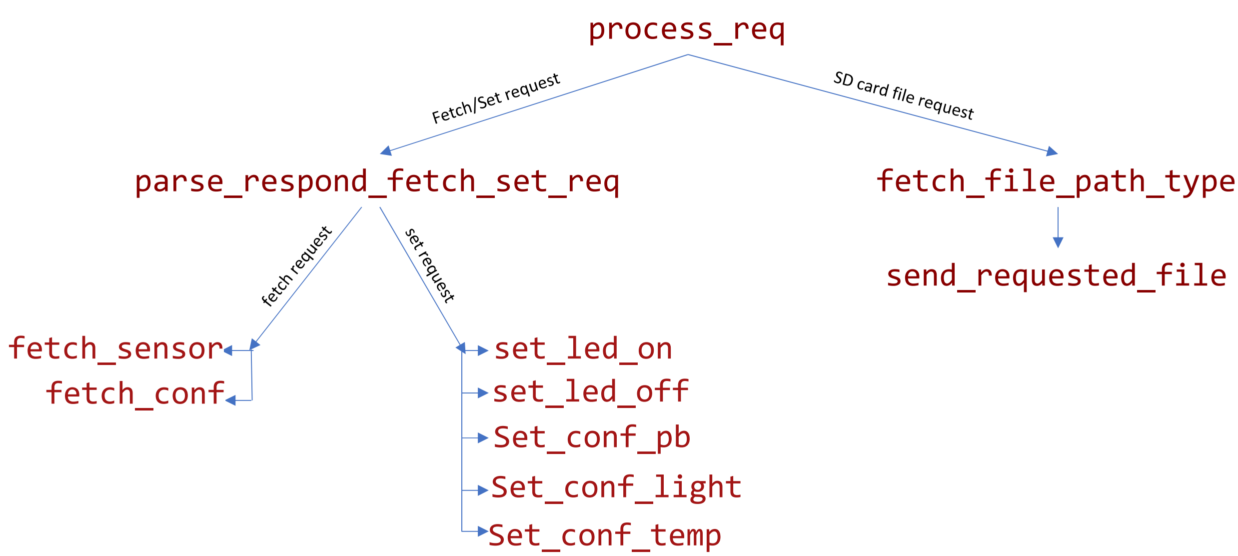


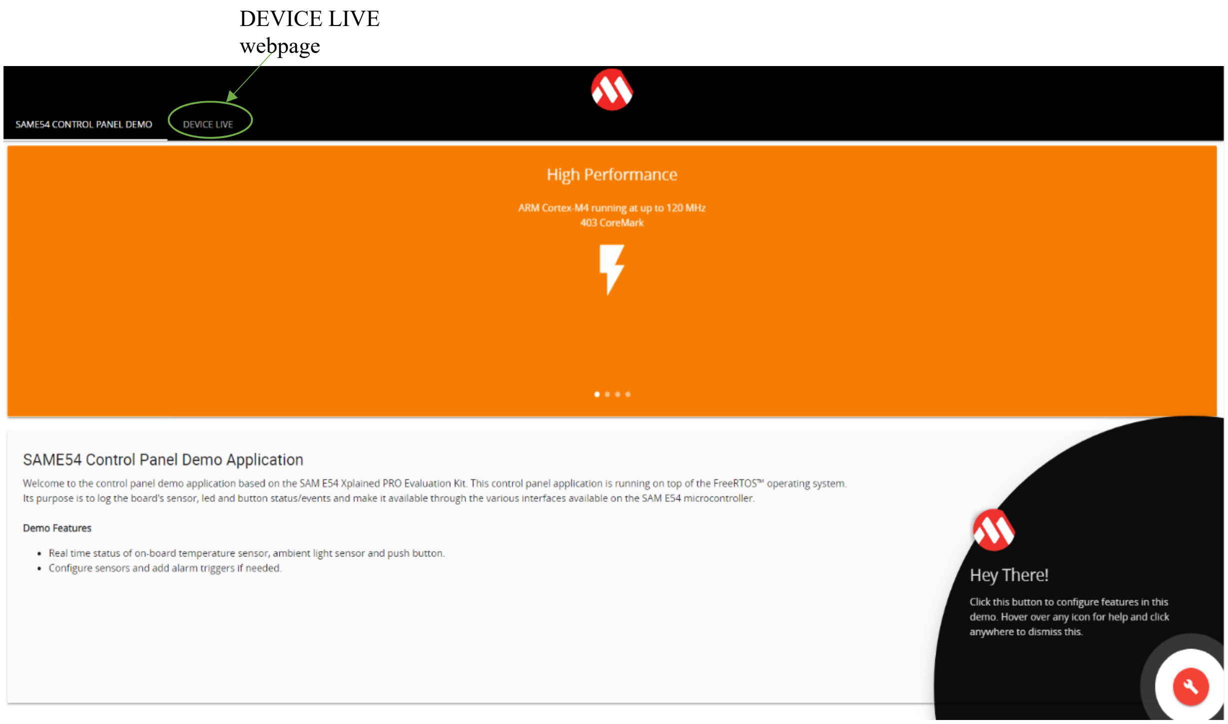
Table 4-3. SD Card File Layout

File name	Description
index.htm	Main html page. Contains the elements of the page and references supporting files that are needed.
core.css	Defines the style of basic elements on the web page
core.js	Used to send, request, and process data to/from server
pack.css	Defines the style of other elements.
pack.js	Used for rendering graphic (chart, icon glow etc.,)
logo.png	Microchip logo for the icon.
log.txt	File for data log
config.txt	Stores configuration data, such as minimum or maximum values.

When the user requests the `index.htm` file to be displayed on the browser, the browser requests a connection to the web server. After the connection establishment, the server receives a request for the .html file. The index page initially runs the image, `pack.js, pack.css and core.css`, which defines the page and other element styles.

```
<link rel = “shortcut icon” type = “image/png” href = “logo.png”>
<script type = “application/javascript” src = “pack.js”></script>
<link href = “pack.css” type = “text/css” rel = “stylesheet”>
<link href = “core.css” type = “text/css” rel = “stylesheet”>
```

Figure 4-4. Web Page Displaying Index Page



- **Application User Actions:** If server gets a *fetch* or *set* request, the `parse_respond_fetch_set_req()` (`Webserver/Web.c`) function is called, and the request is parsed and responded to accordingly. The fetch request part mainly consists of the following two options:
  – `fetch_sensor`: used to update the status of light and temperature sensor values and button status
  – `fetch_conf`: used to send the configuration file if requested

  The *set* request part checks for the following options:

  – `set_led_on`: turns on the led
  – `set_led_off`: turns the led off
  – `set_conf_pb`: configures the push button alarm
  – `set_conf_light`: configures the light minimum or maximum values and alarm
  – `set_conf_temp`: configures temperature minimum or maximum values and alarm

The `core.js` file checks for the user request and performs the necessary actions. It is placed for processing under the `index.htm`, which is triggered when the user requests the URL using a web client, such as a browser.

```
<script type = "application/javascript" src = "core.js">
</script>
```

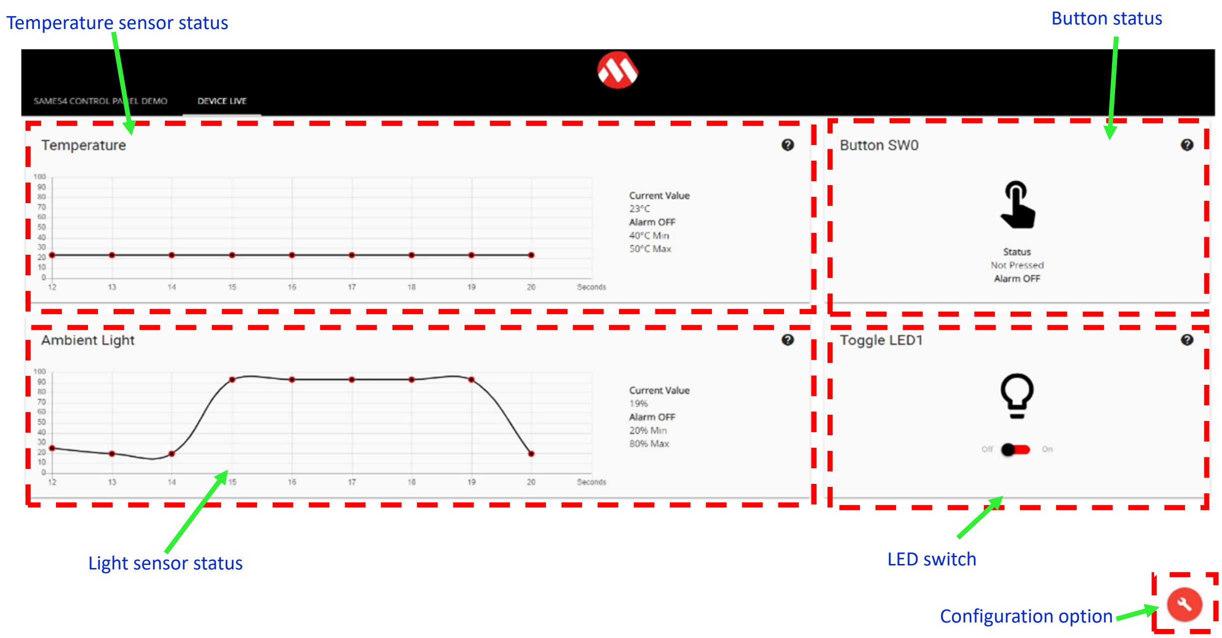
- **User Interface:**The SAM E54 Control Panel Demonstration page provides a small introduction of the SAM E54 Xplained Pro evaluation kit and the demonstration application. Clicking the DEVICE LIVE option on the web page modifies the view. This page requests sensor data periodically from the server, along with user specified control and configuration commands.
  The status updates on the DEVICE LIVE page are as follows:

– Temperature sensor status box shows the temperature values which are updated every second. This also displays the current temperature value as text, alarm status, minimum, and the maximum values.
– Similarly, light sensor values are updated every second in the Light sensor status graph, as shown in the following figure. The current light sensor value, alarm status, minimum, and maximum values are also displayed.
– Button status box shows the status of the SW0 switch in the microcontroller. The icon turns red when the button is pressed.

The control and configuration options on the DEVICE LIVE page are as follows:

– The LED switch box turns the LED ‘ON’ when moving the button icon to ON state, and turns LED ‘OFF’ when moving the button icon to OFF state.
– The Configuration option enables the user to set the minimum and maximum values the sensors can attain and turns the alarm ON and OFF.

Figure 4-5. Control Panel Web Page



• The collected data is stored in text format in the `log.txt` file for future reference, which is accessible from the SD card. The format used to store the data is:
Sensor_idname | Date | Time | Value | max/min.

Note: The last field maximum or minimum is applicable only if the sensor value exceeds the set configuration.

5. Related Articles and Resources

- Details about SAM E54 Xplained Pro evaluation kit:
  https://www.microchip.com/design-centers/32-bit/sam-32-bit-mcus/sam-e-mcus
- Getting started with FreeRTOS :
  http://ww1.microchip.com/downloads/en/appnotes/atmel-42382-getting-started-with-freertos-on-atmel-sam-flash-mcus_applicationnote_at04056.pdf
- Using the LwIP Network Stack:
  https://www.microchip.com/wwwAppNotes/AppNotes.aspx?appnote=en591731
- Use of the Ethernet on SAM4E-EK:
  http://ww1.microchip.com/downloads/en/AppNotes/Atmel-42134-Use-of-Ethernet-on-SAM4E-EK_AT02971_Application-Note.pdf
- TCP/IP Server-Client with CycloneTCP:
  http://ww1.microchip.com/downloads/en/AppNotes/Atmel-42738-TCPIP-Server-Client-with-CycloneTCP_AT16287_ApplicationNote.pdf

Third-party links:

- LwIP source:
  https://savannah.nongnu.org/projects/lwip/
- Web development resources:
  https://www.w3schools.com/whatis/default.asp

6. List of Abbreviations

The following abbreviations are used in this document:

- MCU – Microcontroller unit
- PC – Personal computer
- DHCP – Dynamic Host Configuration Protocol
- PCB – Protocol Control Block
- TCP/IP – Transmission Control Protocol/Internet Protocol
- ICMP – Internet Control Message Protocol
- UDP – User Datagram Protocol
- ROM – Read Only Memory
- LwIP – Light Weight Internet Protocol
- HTML – Hyper Text Markup Language

The Microchip Web Site

Microchip provides online support via our web site at http://www.microchip.com/. This web site is used as a means to make files and information easily available to customers. Accessible by using your favorite Internet browser, the web site contains the following information:

- **Product Support** – Data sheets and errata, application notes and sample programs, design resources, user's guides and hardware support documents, latest software releases and archived software
- **General Technical Support** – Frequently Asked Questions (FAQ), technical support requests, online discussion groups, Microchip consultant program member listing
- **Business of Microchip** – Product selector and ordering guides, latest Microchip press releases, listing of seminars and events, listings of Microchip sales offices, distributors and factory representatives

Customer Change Notification Service

Microchip's customer notification service helps keep customers current on Microchip products. Subscribers will receive e-mail notification whenever there are changes, updates, revisions or errata related to a specified product family or development tool of interest.

To register, access the Microchip web site at http://www.microchip.com/. Under "Support", click on "Customer Change Notification" and follow the registration instructions.

Customer Support

Users of Microchip products can receive assistance through several channels:

- Distributor or Representative
- Local Sales Office
- Field Application Engineer (FAE)
- Technical Support

Customers should contact their distributor, representative or Field Application Engineer (FAE) for support. Local sales offices are also available to help customers. A listing of sales offices and locations is included in the back of this document.

Technical support is available through the web site at: http://www.microchip.com/support

Microchip Devices Code Protection Feature

Note the following details of the code protection feature on Microchip devices:

- Microchip products meet the specification contained in their particular Microchip Data Sheet.
- Microchip believes that its family of products is one of the most secure families of its kind on the market today, when used in the intended manner and under normal conditions.
- There are dishonest and possibly illegal methods used to breach the code protection feature. All of these methods, to our knowledge, require using the Microchip products in a manner outside the operating specifications contained in Microchip's Data Sheets. Most likely, the person doing so is engaged in theft of intellectual property.
- Microchip is willing to work with the customer who is concerned about the integrity of their code.

- Neither Microchip nor any other semiconductor manufacturer can guarantee the security of their code. Code protection does not mean that we are guaranteeing the product as “unbreakable.”

Code protection is constantly evolving. We at Microchip are committed to continuously improving the code protection features of our products. Attempts to break Microchip’s code protection feature may be a violation of the Digital Millennium Copyright Act. If such acts allow unauthorized access to your software or other copyrighted work, you may have a right to sue for relief under that Act.

Legal Notice

Information contained in this publication regarding device applications and the like is provided only for your convenience and may be superseded by updates. It is your responsibility to ensure that your application meets with your specifications. MICROCHIP MAKES NO REPRESENTATIONS OR WARRANTIES OF ANY KIND WHETHER EXPRESS OR IMPLIED, WRITTEN OR ORAL, STATUTORY OR OTHERWISE, RELATED TO THE INFORMATION, INCLUDING BUT NOT LIMITED TO ITS CONDITION, QUALITY, PERFORMANCE, MERCHANTABILITY OR FITNESS FOR PURPOSE. Microchip disclaims all liability arising from this information and its use. Use of Microchip devices in life support and/or safety applications is entirely at the buyer’s risk, and the buyer agrees to defend, indemnify and hold harmless Microchip from any and all damages, claims, suits, or expenses resulting from such use. No licenses are conveyed, implicitly or otherwise, under any Microchip intellectual property rights unless otherwise stated.

Trademarks

The Microchip name and logo, the Microchip logo, AnyRate, AVR, AVR logo, AVR Freaks, BitCloud, chipKIT, chipKIT logo, CryptoMemory, CryptoRF, dsPIC, FlashFlex, flexPWR, Heldo, JukeBlox, KeeLoq, Kleer, LANCheck, LINK MD, maXStylus, maXTouch, MediaLB, megaAVR, MOST, MOST logo, MPLAB, OptoLyzer, PIC, picoPower, PICSTART, PIC32 logo, Prochip Designer, QTouch, SAM-BA, SpyNIC, SST, SST Logo, SuperFlash, tinyAVR, UNI/O, and XMEGA are registered trademarks of Microchip Technology Incorporated in the U.S.A. and other countries.

ClockWorks, The Embedded Control Solutions Company, EtherSynch, Hyper Speed Control, HyperLight Load, IntelliMOS, mTouch, Precision Edge, and Quiet-Wire are registered trademarks of Microchip Technology Incorporated in the U.S.A.

Adjacent Key Suppression, AKS, Analog-for-the-Digital Age, Any Capacitor, AnyIn, AnyOut, BodyCom, CodeGuard, CryptoAuthentication, CryptoAutomotive, CryptoCompanion, CryptoController, dsPICDEM, dsPICDEM.net, Dynamic Average Matching, DAM, ECAN, EtherGREEN, In-Circuit Serial Programming, ICSP, INICnet, Inter-Chip Connectivity, JitterBlocker, KleerNet, KleerNet logo, memBrain, Mindi, MiWi, motorBench, MPASM, MPF, MPLAB Certified logo, MPLIB, MPLINK, MultiTRAK, NetDetach, Omniscient Code Generation, PICDEM, PICDEM.net, PICkit, PICtail, PowerSmart, PureSilicon, QMatrix, REAL ICE, Ripple Blocker, SAM-ICE, Serial Quad I/O, SMART-I.S., SQI, SuperSwitcher, SuperSwitcher II, Total Endurance, TSHARC, USBCheck, VariSense, ViewSpan, WiperLock, Wireless DNA, and ZENA are trademarks of Microchip Technology Incorporated in the U.S.A. and other countries.

SQTP is a service mark of Microchip Technology Incorporated in the U.S.A.

Silicon Storage Technology is a registered trademark of Microchip Technology Inc. in other countries.

GestIC is a registered trademark of Microchip Technology Germany II GmbH & Co. KG, a subsidiary of Microchip Technology Inc., in other countries.

All other trademarks mentioned herein are property of their respective companies.

ISBN: 978-1-5224-4770-2

Quality Management System Certified by DNV

ISO/TS 16949
Microchip received ISO/TS-16949:2009 certification for its worldwide headquarters, design and wafer fabrication facilities in Chandler and Tempe, Arizona; Gresham, Oregon and design centers in California and India. The Company's quality system processes and procedures are for its PIC® MCUs and dsPIC® DSCs, KEELOQ® code hopping devices, Serial EEPROMs, microperipherals, nonvolatile memory and analog products. In addition, Microchip's quality system for the design and manufacture of development systems is ISO 9001:2000 certified.

Worldwide Sales and Service

AMERICAS

Corporate Office
2355 West Chandler Blvd.
Chandler, AZ 85224-6199
Tel: 480-792-7200
Fax: 480-792-7277
Technical Support:
http://www.microchip.com/support
Web Address:
www.microchip.com

Atlanta
Duluth, GA
Tel: 678-957-9614
Fax: 678-957-1455

Austin, TX
Tel: 512-257-3370

Boston
Westborough, MA
Tel: 774-760-0087
Fax: 774-760-0088

Chicago
Itasca, IL
Tel: 630-285-0071
Fax: 630-285-0075

Dallas
Addison, TX
Tel: 972-818-7423
Fax: 972-818-2924

Detroit
Novi, MI
Tel: 248-848-4000

Houston, TX
Tel: 281-894-5983

Indianapolis
Noblesville, IN
Tel: 317-773-8323
Fax: 317-773-5453
Tel: 317-536-2380

Los Angeles
Mission Viejo, CA
Tel: 949-462-9523
Fax: 949-462-9608
Tel: 951-273-7800

Raleigh, NC
Tel: 919-844-7510

New York, NY
Tel: 631-435-6000

San Jose, CA
Tel: 408-735-9110
Tel: 408-436-4270

Canada - Toronto
Tel: 905-695-1980
Fax: 905-695-2078

ASIA/PACIFIC

Australia - Sydney
Tel: 61-2-9868-6733

China - Beijing
Tel: 86-10-8569-7000

China - Chengdu
Tel: 86-28-8665-5511

China - Chongqing
Tel: 86-23-8980-9588

China - Dongguan
Tel: 86-769-8702-9880

China - Guangzhou
Tel: 86-20-8755-8029

China - Hangzhou
Tel: 86-571-8792-8115

China - Hong Kong SAR
Tel: 852-2943-5100

China - Nanjing
Tel: 86-25-8473-2460

China - Qingdao
Tel: 86-532-8502-7355

China - Shanghai
Tel: 86-21-3326-8000

China - Shenyang
Tel: 86-24-2334-2829

China - Shenzhen
Tel: 86-755-8864-2200

China - Suzhou
Tel: 86-186-6233-1526

China - Wuhan
Tel: 86-27-5980-5300

China - Xian
Tel: 86-29-8833-7252

China - Xiamen
Tel: 86-592-2388138

China - Zhuhai
Tel: 86-756-3210040

ASIA/PACIFIC

India - Bangalore
Tel: 91-80-3090-4444

India - New Delhi
Tel: 91-11-4160-8631

India - Pune
Tel: 91-20-4121-0141

Japan - Osaka
Tel: 81-6-6152-7160

Japan - Tokyo
Tel: 81-3-6880- 3770

Korea - Daegu
Tel: 82-53-744-4301

Korea - Seoul
Tel: 82-2-554-7200

Malaysia - Kuala Lumpur
Tel: 60-3-7651-7906

Malaysia - Penang
Tel: 60-4-227-8870

Philippines - Manila
Tel: 63-2-634-9065

Singapore
Tel: 65-6334-8870

Taiwan - Hsin Chu
Tel: 886-3-577-8366

Taiwan - Kaohsiung
Tel: 886-7-213-7830

Taiwan - Taipei
Tel: 886-2-2508-8600

Thailand - Bangkok
Tel: 66-2-694-1351

Vietnam - Ho Chi Minh
Tel: 84-28-5448-2100

EUROPE

Austria - Wels
Tel: 43-7242-2244-39
Fax: 43-7242-2244-393

Denmark - Copenhagen
Tel: 45-4450-2828
Fax: 45-4485-2829

Finland - Espoo
Tel: 358-9-4520-820

France - Paris
Tel: 33-1-69-53-63-20
Fax: 33-1-69-30-90-79

Germany - Garching
Tel: 49-8931-9700

Germany - Haan
Tel: 49-2129-3766400

Germany - Heilbronn
Tel: 49-7131-67-3636

Germany - Karlsruhe
Tel: 49-721-625370

Germany - Munich
Tel: 49-89-627-144-0
Fax: 49-89-627-144-44

Germany - Rosenheim
Tel: 49-8031-354-560

Israel - Ra'anana
Tel: 972-9-744-7705

Italy - Milan
Tel: 39-0331-742611
Fax: 39-0331-466781

Italy - Padova
Tel: 39-049-7625286

Netherlands - Drunen
Tel: 31-416-690399
Fax: 31-416-690340

Norway - Trondheim
Tel: 47-72884388

Poland - Warsaw
Tel: 48-22-3325737

Romania - Bucharest
Tel: 40-21-407-87-50

Spain - Madrid
Tel: 34-91-708-08-90
Fax: 34-91-708-08-91

Sweden - Gothenberg
Tel: 46-31-704-60-40

Sweden - Stockholm
Tel: 46-8-5090-4654

UK - Wokingham
Tel: 44-118-921-5800
Fax: 44-118-921-5820