AVR32753: AVR32 UC3 How to connect to an SSL-server

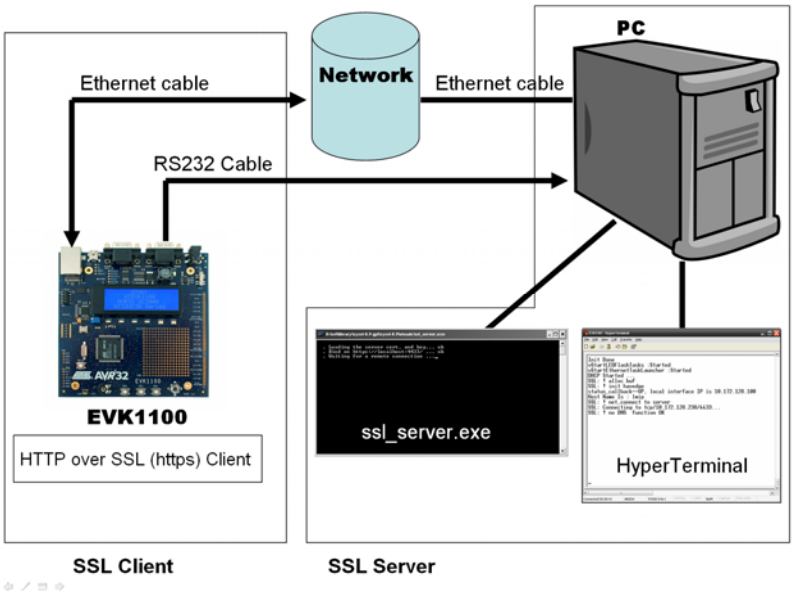
Features

- **Basic HTTPS connection to a web server running on a PC**
  - **POLARSSL Library with support of:**
    - **Block and stream ciphers: AES, ARC4 DES/3DES**
    - **Public-key cryptography: RSA, MPI, Diffie-Hellman**
    - **Cryptographic protocols: SSL/TLS client and server**
    - **Hash functions: MD2, MD4, MD5, SHA-1, SHA-256, SHA-512**
    - **Random Number Generators: HAVEGE**
  - **lwIP TCP/IP Stack 1.3.0**
  - **Free RTOS**

1 Introduction

This application note demonstrates the ability of the UC3 device to exchange messages with a server over TCP/IP connectivity through a secure socket layer connection.

Figure 1-1. SSL application



32-bit AVR® Microcontrollers

Application Note

Rev. 32111C-AVR-11/09

2 Requirements

The software provided with this application note requires several components:

- A computer running Microsoft® Windows® 2000/XP/Vista.
- AVR32Studio and the GNU toolchain (GCC) or IAR Embedded Workbench® for AVR32 compiler.
- A JTAGICE mkII or AVROne! Debugger.
- FLIP Version 3.2.1. (Available on www.atmel.com)
- An AVR UC3 evaluation kit: EVK1100.

3 Acronyms

AES: Advanced Encryption Standard, also known as **Rijndael**, is a block cipher adopted as an encryption standard.
ARC4: Alleged Rivest Cipher 4, a stream cipher developed by Ron Rivest.
DES: Data Encryption Standard, a deprecated cryptographic block cipher.
DHCP: Dynamic Host Configuration Protocol.
DNS: Domain Name System (DNS) associates various information with domain names.
EBI: External Bus Interface.
GPIO: General Purpose Input Output.
HAVEGE: Hardware Volatile Entropy Gathering and Expansion, is a user-level software unpredictable random number generator.
LwIP: Lightweight IP is a widely used open source TCP/IP stack designed for embedded systems.
MACB: Media Access Control version B.
MD2: Message Digest Algorithm 2, is a cryptographic hash function developed by Ronald Rivest in 1989. The algorithm is optimized for 8-bit computers.
MD4: Message Digest Algorithm 4, is a cryptographic hash function developed by Ronald Rivest in 1990.
MD5: Message Digest Algorithm 5, is a cryptographic hash function developed by Ronald Rivest in 1991.
RSA: Rivest, Shamir, & Adleman, public key encryption technology.
SDRAM: Synchronous Dynamic Random Access Memory.
SRAM: Static Random Access Memory.
SHA-1: Secure Hashing Algorithm 1.
SHA-256: one of the four Secure Hashing Algorithm 2.
SHA-512: one of the four Secure Hashing Algorithm 5.
SSL: Secure Sockets Layer, a communications protocol, predecessor to Transport Layer Security.

TC: Timer Counter, 16-bit Timer Counter channels available on the UC3.

TLS: Transport Layer Security, successor to Secure Sockets Layer.
TCP/IP: Transmission Control Protocol/Internet Protocol.
URL: Uniform Resource Locator is a compact string of characters used to represent a resource available on the Internet.
USART: Universal Synchronous & Asynchronous Receiver Transmitter.
POLARSSL: an open source embedded SSL/TLS library with standalone cipher and cryptic algorithm.

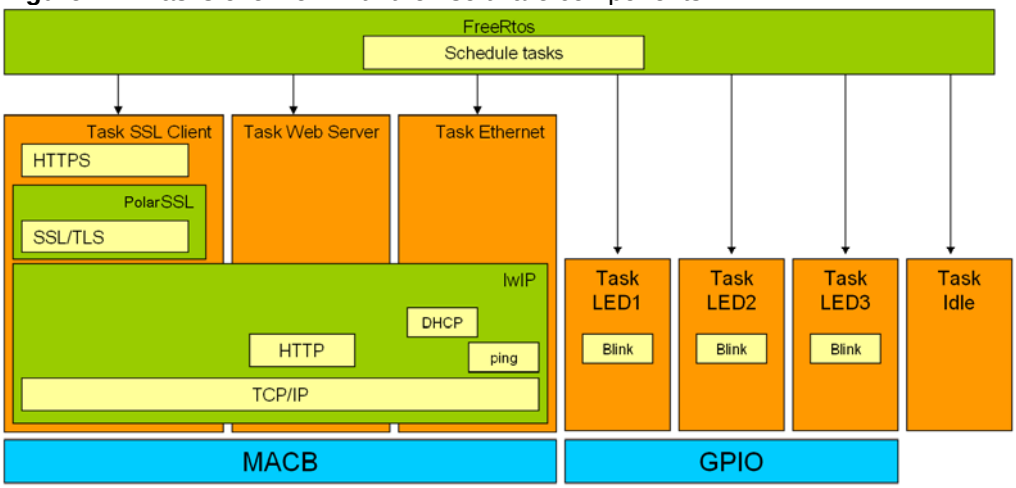
4 SSL Operations

The aim of this application is to demonstrate how to secure a message exchange between the client running on the EVK1100 board and a server. This is done by using the SSL/TLS standard. To realize this objective several major software components are needed.

4.1 Major software components

This demo application is using the software components described below.
In the following figure you can see major software components in green color.

Figure 4-1. Tasks overview with their software components



4.1.1 POLARSSL

This is a small footprint SSL library which contains all needed functions to implement an SSL/TLS server or client.
Each cipher in POLARSSL (AES, MD5, SHA-1, etc.) is self-contained and can be reused as a standalone module.
For example, the file "`aes.c`" contains all functions to operate an AES ciphering.
This demo uses a SSL/TLS client module which contains a set of functions that allow to easily create a client without a strong knowledge in SSL/TLS technologies.

4.1.2 FreeRtos

This is a portable, open source, mini Real Time Kernel.
Thanks to this OS, this demo will create different tasks such as network tasks and the basic SLL task.

4.1.3 lwIP

This is a Lightweight TCP/IP stack designed for embedded systems. This stack has been ported on UC3 to use the MACB controller of the UC3A.
The lwIP stack is executed as a FreeRtos task. This demo uses version 1.3.0, which includes supports for DHCP.

4.2 Client Operation

The Client software runs on the EVK1100. When the board is switched on, the necessary UC3 resources are initialized:

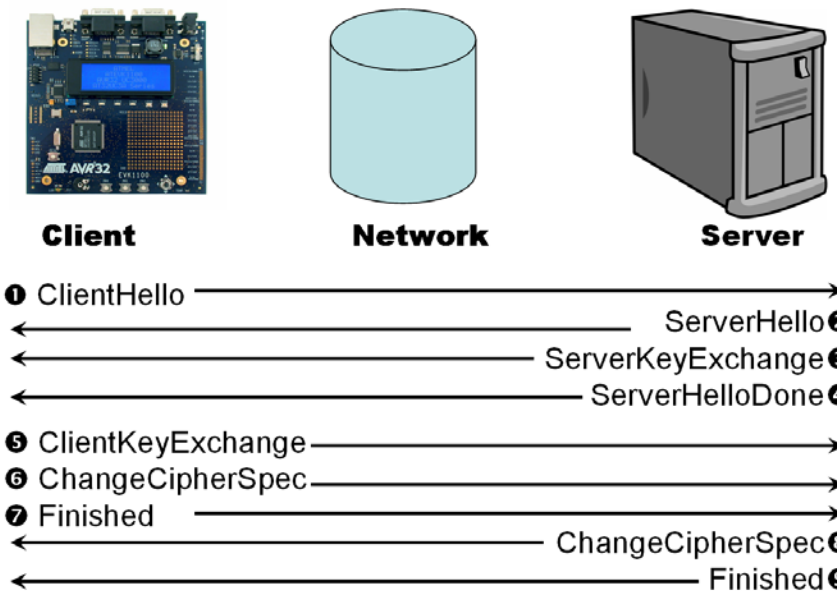
- CPU frequency
- USART
- SDRAM
- GPIO for LED blinking

At this point the program will tell the user through the USART that the initialization is correctly done.
The software will then start FreeRtos tasks:

1. LED blinking tasks: this is a set of three tasks used to visually check if the application is alive. During the demo, each task controls a LED (LED 1, 2 and 3). These LEDs will continuously blink at a different frequency.
2. Ethernet tasks: These two tasks (lwIP and Ethernet) take care of handling the TCP/IP stack, and also manage ping requests and DHCP functionality. The MACB must be initialized before these two tasks are launched.
3. Basic web server: This optional task (see Make file) is a web server which can be used for task monitoring.
4. Basic SSL client: This task handles the secure message exchange. It creates a client task that uses SSL/TLS network layers and performs the following:
   - Initializes SSL/TLS required structure.
   - Connects to a server.
   - Prepares SSL/TLS structure for a secure transaction.
   - Secures the transaction using Simple SSL/TLS handshake. (See next chapter)
   - Writes a message to the server through SSL/TLS layer.
   - Reads a message from the server through SSL/TLS layer.
   - Dumps this message through USART.
   - Closes the connection to the server.
   - Cleans all SSL/TLS requires the structures.

4.3 Secure a transaction with SSL/TLS

Using the POLARSSL client functions, the Simple SSL/TLS handshake and write message can be done in a single function ‘ssl_write’.
A simple SSL/TLS handshaking which establishes an encrypted communication is described hereafter.



1) Client sends a **ClientHello** message specifying the highest TLS protocol version it supports, a random number, a list of suggested cipher suites and compression methods.
2) Server responds with a **ServerHello** message, containing the chosen protocol version, a random number, cipher suite, and compression method from the choices offered by the client.
3) Server sends its public information in **ServerKeyExchange** message
4) Server sends a **ServerHelloDone** message, indicating it is done with handshake negotiation.
5) Client responds with a **ClientKeyExchange** message, this is the session key information encrypted with server's public key.
6) Client now sends a **ChangeCipherSpec** record, essentially telling the Server, "Everything I tell you from now on will be encrypted."
7) Client sends an encrypted **Finished** message, containing a hash and Message Authentication Code (MAC) over the previous handshake messages. The Server will attempt to decrypt the Client's **Finished** message, and verify the hash and Message Authentication Code (MAC). If the decryption or verification fails, the handshake is considered to have failed and the connection should be closed.
8) Server sends a **ChangeCipherSpec** message to activate the negotiated option for all future messages it will send.
9) Server sends a **Finished** message to let the client check the newly activated options.

Note: for more information about SSL transactions see [5].

4.4 Client Software Architecture

4.4.1 Source Code overview

4.4.1.1 Main()

The main() function of the program is located in the file:
src/SERVICES/POLARSSL/EXAMPLES/SSL_EXAMPLE/main.c

- Initialize the clock, SDRAM and UART.
- Start LED blinking tasks.
- Start the Ethernet Task which includes LwIP and MACB setup, web server task and Client SSL task.

4.4.1.2 SSL client task

This task uses the SSL client function given with POLARSSL; source code of the SSL client task can be found in:

- src/SERVICES/POLARSSL/EXAMPLES/SSL_EXAMPLE/network/basicssl/basicssl.c
- src/SERVICES/POLARSSL/EXAMPLES/SSL_EXAMPLE/network/basicssl/basicssl.h

src/NETWORK/BasicSSL/Basicssl.c
This file contains the task called: `vBasicSSLClient`, that does the following:

1. Creates and initializes structures required for SSL/TLS client application.

```
havege_state *hs;  /* Store, random number table */
ssl_context  *ssl; /* SSL sate machine used for handshaking */
ssl_session  *ssn; /* State of the current session*/

/* allocate and init havege structure */
hs = malloc(sizeof(havege_state));
havege_init( hs );

/* allocate ssl contexte struture*/
ssl = malloc(sizeof(ssl_context));
memset( ssl, 0, sizeof( ssl_context ) );

/* allocate ssl session struture*/
ssn = malloc(sizeof(ssl_session));
memset( ssn, 0, sizeof( ssl_session ) );
```

2. Starts to connect to the SSL server using the network connection function. This operation is done every second until the connection request is accepted by the server.

3. Once connected to the server, created structures are initialized with the following code; this operation is required before writing a message.
   The `ssl` structure contains much information:
   - Supported cipher algorithm.
   - SSL/TLS version.
   - State of the transaction.
   - Expiration time
   - Random number table

```
/* set client mode in ssl structure */
ssl_set_endpoint( ssl, SSL_IS_CLIENT );

/* set autentification mode, certificate is not needed */
ssl_set_authmode( ssl, SSL_VERIFY_NONE );

/* add random number structure to ssl strucutre */
ssl_set_rng( ssl, havege_rand, hs );

/* set debug function */
ssl_set_dbg( ssl, my_debug, stdout );

/* set network function and give open socket to ssl structure*/
ssl_set_bio( ssl, net_recv, &server_fd, net_send, &server_fd );

/* set supported ciphering algorithm using ssl_default_ciphers */
ssl_set_ciphers( ssl, ssl_default_ciphers );

/* set session expiration time to 600 seconds */
ssl_set_session( ssl, 1, 600, ssn );
```

4. Now, the client is able to write a message to the server using the function "`write_ssl`". All SSL handshakes are done automatically in this function.

```
/* While there are data to be written over the security layer */
while( ( ret = ssl_write( ssl, buf, len ) ) <= 0 ) {
        /* if Error during written, this is dumped on the UIART */
        if( ret != POLARSSL_ERR_NET_TRY_AGAIN ){
                PRINTF_DBG( " failed\n");
                PRINTF_DBG( "SSL : ! ssl_write returned %x(%x)\n\n",
ret,-ret );
                goto exit;
        }
}
```

5. If write is successfully done, the read operation is performed using "`read_ssl`". All read data are sent to the USART.
   The "`read_ssl`" function takes care of the transaction state.

```
do {
        len = BUF_SIZE - 1;
        /* fill read buffer with 0 */
        memset( buf, 0, sizeof( unsigned char )*BUF_SIZE );

        /* call read function to read data from server */
        ret = ssl_read( ssl, buf, len );

        /* try to read again if requested by the server */
        if( ret == POLARSSL_ERR_NET_TRY_AGAIN )
                continue;

        /* leave is server close the connection */
        if( ret == POLARSSL_ERR_SSL_PEER_CLOSE_NOTIFY )
                break;
        /* leave is other kind of error and dump error to USART */
        if( ret <= 0 ) {
                PRINTF_DBG( "failed\n  ! ssl_read returned %x,(%x)\n\n",
ret,-ret );
                break;
        }

        len = ret;
        PRINTF_DBG( " %d bytes read\n\n%s\n", len, (char *) buf );
}
while( 0 );
```

6. Once all data have been read, the connection is closed and the allocated data structures are freed.

```
net_close( server_fd );

//This function Cleanup all Memory
ssl_free( ssl );
memset( ssl, 0, sizeof( ssl ) );
```

7. A red LED starts blinking to let user know that the secure transaction is done.

```
for(;; ) {
        gpio_clr_gpio_pin(LED6_GPIO);
        vTaskDelay(200);
}
```

4.4.1.3 Drivers

The AT32UC3A drivers' sources are located in the src/DRIVERS directory.
This application requires:

- CPU/CYCLE_COUNTER: needed to compute time or delay
- EBI/SDRAMC: This driver is needed to control the SDRAM.
- FLASHC: Flash controller driver needed during clock setting to sets internal flash wait state.
- GPIO: controls the LEDs.
- INTC: interrupt handling.
- MACB: Ethernet controller for network functions.
- PM: power management, sets clock cycle.
- TC: Timer Counter, needed by freeRTOS
- USART: needed to dump the secure message received from the server to UART1.

4.4.1.4 Components

The hardware components sources are located in the src/COMPONENTS directory.
There is only one component needed for this application:

The SDRAM component used for dynamic memory allocation.

All dynamic memory allocated for this application will be done in the memory range from:

0xD0000000 to 0xD2000000

A dedicated link file:
'src/SERVICES/POLARSSL/EXAMPLE/SSL_EXAMPLE/AT32UC3A0512/GCC/link_uc3a0512.lds' (for GCC and AVR32studio)
'src/SERVICES/POLARSSL/EXAMPLE/SSL_EXAMPLE/AT32UC3A0512/IAR/lnkuc3a0512.xcl' (for IAR™)
is used to enable such functionality. This will move the heap form internal SRAM to SDRAM.

SDRAM is initialized before main in a function call **`_init_startup()`** which is located in the file :
'src/SERVICES/FREERTOS/Source/portable/GCC/AVR32_UC3/port.c'

Table 4-1. Major SSL structures and tasks stacks are stored in this SDRAM:

Structure ssl_conetxte	2168 Bytes
Structure ssl_session	96 Bytes for each session
Structure havege	36880 Bytes
SSL read buffer	16896 Bytes
SSL write buffer	16896 Bytes
Basic Web Server stack	256 Bytes
Basic SSL client stack	1500 Bytes
lwIP stack	512 Bytes
netif stack	256 Bytes

4.4.1.5 SERVICES

This application requires the following services:

POLARSSL 0.12.1: for documentation see [1].

Lwip 1.3.0: for documentation see [2].

FreeRTOS: for documentation see [3].

4.4.2 Board Definition Files

The application is designed to run on the EVK1100.

All projects are configured with the following definition: BOARD=EVK1100.

The EVK1100 definition can be found in the src/BOARDS/EVK1100 directory.

4.4.3 Project Configuration

This application can be customized by changing a few definitions such as the IP address, MAC address, use of DHCP, SSL/TLS server port…
Note: All definitions are not explained in this section, only those pertinent to this application note.

4.4.3.1 src/SERVICES/POLARSSL/EXAMPLES/SSL_EXAMPLE/conf_eth.h

This header file sets all external configurations of the Ethernet module such as the Mac address, server and client IP address.

Table 4-2. Mac address: this address must be unique, given values are in hexadecimal.

#define ETHERNET_CONF_ETHADDR0	0x00
#define ETHERNET_CONF_ETHADDR1	0x14
#define ETHERNET_CONF_ETHADDR2	0x25
#define ETHERNET_CONF_ETHADDR3	0x40
#define ETHERNET_CONF_ETHADDR4	0x40
#define ETHERNET_CONF_ETHADDR5	0x40

This configuration will set MAC address to: **00:14:25:40:40:40**

Table 4-3. Board IP address, required if DHCP is not enabled (see lwipopts.h).

`#define ETHERNET_CONF_IPADDR0`	192
`#define ETHERNET_CONF_IPADDR1`	168
`#define ETHERNET_CONF_IPADDR2`	0
`#define ETHERNET_CONF_IPADDR4`	2

This configuration will set client IP address to: **192.168.0.2**

Table 4-4. SSL server address and port address.

`#define SSL_SERVER_PORT`	**4433**, this is the TPC port where the SSL server will be waiting for the connection
`#define SSL_SERVER_NAME`	**`"192.168.0.1"`**, this is the address of the machine that host the server application. You cannot put the hosts name, because lwIP DNS function is not activated in this application

4.4.3.2 src/SERVICES/POLARSSL/EXAMPLES/SSL_EXAMPLE/conf_lwip_threads.h

Table 4-5. SSL task setting.

`#define lwipBASIC_SSL_CLIENT_STACK_SIZE`	1500, define the stack size for the SSL client task
`#define lwipBASIC_SSL_CLIENT_PRIORITY`	**`( tskIDLE_PRIORITY+6 )`** `defining the task priority`

4.4.3.3 src/SERVICES/FREERTOS/Source/include/FreeRTOSConfig.h

This file contains configuration defines for FreeRtos. There are no SSL client related defines in this file.

4.4.3.4 src/SERVICES/POLARSSL/EXAMPLES/SSL_EXAMPLE/lwipopts.h

Table 4-6. LwIP 1.3.0 configuration.

`#define LWIP_DHCP`	**1 :**(enable), this is used to enable or disable DHCP; **0** :(disable, default value) the client file uses the fixed IP address set in conf_eth.h

4.4.3.5 src/SERVICES/POLARSSL/include/polarssl/config.h

Disable or enable some POLARSSL functionalities. Handle with care because the SSL/TLS client functions need most of the modules.

4.4.4 Preprocessor definition

Some preprocessor definitions are used to enable or disable main features of the application.

Table 4-7. Enable or disable SSL task.

SSL_USED=1	(default) Build the application with the SSL client task
SSL_USED=0	Build the application without the SSL client task. Network tasks will be available for ping, and LEDs will blink but SSL/TLS transaction won't be possible.

Table 4-8. Enable or disable web server task.

HTTP_USED=1	(default) Build the application with the web server task. Connecting to this web server, returns stacks memory status. This is updated every second on a web browser. **Note:** This task, is not using the SSL connection
HTTP_USED=0	Build the application without web server task

4.5 Server Software

This application note comes with a PC SSL server program 'ssl_server.exe' available in "`src/Server`".

This SSL server application is a very simple application based on POLARSSL.

It has been built with debug option **on**, so during transactions, all handshakes information from a server point of view will be dumped on the console.

To run this server, execute '**ssl_server.exe**' in a console and the server does the following:

- It waits for a connection requests on port 4433.
- When it accepts a connection from a client
    - Executes a full handshake with the client to secure the transaction.
    - safely exchanges messages with the client

5 Running the application

5.1 Setting HyperTerminal:

Run HyperTerminal.

Table 5-1. Setup baud rate.

Baud rate	57600
Data bits	8
Parity	None
Stop bits	1
Flow control	None

Connect a RS232 cable to UART_1 of the EVK1100 board

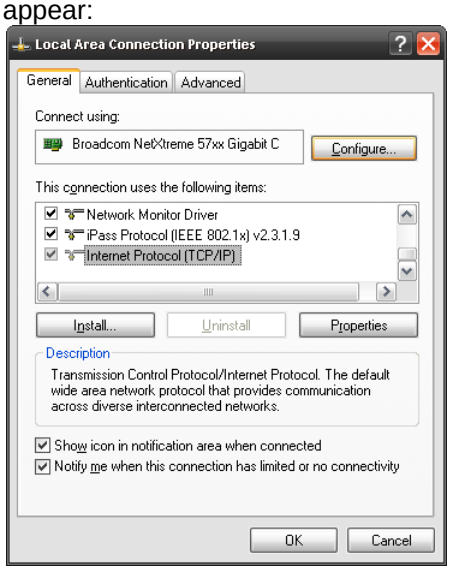
5.2 Network configuration

The PC that runs the server application has to be setup with network parameters. This depends whether the EVK1100 application has been built with or without the DHCP feature.

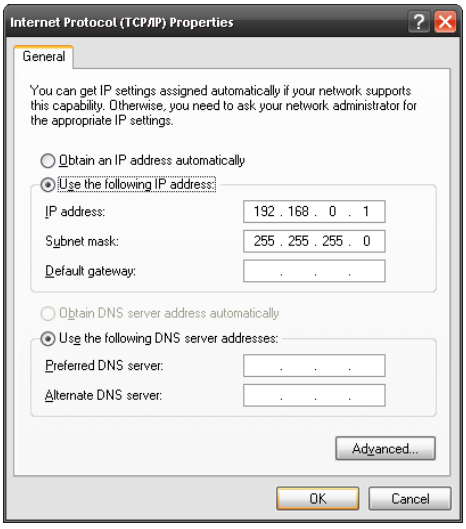
5.2.1 DHCP disabled (default)

In this mode, it is recommended to directly connect the EVK1100 to the PC running the SSL/TLS server.
Take the EVK1100 Ethernet cable, connect it to the EVK1100 connector named "ETH" and the other connector directly to the PC that hosts the SSL/TLS server.
Open the Windows control panel, and select network setting.
Right click on "Local Area Connection" and select the properties menu.

The following menu will appear:



Select “Internet Protocol (TCP/IP)” and press “Properties”.
There is a new dialog box, which must be filled as defined in the following screen shot.



Once this is configured, press OK. Now configuration is ready.
The default server IP address can be changed (see conf_eth.h)
Note: make sure the SSL_SERVER_NAME IP address (conf_eth.h) is the same IP address as the one defined for the PC

5.2.2 DHCP enable

If DHCP mode has been setup in the client application (#define LWIP_DHCP 1), there is nothing to setup. DHCP will give an address to the client application.
Plug one side of the Ethernet cable to EVK1100 connector named “eth” and the other side on an Ethernet switch, hub or router.

⚠The server address must be set in
`src/SERVICES/POLARSSL/EXAMPLES/SSL_EXAMPLE/conf_eth.h.`
To get the server IP address, execute “ipconfig” in a console.
This will give you for example:

```
IP Address. . . . . . . . . . . . : 10.175.128.1
```

Now edit src/config/conf_eth.h and change the following define:

```
#define SSL_SERVER_NAME "10.175.128.1"
```

Remove the line with the ‘#error’ message, save the file and recompile the project.

5.3 Compile Source Code

5.3.1 GCC

The GCC project is located here:

src/SERVICES/POLARSSL/ EXAMPLE/SSL_CLIENT/AT32UC3A0512/GCC/

To build the project under GCC, run the following command in a console:

‘make clean’ to clean up the project

‘make all’ to build up the full application

5.3.2 IAR

The IAR project is located in:

src/SERVICES/POLARSSL/ EXAMPLE/SSL_CLIENT/AT32UC3A0512/IAR/

To build the project under IAR :

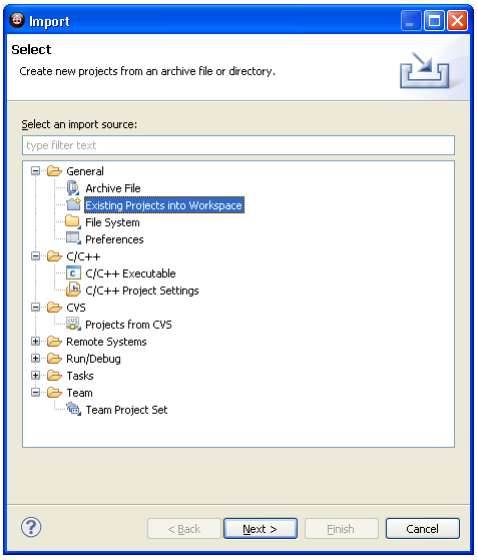
Double click on ‘ssl_example.eww’ or menu ‘File➔open➔workspace…’

Press the make button. 

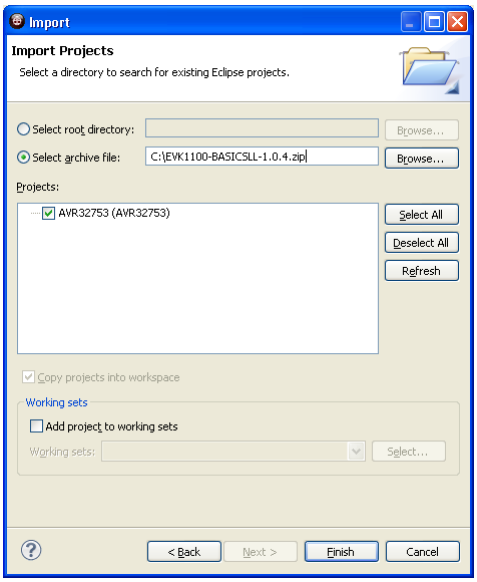
5.3.3 AVR32STUDIO

Run avr32studio.

Import Existing Project into Workspace (File->import…), press the “next” button.



Select the archive file with the browse button. Select the project AVR32753 and press the “finish” button.



The project is now available in AVR32753 project explorer folder.

Press the build button 
Load the Code: Please refer to the application note AVR32015: AVR32 Studio getting started (ref [6])

5.4 Start SSL Server

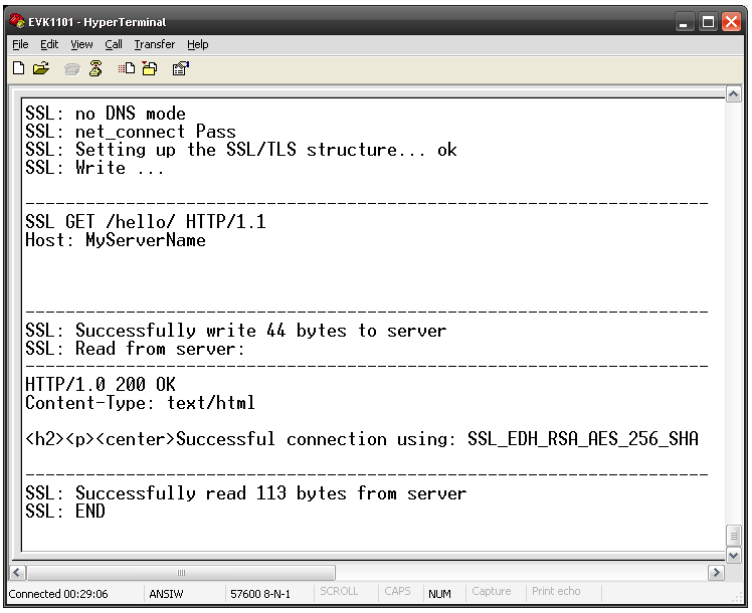
This application comes with a server application. To run this application, you can simply open a DOS console and run “ssl_server.exe” available in src/Server
Once started, this application will wait for a client connection on port 4433.

Note: This is a server application, if you using a firewall, you must be sure that port 4433 accepts connection requests. Otherwise, client connection requests will be refused by the firewall.

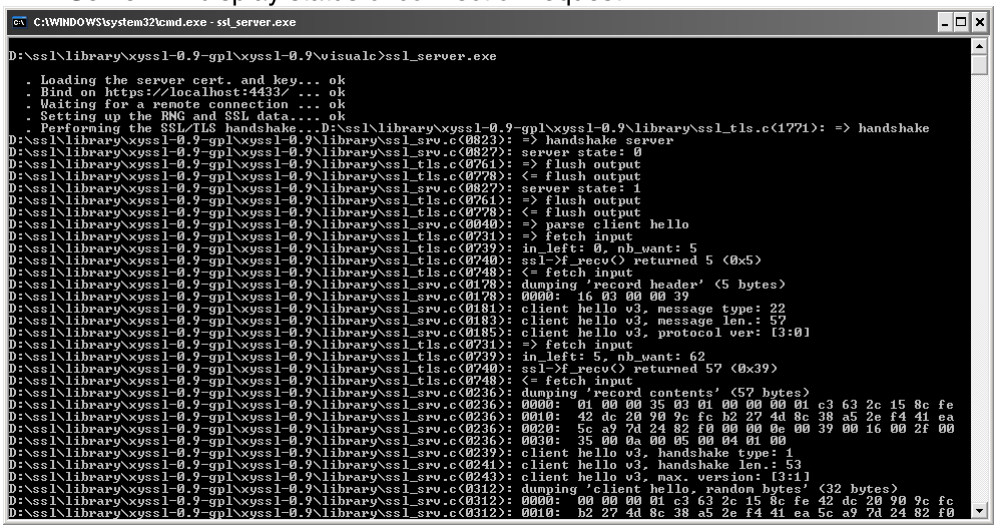
5.5 Running the Application

Power up the board.
The different part of the application should have the following behavior:

- **EVK1100**:
  LED 1, 2 and 3 are blinking, the task `vStartLEDFlashTasks` is running.
- **HyperTerminal:**
  HyperTerminal will display the status of the running application, the write message and the read messages.



- **PC SSL/TLS Server:**
  Server will display status of connection request.



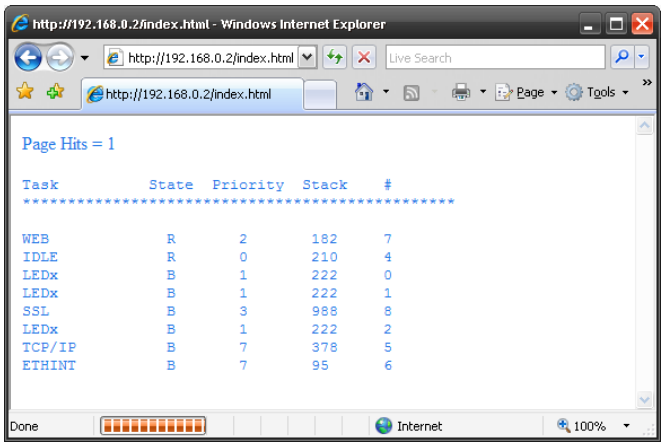
5.6 Troubleshooting

The client is not able to connect to the server:

- If the PC that hosts the server is protected by a firewall, disable it or open port 4433.
- If using DHCP, check on HyperTerminal if the local interface gets an address from the DHCP
- Check if the server address is the same as the one set in conf_eth.h.
  You can check if the client application is running using the ping command:
  e.g.: ping 192.168.0.2

If this demo application has been built with HTTP_USED=1, you can run a web browser and use the following URL: http://192.168.0.2

This will let you see the index.html page with some FreeRtos task information:



6 Reference

[1] POLARSSL
http://www.polarssl.org

[2] LWIP
http://savannah.nongnu.org/projects/lwip/

[3] FreeRTOS
http://www.freertos.org/

[4] Practical TCP/IP and Ethernet Networking for Industry.
By Deon Reynders, Edwin Wright.
ISBN-13: 978-0750658065

[5] SSL and TLS essentials: SSL & TLS Essentials: Securing the Web.
By Stephen A. Thomas
ISBN-13: 978-0471383543

[6] AVR32015: AVR32 Studio getting started
http://www.atmel.com/dyn/resources/prod_documents/doc32086.pdf

7 Appnotes Revision History

Please note that the page numbers referred to in this section are referrring to this document. The referring revision in this section is referring to the document revision.

7.1 Rev. 32111A – 11/09

1 Initial version

Headquarters

Atmel Corporation
2325 Orchard Parkway
San Jose, CA 95131
USA
Tel: 1(408) 441-0311
Fax: 1(408) 487-2600

International

Atmel Asia
Unit 1-5 & 16, 19/F
BEA Tower, Millennium City 5
418 Kwun Tong Road
Kwun Tong, Kowloon
Hong Kong
Tel: (852) 2245-6100
Fax: (852) 2722-1369

Atmel Europe
Le Krebs
8, Rue Jean-Pierre Timbaud
BP 309
78054 Saint-Quentin-en-
Yvelines Cedex
France
Tel: (33) 1-30-60-70-00
Fax: (33) 1-30-60-71-11

Atmel Japan
9F, Tonetsu Shinkawa Bldg.
1-24-8 Shinkawa
Chuo-ku, Tokyo 104-0033
Japan
Tel: (81) 3-3523-3551
Fax: (81) 3-3523-7581

Product Contact

Web Site	***Technical Support***	***Sales Contact***
www.atmel.com	avr32@atmel.com	www.atmel.com/contacts

Literature Request
www.atmel.com/literature