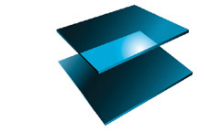
AN017 – eCOG1 uIP Embedded TCP/IP Stack

Version 1.5

Contents

Confidential and Proprietary Information



cyan technology

Revision History

Version	Date	Notes
V1.1	22/07/2004	First release
V1.2	23/09/2004	Updated list of trademarks
V1.3	10/11/2004	Updated memory usage and performance figures
V1.4	11/03/2005	Updated for CyanIDE V1.1
V1.5	09/06/2006	Added references to DOSMakeFS.pl and ActiveState PERL interpreter.

1 Introduction

This application note describes briefly the eCOG1 port of the uIP TCP/IP stack. It also describes a sample application which implements a simple embedded web page server.

The open-source uIP package provides an implementation of the TCP/IP protocol stack for embedded microcontrollers, without sacrificing interoperability or RFC standards compliance. It provides the necessary protocols for Internet communication, with very small code and data memory requirements.

uIP is developed by Adam Dunkels at the Swedish Institute of Computer Science. For more information about uIP, please refer to the uIP documentation or visit the uIP web site at http://www.sics.se/~adam/uip/.

The uIP stack was ported to the eCOG1 evaluation board by Javier Cardona of Cozybit and has been optimised further for the eCOG1 at Cyan.
This implementation is based on uIP version 0.9.

2 Glossary

eCOG1	Cyan Technology target micro controller
HTML	HyperText Markup Language
HTTP	HyperText Transfer Protocol
MAC	Media Access Controller
TCP/IP	Transmission Control Protocol / Internet Protocol
uIP	A low memory implementation of the TCP/IP stack

3 Requirements

- PC running eCOG1 CyanIDE Version 1.1.
- An eCOG1 evaluation board.
- A twisted-pair Ethernet hub or crossover cable

A zip archive file named *AN017SW.zip* contains the uIP documentation and example code for this application note, and is available for download from the Cyan Technology website www.cyantechnology.com. The uIP documentation is also available separately in the archive file *uIP_docs.zip*.

4 An Embedded Web Server Application

4.1 File Locations

Open the archive file *AN017SW.zip* and extract the files to the CyanIDE examples directory for the eCOG1 development board <*C:\Cyan Technology\CyanIDE\examples\ecog1 dev board*>. The files are extracted into a sub-directory <*uip-cyan*>.

Documentation for uIP is available, both in Acrobat format and in html format.

- Acrobat format <*uip-cyan\doc\uip-refman.pdf*>
- HTML format <*uip-cyan\doc\html\index.html*>

The eCOG1 software port is in the directory <*uip-cyan\ecog1*>.

4.2 Configuration

Start CyanIDE and open the embedded web server project, normally located in <*CyanIDE\examples\ecog1 dev board\uip-cyan\ecog1\httpd\http.cyp*>.

Open the file <*uipopt.h*> and look for the constants *UIP_IPADDR0* to *UIP_IPADDR3*. These constants form the IP address for the embedded web server. The default address is 192.168.200.2; this address may be left unchanged, since it is a private address that does not transfer across a router.

This file also contains other configuration parameters for the uIP stack, but no other changes should be necessary in order to compile and run the example web server application. The Ethernet MAC physical address is read from the Ethernet hardware at run-time and does not need to be set in the file.

4.3 HTML Content

The HTML files that are served by the HTTP server application are located in the sub-directory <*uip-cyan\apps\httpd\fs*>. The application includes these as fixed binary data. To change the web page content, modify the HTML source files as required and convert them from HTML to C source data using one of the PERL scripts provided; for a Unix or Linux system, the script file is <*apps\https\makefsfdata*>, for a PC it is <*apps\https\DOSMakeFS.pl*>. The PERL script converts the HTML files into a C source file *fsdata.c* and adds the headers required by the HTTP protocol. The generated file *fsdata.c* is included in the web server project via the file *fs.c*. A suitable PERL script interpreter is ActivePERL, available as a free download from ActiveState at www.activestate.com for AIX, HP-UX, Linux, Mac OS X, Solaris and Windows.

Note that the HTML files are stored as constant data in the eCOG1's flash memory. The example project is configured to hold up to a total of 8Kwords (16Kbytes) of constant data. The total size of the HTML content, including HTTP headers as well as all the constant variables used in your code, should not exceed that size. If it does, then the following warning is given when the project is compiled:

```
Warning     : Some of the program data was not written to a ROM file.
Description : starting at address H'2000
```

It is possible to find out the exact size of the static file system by using a text editor to count the number of occurrences of “0x” in *fsdata.c*.

4.4 Building the Project

The project has two different build configurations, Debug and Release. One or the other can be selected from the Build menu. The Debug configuration shows the Ethernet activity on the evaluation board LEDs (LED0-3), which may be useful for troubleshooting the system. The colour code is as follows:

- Green LED: An Ethernet frame has arrived
- Blue LED: An Ethernet frame was transmitted
- Red LED: An Ethernet frame could not be transmitted and was discarded
- Orange LED: Incoming frames arrive faster than they can be processed

To build the project, just click on the Build button or press the F7 key. The following output messages should appear in the Build window:

```
Building http [Debug]
ecog1cfg devboard.cfg -o temp/devboard.asm -quiet
ecogncc -quiet -zpe1 -I. -I../../../driver_lib -I../../apps/httpd -I../../uip -
I../../ecog1 -otemp/smsc91c111.asm ../smsc91c111.c
ecogncc -quiet -zpe1 -I. -I../../../driver_lib -I../../apps/httpd -I../../uip -
I../../ecog1 -otemp/uip_arch.asm ../uip_arch.c
ecogncc -quiet -zpe1 -I. -I../../../driver_lib -I../../apps/httpd -I../../uip -
I../../ecog1 -otemp/uip_arp.asm ../../uip/uip_arp.c
ecogncc -quiet -zpe1 -I. -I../../../driver_lib -I../../apps/httpd -I../../uip -
I../../ecog1 -otemp/uip.asm ../../uip/uip.c
ecogncc -quiet -zpe1 -I. -I../../../driver_lib -I../../apps/httpd -I../../uip -
I../../ecog1 -otemp/httpd.asm ../../apps/httpd/httpd.c
ecogncc -quiet -zpe1 -I. -I../../../driver_lib -I../../apps/httpd -I../../uip -
I../../ecog1 -otemp/fs.asm ../../apps/httpd/fs.c
ecogncc -quiet -zpe1 -I. -I../../../driver_lib -I../../apps/httpd -I../../uip -
I../../ecog1 -otemp/cgi.asm ../../apps/httpd/cgi.c
ecogncc -quiet -zpe1 -I. -I../../../driver_lib -I../../apps/httpd -I../../uip -
I../../ecog1 -otemp/main.asm main.c
ecogncc -quiet -zpe1 -I. -I../../../driver_lib -I../../apps/httpd -I../../uip -
I../../ecog1 -otemp/timeofday.asm ../timeofday.c
ecogncc -quiet -zpe1 -I. -I../../../driver_lib -I../../apps/httpd -I../../uip -
I../../ecog1 -otemp/putchar.asm putchar.c
ecogncc -quiet -zpe1 -I. -I../../../driver_lib -I../../apps/httpd -I../../uip -
I../../ecog1 -otemp/irqutil.asm ../irqutil.c
ecogcl -pack cstartup -map internal -I. -I../../../driver_lib -I../../apps/httpd -
I../../uip -I../../ecog1 -lib ../../../driver_lib -o out/http -cmdfile temp/files.tmp

out/http.xpv - 0 error(s), 0 warning(s)
eCOG Compiler/Linker V1.1 (17 November 2004)
(c) Cyan Technology Limited 2002, 2003, 2004
Processing command line files ...
Processing library files ...
Packing segments ...
Final assembly ...
NO ERRORS FOUND
Generating ROMs ...
Build finished.
```

4.5 Downloading and Execution

Click on the Run button, or press the F5 key. The following messages should appear in the Build window:

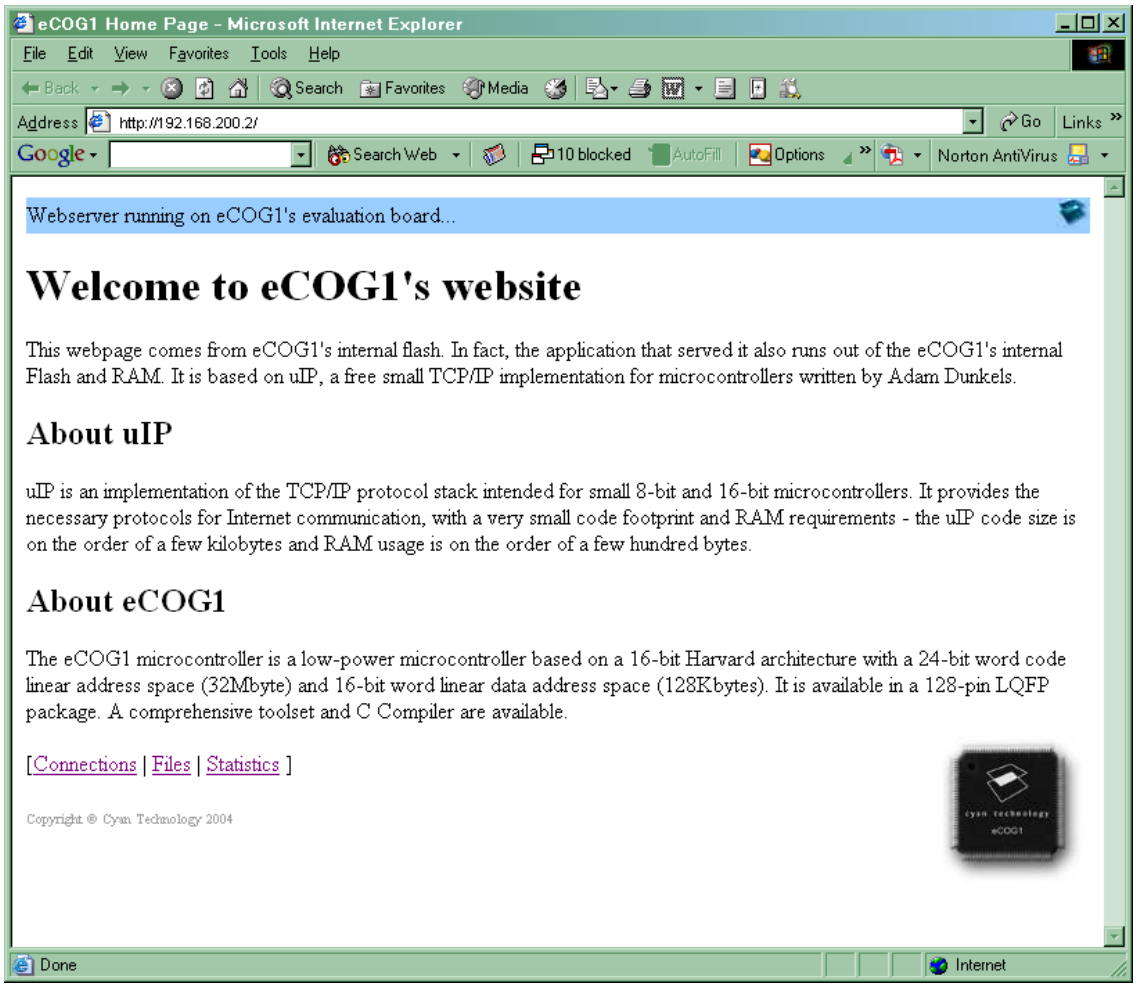
```
Trying eCOG1 on eICE id 0, chip 1...
Connected to eCOG1 on eICE id 0, chip 1
Downloading and verifying flash programmer.....................100%
Downloading and verifying http.rom.....................100%
Loading symbols from
'C:/Cyan Technology/CyanIDE/examples/uip-cyan/ecog1/httpd/out/http.sym'
Running...
```

4.6 Setting up the Host IP Address

The IP address of the host PC should be configured manually to be in the same subnet as your target. If the embedded web server is configured to use the default address (192.168.200.2), then use 192.168.200.1 for the host PC. The net mask should be set to 255.255.255.0.

4.7 Viewing the Embedded Web Pages

Open up a web browser application, and go to http://192.168.200.2. If the embedded server is running correctly and the network connection is ok, the following web page should be seen in the browser.



4.8 Memory Usage

The httpd example project (Release build) uses memory as follows:

Code size	**23284 bytes**
C startup and interrupt vectors	790 bytes
Peripheral configuration	1536 bytes
Ethernet device driver	5706 bytes
uIP stack	6124 bytes
httpd server application	3198 bytes
I/O	246 bytes
Std library	4852 bytes
LCD routines	832 bytes

Constant data	**7872 bytes**
Constant data size	7816 bytes
Initialized data size	56 bytes

Total size of code + constant data in flash memory = 31156 bytes.

Data size (in RAM)	**3114 bytes**
Interrupt stack (max.)	128 bytes
User stack (max.)	512 bytes
Heap (max.)	256 bytes
Initialised data	56 bytes
Static variables	2062 bytes

5 Building a New Application

5.1 Overview

This section describes briefly how to create a new application project that uses the uIP TCP/IP stack. A simple test program is used as an example.

5.2 Creating the Project

To create a new project within CyanIDE, select *Project/New* from the main menu. Select the *Eval Board Project* template from the available eCOG1 Microcontroller projects.

Enter a name for the project, for example *test1*, and set the project directory to <*C:\Program Files\Cyan Technology\CyanIDE\examples\ecog1 dev board\uip-cyan\ecog1*>. The new project is created in a sub-directory <*test1*>, and the following files are created.

- cstartup.asm
- evalboard.cfg
- internal.map
- irq.asm
- main.c
- test1.cyp

5.3 File Locations

The files used by the *test1* application are best located in the following directories, for consistency with the uIP source code and header files. This allows the common files to be shared between projects without having to duplicate the source code in the different projects.

- <*uip-cyan\apps\test1*>
  This directory contains the files *test1.c* and *test1.h*.
  The file *test1.c* contains initialisation functions, and the application entry point which is called in response to events.
  The file *test1.h* defines the symbol *UIP_APPCALL* as the application function name, contains a structure to store the application state, and the size of this structure.
- <*uip-cyan\ecog1\test1*>
  This directory contains the files created by the new project, as listed above. File *main.c* is the main entry point of the application code. The main function in the web server example can be reused as a starting point for other projects.
  File *cstartup.asm* is created automatically by CyanIDE.
  File *devboard.cfg* is also created automatically, and contains the chip configuration data as defined in the configuration editor tool.
  File *irq.asm* contains the interrupt vectors. The template creates a default set of interrupt vectors which all point to a minimal interrupt handler routine. Any interrupts used by the application must have their interrupt handler routines entered into the vector table.
- <*uip-cyan\ecog1*>
  Contains the TCP/IP stack files plus some other files with different functions.
  File *irqutil.c* contains a copy of the *gpio_wr* function which can be called from the interrupt service routines.
  File *smsc91c111.c* contains the LAN controller device driver functions.
  File *timeofday.c* contains the time-of-day interrupt handler (*tmr_handler*).
  File *uip.c* contains the uIP TCP/IP stack code.
  File *uip_arch.c* contains CPU architecture dependent uIP functions.
  File *uip_arp.c* implements ARP, the address resolution protocol.
  File *uipopt.h* contains configuration options for uIP. This file must be modified to set the required IP address settings, and to include the suitable *appname.h* header file.

All the directories used for project files need to be added to the compiler and linker search paths within the CyanIDE environment. Select *Project/Properties* from the main menu. Select the *Compiler/Directories* item in the left pane of the dialogue to display the current list of include directories for the project. Click in the *value* field in the right pane, and set the list of include directories as follows:

```
../../../driver_lib;../../uip;../../ecog1;../../ecog1/test1
```

Select the *Linker/Directories* item in the left pane, and set the list of library directories as follows:

```
../../../driver_lib
```

It should now be possible to compile and build the project successfully, provided all the file dependencies are correct, and the code and data segment sizes are not exceeded.

5.4 Test Application

The sample project *test1* shows how to implement a simple application on top of the uIP TCP/IP stack. It is a modified version of *example1* from the uIP documentation. It opens a connection by listening to port 1234. Whenever data is received from the remote host it replies by sending the string “OK\r\n” in an Ethernet frame. If the project is built in a Debug configuration, then it also echoes the contents of the received frame stored in uip_buf to the serial port.

If the project is located under the folder <*examples\ecog1 dev board\uip-cyan*>, ensure that the application header file name included in the *uipopt.h* file is *test1.h*. By default it is *httpd.h* for the embedded web server application.

To test this simple application, a suitable test program may be used on the host machine. One example is the Socket Workbench package from Sigma Solutions. A free trial version can be downloaded from their web site at http://www.sigmasolutions.com.au. It can be configured as a client or server on the specified port. It allows the user to connect a socket to a remote host on a specified port and send data to it, and to see all the sent and received data.

For this example, Socket Workbench is used in a client configuration. The server IP address specified must match the one configured in the evaluation board (as specified in *uipopt.h* provided that *UIP_FIXEDADDR* is 1). If the evaluation board is configured for the suggested default address, then set the host machine to 192.168.200.1. The host port number must be set to the same port as the one to which the application is listening (1234). The socket workbench allows a socket connection to be made to the remote host (evaluation board) on the specified port. Once it is connected, it allows test data to be sent and received.

When running both the example application on the evaluation board and the Socket Workbench on the host machine with the settings specified above, data sent from Socket Workbench is received by the example program. If the test1 application is built in a Debug configuration, then received Ethernet frames are echoed via the serial port. A terminal emulator program such as HyperTerminal can be used to view the received frames. The normal settings are 9600 baud, 8 data bits, 1 stop bit, no parity and no flow control.

The test project can be modified or extended to experiment with the different events that cause the application to be called.

5.5 Source Code

The main source code for this test application is listed here:

```
/*---------------------------------------------------------------------------*/
/**
 * Initialisation function
 *
 * Starts to listen for incoming connection requests on TCP port 1234
 */
/*---------------------------------------------------------------------------*/
void test1_init(void)
{
    uip_listen(HTONS(1234));
}


/*---------------------------------------------------------------------------*/
/**
 * Application function
 *
 * If the application was called because data was received, send "OK\r\n" to the
 * remote host and dump the received frame to the serial port
 */
/*---------------------------------------------------------------------------*/
void test1_appcall(void)
{
    switch(uip_conn->lport)
    {
        case HTONS(1234):

        if (!pckt)
        {
            stats();
        }

        if uip_newdata()
        {
            pckt++;

            uip_send((u8_t *)"OK\r\n", 4);

#ifdef DEBUG
            if ((pckt == 1) || (uip_len != 1460))
            {
                print_uip_buf(1);
                printf("uip_len=%d\r\n", uip_len);
            }
#endif
        }

        if ((uip_len ==1))
            stats();
        break;

        default:
            break;
    }
}
```

5.6 Memory Usage

The test1 example project (Release build) uses memory as follows:

Code size	**20724 bytes**
C startup and interrupt vectors	570 bytes
Peripheral configuration	1536 bytes
Ethernet device driver	5706 bytes
uIP stack	6124 bytes
test1 application	778 bytes
I/O	246 bytes
Std library	4854 bytes
LCD routines	838 bytes

Constant data	**294 bytes**
Constant data size	284 bytes
Initialized data size	10 bytes

Total size of code + constant data in flash memory = 21018 bytes.

Data size (in RAM)	**2942 bytes**
Interrupt stack	128 bytes
User stack	512 bytes
Heap	256 bytes
Initialised data	10 bytes
Static variables	2036 bytes

5.7 Performance

Socket Workbench can be set for a periodic transmission of a packet containing a fixed amount of data. Any received packets are displayed in a log window. The repetitive transmission rate can then be varied in Socket Workbench to determine approximately the maximum rate at which packets are sent and acknowledged correctly by the *test1* program running on the evaluation board.

A payload of 1460 bytes was set in Socket Workbench to observe the behaviour of the *test1* example application. This is the payload for an Ethernet frame of the maximum receive buffer size (1514 bytes including headers). With this packet size, a release build of the *test1* program receives and acknowledges packets successfully at the maximum repetition rate available in the Socket Workbench program of 1 frame per millisecond (approximately 1.5 Mbps). The release build does not echo any received packet data to the serial port.

A debug build of the *test1* program echoes any received packet data to the serial port. In this case the fastest frame repetition rate for proper behaviour of the application (correct reception of frames with 1460 bytes of payload) was 1 frame every 20 milliseconds (approximately 75.7 Kbps). At one frame every 10 milliseconds the received packets were not extracted from the buffer correctly before the next frame was received, but this is due to the delay in printing the packet data via the serial port.

Note that the achieved performance depends significantly on the complexity of the application code. This example is a very simple application.

6 The Dumper Utility

6.1 Overview

The dumper utility echoes to the serial port every Ethernet frame that is received by the device driver, regardless of the destination IP address. It configures the LAN controller in promiscuous mode to receive all packets.

This utility allows the user to check that the driver reads the frames with no problem, as well as the LAN connection activity. If the driver is working properly it should respond to the ping requests (ICMP packets) sent to its IP address.

6.2 Memory Usage

The dumper utility (Release build) uses memory as shown in the tables below. Note that it uses only the Ethernet device driver, not the uIP TCP/IP stack code.

Code size	**15002 bytes**
C startup and interrupt vectors	768 bytes
Peripheral configuration	1614 bytes
Ethernet device driver	6360 bytes
uIP stack	0 bytes
dumper utility	772 bytes
I/O	246 bytes
Std library	4328 bytes
LCD routines	842 bytes

Constant data	**204 bytes**
Constant data size	202 bytes
Initialized data size	2 bytes

Total size of code + constant data in flash memory = 15206 bytes.

Data size (in RAM)	**1984 bytes**
Interrupt stack	64 bytes
User stack	256 bytes
Heap	0 bytes
Initialised data	2 bytes
Static variables	1662 bytes