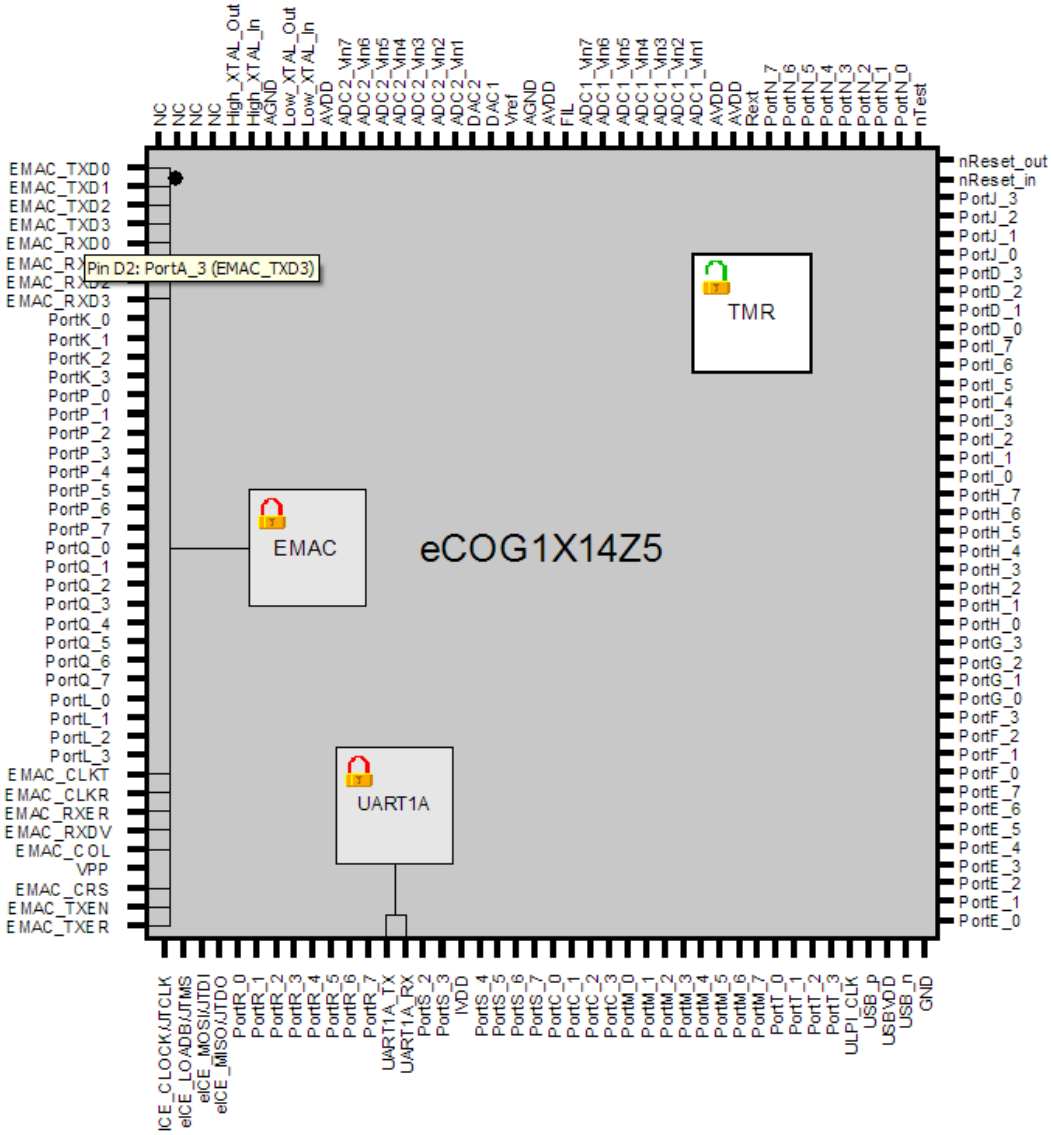
AN057 – uIP V1.0 TCP/IP Stack for eCOG1k and eCOG1X

Version 1.0

This application note describes how to use the uIP V1.0 TCP/IP Library with the eCOG1k and eCOG1X microcontrollers.



Confidential and Proprietary Information

Revision History

Version	Date	Notes
V1.0	05/07/2007	Initial Release

Contents

1 Introduction

This application note describes the uIP V1.0 TCP/IP stack implementation for the eCOG1k and eCOG1X microcontrollers. It also describes the following simple applications:

- Hello World example
- Telnet server (Telnet daemon).
- Web Server (HTTP daemon).
- DNS Resolver example

The uIP TCP/IP stack is implemented as a library in the CyanIDE development tools, and is suitable for both eCOG1k and eCOG1X. Included within the uIP library are the following clients that are available for all applications to use:

- DHCP Client
- DNS Client

Additional services, protocols and application layer components will be added in future.

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 visit the uIP web site at http://www.sics.se/~adam/uip/. The uIP reference manual is available at http://www.sics.se/~adam/download/uip-1.0-refman.pdf.

2 Glossary

DHCP	Dynamic Host Configuration Protocol for allocating IP addresses
DNS	Directory Name System for translating hostnames to IP addresses
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 1.4.1 or later.
- An eCOG1k development board (Iss. 2) or an eCOG1X development board (Rev. B)
- An Ethernet connection

A zip archive file named *AN057LIB.zip* contains the common uIP library code for this application note, and an updated version of the eCOG1X EMAC peripheral support library. A second zip archive file named *AN057SW.zip* contains the source code for use with this application note as a number of example projects. These two zip files are available for download from the Cyan Technology website www.cyantechnology.com.

4 Software Installation

Extract the library software file *AN057LIB.zip* to the CyanIDE install directory, usually *C:\Program Files\Cyan Technology\CyanIDE*. This creates a new library directory *uIP1.0* alongside the existing *eCOG1k* and *eCOG1X* library directories, and updates the files in the eCOG1X EMAC peripheral library.

Extract the example software file *AN057SW.zip* to any convenient location, such as the CyanIDE projects directory *C:\Documents and Settings\<username>\My Documents\CyanIDE Projects*. This creates a new directory *AN057SW* with three further subdirectories. Two directories contain the example projects for the eCOG1k development board V2.1 and for the eCOG1X development board rev B. The third directory contains project templates suitable for applications using uIP; to add these to the standard templates, simply copy them to the CyanIDE templates directory *C:\Program Files\Cyan Technology\CyanIDE\templates*.

5 uIP Configuration

All uIP projects must contain a configuration file for the library software. This is included in the project as ***uip-conf.h***, and contains all the definitions used by uIP. The most significant of these definitions are described below.

5.1 UIP_CONF_FIXEDADDR

Set this flag to True (non-zero) if the IP address of the uIP Stack is fixed at compile time. Set it to False (zero) if the IP address can be assigned dynamically at run-time, for example when using the DHCP client.

If the IP address is fixed, then the UIP_IPADDRn, UIP_NETMASKn and UIP_DRIPADDRn symbols should be set in the header file to define the IP Address, the Net Mask and Default Router respectively.

5.2 UIP_CONF_PINGADDRCONF

If this flag is set to True (non-zero), then the IP address of the uIP stack is defined by the destination IP address of the first PING packet received by the stack after initialisation. Note that if this is used then UIP_CONF_FIXEDADDR must be set to False.

5.3 UIP_CONF_UDP

Set this flag to True (non-zero) if the uIP stack should include the code to handle UDP connections. This is required if either the DHCP or DNS clients are used by the application.

5.4 UIP_CONF_UDP_CONNS

This value defines the maximum number of UDP connections supported by the uIP stack at any one time. The more connections supported, the more memory is required by the uIP stack. Fourteen bytes of RAM are required by each connection.

5.5 UIP_CONF_ACTIVE_OPEN

Set this flag to True (non-zero) if the uIP stack should include the code to allow the application to open connections to other devices. Set it to False (zero) if the application only receives connections from other devices.

5.6 UIP_CONF_MAX_CONNECTIONS

This value defines the maximum number of TCP connections supported by the uIP stack at any one time. The more connections supported, the more memory is required by the uIP stack. Thirty-two bytes of RAM are required by each connection.

5.7 UIP_CONF_MAX_LISTENPORTS

This value defines the maximum number of TCP ports on which the uIP stack can listen for connections. This differs from the maximum number of connections, as a single TCP port may have multiple connections.

5.8 UIP_CONF_BUFFER_SIZE

This value defines the size of the packet buffer used by uIP. The uIP stack uses one buffer to handle all its IP transactions. The maximum useful size for this is equal to the maximum size of an Ethernet packet, 1518 bytes.

In applications running on the eCOG1k (which has less internal RAM available than the eCOG1X), this value can be reduced to keep more memory available for other uses.

5.9 UIP_CONF_STATISTICS

Set this flag to True (non-zero) if the uIP stack should include support for network packet statistics. This requires an extra 44bytes of RAM, or 52bytes of RAM if the UDP support is also included in the uIP stack.

6 uIP Callbacks

The uIP stack makes use of callback functions to interact with the application. These are described below.

6.1 network_device_init

```
void network_device_init(void)
```

This is called by the initialisation of the uIP stack. It should initialise the Ethernet hardware interface, set the MAC address in the Ethernet interface and inform the uIP stack of the MAC address using the `uip_setethaddr()` function.

6.2 network_device_read

```
u16_t network_device_read(void)
```

This is called by the uIP stack to check whether an Ethernet packet has been received by the interface. If a packet has been received, then it should be copied into `uip_buf` and the number of bytes in the packet returned. If no data has been received, then zero should be returned.

6.3 network_device_send

```
void network_device_send(void)
```

This is called by the uIP stack to send an Ethernet packet. The Ethernet header and IP header are contained in `uip_buf` and the data is pointed to by `uip_appdata` (which may or may not point to `uip_buf`). The function should return once the data has been copied from the buffers and they are available for use again.

6.4 uip_tcp_appcall

```
void uip_tcp_appcall(void)
```

This is called by the uIP stack to inform the application of a TCP event. The application should use the connection information pointed to by `uip_conn` to determine which port and what type of event has occurred.

6.5 uip_udp_appcall

```
void uip_udp_appcall(void)
```

This is called by the uIP stack to inform the application of a UDP event. The application should use the connection information pointed to by `uip_udp_conn` to determine which port and what type of event has occurred.

If the DHCP client is used by the application, this callback should include a call to the `dhcpc_appcall()` function.

If the DNS client is used by the application, this callback should include a call to the `resolv_appcall()` function.

6.6 dhcpc_configured

```
void dhcpc_configured(const struct dhcpc_state *s)
```

This callback function is called by the DHCP client (if used by the application) to inform the application that an IP address has been acquired by the client. The `dhcpc_state` structure contains the IP addresses assigned, the netmask and default router IP address. It also contains the IP address of the DNS server that can be passed on to the DNS client.

If DHCP is not used, then these parameters must be configured by the application in some other way.

6.7 resolv_found

```
void resolv_found(char *name, uip_ipaddr_t *ipaddr)
```

This callback function is called by the DNS client to inform the application of success or failure in resolving a name queried by the `resolv_lookup()` function. If the `ipaddr` pointer is NULL then the lookup failed, otherwise it points to the IP address.

7 uIP Application Data

If the application is providing a server function (such as the Web Server example), then the application has to store some information on a per connection basis.

To facilitate this, the `uip_conns[]` data structure array (for TCP connections) and `uip_udp_conns[]` data structure array (for UDP connections) contain an element which is a void pointer `appdata` that can be used to link a connection with application specific data.

The main application initialises these pointers to a union of all the application data structures. A union is used as each connection can only be used by one application at a time.

When an application receives a callback from the uIP stack for connection (via the `uip_tcp_appcall()` callback for TCP connections or via `uip_udp_appcall()` callback for UDP connections), it can then cast the `appdata` pointer (`uip_conn->appdata` for TCP connections and `uip_udp_conn->appdata` for UDP connections) to the correct type for the appropriate connection. It can then maintain the per connection information for that connection.

Application code that acts as a client does not need to store any per connection data and so these unions do not require initialisation.

8 uIP Timer

The uIP Stack requires a timer to allow it to implement timeouts on transactions. This is implemented by the application project, as it is hardware-specific, and is contained in the files *clock-arch.c* and *clock-arch.h*.

8.1 clock-arch.h

This defines the type of the clock timer (usually type u16_t), as well as prototypes for the functions `clock_time()` and `clock_init()`. It also contains a symbol definition `CLOCK_CONF_SECOND` that defines the number of clock ticks in one second. For example, if the clock timer is incremented every millisecond, then `CLOCK_CONF_SECOND` is defined as 1000.

Applications should use `CLOCK_CONF_SECOND` to scale any time based measurements they perform using the uIP timer, rather than assuming the time base of the uIP timer.

8.2 clock-arch.h

This contains the definitions of the `clock_time()` and `clock_init()` functions.

8.2.1 clock_time()

```
clock_time_t clock_time(void)
```

This returns the current clock time in clock ticks.

8.2.2 clock_init()

```
void clock_init(void)
```

This function initialises the clock timer and starts it running. It is called from the uIP initialisation functions and does not need to be called explicitly.

9 Example Applications

All the example applications are provided for both the eCOG1k and eCOG1X development boards.

The eCOG1k development board uses an external SMSC LAN91C111 integrated Ethernet MAC and PHY device for its Ethernet interface.

The eCOG1X development board uses the EMAC peripheral integrated into the eCOG1X together with an external PHY device for its Ethernet interface.

All projects have three build configurations. These are:

Release – Release build with minimal debug information.

Debug – Debug build with most of the debug information. The RS232 serial port is used to display the debug information, which consists of a '.' whenever the uIP stack performs its periodic functions every half a second, an 'R' when the uIP stack receives data and a 'T' when the uIP stack transmits data. On the eCOG1k development board, status is also indicated on the LEDs (LED0-3). 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
- Yellow LED Incoming frames arrive faster than they can be processed

Full Debug – The full debug information is provided, which includes printing the contents of all transmitted and received packets.

9.1 Hello World Example

The 'Hello World' example is a demonstration of how to write uIP applications with protosockets, a feature introduced to uIP in the V1.0 release. Protosockets provide a sequential programming interface to uIP that makes application programming with uIP easier. In addition, protosockets add a very small memory overhead: about 1Kbytes of extra code and 26 bytes of extra data per TCP connection. For more information, see http://www.sics.se/~adam/uip/protosockets.html

This application uses a fixed IP address, so UIP_CONF_FIXEDADDR is set to 1 (true) and the UIP_IPADDRn, UIP_NETMASKn and UIP_DRIPADDRn values should be set appropriately for the network.

To test the application build it and run it, then open a Windows command shell by choosing *Start->Run*. Enter `cmd` and select `OK`. At the command line, enter:

```
telnet ipaddress 1000    (where ipaddress is UIP_IPADDRn defined in uip-conf.h)
```

The following message is then displayed:

```
Hello. What is your name ?
```

Enter your name. The application responds and then terminates the connection:

```
Hello. What is your name ? John
Hello John

Connection to host lost.
```

9.2 Web Server (HTTP Daemon)

This example demonstrates a simple embedded web server application.

This application uses a fixed IP address, so UIP_CONF_FIXEDADDR is set to 1 (true) and the UIP_IPADDRn, UIP_NETMASKn and UIP_DRIPADDRn values should be set appropriately for the network.

The HTML files that are served by the webserver application are located in a subdirectory of the project <*httpd-fs*>. The application includes these files 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 the PERL script <*DOSMakeFS.pl*>. The PERL script converts the HTML files into a C source file *httpd-fsdata.c*. The generated file *httpd-fsdata.c* is included in the web server project via the file *httpd-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. On the eCOG1k this 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 *httpd-fsdata.c*. Alternatively, the file <*out\project.sec*> lists the sizes and start addresses of all the code, constant and static data segments in the project.

On the eCOG1X, 32Kwords (64Kbytes) of constant data are allocated and it is much less likely that this will be filled by a simple web site.

To test the application, build it and run it, then open a web browser application and enter the IP address defined in *uip-conf.h*. If the embedded server is running correctly and the network connection is ok, the following web page should be seen in the browser:

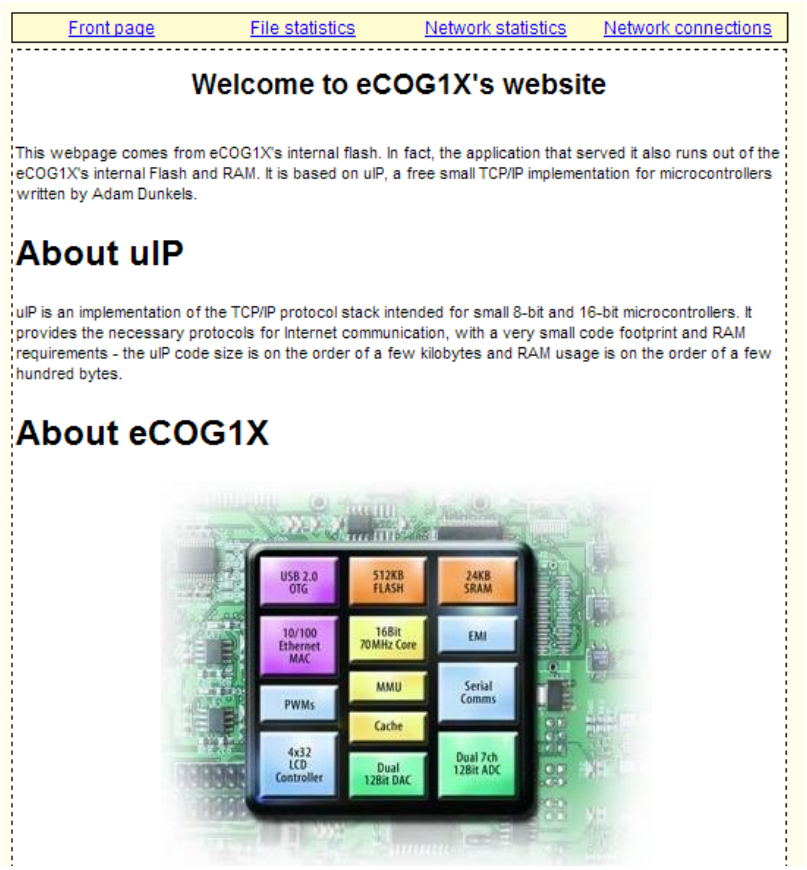


Figure 1. eGOG1X Example Web Site

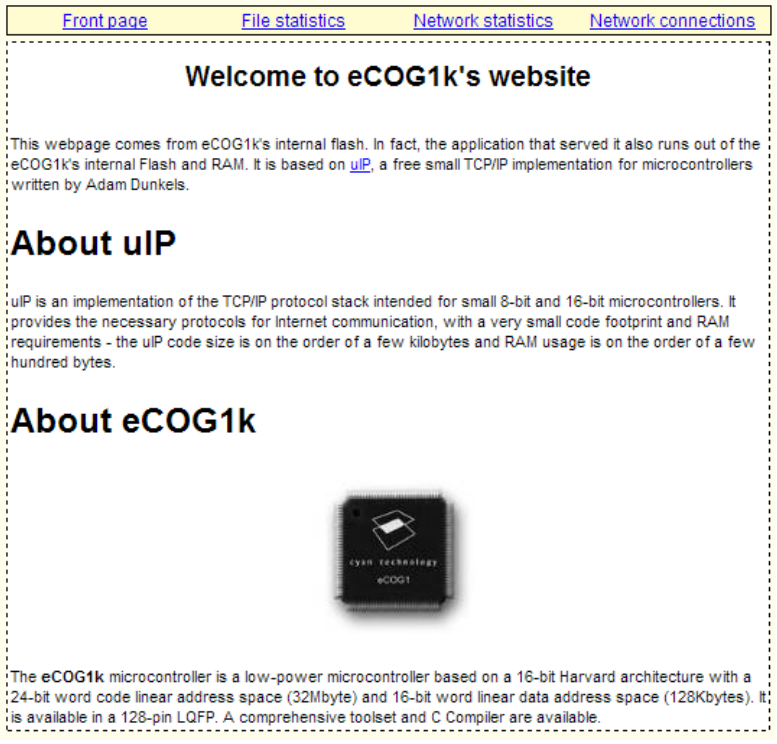


Figure 2. eCOG1k Example Web Site

9.3 Web Server (HTTP Daemon) with DHCP

This example is the same as the Web Server example, except that that it is combined with the DHCP Client that is provided with the uIP stack library.

In the include file *uip-conf.h*, UIP_CONF_FIXEDADDR is set to zero (false) and UIP_CONF_UDP is set to 1 (true) to enable UDP support, as required for the DHCP client.

The `main()` function contains a call to the function `dhcpc_init()` to initialise the DHCP client with the Ethernet MAC address of the board.

In the file *main.c*, the `dhcpc_configured()` callback function is provided to configure the uIP stack with the IP address, net mask and default router IP addresses when the DHCP client acquires this information. The `uip_udp_appcall()` callback function includes a call to `dhcpc_appcall()`.

Once the application is built and running, it attempts to acquire an IP address with DHCP. When it does, it prints the message “IP Address a.b.c.d acquired OK”. Connecting to this IP address with a web browser then brings up the demo web pages as before.

Note that the Network Statistics page now includes the UDP statistics.

9.4 Telnet Server Example

This example demonstrates a simple Telnet server application, in combination with the DHCP client.

Once the application is built and running, it waits for a connection on Port 23.

Open a windows command shell by choosing *Start->Run*. Enter `cmd` and select `OK`.
At the command line, enter:

```
telnet ipaddress
```

where `ipaddress` is the IP address acquired by the DHCP client.

The following message is then displayed:

```
Cyan Command Shell
Type HELP and return for help

CMD>
```

Enter ‘`HELP`‘ at the prompt to display a list of available commands:

```
CMD> HELP
QUIT - Quit the shell
HELP - Display this information
BINARY - Select binary as current base
HEX - Select hexadecimal as current base
DECIMAL - Select decimal as current base
. - Pop and print top of stack
CMD>
```

9.5 DNS Resolver Example

This example demonstrates the DNS client included in the uIP library.

Once the application is built and running, DNS queries can be entered through the serial interface. The interface is based on a cut down version of the Command Line Interface described in Application Note AN020.

Once the application has acquired an IP address using the DHCP Client, lookup requests may be entered at the command line in the following way:

LOOKUP <name>

The application then searches its local name cache, and if the name is not in the local cache then it sends a request to the DNS Server.

```
uIP DNS Resolver demo for eCOG1X
CMD>
eCOG1X Ethernet MAC initialised
IP Address a.b.c.d acquired OK
CMD>LOOKUP google.com
Failed to resolve GOOGLE.COM, so adding it to the list
CMD>
Resolved GOOGLE.COM as 72.14.207.99
```

10 Building a New Application

This section describes briefly how to create a new application project that uses the uIP stack library, and discusses file locations and include directories.

10.1 Creating a Project

Included with the *AN057SW.zip* file are two template files for use with CyanIDE to help create new uIP applications for the eCOG1k and eCOG1X Development Boards. If these are placed in the templates folders, then they can be used as the starting point for new uIP applications.

Select *Project->New* from the main menu, select the project type from one of the available templates, enter a name for the new project and set the project directory to a suitable location within the CyanIDE projects directory.

10.2 Include Directories

When adding the uIP library to an application project, the directories used for various library files need to be added to the compiler and linker search paths within the CyanIDE project 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:

```
${PERIPHERAL_LIBRARY_PATH}/uIP1.0;${PROJECT_PATH}
```

Note that the project path is included here, so that the library can find the header file *uip-conf.h*.

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

```
${PERIPHERAL_LIBRARY_PATH}/uIP1.0
```

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.

10.3 Timer Functions

The uIP stack contains within it a mechanism for measuring time. This can be used by any application using the uIP stack.

Create a variable of type `struct timer`, and initialise it with the function `timer_set()`. The timer can then be tested with the function `timer_expired()`. To restart or reset the timer, use the `timer_restart()` and `timer_reset()` functions.