AN10799

Porting uIP1.0 to LPC23xx/24xx

Rev. 01 — 17 February 2009 Application note

Document information

Info	Content
Keywords	uIP, TCP/IP Stack, LPC23xx, LPC24xx
Abstract	This application note describes the steps and details of porting uIP (a light-weight TCP/IP Stack) to LPC23xx/24xx. A simple web server is implemented as a demo.

founded by Philips

Revision history

Rev	Date	Description
01	20090217	Initial version

Contact information

For additional information, please visit: http://www.nxp.com

For sales office addresses, please send an email to: salesaddresses@nxp.com

1. Introduction

The Ethernet block of LPC23xx/24xx contains a full featured 10/100 Mbps Ethernet MAC designed to provide optimized performance through the use of DMA hardware acceleration. Features include a generous suite of control registers, half or full duplex operation, flow control, control frames, hardware acceleration for transmit retry, receive packet filtering and wake-up on LAN activity. Automatic frame transmission and reception with Scatter-Gather DMA off-loads many operations from the CPU.

The uIP TCP/IP stack is an extremely small implementation of the TCP/IP protocol suite intended for embedded systems running low-end 8 or 16-bit microcontrollers. The code size and RAM requirements of uIP is an order of magnitude smaller than other generic TCP/IP stacks today. It can only handle a single network interface and contains the IP, ICMP, UDP and TCP protocols.

This application note describes the steps and details for porting the uIP TCP/IP stack (latest version is 1.0) to LPC23xx/24xx. As a demo, a simple web server is implemented.

Based on this document, users can easily port uIP to other NXP MCU with Ethernet block.

2. Porting uIP

2.1 Overview

There is no need to make any changes to the actual uIP TCP/IP code, but a device driver for the target's network device (Ethernet controller/serial port/whatever) has to be written. The actual system integration part (i.e., the main control loop, which should call the uIP functions when data arrives and when the periodical timer expires) also has to be prepared.

2.2 Porting description

2.2.1 Preparation

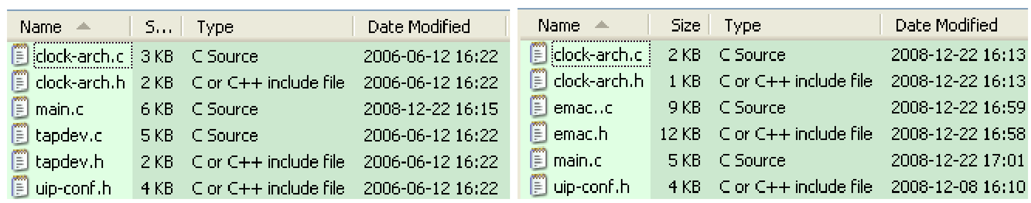
The uIP can be downloaded from its official website: http://www.nxp.com/redirect/sics.se/uip. Read the reference manual to understand the file structure, work flow and especially the main control loop.

Fig 1. File structure of uIP

You also need to be familiar with the LPC Ethernet and Timer blocks by reading the appropriate sections of the LPC23xx/24xx user manual. The latest user manual can be downloaded from NXP website: http://www.standardics.nxp.com/support/documents/microcontrollers/.

2.2.2 Modify porting directory and files

1. Modify the directory name of “unix” to “lpc24xx_port”.
2. Modify the file name of “tapdev.c” and “tapdev.h” to “emac.c” and “emac.h” respectively.



Name	S...	Type	Date Modified
clock-arch.c	3 KB	C Source	2006-06-12 16:22
clock-arch.h	2 KB	C or C++ include file	2006-06-12 16:22
main.c	6 KB	C Source	2008-12-22 16:15
tapdev.c	5 KB	C Source	2006-06-12 16:22
tapdev.h	2 KB	C or C++ include file	2006-06-12 16:22
uip-conf.h	4 KB	C or C++ include file	2006-06-12 16:22

Name	Size	Type	Date Modified
clock-arch.c	2 KB	C Source	2008-12-22 16:13
clock-arch.h	1 KB	C or C++ include file	2008-12-22 16:13
emac..c	9 KB	C Source	2008-12-22 16:59
emac.h	12 KB	C or C++ include file	2008-12-22 16:58
main.c	5 KB	C Source	2008-12-22 17:01
uip-conf.h	4 KB	C or C++ include file	2008-12-08 16:10

Fig 2. Files in “unix” directory (left) and in “lpc24xx_port”directory (right)

2.2.3 Porting Timer driver

The timer library provides functions for setting, resetting and restarting timers, and for checking if a timer has expired.

Two timer functions must be implemented (prototypes are defined in header file “clock.h” in the “uip” directory):

1. clock_time_t **clock_time**(void);
2. void **clock_init**(void);

The function **clock_time**() returns the current system clock time which is measured in system ticks.

The function **clock_init**() initializes the clock library and should be called from the main() function of the system.

The timer driver is implemented in file “clock-arch.h” and “clock-arch.c”. The files containing “arch” mean that are architecture specific implementations.

In this demo, LPC23xx/24xx Timer0 is used and fired every 10 milliseconds.

2.2.4 Porting Ethernet driver

The Ethernet library provides functions for initializing Ethernet block, transmitting and receiving Ethernet frames in LPC23xx/24xx.

Three Ethernet functions must be implemented which are defined in the previous file “tapdev.h” (it has been changed to “emac.h”):

1. BOOL_8 **tapdev_init**(void);
2. UNS_32 **tapdev_read**(void * pPacket);
3. BOOL_8 **tapdev_send** (void *pPacket, UNS_32 size);

The function **tapdev_init**() initializes the LPC23xx/24xx Ethernet block (including MAC and DMA controller) and PHY chip.

The function **tapdev_read**() receives an Ethernet frame from MAC/DMA controller.

The function **tapdev_send**() transmits an Ethernet frame to MAC/DMA controller.

The Ethernet driver is implemented in “emac.h” and “emac.c”.

The demo was tested on the Keil MCB2300 board. The PHY chip is DP83848.

2.2.4.1 Porting tapdev_init()

After reset, the Ethernet software driver needs to initialize the Ethernet block. During initialization the software needs to:

1. Remove the soft reset condition from the MAC
2. Configure the PHY via the MIIM interface of the MAC
3. Select RMII or MII mode
4. Configure the transmit and receive DMA engines, including the descriptor arrays
5. Configure the host registers (MAC1, MAC2, etc.) in the MAC
6. Enable the receive and transmit datapaths.

For rev ‘-’ devices, the default configuration does not enable the interface in RMII mode. The workaround can be found in the LPC23xx/24xx errata sheet.

Ethernet.1: **Setting up the Ethernet interface in RMII mode**

Introduction: The LPC2468 has an Ethernet interface, which can be interfaced with an off-chip PHY using the RMII interface.

Problem: The default configuration of the device does not enable the RMII interface.

Workaround: To use the Ethernet interface in RMII mode write a 1 to bit 12 (P1.16) in PINSEL2 register (located at 0xE002 C008). This workaround only applies for Rev ‘-’ devices and does not apply for Rev ‘A’ and newer devices. In order to have both Rev ‘-’ and other revisions coexist in the same piece of software, the MAC module ID can be used to identify the part and determine if port pin P1.6 needs to be set or not.

Here are the steps (along with some sample code) to initialize the MAC based on the module ID:

1. In master header file lLPC24xx.h, make sure Module ID is defined (Please note, this ID register is not documented in the User's Manual).

```
#define MAC_BASE_ADDR          0xFFE00000
#define MAC_MODULEID (*(volatile unsigned long *)(MAC_BASE_ADDR +
0xFFC)) /* Module ID reg (RO) */
```

Fig 3. Ethernet errata for rev ‘-‘ LPC23xx/24xx devices.

Depending on the PHY, the software needs to initialize registers in the PHY via the MII Management interface. The software can read and write PHY registers by programming the MCFG, MCMD, MADR registers of the MAC. The MIIM clock divider ranges from 4 to 28 which will result in a MIIM clock ranging from 0.6MHz to 4.5MHz if PCLK is set to 18MHz. Although IEEE802.3u defines the clock to be no faster than 2.5MHz, some PHYs may support the clock up to 12.5MHz, for example DP83848.

For the physical address of DP83848, since the PHYAD[0] pin has weak internal pull-up resistor and PHYAD[4:1] pins have weak internal pull-down resistors, the default setting for the PHY address is 00001 (0x1).

After initializing the receive descriptor and status arrays to receive frames from the Ethernet connection, both the receive DMA controller and receive datapath of the MAC need to be enabled. To prevent overflow in the receive DMA engine, the receive DMA engine should be enabled by setting the RxEnable bit in the Command register before enabling the receive datapath in the MAC by setting the RECEIVE ENABLE bit in the MAC1 register.

```
        rx_descr_init();
        tx_descr_init();

        /* Enable receive and transmit mode of MAC Ethernet core */
        MAC_COMMAND   |= (CR_RX_EN | CR_TX_EN);
        MAC_MAC1      |= MAC1_REC_EN;

        return(TRUE);
```

Fig 4. Enable receive DMA controller and receive datapath of MAC

2.2.4.2 Porting tapdev_send()

Fig 5 shows the transmit process (in the view of software only).

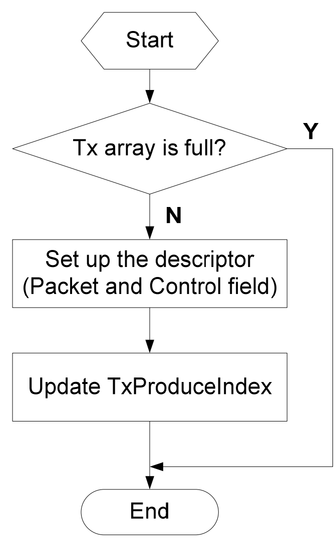


Fig 5. Flow chart of transmit process

Note: If the transmit datapath is enabled the Ethernet block will start transmitting the frame as soon as it detects the TxProduceIndex is not equal to TxConsumeIndex - both were zero after reset.

```
BOOL_8 tapdev_send(void *pPacket, UNS_32 size)
{
    UNS_32 TxProduceIndex = MAC_TXPRODUCEINDEX;
    UNS_32 TxConsumeIndex = MAC_TXCONSUMEINDEX;

  if(size == 0)
    return(TRUE);

  TxProduceIndex++;
  if(TxProduceIndex == MAC_TXDESCRIPTORNUM)
    TxProduceIndex = 0;

  if(TxProduceIndex == TxConsumeIndex)   //full
    return(FALSE);

  size = MIN(size,EMAC_MAX_PACKET_SIZE);
  memcpy((unsigned int *)TX_BUF(TxProduceIndex),pPacket,size);
  TX_DESC_CTRL(TxProduceIndex) &= ~0x7ff;
  TX_DESC_CTRL(TxProduceIndex) |= (size - 1) & 0x7ff;

  MAC_TXPRODUCEINDEX = TxProduceIndex;   // update produce index,
                                         // and transmit now

  return(TRUE);
}
```

Fig 6. Implementation of function tapdev_send()

2.2.4.3 Porting tapdev_read()

Fig 7 shows the receive process (in the view of software).

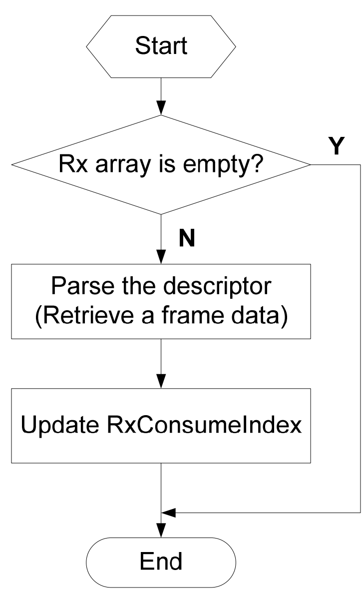


Fig 7. Flow chart of receive process

Note: If the receive datapath is enabled the Ethernet block will continue to receive frames until the receive descriptor array is full.

```
UNS_32 tapdev_read(void * pPacket)
{
    UNS_32 RxProduceIndex = MAC_RXPRODUCEINDEX;
    UNS_32 RxConsumeIndex = MAC_RXCONSUMEINDEX;
    UNS_32 Size = EMAC_MAX_PACKET_SIZE;
    UNS_32 in_size;

  if(RxConsumeIndex == RxProduceIndex)  // empty
  {
    return(0);
  }

  in_size = (RX_STAT_INFO(RxConsumeIndex) & 0x7ff)+1;
  Size = MIN(Size,in_size);
  memcpy(pPacket,(unsigned int *)RX_BUF(RxConsumeIndex),Size);

  RxConsumeIndex++;
  if(RxConsumeIndex > MAC_RXDESCRIPTORNUM)
    RxConsumeIndex = 0;

  MAC_RXCONSUMEINDEX = RxConsumeIndex;
  return(Size);
}
```

Fig 8. Implementation of function tapdev_read()

2.2.5 Main loop control

The main loop control is in the source file main.c. Before entering the main loop, all system functions and devices must be initialized, including LPC23xx/24xx vector interrupt controller, serial port (for debug), timer, Ethernet and HTTP server, etc. MAC and IP address should be also set here.

The uIP stack can be run either as a task in a multitasking system, or as the main program in a single-tasking system. In both cases, the main control loop does two things repeatedly:

- Check if a packet has arrived from the network
- Check if a periodic timeout has occurred

2.3 Demo description

For the purposes of a demo, we created a new uVision3 project and implemented a web server based on uIP1.0.

Note: The function **snprintf**() is used in the source file httpd-cgi.c but not implemented by Keil. Therefore, MicroLIB can not be used in the project:

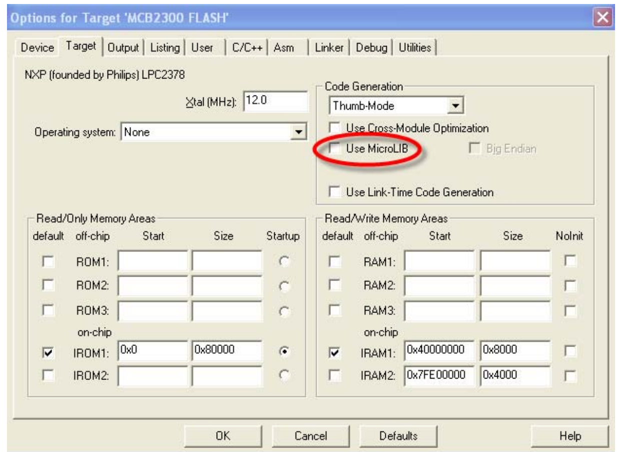


Fig 9. Project options without use of MicroLIB

Modify the IP configuration of your laptop to be in the same LAN with MCB2300 Board.

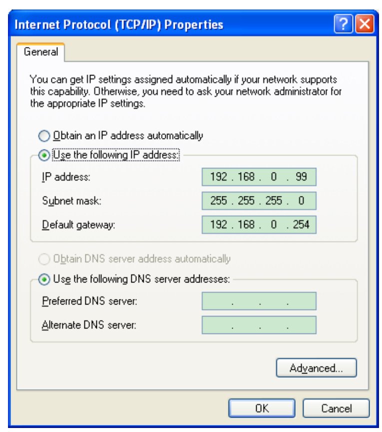


Fig 10. IP Configuration in laptop

Build the project and start a debug session. Then type //192.168.0.100 in the address bar. You will see the main page of uIP web server.

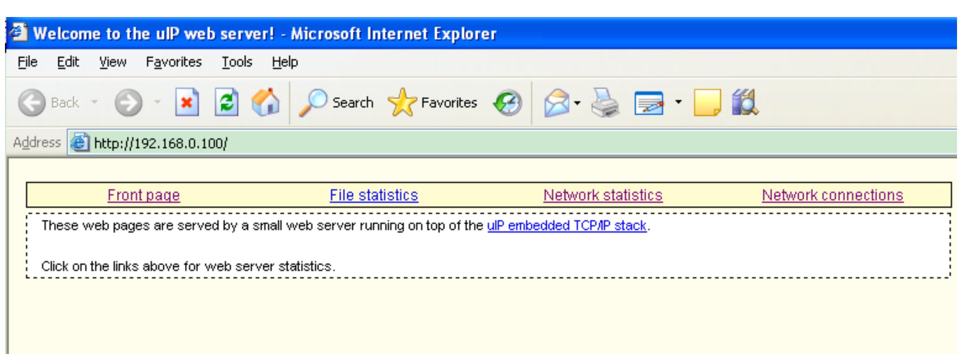


Fig 11. Main page of uIP web sever

Click the hyper-link "Network statistics", the current network statistics will be displayed:

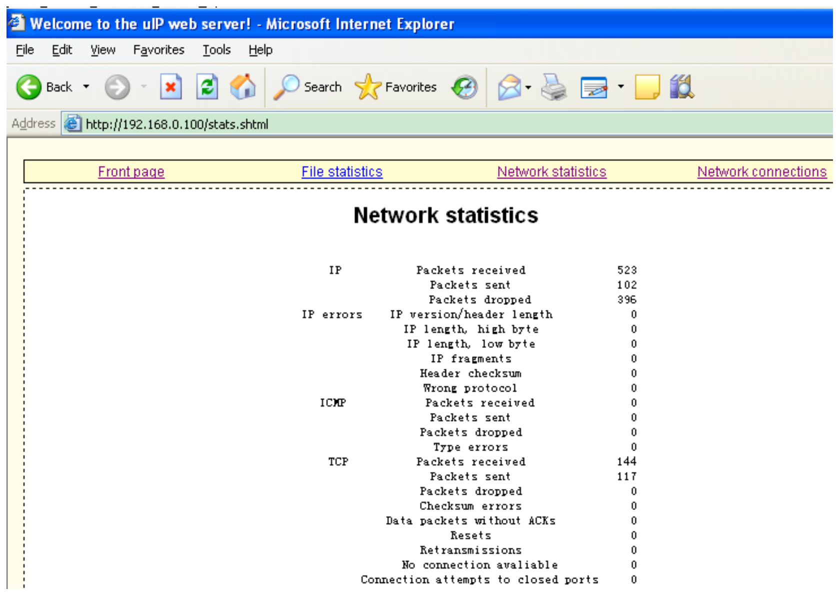


Fig 12. Network statistics of uIP web server

Click the hyper-link "Network connections", the current connections will be displayed:

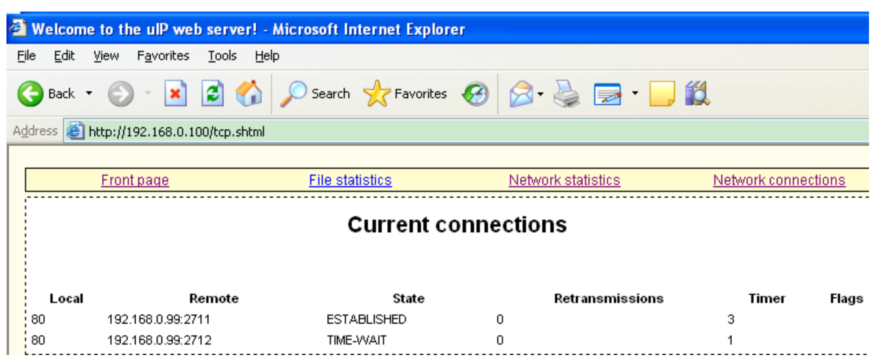


Fig 13. Network connections of uIP web sever

3. Conclusion

With the success of the Internet, the TCP/IP protocol suite has become a global standard for communications. For embedded systems, being able to run native TCP/IP makes it possible to connect the system directly to an intranet or even the global Internet. Embedded devices with full TCP/IP support will be first-class network citizens, thus being able to fully communicate with other hosts in the network.

Compared with other TCP/IP stack implementation, uIP consumes a lot less memory while still providing basic TCP/IP stack capabilities making it suitable for small memory footprint embedded devices like those in the LPC23xx/24xx series.

4. Legal information

4.1 Definitions

Draft — The document is a draft version only. The content is still under internal review and subject to formal approval, which may result in modifications or additions. NXP Semiconductors does not give any representations or warranties as to the accuracy or completeness of information included herein and shall have no liability for the consequences of use of such information.

4.2 Disclaimers

General — Information in this document is believed to be accurate and reliable. However, NXP Semiconductors does not give any representations or warranties, expressed or implied, as to the accuracy or completeness of such information and shall have no liability for the consequences of use of such information.

Right to make changes — NXP Semiconductors reserves the right to make changes to information published in this document, including without limitation specifications and product descriptions, at any time and without notice. This document supersedes and replaces all information supplied prior to the publication hereof.

Suitability for use — NXP Semiconductors products are not designed, authorized or warranted to be suitable for use in medical, military, aircraft, space or life support equipment, nor in applications where failure or malfunction of a NXP Semiconductors product can reasonably be expected to result in personal injury, death or severe property or environmental damage. NXP Semiconductors accepts no liability for inclusion and/or use of NXP Semiconductors products in such equipment or applications and therefore such inclusion and/or use is for the customer's own risk.

Applications — Applications that are described herein for any of these products are for illustrative purposes only. NXP Semiconductors makes no representation or warranty that such applications will be suitable for the specified use without further testing or modification.

4.3 Trademarks

Notice: All referenced brands, product names, service names and trademarks are property of their respective owners.

5. Contents

Date of release: 17 February 2009

Document identifier: AN10799_1