

AN10403

Connecting ethernet interface with LPC2000

Rev. 01 — 7 February 2007

Application note

Document information

Info	Content
Keywords	ARM, LPC2000, Ethernet, RTL8019
Abstract	This application note describes how to connect Ethernet interface with NXP LPC2000 series ARM MCU. Reference schematics and Ethernet driver source code is attached, with RTL8019AS as Ethernet controller. The document also shows how to configure uClinux Ethernet options in LPC22xx development system.

Revision history

Rev	Date	Description
01	20070207	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

1.1 About LPC2000

The 16/32-bit LPC2000 family is based on a 1.8 V ARM7TDMI-S core operating at up to 60 MHz together with a wide range of peripherals including multiple serial interfaces, 10-bit ADC and external bus options.

For more about LPC2000, refer to <http://www.nxp.com/>.

1.2 Ethernet

Ethernet is a local area network (LAN) technology that transmits information between computers by use of either coaxial or twisted pair cable. Ethernet uses a bus or star topology and supports data transfer rates of 10 Mbps, though newer systems use 100 Mbps.

The Ethernet specification serves as the basis for the IEEE 802.3 standard, which specifies the physical and lower software layers.

The RTL8019AS - 10 Mbit Ethernet controller by Realtek company - is a highly integrated Ethernet controller which offers a simple solution to implement a Plug and Play NE2000 compatible adapter with full-duplex and power down features.

You can find specification document and more products info on Realtek website.

2. Connecting Ethernet with LPC2000

2.1 Connecting Ethernet with LPC21xx

The LPC21xx series, with tiny 64-pin package, are equipped with a variety of peripherals as UART, I2C, SPI, Timers, ADC/DAC, USB, etc, but without external bus interface. So GPIO is used to interface with RTL8019AS and 8-bit slots are selected in order to save GPIO.

The operation address of the RTL8019AS is 300H~31FH. [Fig 1](#) gives an example schematic connecting LPC21xx with RTL8019AS.

(Zoom in to see details.)

002aac810

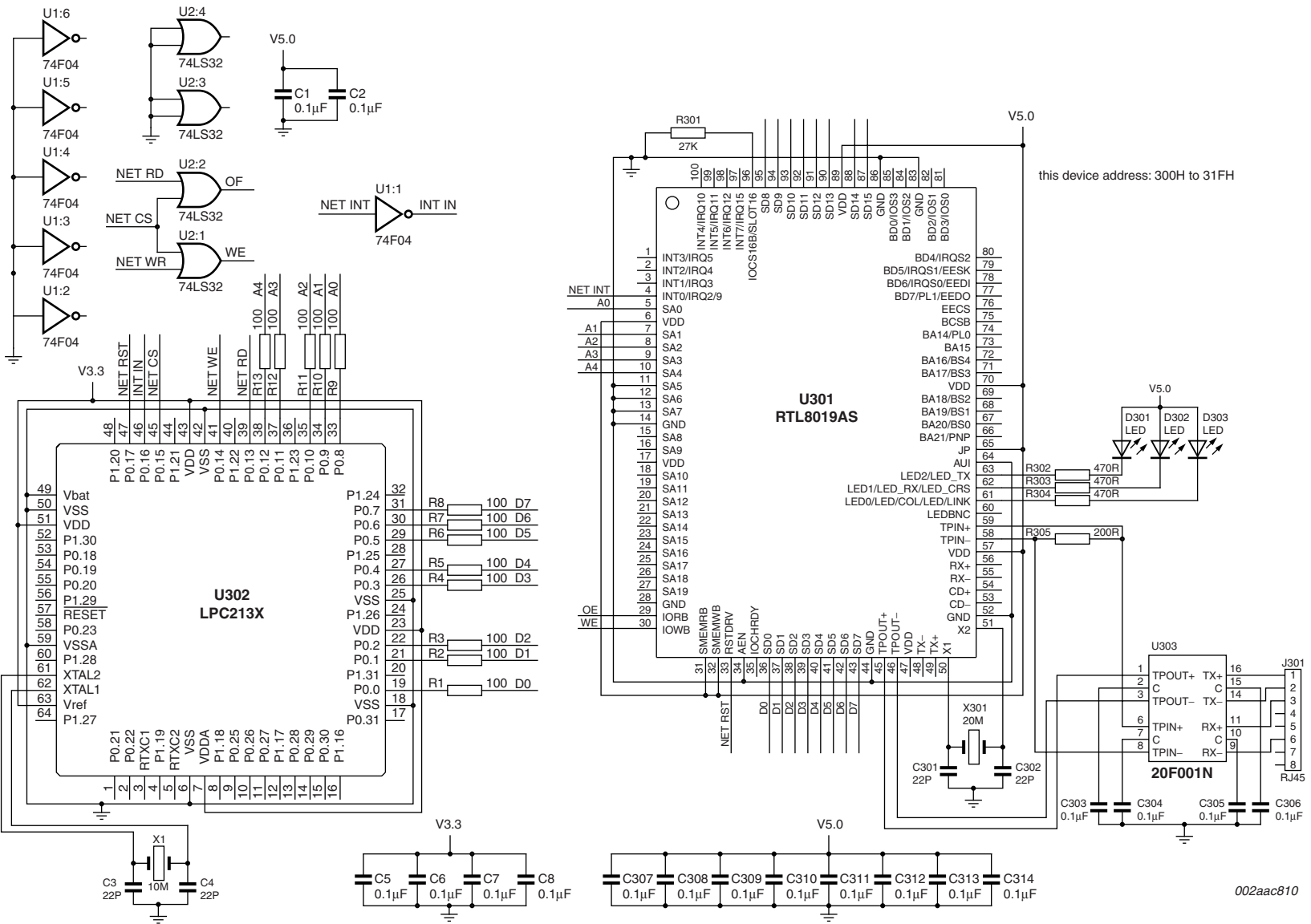


Fig 1. LPC21xx + RTL8019AS

2.2 Connecting Ethernet with LPC22xx

With their 144-pin package, LPC22xx series have configurable external memory interface with up to four banks, each up to 16 MB and 8/16/32 bit data width.

[Fig 2](#) gives example schematics of connecting LPC22xx with RTL8019AS.

(Zoom in to see details.)

The address map of RTL8019AS in the system is as:

1	CS0:	For external flash memory.	
2	0x80000000 - 0x807FFFFFFF	8 MB	
3	0x80400000 - 0x80FFFFFFF	reserve	
4			
5	CS1:	For external SRAM	
6	0x81000000 - 0x811FFFFFFF	2 MB	
7	0x81200000 - 0x81FFFFFFF	reserve	
8			
9	CS2:	For Ethernet	
10	0x82000000 - 0x821FFFFFFF	2 MB	
11	0x82200000 - 0x82FFFFFFF	reserve	
12			
13	CS3:	For others	
14	0x83000000 - 0x8303FFFFFF	LCD	
15	0x83040000 - 0x8307FFFFFF	USB	
16	0x83080000 - 0x830FFFFFFF	IDE	
17	0x83100000 - 0x831FFFFFFF	CF Card	
18	0x8320 0000 to 0x83FF FFFF	reserve	



3. Ethernet in uClinux system

Linux operating system has superior network function support, including full TCP/IP, PPP stack, etc. Its branch uClinux – Linux without MMU support – retains this advantage.

This section lists how to configure Ethernet options in uClinux system when connecting Ethernet controller with LPC22xx. To know about uClinux for LPC22xx, please read related application note “AN_Getting Started uClinux with LPC22xx”.

In the uClinux-dist directory on your Linux PC, type below command:

```
19 [root@mylinux uClinux-dist]# make menuconfig
```

At the first Main Menu, set vendor/product selection as below:

```
20 Vendor/Product Selection --->
21     (Philips) Vendor
22     (LPC22xx) Philips Product
```

Set “Kernel/Library/Defaults Selection” as below.

```
23 Kernel/Library/Defaults Selection --->
    24 (linux-2.6.x) Kernel Version
    25 (uClibc) Libc Version
    26 [ ] Default all settings
    27 [*] Customize Kernel Settings
    28 [*] Customize Vendor/User Settings
    29 [ ] Update Default Vendor Settings
```

Please toggle at least the two menus about kernel and user settings.

In the first popup window about ‘Linux Kernel Configuration’, enter the ‘Networking support’. Set and toggle the options as below. Especially TCP/IP networking and Network device options are definitely needed.

```
30 Networking support --->
31     [*] Networking support
32         Networking options --->
33             [*] Packet socket
34             ...
35             [*] Unix domain sockets
36             ...
37             [*] TCP/IP networking
38     [ ] Amateur Radio support --->
39     [ ] IrDA(infrared) subsystem support --->
40     [ ] Bluetooth subsystem support --->
41     [*] Network device support
42         .....
43         Ethernet (10 or 100Mbit) --->
44             [ ] Ethernet (10 or 100Mbit)
45             [*] NE2000/NE1000 support
46             [*] RT8019AS support
```

For the Ethernet options in network device support, user can use ‘NE2000/NE1000 support’ provided by original uClinux system.

Or user can create his own RTL8019AS option and toggle it. The difference is to use different device driver programs. See related description in next section.

In the second popup window for user settings, enter the Network Applications and toggle any application program you need. These application programs are provided in the official uClinux distribution package. In case the application is toggled and compiled into the Linux file system, it can be used as a Linux command in the console such as “ping, arp, ifconfig, etc.”

E.g.

```
47 Network Applications --->
48      [*] arp
49      [*] httpd
50      [*] ifconfig
51      [*] ping
52      [*] telnet
```

If selecting “httpd”, we can also create a web server on the board and display some web pages in the web browser on PC.

E.g.



Fig 3. Web browser example

4. Ethernet driver

To drive an Ethernet controller, an Ethernet driver program should run on the microcontroller. The appendix gives an example source code based on [Fig 2](#) connecting scheme.

Please note that this reference source code is only a draft hint but not prefect. It provides the basic driver flow such as init, transmit, receive, interrupt, etc.

In the uClinux system, we can find the NE2000 compatible Ethernet driver at `./linux-2.6.x/drivers/net/ne.c, 8390.c, 8390.h`.

Since RTL8019AS is NE2000 compatible, these codes can be used as the RTL8019AS driver after adding some necessary initialization and definitions extra for RTL8019AS.

Or user can develop his own Ethernet device driver under uClinux.

Appendix A Reference RTL8019AS Driver Program

```

53  /*****
54      rtl8019.h
55      This software may be used and distributed according to the terms
56      of the GNU General Public License, incorporated herein by reference.
57      *****/
58  #define SHIFT(x) (x<<1)
59
60  #define BaseAddr (0x82000000+SHIFT(0x300))
61  #define RWPORT (BaseAddr+SHIFT(0x10)) /* dma read write address,
62      form 0x10 - 0x17 */
63
64  #define RstAddr (BaseAddr+SHIFT(0x18)) /* reset register, 0x18,
65      0x1a, 0x1c, 0x1e even address is recommended */
66
67  /* page 0 */
68  #define Pstart (BaseAddr+SHIFT(1)) /* page start */
69  #define Pstop (BaseAddr+SHIFT(2)) /* page stop */
70  #define BNRy (BaseAddr+SHIFT(3))
71  #define TPSR (BaseAddr+SHIFT(4)) /* transmit page start */
72  #define TBCR0 (BaseAddr+SHIFT(5))
73  #define TBCR1 (BaseAddr+SHIFT(6))
74  #define ISR (BaseAddr+SHIFT(7)) /* interrupt status register */
75
76  #define RSAR0 (BaseAddr+SHIFT(8)) /* dma read address */
77  #define RSAR1 (BaseAddr+SHIFT(9))
78  #define RBCR0 (BaseAddr+SHIFT(10)) /* dma read byte count */
79  #define RBCR1 (BaseAddr+SHIFT(11))
80
81  #define RCR (BaseAddr+SHIFT(12)) /* receive config */
82  #define TCR (BaseAddr+SHIFT(13)) /* transmit config */
83  #define DCR (BaseAddr+SHIFT(14)) /* data config */
84  #define IMR (BaseAddr+SHIFT(15)) /* interrupt mask */
85
86  #define ID8019L (BaseAddr+SHIFT(10))
87  #define ID8019H (BaseAddr+SHIFT(11))
88
89  /* page 1 */
90  #define PAR0 (BaseAddr+SHIFT(1))
91  #define PAR1 (BaseAddr+SHIFT(2))
92  #define PAR2 (BaseAddr+SHIFT(3))
93  #define PAR3 (BaseAddr+SHIFT(4))
94  #define PAR4 (BaseAddr+SHIFT(5))
95  #define PAR6 (BaseAddr+SHIFT(6))
96
97  #define CURR (BaseAddr+SHIFT(7))
98  #define MAR0 (BaseAddr+SHIFT(8))
99  #define MAR1 (BaseAddr+SHIFT(9))
100 #define MAR2 (BaseAddr+SHIFT(10))
101 #define MAR3 (BaseAddr+SHIFT(11))
102 #define MAR4 (BaseAddr+SHIFT(12))
103 #define MAR5 (BaseAddr+SHIFT(13))

```

```
101 #define    MAR6  (BaseAddr+SHIFT(14))
102 #define    MAR7  (BaseAddr+SHIFT(15))
103
104 /* page 2 */
105
106 /* page 3 */
107 #define    CR9346  (BaseAddr+SHIFT(1))
108 #define    CONFIG0 (BaseAddr+SHIFT(3))
109 #define    CONFIG1 (BaseAddr+SHIFT(4))
110 #define    CONFIG2 (BaseAddr+SHIFT(5))
111 #define    CONFIG3 (BaseAddr+SHIFT(6))
112
```

```

113  /*****
114      rtl8019.c
115      This software may be used and distributed according to the terms
116      of the GNU General Public License, incorporated herein by reference.
117      *****/
118
119
120  #include "rtl8019.h"
121
122  #define RTL8019_OP_16    1
123
124  #undef  DEBUG
125  #define DEBUG    1
126  #ifdef    DEBUG
127  #define TRACE(str, args...) printk(str, ## args)
128  #else
129  #define TRACE(str, args...)
130  #endif
131
132  #define    outportb(port, data) *((volatile u8 *)(port)) = (u8)(data)
133  #define    inportb(port)        *((volatile u8 *)(port))
134
135  #define    outportw(port, data) *((volatile u16 *)(port)) = (u16)(data)
136  #define    inportw(port)        *((volatile u16 *)(port))
137
138  #define    ETH_FRAME_LEN        1514
139
140  #define    RPSTART                0x4c
141  #define    RPSTOP                 0x80
142  #define    SPSTART                0x40
143
144  static int timeout = 100; // tx watchdog ticks 100 = 1s
145  static char *version = "Rtl8019as driver for Philips LPC22xx: version 0.1\n";
146
147  /*
148   * This structure is private to each device. It is used to pass
149   * packets in and out, so there is place for a packet
150   */
151  struct nic_8019_priv {
152      struct net_device_stats stats;
153      spinlock_t lock;
154      struct sk_buff *skb;
155  };
156
157  /*****
158   static u8 rBNRY;
159   static u8 SrcMacID[ETH_ALEN] = {0x12,0x34,0x56,0x78,0x90,0xAB,};
160
161   static void SetRegPage( u8 PageIdx)
162   {
163       u8 temp;

```

```

164
165     temp = inportb(BaseAddr);
166     temp = (temp&0x3b)|(PageIdx<<6);
167     outportb(BaseAddr, temp);
168 }
169
170 irqreturn_t nic_8019_rx(int irq, void *dev_id, struct pt_regs *regs)
171 {
172     u8 RxPageBeg, RxPageEnd;
173     u8 RxNextPage;
174     u8 RxStatus;
175     u16 *data, temp;
176     u16 i, RxLength, RxLen;
177
178     struct sk_buff *skb;
179     struct net_device *dev = (struct net_device *) dev_id;
180     struct nic_8019_priv *priv = (struct nic_8019_priv *) dev->priv;
181
182     TRACE("TX/RX Interupt!\n");
183     spin_lock(&priv->lock);
184     SetRegPage(0);
185     outportb(BNRY, rBNRY);
186     RxStatus = inportb(ISR);
187     printk("RxStatus=0x%x\n", RxStatus);
188     if (RxStatus & 2) {
189         outportb(ISR, 0x2);          //clr TX interupt
190         priv->stats.tx_packets++;
191         TRACE("transmit one packet complete!\n");
192     }
193
194     if (RxStatus & 1) {
195         TRACE("Receivex packet....\n");
196         outportb(ISR, 0x1);          //clr Rx interupt
197         SetRegPage(1);
198         RxPageEnd = inportb(CURR);
199
200         SetRegPage(0);
201         RxPageBeg = rBNRY+1;
202         if(RxPageBeg>=RPSTOP)
203             RxPageBeg = RPSTART;
204         outportb(BaseAddr, 0x22);    // stop    remote dma
205
206         //outport(RSAR0, RxPageBeg<<8);
207         //outport(RBCR0, 256);
208         outportb(RSAR0, 0);
209         outportb(RSAR1, RxPageBeg);
210         outportb(RBCR0, 4);
211         outportb(RBCR1, 0);
212         outportb(BaseAddr, 0xa);
213
214     #ifdef RTL8019_OP_16

```

```

215         temp      = inportw(RWPORT);
216         RxNextPage = temp>>8;
217         RxStatus   = temp&0xff;
218         RxLength   = inportw(RWPORT);
219     #else
220         RxStatus   = inportb(RWPORT);
221         RxNextPage = inportb(RWPORT);
222         RxLength   = inportb(RWPORT);
223         RxLength   |= inportb(RWPORT)<<8;
224     #endif
225     TRACE("\nRxBeg = %x, RxEnd = %x, nextpage = %x, size = %i\n",
RxPageBeg, RxPageEnd, RxNextPage, RxLength);
226     RxLength -= 4;
227     if (RxLength>ETH_FRAME_LEN) {
228         if (RxPageEnd==RPSTART)
229             rBNRY = RPSTOP-1;
230         else
231             rBNRY = RxPageEnd-1;
232
233         outportb(BNRY, rBNRY);
234         TRACE("RxLength more long than %x\n", ETH_FRAME_LEN);
235         return IRQ_HANDLED;
236     }
237
238     skb = dev_alloc_skb(RxLength+2);
239     if (!skb) {
240         TRACE("Rtl8019as eth: low on mem - packet dropped\n");
241         priv->stats.rx_dropped++;
242         return IRQ_HANDLED;
243     }
244
245     skb->dev = dev;
246     skb_reserve(skb, 2);
247     skb_put(skb, RxLength);
248     data = ( u16 *)skb->data;
249
250     //      eth_copy_and_sum(skb, data, len, 0);
251     outportb(RSAR0, 4);
252     outportb(RSAR1, RxPageBeg);
253     outportb(RBCR0, RxLength);
254     outportb(RBCR1, RxLength>>8);
255     outportb(BaseAddr, 0xa);
256     #ifdef RTL8019_OP_16
257         i = 2;
258         data -= 2;
259         RxLen=(RxLength+1)/2;
260     #else
261         i = 4;
262         data -= 4;
263         RxLen=RxLength;
264     #endif

```

```

265         for(; RxLen--;) {
266 #ifdef RTL8019_OP_16
267             static const int cmp_val = 0x7f;
268 #else
269             static const int cmp_val = 0xff;
270 #endif
271             if (!(i & cmp_val)) {
272                 outportb(BNRY, RxPageBeg);
273                 RxPageBeg++;
274                 if(RxPageBeg>=RPSTOP)
275                     RxPageBeg = RPSTART;
276             }
277 #ifdef RTL8019_OP_16
278             data[i++] = inportw(RWPORT);
279             TRACE("%2X,%2X,", data[i-1]&0xff,data[i-1]>>8);
280 #else
281             data[i++] = inportb(RWPORT);
282             TRACE("%2X,", data[i-1]);
283 #endif
284         }
285
286         TRACE("\n");
287         outportb(BNRY, RxPageBeg);
288         rBNRY = RxPageBeg;
289
290         skb->protocol = eth_type_trans(skb, dev);
291         TRACE("\nprotocol=%x\n", skb->protocol);
292         priv->stats.rx_packets++;
293         priv->stats.rx_bytes +=RxLength;
294         netif_rx(skb);
295     } else {
296         outportb(ISR, 0xfe);
297     }
298
299     spin_unlock(&priv->lock);
300     return IRQ_HANDLED;
301 }
302
303
304 /*
305  * Open and Close
306  */
307 static int nic_8019_open(struct net_device *dev)
308 {
309     int i,j;
310
311     MOD_INC_USE_COUNT;
312     TRACE("open\n");
313     // Disable irqs
314     disable_irq(dev->irq);
315     // register rx isr

```

```

316     if (request_irq(dev->irq, &nic_8019_rx, SA_INTERRUPT, "eth rx isr", dev)) {
317         printk(KERN_ERR "Rtl8019: Can't get irq %d\n", dev->irq);
318         return -EAGAIN;
319     }
320
321     // wake up Rtl8019as
322     SetRegPage(3);
323     outportb(CR9346, 0xcf);          //set eem1-0, 11 ,enable write config
register
324     outportb(CONFIG3, 0x60);        //clear pwrnd, sleep mode, set led0 as led_col,
led1 as led_crs
325     outportb(CR9346, 0x3f);        //disable write config register
326
327     // initialize
328     outportb(RstAddr, 0x5a);
329     i = 20000;
330     while(i--);
331
332     SetRegPage(0);
333     inportb(ISR);
334     outportb(BaseAddr, 0x21);      /* set page 0 and stop */
335     outportb(Pstart, RPSTART);    /* set Pstart 0x4c */
336     outportb(Pstop, RPSTOP);      /* set Pstop 0x80 */
337     outportb(TPSR, SPSTART);      /* SPSTART page start register, 0x40 */
338     outportb(BNRY, RPSTART);      /* BNRY-> the last page has been read */
339     outportb(TCR, 0xe0);          /* set TCR 0xe0 */
340     outportb(DCR, 0xc9);          /* set DCR 0xc9, 16bit DMA */
341
342     outportb(IMR, 0x03);          /* set IMR 0x03, enable tx rx int */
343     outportb(ISR, 0xff);          /* clear ISR */
344
345     SetRegPage(1);
346     for(i=0; i<6; i++)
347         outportb(BaseAddr+(1+i)*2, dev->dev_addr[i]); // set mac id
348
349     outportb(CURR, RPSTART+1);
350     outportb(MAR0, 0x00);
351     outportb(MAR1, 0x41);
352     outportb(MAR2, 0x00);
353     outportb(MAR3, 0x80);
354     outportb(MAR4, 0x00);
355     outportb(MAR5, 0x00);
356     outportb(MAR6, 0x00);
357     outportb(MAR7, 0x00);
358     outportb(BaseAddr, 0x22);      /* set page 0 and start */
359     rBNRY = RPSTART;
360     enable_irq(dev->irq);
361     // Start the transmit queue
362     netif_start_queue(dev);
363
364     return 0;

```

```

365 }
366
367 static int nic_8019_stop(struct net_device *dev)
368 {
369     TRACE("stop\n");
370     SetRegPage(3);
371     outportb(CR9346, 0xcf);          // set eem1-0, 11 ,enable write config
register
372     outportb(CONFIG3, 0x66);        // enter pwrnd, sleep mode, set led0 as led_col,
led1 as led_crs
373     outportb(CR9346, 0x3f);        // disable write config register
374
375     free_irq(dev->irq, dev);
376     netif_stop_queue(dev);
377     MOD_DEC_USE_COUNT;
378
379     return 0;
380 }
381
382 static int nic_8019_start_xmit(struct sk_buff *skb, struct net_device *dev)
383 {
384     int i;
385     u16 len,TxLen;
386     u16 *data;
387     struct nic_8019_priv *priv = (struct nic_8019_priv *) dev->priv;
388
389     TRACE("start_xmit\n");
390
391     len = skb->len < ETH_ZLEN ? ETH_ZLEN : skb->len;
392     TRACE("\nTx Length = %i,%x,%x\n", len, skb->data[12], skb->data[13]);
393     data =(u16*) skb->data;
394
395     outportb(BaseAddr,0x22);        //switch to page 0 and stop remote dma
396     if (inportb(BaseAddr)&4)        // last remote dma not complete,return 1 echo
busy(error),retransmit next
397         return 1;
398 #ifdef bug_fix_for_write
399     //read page 42,0,42,0 before write if you have problem
400 #endif
401     outportb(RSAR0, 0);
402     outportb(RSAR1, SPSTART);
403     outportb(RBCR0, len&0xff);
404     outportb(RBCR1, len>>8);
405     outportb(BaseAddr, 0x12);        //begin remote write
406     dev->trans_start = jiffies;
407 #ifdef RTL8019_OP_16
408     TxLen=(len+1)/2;
409 #else
410     TxLen=len;
411 #endif
412     for(i=0; i<TxLen; i++) {

```

```

413 #ifdef RTL8019_OP_16
414     outportw(RWPORT, data[i]);        // copy data to nic ram
415     TRACE("%2X,%2X,", data[i]&0xff, data[i]>>8);
416 #else
417     outportb(RWPORT, data[i]);        // copy data to nic ram
418     TRACE("%2X,", skb->data[i]);
419 #endif
420 }
421
422 TRACE("\n");
423 outportb(TPSR, SPSTART);              // transmit begin page 0x40
424 outportb(TBCR0, len&0xff);
425 outportb(TBCR1, len>>8);
426 outportb(BaseAddr, 0x1e); // begin to send packet
427 dev_kfree_skb(skb);
428 return 0;
429 }
430
431 static struct net_device_stats *nic_8019_get_stats(struct net_device *dev)
432 {
433     struct nic_8019_priv *priv = (struct nic_8019_priv *) dev->priv;
434     TRACE("get_stats\n");
435     return &priv->stats;
436 }
437
438 /*****
439 static int nic_8019_init(struct net_device *dev)
440 {
441     int i;
442     TRACE("init\n");
443     ether_setup(dev); // Assign some of the fields
444
445     // set net_device methods
446     dev->open = nic_8019_open;
447     dev->stop = nic_8019_stop;
448     dev->get_stats = nic_8019_get_stats;
449     dev->hard_start_xmit = nic_8019_start_xmit;
450
451     // set net_device data members
452     dev->watchdog_timeo = timeout;
453     dev->irq = 14; //LPC22xx: EXT0
454     dev->dma = 0;
455
456     // set MAC address manually
457     printk(KERN_INFO "%s: ", dev->name);
458     for(i=0; i<6; i++) {
459         dev->dev_addr[i] = SrcMacID[i];
460         printk("%2.2x%c", dev->dev_addr[i], (i==5) ? ' ' : ':');
461     }
462     printk("\n");
463 }

```

```
464     SET_MODULE_OWNER(dev);
465
466     dev->priv = kmalloc(sizeof(struct nic_8019_priv), GFP_KERNEL);
467     if(dev->priv == NULL)
468         return -ENOMEM;
469
470     memset(dev->priv, 0, sizeof(struct nic_8019_priv));
471     spin_lock_init(&((struct nic_8019_priv *) dev->priv)->lock);
472     return 0;
473 }
474
475 static struct net_device nic_8019_netdevs = {
476     init: nic_8019_init,
477 };
478
479 /*
480  * Finally, the module stuff
481  */
482 int __init nic_8019_init_module(void)
483 {
484     int result;
485     TRACE("init_module\n");
486
487     //Print version information
488     printk(KERN_INFO "%s", version);
489
490     //register_netdev will call nic_8019_init()
491     if((result = register_netdev(&nic_8019_netdevs)))
492         printk("Rtl8019as eth: Error %i registering device \"%s\"\n", result,
493             nic_8019_netdevs.name);
494
495     return result ? 0 : -ENODEV;
496 }
497
498 void __exit nic_8019_cleanup(void)
499 {
500     TRACE("cleanup\n");
501     kfree(nic_8019_netdevs.priv);
502     unregister_netdev(&nic_8019_netdevs);
503     return;
504 }
505
506 module_init(nic_8019_init_module);
507 module_exit(nic_8019_cleanup);
508 MODULE_DESCRIPTION("Rtl8019as ethernet driver");
```

5. Legal information

5.1 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.

5.2 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.

5.3 Trademarks

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

6. Contents

1. Introduction3

1.1 About LPC20003

1.2 Ethernet.....3

2. Connecting Ethernet with LPC20003

2.1 Connecting Ethernet with LPC21xx.....3

2.2 Connecting Ethernet with LPC22xx.....5

3. Ethernet in uClinux system7

4. Ethernet driver9

Appendix A Reference RTL8019AS Driver Program.10

5. Legal information20

5.1 Definitions.....20

5.2 Disclaimers.....20

5.3 Trademarks20

6. Contents.....21

Please be aware that important notices concerning this document and the product(s) described herein, have been included in the section 'Legal information'.



© NXP B.V. 2007. All rights reserved.

For more information, please visit: <http://www.nxp.com>
For sales office addresses, email to: salesaddresses@nxp.com

Date of release: 7 February 2007
Document identifier: AN10403_1