

AN10605

Running eCos on LPC2214

Rev. 01 — 1 August 2007

Application note

Document information

Info	Content
Keywords	eCos, RTOS, LPC2214, Olimex
Abstract	Describes the steps to run eCos on Olimex LPCE-2214 evaluation board.

Revision history

Rev	Date	Description
01	20070801	Initial revision

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 purpose of this document is describing the method of porting eCos operating system on Olimex LPCE-2214 evaluation board with ARM7-TDMI MCU and CS8900A Ethernet controller onboard.

2. Introduction into eCos

eCos is an open source real-time operating system (RTOS) available under the terms of GNU GPL License. eCos is based on new technology configuration system – one of the most important features in it. The configuration system allows developer to impose his own requirements on functional parts of eCos. In other words, the developer is able to create a very specialized operating system for specific purposes. The configuration system also guarantees that no unnecessary components will be included into operating system. The output binary image size will be reduced as a result. Intercomponental communications are strictly documented, which allows for developer to use any third-party components or develop his own ones.

eCos supports a wide spectrum of architectures: 16, 32 and 64 bit wide; MPU, MCU and DSP cores.

eCos kernel, libraries and run-time components use Hardware Abstraction Level (HAL). So they are able to run on any systems with drivers ported into HAL.

The main functions of eCos:

- Hardware Abstraction Level (HAL).

- Real-time kernel:

 - Interrupt handling.

 - Exception handling.

 - Task Scheduler configuration.

 - Thread support.

 - Kinds of synchronization primitives.

 - Timers, counters and signals support.

 - Memory allocation subsystem configuration.

 - Debug and profiling support.

- uTRION 3.0 compatible API.

- POSIX compatible API.

- ISO C math libraries.

- Drivers for serial ports, Ethernet, wallclock and watchdog are included.

- USB Slave support.

- Contains 3 different TCP/IP stacks.

- GDB Debugger support.

All eCos applications are executed with RedBoot bootloader, which runs parallel to operating system and can have its own IP-address. Compiled RedBoot with TCP/IP stack inside takes about 100 kB of memory. RedBoot can be executed both from RAM and ROM.

All necessary eCos and RedBoot documentation can be found at <http://www.nxp.com/redirect/ecos.sourceware/>.

Remark: Operating system eCos, bootloader RedBoot, and user application can only be compiled with GCC (arm-toolchain redaction) under Linux or Cygwin. All compilation instructions are available from <http://www.nxp.com/redirect/ecos.sourceware/>.

3. Olimex LPCE-2214 evaluation board

3.1 Board design

Olimex LPCE-2214 evaluation board is shown on Figure 1.

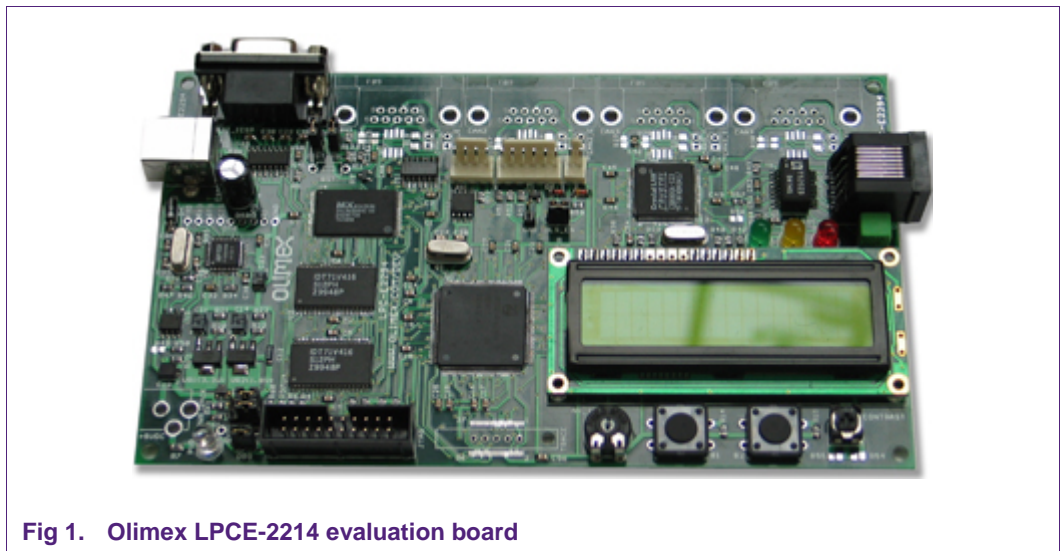


Fig 1. Olimex LPCE-2214 evaluation board

3.2 Key features

Key features of Olimex LPCE-2214 board:

1. MCU: **LPC2214** 16/32 bit ARM7TDMI-S™t with 256K Bytes Program Flash, 16K Bytes RAM, EXTERNAL MEMORY BUS, RTC, 4x 10 bit ADC 2.44 uS, 2x UARTs, I2C, SPI, 2x 32bit TIMERS, 7x CCR, 6x PWM, WDT, 5V tolerant I/O, up to 60MHz operation
2. standard JTAG connector with ARM 2x10 pin layout for programming/debugging with ARM-JTAG
3. optional ETM connector (not installed)
4. 1MB (256Kx32bit) 12 ns 71V416 SRAM
5. 1MB (512Kx16bit) 55 ns MX26LV800BTC FLASH
6. standard JTAG connector with ARM 2x10 pin layout for programming/debugging with ARM-JTAG
7. USB to RS232 converter, board can take power only from USB port
8. Via the USB-RS232 virtual port, the RESET circuit can be externally controlled via the NXP ISP utility
9. Jumpers for boot select from external memory

10. Jumpers for ISP/RUN mode
11. Ethernet controller with CS8900A and RJ45 connector
12. LCD 16x2 DISPLAY with BACKLIGHT
13. 2 BUTTONS
14. POTENTIOMETER connected to ADC
15. SPI connector
16. RS232 driver and connector
17. DALLAS i-BUTTON interface and connector
18. I2C 24LC515 EEPROM on board
19. two on board voltage regulators 1.8V and 3.3V with up to 800mA current

4. Compilation instructions

To start working with eCos on any MCU with specific peripherals, it's necessary:

1. Prepare RedBoot bootloader.
2. Configure and compile eCos kernel.
3. Write your own application and run it on the board.

Remark: many widespread MCU cores and evaluation boards are already supported by RedBoot and eCos. The list is available on <http://www.nxp.com/redirect/ecos.sourceware/>. Usually, there is no necessity to compile RedBoot and eCos – the precompiled differently configured images are available with eCos distribution.

4.1 Preparing

Prepare the compiler and eCos sources. Detailed instruction is available at <http://www.nxp.com/redirect/ecos.sourceware/getstart>.

Remark: as a rule, there are no direct links to the latest eCos distribution. It's possible to get the latest eCos snapshot from CVS public repository. All instructions are available at eCos site. It's necessary to get the latest snapshot to make eCos running on Olimex LPCE-2214. Weekly snapshots are available from <http://www.nxp.com/redirect/ecoscentric/devzone/snapshots/>.

4.2 Modifying eCos sources

In case of correct execution of instructions above, we have eCos sources (usually in /opt/ecos directory) and arm-linux-gcc compiler. It's necessary to make changes in eCos sources to make it possible to run it on Olimex LPCE-2214.

4.2.1 Add CS8900A driver (included).

Execute the following commands in the shell.

```
1 ECOS_REPOSITORY=/opt/ecos/packages
2 zcat ./devs_eth_cl_cs8900a.patch.gz | patch -d $ECOS_REPOSITORY -p0
```

4.2.2 Add Olimex LPCE-2214 Template.

Execute the following commands in the shell.

```
1 tclsh $ECOS_REPOSITORY/ecosadmin.tcl --accept_license add ./olpce2294-1.0.epk
```

4.2.3 Change flash-memory type support.

Execute the following commands in the shell (all required files are included).

```
2 ECOS_REPOSITORY=/opt/ecos/packages
3 cp ./flash_olpce2294.cdl $ECOS_REPOSITORY/devs/flash/arm/olpce2294/v1_0/cdl/
4 cp ./arm_olpce2294_flash.c $ECOS_REPOSITORY/devs/flash/arm/olpce2294/v1_0/src
5 cp ./ecos.db $ECOS_REPOSITORY/
```

4.3 Building RedBoot

We have all sources prepared to compile. Make RedBoot:

```
1 ECOS_REPOSITORY=/opt/ecos/packages
2 cd $ECOS_REPOSITORY
3 ./ecosconfig new olpce2294 redboot
4 ./ecosconfig import \
5 $ECOS_REPOSITORY/hal/arm/lpc2xxx/olpce2294/v1_0/misc/redboot_ROM.ecm
6 ./ecosconfig resolve
7 ./ecosconfig tree
8 make
9 arm-elf-objcopy install/bin/redboot.elf -O ihex install/bin/redboot.hex
10 ls -l $ECOS_REPOSITROY/install/bin
```

Remark: “ecosconfig” file included in case of its absence.

Compiled redboot.hex can be burned to internal MCU flash using NXP Flash Utility. RedBoot console is available via HyperTerminal (or another terminal), configured as 57600 8-N-1.

Remark: by defaults, RedBoot starts searching DHCP and BOOTP servers in Local Area Network. In case of their absence it's necessary to execute the following actions:

```
1 fis init
2 fconfig
3 [input required parameters]
```

Required parameters are local and server's IP addresses. You can run “ping” to check the correctness of given parameters. Use “help ping” in RedBoot console.

4.4 Building the kernel

Execute the following commands in the shell.

```
1 ./ecosconfig new olpce2294 kernel
2 ./ecosconfig resolve
3 ./ecosconfig tree
4 make
5 ./ecosconfig new olpce2294 net
6 ./ecosconfig resolve
7 ./ecosconfig tree
8 make
```

4.5 User application compilation.

It's necessary to prepare Makefile first. We'll take the makefile from one of the eCos application examples (see Listing. 1). Test application source is shown in Listing. 2.

To compile and link your application, it's necessary to copy makefile and sources to the same directory and execute "make" command.

Compiled file "test" is the user application with kernel modules inside.

4.6 Loading and execution the application.

You can load the application using Ethernet interface:

```
1 load test
2 go
```

In this case, file "test" will be loaded from TFTP server, which was pointed while executing "fconfig". Simple TFTP/DHCP server is included.

It's possible to debug the application using GDB debugger. You can get access to it by connecting to board IP address (as it was configured with "fconfig") using port 9000.

Loading the application via UART:

```
1 load -v -r -b [addr] -m ymodem
2 go
```

where "addr" – is an address of any unused (free) RAM block of required length.

Remark: all those operations can be executed automatically with startup script, which is configurable via "fconfig". It's also possible to burn an application into ROM after "load ..." it into RAM:

```
1 fis create [appl_name]
```

After that you can use the following commands to execute your application even after reset (as it was stored in non-volatile memory):

```
1 fis load [appl_name]
2 go
```


5. Listing 1. Makefile example

```

1  # You can edit only these lines:
2  #####
3
4  TARGET          = test
5  OBJECTS         = test.o
6  INSTALL_DIR=/opt/ecos/packages/install
7
8  #####
9
10
11 include $(INSTALL_DIR)/include/pkgconf/ecos.mak
12
13 XCC              = $(ECOS_COMMAND_PREFIX)gcc
14 XCXX             = $(XCC)
15 XLD              = $(XCC)
16 STRIP           = $(ECOS_COMMAND_PREFIX)strip
17
18 CFLAGS           = -I$(INSTALL_DIR)/include
19 CXXFLAGS         = $(CFLAGS)
20 LDFLAGS          = -nostartfiles -L$(INSTALL_DIR)/lib -Ttarget.ld
21
22
23 .PHONY: all clean
24
25 all: $(TARGET)
26
27 clean:
28     -rm -f ./*.o
29     -rm -f $(TARGET)
30
31 %.o: %.c
32     $(XCC) -c -o $*.o $(CFLAGS) $(ECOS_GLOBAL_CFLAGS) $<
33
34 %.o: %.cxx
35     $(XCXX) -c -o $*.o $(CXXFLAGS) $(ECOS_GLOBAL_CFLAGS) $<
36
37 %.o: %.C
38     $(XCXX) -c -o $*.o $(CXXFLAGS) $(ECOS_GLOBAL_CFLAGS) $<
39
40 %.o: %.cc
41     $(XCXX) -c -o $*.o $(CXXFLAGS) $(ECOS_GLOBAL_CFLAGS) $<
42
43 $(TARGET): $(OBJECTS)
44     $(XLD) $(LDFLAGS) $(ECOS_GLOBAL_LDFLAGS) -o $@ $@.o
45     $(STRIP) $(TARGET)

```

6. Listing 2. Application example

```
1  #include <errno.h>
2  #include <sys/param.h>
3  #include <cyg/infra/diag.h>
4  #include <cyg/kernel/kapi.h>
5  #include <pkgconf/system.h>
6  #include <network.h>
7  #include <stdio.h>
8  #include <stdlib.h>
9  #include <unistd.h>
10 #include <errno.h>
11 #include <sys/types.h>
12 #include <sys/socket.h>
13 #include <sys/sockio.h>
14 #ifdef CYGPKG_NET_FREEBSD_SYSCTL
15 #include <sys/sysctl.h>
16 #endif
17 #include <sys/param.h>
18 #include <netdb.h>
19 #include <arpa/inet.h>
20 #include <net/if.h>
21 #include <net/if_dl.h>
22 #include <net/if_types.h>
23 #include <net/route.h>
24 #include <netinet/in.h>
25
26
27 #define ETHER_ADDR_LEN      6
28 #define MACSTRING 18
29
30
31 char* ether_print(const u_char cp[ETHER_ADDR_LEN], char *etheraddr, const unsigned int len)
32 {
33     snprintf(etheraddr, len, "%02x:%02x:%02x:%02x:%02x:%02x", cp[0], cp[1], cp[2], cp[3], cp[4], cp[5]);
34     return(etheraddr);
35 }
36
37 int IP_SetMAC(const char* interface, unsigned char mac[ETHER_ADDR_LEN])
38 {
39     int test_sock=0;           // Socket PF_INET/SOCK_DGRAM
40     unsigned char i=0;
41     struct ifreq ifr;
42     unsigned char display[MACSTRING];
43
44     test_sock = socket( PF_INET, SOCK_DGRAM, 0 );
45     if( test_sock == -1 )
46     {
47         diag_printf("Cannot obtain socket");
48         return (-1);
49     }
50 }
```

```

49     }
50
51     memset(&ifr,0,sizeof( struct ifreq ) );
52     strncpy(ifr.ifr_name,interface,IFNAMSIZ);
53     for (i=0;i<ETHER_ADDR_LEN;i++)
54         ifr.ifr_hwaddr.sa_data[i]=mac[i];
55     ether_print(mac,display,MACSTRING);
56     if( ioctl( test_sock, SIOCSIFHWADDR, &ifr ) == -1 )
57     {
58         diag_printf("Cannot set MAC address for '%s' to '%s' because '%s'", interface, display,
59             strerror(errno));
60         close(test_sock);
61         return (-1);
62     }
63     else diag_printf("MAC Adress for '%s' set to %s",interface,display);
64     close(test_sock);
65     return(0);
66 }
67
68 int IP_GetIP (const char *interface, char *ip_address)
69 {
70     int s;
71     struct ifreq ifr;
72     struct sockaddr_in* addr=NULL;
73
74     s = socket(AF_INET, SOCK_DGRAM, 0);
75     if (s < 0)
76     {
77         printf("Cannot obtain IP socket\n");
78         return -1;
79     }
80
81     strcpy(ifr.ifr_name, interface);
82
83     if( ioctl( s, SIOCGIFADDR, &ifr ) == -1 )
84     {
85         printf("Cannot obtain IP address\n");
86         return -1;
87     }
88     else
89     {
90         addr= (struct sockaddr_in *)&(ifr.ifr_addr);
91         strcpy(ip_address,inet_ntoa(addr->sin_addr));
92         printf("IP address for '%s' is '%s'\n",interface,inet_ntoa(addr->sin_addr));
93         show_bootp(eth0_name, &eth0_bootp_data);
94         return 0;
95     }
96 }
97
98 int IP_SetIP(const char* interface,const char * address)
99 {

```

```

100     int test_sock=0;
101     struct sockaddr_in* addr=NULL;
102     struct ifreq ifr;
103
104     test_sock = socket( PF_INET, SOCK_DGRAM, 0 );
105     if( test_sock == -1 )
106     {
107         fprintf(stderr,"Cannot obtain socket");
108         return (-1);
109     }
110
111     memset(&ifr,0,sizeof( struct ifreq ) );
112     strncpy(ifr.ifr_name,interface,IFNAMSIZ);
113
114     if( ioctl( test_sock, SIOCGIFADDR, &ifr ) == -1 )
115     {
116         diag_printf("Cannot obtain IP address of '%s' because '%s'",interface,strerror(errno));
117     }
118     else
119     {
120         if( ioctl( test_sock, SIOCDFADDR, &ifr ) != 0 )
121         {
122             diag_printf("Cannot suppress old IP for '%s' because '%s'",interface,strerror(errno));
123         }
124     }
125
126     memset( &ifr, 0, sizeof( struct ifreq ) );
127     addr= (struct sockaddr_in *)&(ifr.ifr_addr);
128     memset(addr, 0, sizeof( struct sockaddr_in ) );
129     addr->sin_len=sizeof(struct sockaddr_in);
130     addr->sin_family=AF_INET;
131     addr->sin_addr.s_addr=inet_addr(address);
132     strncpy(ifr.ifr_name,interface,IFNAMSIZ);
133
134     if( ioctl( test_sock, SIOCSIFADDR, &ifr ) != 0 )
135     {
136         diag_printf("Cannot set IP address of '%s' to '%s' because '%s'", interface, address,
137             strerror(errno));
138         close(test_sock);
139         return (-1);
140     }
141     else diag_printf("IP address for '%s' set '%s' ",interface,inet_ntoa(addr->sin_addr));
142     close(test_sock);
143     return(0);
144 }
145
146
147 #ifndef CYGPKG_LIBC_STDIO
148 #define perror(s) diag_printf(#s ": %s\n", strerror(errno))
149 #endif
150

```

```
151 #define STACK_SIZE (CYGNUM_HAL_STACK_SIZE_TYPICAL + 0x1000)
152 static char stack[STACK_SIZE];
153 static cyg_thread thread_data;
154 static cyg_handle_t thread_handle;
155
156 extern void cyg_test_exit(void);
157
158 void pexit(char *s)
159 {
160     perror(s);
161     cyg_test_exit();
162 }
163
164 static void server_test(struct bootp *bp)
165 {
166     int s, client;
167     socklen_t client_len;
168     struct sockaddr_in client_addr, local;
169     char buf[1024*101];
170     int one = 1;
171     fd_set in_fds;
172     int num, len;
173     struct timeval tv;
174
175     printf ("changing MAC...\n");
176     char mac [ETHER_ADDR_LEN] = {0x08, 0x88, 0x12, 0x34, 0x56, 0x78};
177     IP_SetMAC ("eth0", mac);
178
179     char cur_ip [100];
180     IP_GetIP ("eth0", cur_ip);
181     printf ("\nCurrent IP: %s\n", cur_ip);
182
183
184     char new_IP[4] = {192, 168, 100, 102};
185     char new_IP_str[16];
186     sprintf (new_IP_str, "%d.%d.%d.%d\n", new_IP[0], new_IP[1], new_IP[2], new_IP[3]);
187     IP_SetIP ("eth0", new_IP_str);
188
189     s = socket(AF_INET, SOCK_STREAM, 0);
190     if (s < 0) {
191         printf("stream socket\n");
192     } else printf ("stream ok\n");
193     if (setsockopt(s, SOL_SOCKET, SO_REUSEADDR, &one, sizeof(one))) {
194         printf("setsockopt SO_REUSEADDR\n");
195     } else printf ("reuse ok\n");
196     if (setsockopt(s, SOL_SOCKET, SO_REUSEPORT, &one, sizeof(one))) {
197         printf("setsockopt SO_REUSEPORT\n");
198     } else printf ("reuse2 ok\n");
199     memset(&local, 0, sizeof(local));
200     local.sin_family = AF_INET;
201     local.sin_len = sizeof(local);
```

```
202     local.sin_port = htons(7734);
203     local.sin_addr.s_addr = INADDR_ANY;
204     if(bind(s, (struct sockaddr *) &local, sizeof(local)) < 0) {
205         printf("bind error\n");
206     }else printf ("bind ok\n");
207     listen(s, SOMAXCONN);
208
209     while (true)
210     {
211         client_len = sizeof(client_addr);
212         if ((client = accept(s, (struct sockaddr *)&client_addr, &client_len)) < 0) {
213             printf("accept");
214         }
215         client_len = sizeof(client_addr);
216         getpeername(client, (struct sockaddr *)&client_addr, &client_len);
217         diag_printf("connection from %s:%d\n", inet_ntoa(client_addr.sin_addr),
218             ntohs(client_addr.sin_port));
219
220     #ifdef CYGPKG_LIBC_STDIO
221         sprintf(buf, "Hello %s:%d\n", inet_ntoa(client_addr.sin_addr), ntohs(client_addr.sin_port));
222     #else
223         strcpy(buf, "Hello ");
224         strcat(buf, inet_ntoa(client_addr.sin_addr));
225         strcat(buf, ":");
226         strcat(buf, atoi(ntohs(client_addr.sin_port)));
227         strcat(buf, "\n");
228     #endif
229         write(client, buf, strlen(buf));
230         close(client);
231     }
232
233     close(s);
234 }
235
236 void net_test(cyg_addrword_t param)
237 {
238     diag_printf("Start SERVER test\n");
239
240
241     init_all_network_interfaces();
242     #ifdef CYGHWR_NET_DRIVER_ETH0
243         if (eth0_up) {
244             server_test(&eth0_bootp_data);
245         }
246     #endif
247
248     cyg_test_exit();
249 }
250
251 void cyg_start(void)
252 {
```

```
253     printf ("START!\n");
254     // Create a main thread, so we can run the scheduler and have time 'pass'
255     cyg_thread_create(10,           // Priority - just a number
256                     net_test,     // entry
257                     0,             // entry parameter
258                     "Network test", // Name
259                     &stack[0],     // Stack
260                     STACK_SIZE,    // Size
261                     &thread_handle, // Handle
262                     &thread_data   // Thread data structure
263     );
264     cyg_thread_resume(thread_handle); // Start it
265     cyg_scheduler_start();
266 }
```

7. Legal information

7.1 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.

7.2 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.

7.3 Trademarks

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

8. Index

C	
cyg_start	13
E	
eCos	3
ether_print.....	9
Evaluation board.....	4
I	
IP_GetIP	10
IP_SetIP.....	10
IP_SetMAC.....	9
N	
net_test.....	13
S	
server_test.....	12

9. Contents

1.	Introduction	3
2.	Introduction into eCos	3
3.	Olimex LPCE-2214 evaluation board	4
3.1	Board design	4
3.2	Key features	4
4.	Compilation instructions	5
4.1	Preparing.....	5
4.2	Modifying eCos sources	5
4.2.1	Add CS8900A driver (included)	5
4.2.2	Add Olimex LPCE-2214 Template	6
4.2.3	Change flash-memory type support.	7
4.3	Building RedBoot.....	7
4.4	Building the kernel.....	7
4.5	User application compilation.....	7
4.6	Loading and execution the application.	8
5.	Listing 1. Makefile example	9
6.	Listing 2. Application example.....	10
7.	Legal information	16
7.1	Definitions.....	16
7.2	Disclaimers.....	16
7.3	Trademarks	16
8.	Index.....	17
9.	Contents.....	18

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

