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.

founded by Philips

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.
- uITRION 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.

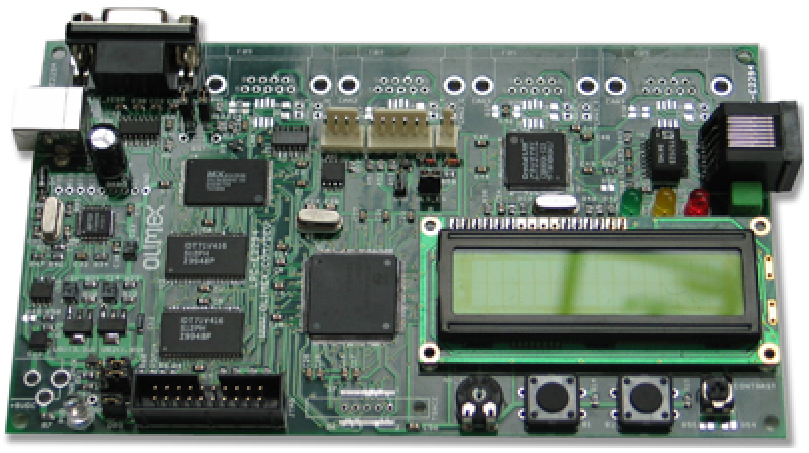

Fig 1. Olimex LPCE-2214 evaluation board

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.

```
ECOS_REPOSITORY=/opt/ecos/packages
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.

```
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).

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

4.3 Building RedBoot

We have all sources prepared to compile. Make RedBoot:

```
ECOS_REPOSITORY=/opt/ecos/packages
cd $ECOS_REPOSITORY
./ecosconfig new olpce2294 redboot
./ecosconfig import \
$ECOS_REPOSITORY/hal/arm/lpc2xxx/olpce2294/v1_0/misc/redboot_ROM.ecm
./ecosconfig resolve
./ecosconfig tree
make
arm-elf-objcopy install/bin/redboot.elf -O ihex install/bin/redboot.hex
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:

```
fis init
fconfig
[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.

```
./ecosconfig new olpce2294 kernel
./ecosconfig resolve
./ecosconfig tree
make
./ecosconfig new olpce2294 net
./ecosconfig resolve
./ecosconfig tree
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:

```
load test
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:

```
load -v -r -b [addr] -m ymodem
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:

```
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):

```
fis load [appl_name]
go
```

5. Listing 1. Makefile example

```
#  You can edit only these lines:
#############################################

TARGET          = test
OBJECTS         = test.o
INSTALL_DIR=/opt/ecos/packages/install

##############################################


include $(INSTALL_DIR)/include/pkgconf/ecos.mak

XCC           = $(ECOS_COMMAND_PREFIX)gcc
XCXX          = $(XCC)
XLD           = $(XCC)
STRIP        = $(ECOS_COMMAND_PREFIX)strip

CFLAGS        = -I$(INSTALL_DIR)/include
CXXFLAGS      = $(CFLAGS)
LDFLAGS       = -nostartfiles -L$(INSTALL_DIR)/lib -Ttarget.ld


.PHONY: all clean

all: $(TARGET)

clean:
      -rm -f ./*.o
      -rm -f $(TARGET)

%.o: %.c
      $(XCC) -c -o $*.o $(CFLAGS) $(ECOS_GLOBAL_CFLAGS) $<

%.o: %.cxx
      $(XCXX) -c -o $*.o $(CXXFLAGS) $(ECOS_GLOBAL_CFLAGS) $<

%.o: %.C
      $(XCXX) -c -o $*.o $(CXXFLAGS) $(ECOS_GLOBAL_CFLAGS) $<

%.o: %.cc
      $(XCXX) -c -o $*.o $(CXXFLAGS) $(ECOS_GLOBAL_CFLAGS) $<

$(TARGET): $(OBJECTS)
      $(XLD) $(LDFLAGS) $(ECOS_GLOBAL_LDFLAGS) -o $@ $@.o
      $(STRIP) $(TARGET)
```

6. Listing 2. Application example

```
#include <errno.h>
#include <sys/param.h>
#include <cyg/infra/diag.h>
#include <cyg/kernel/kapi.h>
#include <pkgconf/system.h>
#include <network.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <errno.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <sys/sockio.h>
#ifdef CYGPKG_NET_FREEBSD_SYSCTL
#include <sys/sysctl.h>
#endif
#include <sys/param.h>
#include <netdb.h>
#include <arpa/inet.h>
#include <net/if.h>
#include <net/if_dl.h>
#include <net/if_types.h>
#include <net/route.h>
#include <netinet/in.h>


#define ETHER_ADDR_LEN      6
#define MACSTRING 18


char* ether_print(const u_char cp[ETHER_ADDR_LEN], char *etheraddr,const unsigned int len)
{
      snprintf(etheraddr,len,"%02x:%02x:%02x:%02x:%02x:%02x",cp[0],cp[1],cp[2],cp[3],cp[4],cp[5]);
      return(etheraddr);
}

int IP_SetMAC(const char* interface,unsigned char mac[ETHER_ADDR_LEN])
{
      int test_sock=0;            // Socket PF_INET/SOCK_DGRAM
      unsigned char i=0;
      struct ifreq ifr;
      unsigned char display[MACSTRING];

      test_sock = socket( PF_INET, SOCK_DGRAM, 0 );
      if( test_sock == -1 )
      {
            diag_printf("Cannot obtain socket");
            return (-1);
```

```
        }

        memset(&ifr,0,sizeof( struct ifreq ) );
        strncpy(ifr.ifr_name,interface,IFNAMSIZ);
        for (i=0;i<ETHER_ADDR_LEN;i++)
              ifr.ifr_hwaddr.sa_data[i]=mac[i];
        ether_print(mac,display,MACSTRING);
        if( ioctl( test_sock, SIOCSIFHWADDR, &ifr ) == -1 )
        {
              diag_printf("Cannot set MAC address for '%s' to '%s' because '%s'", interface, display,
                    strerror(errno));
              close(test_sock);
              return (-1);
        }
        else diag_printf("MAC Adress for '%s' set to %s'",interface,display);
        close(test_sock);
        return(0);
  }

  int IP_GetIP (const char *interface, char *ip_address)
  {
        int s;
        struct ifreq ifr;
        struct sockaddr_in* addr=NULL;

        s = socket(AF_INET, SOCK_DGRAM, 0);
        if (s < 0)
        {
              printf("Cannot obtain IP socket\n");
              return -1;
        }

        strcpy(ifr.ifr_name, interface);

        if( ioctl( s, SIOCGIFADDR, &ifr ) == -1 )
        {
              printf("Cannot obtain IP address\n");
              return -1;
        }
        else
        {
              addr= (struct sockaddr_in *)&(ifr.ifr_addr);
              strcpy(ip_address,inet_ntoa(addr->sin_addr));
              printf("IP address for '%s' is '%s'\n",interface,inet_ntoa(addr->sin_addr));
              show_bootp(eth0_name, &eth0_bootp_data);
              return 0;
        }
  }

  int IP_SetIP(const char* interface,const char * address)
  {
        int test_sock=0;
        struct sockaddr_in* addr=NULL;
        struct ifreq ifr;

        test_sock = socket( PF_INET, SOCK_DGRAM, 0 );
        if( test_sock == -1 )
        {
              fprintf(stderr,"Cannot obtain socket");
              return (-1);
        }

        memset(&ifr,0,sizeof( struct ifreq ) );
        strncpy(ifr.ifr_name,interface,IFNAMSIZ);

        if( ioctl( test_sock, SIOCGIFADDR, &ifr ) == -1 )
        {
              diag_printf("Cannot obtain IP address of '%s' because '%s'",interface,strerror(errno));
        }
        else
        {
              if( ioctl( test_sock, SIOCDIFADDR, &ifr ) != 0 )
              {
                    diag_printf("Cannot suppress old IP for '%s' because '%s'",interface,strerror(errno));
              }
        }

        memset( &ifr, 0, sizeof( struct ifreq ) );
        addr= (struct sockaddr_in *)&(ifr.ifr_addr);
        memset(addr, 0, sizeof( struct sockaddr_in) );
        addr->sin_len=sizeof(struct sockaddr_in);
        addr->sin_family=AF_INET;
        addr->sin_addr.s_addr=inet_addr(address);
        strncpy(ifr.ifr_name,interface,IFNAMSIZ);

        if( ioctl( test_sock, SIOCSIFADDR, &ifr ) != 0 )
        {
              diag_printf("Cannot set IP address of '%s' to '%s' because '%s'", interface, address,
                    strerror(errno));
              close(test_sock);
              return (-1);
        }
        else diag_printf("IP address for '%s' set '%s' ",interface,inet_ntoa(addr->sin_addr));
        close(test_sock);
        return(0);
  }


  #ifndef CYGPKG_LIBC_STDIO
  #define perror(s) diag_printf(#s ": %s\n", strerror(errno))
  #endif

```

```
#define STACK_SIZE (CYGNUM_HAL_STACK_SIZE_TYPICAL + 0x1000)
static char stack[STACK_SIZE];
static cyg_thread thread_data;
static cyg_handle_t thread_handle;

extern void cyg_test_exit(void);

void pexit(char *s)
{
    perror(s);
    cyg_test_exit();
}

static void server_test(struct bootp *bp)
{
    int s, client;
    socklen_t client_len;
    struct sockaddr_in client_addr, local;
    char buf[1024*101];
    int one = 1;
    fd_set in_fds;
    int num, len;
    struct timeval tv;

    printf ("changing MAC...\n");
    char mac [ETHER_ADDR_LEN] = {0x08, 0x88, 0x12, 0x34, 0x56, 0x78};
    IP_SetMAC ("eth0", mac);

    char cur_ip [100];
    IP_GetIP ("eth0", cur_ip);
    printf ("\nCurrent IP: %s\n", cur_ip);


    char new_IP[4] = {192, 168, 100, 102};
    char new_IP_str[16];
    sprintf (new_IP_str, "%d.%d.%d.%d\0", new_IP[0], new_IP[1], new_IP[2], new_IP[3]);
    IP_SetIP ("eth0", new_IP_str);

    s = socket(AF_INET, SOCK_STREAM, 0);
    if (s < 0) {
        printf("stream socket\n");
    } else printf ("stream ok\n");
    if (setsockopt(s, SOL_SOCKET, SO_REUSEADDR, &one, sizeof(one))) {
        printf("setsockopt SO_REUSEADDR\n");
    }else printf ("reuse ok\n");
    if (setsockopt(s, SOL_SOCKET, SO_REUSEPORT, &one, sizeof(one))) {
        printf("setsockopt SO_REUSEPORT\n");
    }else printf ("reuse2 ok\n");
    memset(&local, 0, sizeof(local));
    local.sin_family = AF_INET;
    local.sin_len = sizeof(local);
    local.sin_port = htons(7734);
    local.sin_addr.s_addr = INADDR_ANY;
    if(bind(s, (struct sockaddr *) &local, sizeof(local)) < 0) {
        printf("bind error\n");
    }else printf ("bind ok\n");
    listen(s, SOMAXCONN);

    while (true)
    {
        client_len = sizeof(client_addr);
        if ((client = accept(s, (struct sockaddr *)&client_addr, &client_len)) < 0) {
            printf("accept");
        }
        client_len = sizeof(client_addr);
        getpeername(client, (struct sockaddr *)&client_addr, &client_len);
        diag_printf("connection from %s:%d\n", inet_ntoa(client_addr.sin_addr),
           ntohs(client_addr.sin_port));

#ifdef CYGPKG_LIBC_STDIO
        sprintf(buf, "Hello %s:%d\n", inet_ntoa(client_addr.sin_addr), ntohs(client_addr.sin_port));
#else
        strcpy(buf, "Hello ");
        strcat(buf, inet_ntoa(client_addr.sin_addr));
        strcat(buf,":");
        strcat(buf, atoi(ntohs(client_addr.sin_port)));
        strcat(buf,"\n");
#endif
        write(client, buf, strlen(buf));
        close(client);
    }

    close(s);
}

void net_test(cyg_addrword_t param)
{
    diag_printf("Start SERVER test\n");


    init_all_network_interfaces();
#ifdef CYGHWR_NET_DRIVER_ETH0
    if (eth0_up) {
        server_test(&eth0_bootp_data);
    }
#endif

    cyg_test_exit();
}

void cyg_start(void)
{
    printf ("START!\n");
    // Create a main thread, so we can run the scheduler and have time 'pass'
    cyg_thread_create(10,                   // Priority - just a number
                      net_test,             // entry
                      0,                    // entry parameter
                      "Network test",       // Name
                      &stack[0],            // Stack
                      STACK_SIZE,           // Size
                      &thread_handle,       // Handle
                      &thread_data          // Thread data structure
            );
    cyg_thread_resume(thread_handle);  // Start it
    cyg_scheduler_start();
}
```

7. Legal information

7.1 Definitions

7.2 Disclaimers

7.3 Trademarks

8. Index

9. Contents

Date of release: 1 August 2007

Document identifier: AN10605_1

founded by PHILIPS