Visit us at www.hitex.de, www.hitex.co.uk or www.hitex.com

hitex DEVELOPMENT TOOLS

Hitex Germany (Head Quarters)
Greschbachstr. 12 ☎ +049-721-9628-0
D-76229 Karlsruhe Fax +049-721-9628-149
✉ Sales@hitex.de

Hitex USA
2062 Business Center ☎ 800-45-HITEX
Drive, Suite 230 ☎ +1-949-863-0320
Irvine, CA 92612 Fax +1-949-863-0331
✉ Info@hitex.com

Hitex UK
Warwick University ☎ +44-24-7669-2066
Science Park Fax +44-24-7669-2131
GB Coventry CV47EZ ✉ Info@hitex.co.uk

Hitex Asia
25 International ☎ +65-6566-7919
Business Park, #04-62A Fax +65-6563-7539
German Centre ✉ Sales@hitexasia.com.sg
Singapore 609916

Embedding Software Quality

Application Example

for

LPC2378

MCB2300 / FreeRTOS (GCC)

This documentation describes the example port of the FreeRTOS operating system for LPC23xx from NXP.

Architecture:	ARM
Device:	LPC2378
Chip Manufacturer:	NXP
Peripherals:	LEDs, Speaker, LCD
Compiler:	GNU GCC ARM-elf v.4.0.3

Author:	Stefan Grohmann /Hitex Germany
Contact:	applications@hitex.de
Revision:	03/2007 - 002

Preface

In order to support you when working with our products, we provide documents containing specific examples, additional topics, special hints, detailed procedures etc.

This document explains features and functions of an application example to run with the HiTOP user interface.

For more information on the current software and hardware revisions, as well as our update service, please contact www.hitex.de, www.hitex.co.uk or www.hitex.com.

Contents

1	**Scope and Disclaimer**	**3**
2	**Original Software and Documentation**	**4**
3	**Used Modules**	**5**
3.1	C Sources	5
3.1.1	main.c	5
3.2.1	startup.s	6
3.2.2	portISR.c	6
3.2.3	port.c	6
3.2.4	lcd_mcb23xx.c	6
4	**Header Files**	**7**
5	**Memory**	**7**
5.1	Stack	7
5.2	Heap	7
6	**API Usage**	**8**
7	**Customizing**	**8**
8	**Board Configuration**	**8**
9	**Compiler-Specific Keys and Options**	**9**
9.1	__attribute__((naked))	9
9.2	__attribute__((interrupt(IRQ)))	9

1 Scope and Disclaimer

This software is a port of the current version of the FreeRTOS operating system for the ARM-based LPC23xx controller by NXP (Philips Semiconductors). The port includes the hardware-related driver and definitions as well as the standard tasks of the demo applications.

Subject Matter

The design is made to show the implementation and configuration of the real time operating system on the Keil MCB2300 board populated with the LPC2378 device. Basic software is GPL licensing model, laid open on the according WEB Sites. The implementation is 'as it is' and no warranties are made. The use and the change of the code is free and not under control of the originators.

Requirements

- Hitex HiTOP5.2 ARM Development System with Tanto or Tantino
- Keil MBC2300 board
- LPC2378 controller
- System tool provided by MS-Windows OS

Features and Components

- Platform: LPC2378 ARM core
- Keil MCB2300 board or equivalent
- GNU ARM compiler GCC-hitex-elf version 4.0.3
- LIBs (Interwork / GNU ARM compiler)
- LCD, LED, Speaker peripherals are used

Procedure

Installing

1. Step: Install HiTOP 52 ARM.
2. Step: Configure the Keil evaluation board.
3. Step: Supply power to the evaluation board.

Starting

1. Step: Copy the project data to the working directory.
2. Step: Load the project with the HiTOP 5 user interface.
3. Step: Run the project with the preconfigured settings.

Customizing

1. Step: Enable or disable selected tasks in the main.c file by commenting them according to your individual needs.
2. Step: Adjust the stack size to the enabled tasks in the file startup.s, section “Stack size definitions”.
3. Step: Adjust the heap size if necessary in the file FreeRTOSConfig.h (see configTOTAL_HEAP_SIZE definition).
4. Step: Recompile the project, download to the target hardware and run.

2 Original Software and Documentation

For more information on the FreeRTOS operating system see the related documents on www.freertos.org.

The current version 4.0.3 and the port to the LPC23xx hardware offer basic functions of real-time handling, task management and memory management based on the original codes. Other applicational functions can be applied on this basic software.

The software also contains demo applications running in different OS modes.

3 Used Modules

This chapter describes source files changed or adapted for the FreeRTOS demonstration project. For information on all other files not described here, refer to www.freertos.org.

3.1 C Sources

All sources are written in pure C following the ANSI rules nearly strictly. For all sources originated by the GPL code of the original FreeRTOS implementation the originator is responsible.

Startup.s	ASM file for setup routines
lcd_mcb2300.c	Task- and hardware-specific routines for the LCD module
port.c	Controller-specific routines for basic RTOS functions
portISR.c	Controller-specific routines for interrupt processing
croutine.c	Cooperative routines
heap_2.c	Memory management
list.c	Task list functions
queue.c	Queue management
tasks.c	Kernel control
flash.c	Task: Demo to flash LEDs
integer.c	Task: Demo on calculation tasks
ParTest.c	Help routines to access IO
print.c	Task: Demo for queuing task
semtest.c	Task: Demo for semaphore task

3.1.1 main.c

The main program (main.c) provides the following routines:

prvLCDOutHitexLogo()

Prints out an example text to the LCD module.

Lprintf()

Should be used to print single line messages to the LCD.

Main()

The main routine contains a call for all related hardware setup routines (**prvSetupHardware()**). The startup code in the file startup.s initializes hardware components like PLL and clock. After LCD initialization and LCD output, some of the Demo tasks are created with their selected priority. After all applicational tasks have been started, the highest prioritized task **vErrorChecks** is started. At this point the operating system is ready to run. All tasks are created, i.e. initial routines are done and memory is allocated. Now the IDLE task has to be created and the timer has to be started. This is done in the function **vTaskStartScheduler()**. Note that the routine is not written as an endless loop, because the operating system is invoked and the routine is left.

vErrorChecks()

This routine is the highest priority task calling the routine **prvCheckOtherTasksAreStillRunning()**.

prvSetupHardware

This routine is called at the entry of the main function to configure the controller's peripherals if they are not configured with the task creation process.

prvToggleOnBoardLED()

This routine is not used.

prvCheckOtherTasksAreStillRunning()

This routine checks for all the applications if they are still running or if an error occurred.

vMemCheckTask()

This dynamic task routine is created and deleted during each cycle of the **vErrorChecks()** task to check the operation of the memory locator.

3.1.2 startup.s

One of the main aspects on ARM controllers is the usage of appropriate startup code. The basic initialization and memory allocation is done via this file.

3.1.3 portISR.c

This module includes the routine to initialize and handle the VIC interrupt of the timer 0 unit. Also the disabling and enabling of the interrupts is handled.

3.1.4 port.c

This module contains the controller-specific implementation of the VIC and timer 0 initialization and stack initialization routine

3.1.5 lcd_mcb23xx.c

The source contains all generic routines to enable communication with the onboard LCD module. Initialization and generic character output is implemented. Another task is realized to provide a progress bar and a running text into the second line of the LCD display.

4 Header Files

Defines.h	General definitions and types
Peripherals_lpc23xx.h	Hardware-related stuff
lcd_mcb2300.h	LCD routine stuff
portmacro.h	Routines to switch context
portable.h	Memory allocation
croutine.h	Cooperative task stuff
FreeRTOS.h	Main include for FreeRTOS
FreeRTOSConfig.h	Configuration file
list.h	Structs and macro for list stuff
projdefs.h	Project definitions
queue.h	Queue stuff
semphr.h	Semaphore stuff
task.h	Type definitions and macros for the kernel
flash.h	Task predeclaration and const
integer.h	Task predeclaration and const
partest.h	Task predeclaration and const
print.h	Predeclaration and const
semtest.h	Task predeclaration and const

5 Memory

In general the amount of RAM on embedded systems is limited and/or they have different timing restrictions if external RAM is used. The Free RTOS port offers a single scheme to handle the allocation algorithm. Several schemes are available but the HEAP_2 is chosen.

5.1 Stack

The overall size of the reserved stack memory is declared in the startup.s file. The ARM architecture enables to define stack for several processor modes. It is recommended (even for good coding style) to allocate the stack in the appropriate modes. The timer of the RTOS uses IRQ especially by using the VIC unit. In this case, the stack should be allocated on IRQ mode section. If the stack is accessed and there is no special routine for checking the processor mode, the stack is treated as a linear memory area with descending addressing. The amount of stack to allocate is depending strictly on the count of tasks and the individual configured stack sizes in the tasks.

5.2 Heap

The size of allocated heap memory is defined in the file FreeRTOSconfig.h with the definition of configTOTAL_HEAP_SIZE. This memory block takes quite a lot of the controller resources.

In general, it is application-specific how many heap should be allocated. In detail, the kernel control task uses the highest amount of heap memory.

6 API Usage

The usage of the implemented co-routines is set off with the option "configUSE_CO_ROUTINES 0" in the file FreeRTOSConfig.h. The co-routines provide a set of prioritized cooperative routines. They come along with reduced RAM usage. Otherwise tasks are fully preemptive and are similar to the so-called traditional scheduling model. The FreeRTOS API allows general task creation, control, queuing and semaphores as well as kernel control and some task utilities. Further documentation is provided on www.freertos.org.

7 Customizing

The OS is roughly configurable. The configurations and options are described in the related documents on www.freertos.org. In this document, only the options used are described. All these options have strong effects on the behavior of the OS and should be changed carefully. All the following options are set in the file FreeRTOSConfig.h. The options used are the following:

- **configUSE_PREEMPTION 1**
  Needed if preemptive tasking is used.
- **configUSE_IDLE_HOOK 0**
  Set to zero since the predefined prototype vApplicationIdleHook() is not used. Instead of implementing tasks with the “IDLE priority”, the definition is used.
- **configUSE_TICK_HOOK 0**
  Used for executing intermediate code in the tick interrupt. The option is set to zero.
- **configCPU_CLOCK_HZ ((unsigned portLONG) 48000000)**
  The PLL and Clock configuration is set to 48 MHz. Since the USB peripheral unit device is used, the frequence should be adjusted to 48 MHz. The APB divider is set to the system clock frequency.
- **configTICK_RATE_HZ ((portTickType) 1000)**
  Causes a minimal tick of 1 ms on the timer interrupt.
- **configMAX_PRIORITIES ((unsigned portBASE_TYPE) 5)**
  Sets the maximum priority to the given value.
- **configTOTAL_HEAP_SIZE ((size_t) (10 * 1024))**
  Affects the memory consumption of the OS enormously. If the size is set too small, the RTOS does not work correctly. If set too high a 'run out of memory' may occur.
- **configUSE_TRACE_FACILITY 0**
  Allows additional so-called debugging features on the OS. The trace can be used to log a specified time of the task management. The data can be given to a tool and converted to spread sheet format.
- **configIDLE_SHOULD_YIELD 1**
  Has only effects if the preemptive scheduling is used. A setting prevents the scheduler from ending a task before the time slice is run out.

8 Board Configuration

The Keil evaluation board MCB2300 must be configured for using a JTAG wiggler like Tanto or Tantino. To enable the port2 function, remove the jumper J13 (labeled as 'JTAG'). This jumper enables the trace port of the LPC23xx device. If the jumper is closed, the port2 is occupied by the control function of the trace module and no access to the onboard LED is possible.

9 Compiler-Specific Keys and Options

This chapter describes the compiler-related specific keys and options to handle the code with the GCC compiler.

9.1 __attribute__((naked))

The attribute “naked” prevents the compiler from using prolog and epilog. This attribute is to use for interrupt handling without access to the stack. Stack handling must be performed by the software.

9.2 __attribute__((interrupt(IRQ)))

This attribute forces the compiler to generate an interrupt function with entry and exit routine. This routines affecting the stack can only be used for normal interrupt handling with stack operation. The OS requires to add the attribute “naked” to this attribute. In this case, the compiler will not create prolog and epilog causing the stack and link register contents remaining unchanged. At this point of the program the task switching can be done. Allowed values for this parameter are: IRQ, FIQ, SWI, ABORT and UNDEF.

■