# Notes

**Fr**: A.C. Verbeck
**To**: File
**Re**: FreeRTOS example notes
**Port**: NXP LPC1788
**Board**:Haoyu Electronics HY-LPC1788-Core / HY-LPC1788-SDK
**Date**: April 20, 2013

**Introduction**
The examples in this port are from *Using the FreeRTOS Real Time Kernel – A Practical Guide.* This book can be purchased from the FreeRTOS shop. The book is a primer on RTOS usage targeted at ARM Cortex M3 processors. It comes with sixteen excellent examples. These are designed to run on the Keil µVision4 simulator. It is possible to get a demo version of the Keil MDK for free. However, it's very limited. The full version of the base Keil product is 5000USD. The more capable Professional Edition is even more.

Rowley Associates have a similar platform based on gcc that costs 1500USD for the complete professional version. The single-user Personal License version is a very reasonable 150USD. I have been using this system for about six months now and I find it as good as the Keil offering. I can move projects from Keil to CrossWorks in minutes.

The HY-LPC1788-Core / HY-LPC1788-SDK board set is an excellent system for working with NXP processors. The core processor board has: 12 MHz crystal main oscillator, 32.768 kHz crystal for Real Time Clock, LCD interface with back-light and touch panel circuit, EMC 32MB 16bit data bus SDRAM memory, EMC 128M NAND FLASH memory, EMC 8MB NOR FLASH memory, 2MB SPI FLASH memory, and 256Mbit EEPROM memory. The SDK board has 100/10M Ethernet interface, USB to serial port, connect UART0 (for ISP and debugging), two DB9 RS232 serial ports, two mini USB 2.0 device interface, two USB Host 2.0 interface, micro SD card slot, standard 2.54mm JTAG interface, $I^2S$ interface based on UDA1380HN, 3.5mm stereo audio output, 3.5mm stereo audio input, 3.5mm microphone input, two CAN 2.0B bus interface, one RS485 bus interface, 7 user buttons and 2MB SPI FLASH. It costs 120USD with a 4.3in LCD; 135USD with a 5in LCD, and 150USD with a 7in LCD. There is also a 32bit SDRAM version as well for 10USD more. Shipping is free.

In this port, the basic idea was to use as little of the platform as possible: this is done to completely eliminate any issues that may arise do to specific hardware configuration. This port uses the Cortex M3 core, internal RAM and FLASH, the SysTick timer, and last the

I have ported these examples to the following configuration:

| Tool/System | Manufacturer | Version |
|---|---|---|
| IDE | Rowley Associates | CrossWorks v2.3.1 |
| RTOS | Real Time Engineers Ltd. | FreeRTOS v7.4.0 |
| Processor | NXP | LPC1788 |
| Processor Board | HAOYU | HY-LPC1788-Core |
| Base board | HAOYU | HY-LPC1788-SDK |

**FreeRTOS v7.4.0 port notes**
There is still no prvSetupHardware() function in this FreeRTOS port. Normally, this function is used to configure the hardware prior to starting the RTOS. Rowley use the traditional CMSIS SystemInit() to initialize the processor in LPC177x_8x_Startup.s: I have added the SystemCoreClockUpdate() to LPC177x_8x_Startup.s to verify that the clock is at the expected frequency.

```
#define configCPU_CLOCK_HZ          ( 120000000 )
#define configTICK_RATE_HZ          ( ( portTickType ) 100 )
```

These are the two defines associated with the SysTick interrupt. The `configCPU_CLOCK_HZ` value must be set to "*..the frequency of the clock that drives the peripheral used to generate the kernels periodic tick interrupt. This is very often, but not always, equal to the main system clock frequency.*"

Notes:
1. The external crystal is 12MHz.
2. The PLL is the clock source for the processor. It is configured to operate at 120MHz.
3. The config define `configTICK_RATE_HZ` defaults to 1000. This is far too fast for a useful implementation. I have set this to a more reasonable 100.

FreeRTOS doesn't need to know the Core CPU frequency. It does need to know the frequency of the tick timer. For ARM Cortex processors, the `configCPU_CLOCK_HZ` define is very misleading because it implies that the processor clock is used to define the SysTick timer frequency. In this case (as well as nearly all Cortex processors) the SysTick timer frequency can be significantly different from the CPU frequency.

LPC177x_8x_Startup.s has been imported to each example. This was done to add the SystemCoreClockUpdate() call.

**Example Application Porting Notes**

Three interrupts are used for most of the examples. The fourth interrupt is used in three examples:

```
#define xPortSysTickHandler             SysTick_Handler
#define xPortPendSVHandler              PendSV_Handler
#define vPortSVCHandler                 SVC_Handler
#define vSoftwareInterruptHandler       WDG_IRQHandler
```

1. **SysTick_Handler**: is the CMSIS 10ms system tick. This is used to advance the RTOS tick counter. This value controls time-out delays for things like queue's and semaphore's. It is also used to set the time-slice for the round-robin scheduler. In this port, it is set to 1ms. For the next version of this implementation, it will be set to 10ms.
2. **PendSV_Handler**: This handler is written in assembly language and appears to be the context switch function for the RTOS.
3. **SVC_Handler**: SVC 0 is used to start the first task. It does not appear that any other SVC's are used/available.
4. **WDG_IRQHandler**: This is the watchdog timer. It is used to generate a general interrupt (not a watchdog timer interrupt). It is used in example 12, 13, and 14. It is the first interrupt past the standard Cortex-M3 system interrupts. That's the only reason this interrupt vector was selected.

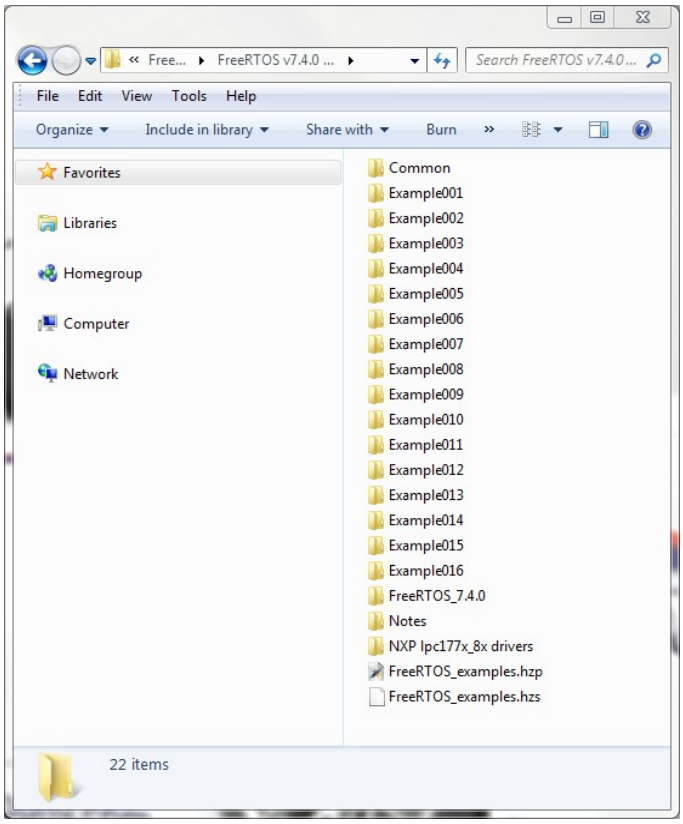
I have looked in current versions of both the *Using the FreeRTOS Real Time Kernel* and *FreeRTOS Reference Manual*. There is no description of the internals of FreeRTOS, so my notes above are open to interpretation. However, there is a reference to these three interrupts in *Using the FreeRTOS Real Time Kernel*. On page 5, it states: *FreeRTOS makes use of the SysTick, PendSV, and SVC interrupts. These interrupts are not available for use by the application.*

There were a number of changes to the example applications:

1. I had to delete references to a TI Stellaris BSP that was not in the system.
2. I modified basic_io.c to use the CrossWorks v2.3.1 debug_print() function. Incidentally, the <cross_studio_io.h> I/O system is a very comprehensive facility that works on file input and output as well as console I/O.
3. I added a directory called Common that contains common files for all of the examples. The only files that are in the example directories are files that are unique to the example.
4. I have added an “example print” it's clear which example is running.
5. SystemInit() is called in LPC177x_8x_Startup.s. This function sets the processor clock to be the PLL and configures the PLL to 120MHz.
6. After configuration for 120MHz, the function SystemCoreClockUpdate() is called. This function reads the PLL clocking registers and updates *SystemCoreClock* with the real speed of the core clock.

**ZIP File Structure**

Here is the directory tree of the zipped file:




At the top is the common directory, in the middle are the FreeRTOS examples. At the bottom is the RTOS itself, notes (home for this document), the NX lpc177x_8x peripheral drivers and the CrossWorks project files.

The examples are arranged just as they are in the book: Example001 to Example016.

**Compiling / Running the Examples**

1. Make sure that the HY-LPC1788-SDK board+core boards are properly assembled, connected, and that the JTAG/SWD drivers are installed. I use a JLINK in my system. Note: on the HY-LPC1788-SDK platform SWD is the only debug mode that operates correctly. *JTAG does not work.*

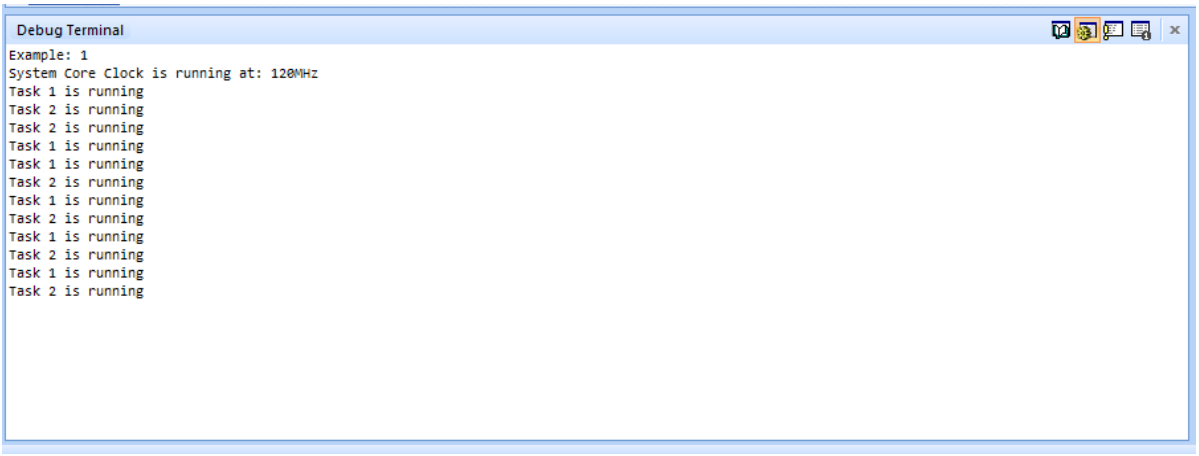
2. Make sure that Rowley Associates CrossWorks v2.3.1 is installed and configured as well. Specifically, I use tabs (language settings for C/C++), tabs are tabs (not spaces) and indent and tabsize are set to 4.

3. The easiest way to start the IDE is to double click on FreeRTOS_examples.hzp. This opens the Rowley solution workspace.

4. Select the example you'd like to run: Double click on the project icon, or right click on the project and click the "Set as Active Project" entry.

5. Compile the example: One way is to right click on the project. Then select Debug → Start Debugging. This will compile the example (if it's out of date) and begin debugging. Another way locate the project explorer cursor on the solution workspace entry (Solution 'FreeRTOS examples') then select the Main Menu → Build → Rebuild Solution to build everything.

6. Most of the output from the examples is shown in the CrossWorks Debug Terminal window. It looks like this:



**Bibliography**

[1] The FreeRTOS Reference Manual
Version 1.2.1
Copyright Real Time Engineering 2011
This book is available for 30USD from http://shop.freertos.org
Don't hesitate: If you are going to use FreeRTOS; buy this book.

[2] Using the FreeRTOS Real Time Kernel – A Practical Guide
Third Edition
Copyright Real Time Engineering 2011
This book is available for 35USD from http://shop.freertos.org
If you are going to use FreeRTOS, and this is the first time you've used an RTOS, buy this book.

**Bill of Materials**

[1] LPC1788 system: Haoyu Electronics HY-LPC1788-Core / HY-LPC1788-SDK
This is a direct link to the development platform. This particular version has the 7in LCD. It can be purchased for $170. To me this is an amazing price for this system.
http://www.hotmcu.com/hylpc1788-development-board-with-7-touch-screen-tft-lcd-p-39.html?cPath=1_21

[2] IDE: Rowley & Associates CrossWorks for ARM v2.3.1
This is a direct link to the IDE vendor Rowley & Associates. CrossWorksv2.3.1 is an excellent platform. The Personal License version is $150. Another advantage of this platform is that it runs on Windows, Linux, and Mac OSX.
http://www.rowley.co.uk/

[3] Debugger: Segger JLINK is the classic system for JTAG/SWD debugging. More, it can be used by nearly any IDE (Keil MDK, IAR EWARM, and of course CrossWorks). It has excellent tools that can be used with debuggers like GDB. There are a lot of debuggers out there, but this is the only one to own.
http://shop-us.segger.com/J_Link_EDU_p/8.08.90.htm

This paper was produced with LibreOffice v3.5.4.2. It is an excellent tool.