Notes

Fr: A.C. Verbeck
To: File
Re: FreeRTOS example notes
Date: December 17, 2012

Introduction
Over the last three or four months, I have been casting about to find the right embedded development solution. I have several personal projects I want to do and I do not want to own a GCC / GDB / IDE port or have to pay 5000USD to get a reasonable development platform.

Part of my search was to purchase *Using the FreeRTOS Real Time Kernel - A Practical Guide* 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 much more.

Rowley Associates have a similar platform based on gcc that costs 1500USD for the complete professional version. The single-user home-enthusiast version is a very reasonable 150USD. I have been using this system for about three months now and I find it as good as the Keil offering.

After reviewing Cortex M3 and M4 parts from a number of manufacturers including NXP, ST, TI and FreeScale, I settled on the ST STM32F407VG. This part is based on a Cortex M4F core. It has an excellent set of peripherals, 1M flash, 192Kb RAM. ST have an evaluation board, the STM32F4-discovery, that has a number of interesting peripherals like a MEMs microphone, Cirrus audio DAC, MEMs motion sensor, and more. I was fortunate enough to acquire one of these boards without charge. But even the list price is a reasonable 14.90USD at DigiKey.

After extensive searching, I was not able to locate a port of the latest version of FreeRTOS for the STM32F407VG. In addition, the book uses version 6.1 of FreeRTOS, a version of the RTOS that is far out of date. I determined that I would have to port the examples to the latest version of FreeRTOS, the latest version of the CrossWorks IDE, and finally the STM32F407VG on an STM32F4-discovery board. The examples (Example001 to Example016) are now targeted to the following:

Tool/System	Manufacturer	Version
IDE	Rowley Associates	CrossWorks v2.3
RTOS	Real Time Engineers Ltd.	FreeRTOS v7.3.0
Processor	STMicroelectronics	STM32F407VG
Board	STMicroelectronics	STM32F4-discovery

FreeRTOS v7.3.0 port notes

There is no prvSetupHardware() function in this FreeRTOS port. Normally, this function is used to configure the hardware prior to starting the RTOS. These examples depend on the reset state of the processor. The examples assumes that the CPU is running at 16MHz – the frequency of the HSI clock.

```
#define configCPU_CLOCK_HZ          ( 16000000 )
#define configTICK_RATE_HZ          ( ( portTickType ) 1000 )
```

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.*"

1. The default processor clock on an STM32F407VG is the HSI (high speed internal clock). This clock runs at ~16MHz.
2. SysTick (or Cortex System timer) has two potential clock sources:
    1. RM0090 (STM32F4xx Reference Manual, Figure 13. Clock tree) implies that SYSCLK is divided by 8 and sent to the Cortex System timer (SysTick). This would make the SysTick clock 2MHz.
    2. PM0214 (STM32F4xx programmers manual) notes that there are really two sources for SysTick and the default is to run at the AHB clock rate. Note, this information is inexplicably not in RM0090 clock tree.
3. The config define `configTICK_RATE_HZ` defaults to 1000. This is far too fast for a useful implementation. To keep things simple, I did not modify this. A better value would be either 100 or even 50.

In summary:

- The processor frequency for this port is 16MHz.
- The SysTick frequency for this port is also 16MHz.
- The TICK_RATE frequency is 1kHz.

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 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.

My intention is to create a SystemInit() / prvSetupHardware() that does the following:

- Sets the processor frequency to 160MHz.
- Sets the SysTick frequency to 20MHz.
- Sets the TICK_RATE frequency to 100Hz.

Once available, I will update the examples and post it.

CrossWorks v2.3 Configuration Notes

The infamous CrossWorks pre-processor define STARTUP_FROM_RESET is set. If you haven't read the header in STM32_Startup.s, here's a copy:

```
STARTUP_FROM_RESET

  If defined, the program will startup from power-on/reset. If not defined
  the program will just loop endlessly from power-on/reset.

  This definition is not defined by default on this target because the
  debugger is unable to reset this target and maintain control of it over the
  JTAG interface. The advantage of doing this is that it allows the debugger
  to reset the CPU and run programs from a known reset CPU state on each run.
  It also acts as a safety net if you accidently download a program in FLASH
  that crashes and prevents the debugger from taking control over JTAG
  rendering the target unusable over JTAG. The obvious disadvantage of doing
  this is that your application will not startup without the debugger.

  We advise that on this target you keep STARTUP_FROM_RESET undefined whilst
  you are developing and only define STARTUP_FROM_RESET when development is
  complete.
```

What this means is that the application will run from power-up / reset without the debugger connected. The default loops at start-up and requires the debugger to load / run an application. To me this was counter intuitive. I would expect that unless I added a define to lock the processor at start-up that the application would run. Once I discovered what Rowley was doing, I was impressed. I have created defects that halt the system clock. Once halted the JTAG port cannot restart the processor.

The define USE_STDPERIPH_DRIVER is also defined. This selects the type of register accesses will be used in the application: direct register access or API. Here's the note from stm32f4xx.h

```
The file is the unique include file that the application programmer
is using in the C source code, usually in main.c. This file contains:
 - Configuration section that allows to select:
    - The device used in the target application
    - To use or not the peripheral's drivers in application code(i.e.
      code will be based on direct access to peripheral's registers
      rather than drivers API), this option is controlled by
      "#define USE_STDPERIPH_DRIVER"
    - To change few application-specific parameters such as the HSE
      crystal frequency
```

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   WWDG_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. **WWDG_IRQHandler**: This is the windowed 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-M4 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.*

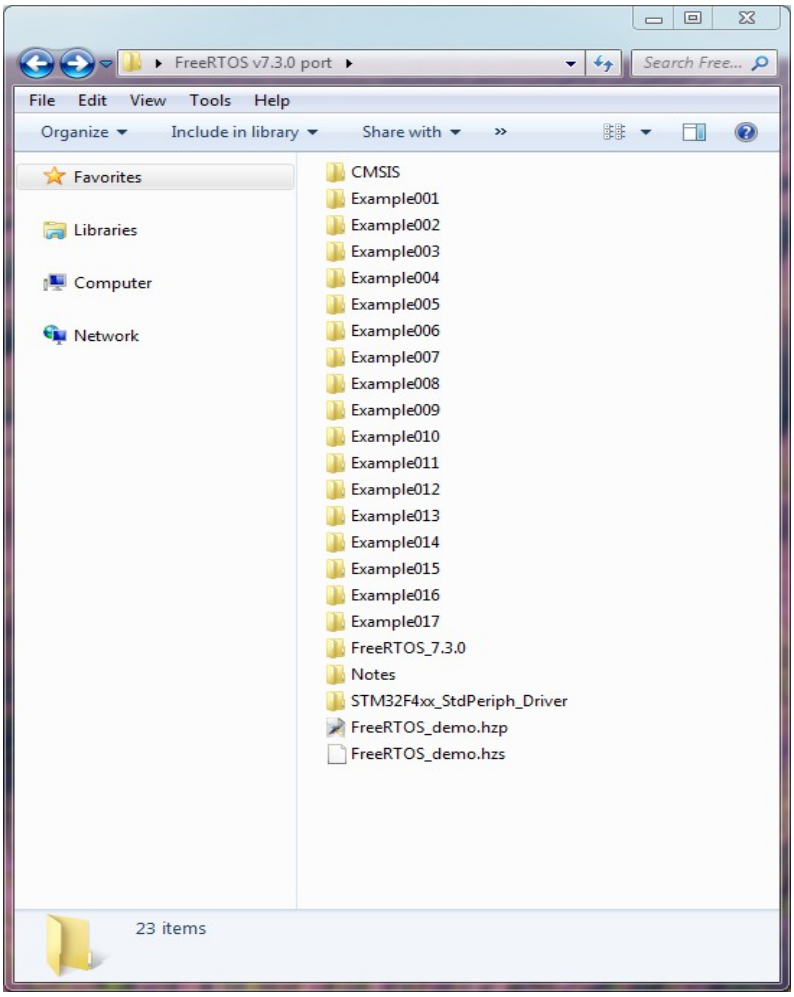
There were two other changes to the example applications:

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 debug print processing.

Otherwise the applications are exactly as they are published for FreeRTOS v6.1.0.

ZIP File Structure

Here is the directory tree of the zipped file:



At the top is the CMSIS library, in the middle are the FreeRTOS examples. At the bottom is the RTOS itself, notes (home for this document), the STM32F4xx peripheral drivers and the CrossWorks project files.

The examples are arranged just as they are in the book: Example001 to Example016. Example017 is the Finn Bindeballe's FreeRTOS v7.1.1 example application.

Note that CMSIS, FreeRTOS 7.3.0 and the STM32F4xx_StdPeriph_Driver directories have been trimmed to minimal required files. You should consider acquiring these directories independently. The CMSIS library (for example) has the floating point and DSP libraries. A link to this download is in the bibliography.

Compiling / Running the Examples

1. Make sure that the STM32F4-discovery board is plugged in and that the STLINK drivers are installed. See the bibliography for a link to the STLINK drivers.

2. Make sure that Rowley Associates CrossWorksv2.3 is installed and configured as well.

3. The easiest way to start the IDE is to double click on FreeRTOS_demo.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.

Bibliography

[1] RM0090 Reference manual, STM32F40x, STM32F41x, STM32F42x, STM32F43x advanced ARM-based 32-bit MCUs, Doc ID 018909 Rev 3, STMicroelectronics
http://www.st.com/internet/com/TECHNICAL_RESOURCES/TECHNICAL_LITERATURE/REFERENCE_MANUAL/DM00031020.pdf

[2] PM0214 Programming manual, STM32F3xxx and STM32F4xxx Cortex-M4 programming manual, Doc ID 022708 Rev 3, STMicroelectronics
http://www.st.com/internet/com/TECHNICAL_RESOURCES/TECHNICAL_LITERATURE/PROGRAMMING_MANUAL/DM00046982.pdf

[3] STM32F4DISCOVERY Schematic
Number: MB997 Rev: B.2, STMicroelectronics
http://www.st.com/internet/com/TECHNICAL_RESOURCES/TECHNICAL_LITERATURE/USER_MANUAL/DM00039084.pdf

[4] ST-LINK/V2ST-LINK/V2 in-circuit debugger/programmer for STM8 and STM32
http://www.st.com/internet/evalboard/product/251168.jsp
The drivers are at the bottom of the the “Design Support” tab.

[5] STM32F407VG Peripheral Library and CMSIS files
http://www.st.com/internet/mcu/product/252140.jsp
The file is near the bottom of the “Design Support” tab in the “FIRMWARE” section. **Make sure to get the** *STM32F4 DSP and standard peripherals library, including 82 examples for 26 different peripherals and template project for 5 different IDEs*, **not the** *STM32F4DISCOVERY board firmware package, including 22 examples (covering USB Host, audio, MEMS accelerometer and microphone…) and preconfigured projects for 4 different IDEs*. **The STM32F4DISCOVERY firmware is out of date.**

[6] 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.

[7] 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.

This paper was written with LibreOffice Writer. It is an excellent tool.