Programming Embedded Systems 2014/ JB

Exercise 5 / 12/14.2.2014 / Deadline for submitting report 28.2.2014

Return report electronically on address: https://abacus.abo.fi/ro.nsf. If you do not have an ÅA account, please email report to Åke Syysloiste <agustavs@abo.fi>.

Advisor/labs: Åke Syysloiste. Åke will be available during lab hours, at other times he can be found in room A5031.

This time we will work on two (2) alternative implementations of embedded systems.

1) Texas Instruments TivaC development card + homebrew daughtercard.
2) RaspberryPi

For each device, we will do an exercise.

TivaC + daughtercard

The TI TivaC can be programmed by standard TI tools, but here we instead use the port of the Arduino-based developing environment for TI. This is called Energia, end is found on the web-site energia.nu.

The daughterboard has the following components: pushbuttons, leds, NTC-resistor and and 3-axis accelerometer. In this exercise, we will use the accelerometer.

Task 1

Construct software for reading the accelerometer. In **normal conditions** (acceleration 9,81 m/s^2), **green led** is lit. In **high acceleration** (> 12 m/s^2) also the **yellow led** is lighted. In even higher, also the red led is on.

The accelerometer measures acceleration in 3 axis (x,y,z), and is read using the ADC, using pins (A9, A8, A10). The leds (green, yellow, red) are accessed through the pins (PB2, PE0, PB7). The accelerometer measures the the force affecting a mass, so that if the force is 0, the analog output if half of the input voltage to the sensor. The total acceleration vector affecting the mass is hence given by k* sqrt((x-x_0)^2 + (y-y_0)^2 + (z-z_0^2)), where k is a gain constant and (x_0, y_0, z_0) describes the value of respective axes when no force is on the mass in that direction. Hence, you must try to make estimates of k, x_0, y_0 and z_0.

The Energia software is using to basic software function: `setup()` where all setup should be made and `loop()` which is called iteratively. No `main()` function is provided in this system by the programmer, but that is included in the Energia-environment. Debug information is easy to output using the inbuilt serial port over USB.

Tasks 2

1. Build and install the default Demo application available in the FreeRTOS port for RaspberryPi. The defualt Demo program starts two tasks, which blinks the status led on the RaspberryPI.

2. Modify and rebuild the demo application.
 a) Create a system where pressing a button lights up a led
 b) A system that counts button presses since start, and shows the count by repeatingly flashing the led the number of times buttons where pressed

Equipment

 * RaspBerryPI + SDCard + Power supply
 * Daughter-boad (3 push buttons, 3 leds, TTL Serial port - USB)
 * SD-card reader
 * Linux computer

Guidelines

1**. A Windows / Linux computer with a SD-card reader. These instructions assumes that you have a**

 * Linux computer with "make" and "python" installed
 * SD-card readers. The computers in the lab does not have a SD-card readers, but there are a few spare in the lab.

Note that the machines in the laboratory are dual-boot configured. To run Linux, restart the computer and select Linux on bootup.
If you need to install tools, ask Åke for root password.

2. FreeRTOS for Raspberry PI

The FreeRTOS port is not yet a official one, but one that was posted on GIT. Hence, the building of it is not "out of the box", but needs some modification. The port of FreeRTOS for RaspberryPI is found here:
https://github.com/jameswalmsley/RaspberryPi-FreeRTOS

Make a own directory in the home folder

$HOME->$MYFOLDER->pi (e.g. /home/labuser/group_xyz/pi)

Get the source, using git
% git clone https://github.com/jameswalmsley/RaspberryPi-FreeRTOS

3. Toolchain for ARM (for compiling on Linux)
https://launchpad.net/gcc-arm-embedded/

A toolchain mean that you have tools for building a executable file/image/set of files for a certain architecture. As we are targeting the RaspberryPI, we need a toolchain for building code (i.e. array of bytes) that the Raspberry PI can use. The Raspberry PI's "heart" is a Broadcom BCM2835 SoC (System on Chip), i.e. it has several devices placed on a single piece of silicon. The most important part from our point of view is the CPU, a ARM1176JZF 700 MHz processor. So our toolchain should hence produce executables/images that the ARM can use.
The toolchain includes a cross-compiler, i.e. a compiler that works on one architecture, but produces executables for another architecture. The toolchain above is a pre-built, i.e. the toolchain itself need not to be compiled from source, but is already compiled, and directly usable. Please make sure that you get the pre-built toolchain that works on your architecture.

Install the toolchain in directory
$HOME/$MYFOLDER/

To make the Demo application, to to the FreeRTOS folder, subfolder "Demo" and issue
% make

This will not work out of the box, but paths (to compiler toolchain and libraries) must be changed. Ask Åke for help with this if needed.

4. How to make the RaspberryPI use our code (the FreeRTOS we built)?

Each device (PC/laptop/router/ipod/VOIP phone etc.) uses their own WELL DEFINED way to start running software on them. This process is called the booting process. The booting process is a combination of hardware / media / firmware / boot loaders.

At the very bottom is the hardware itself. The Microchip PIC in the lab just started from memory address 0 when it got "power good" reset. We have to make sure that some working executables are in that position when it get it reset from "power good". Otherwise the CPU just do nonsense, and eventually get into some state where it is frozen. In bootlader enabled device (like the Modtronix in the lab), there was a boot loader at position 0, which in turn checked if any other device tried to contact to upload the work software. If no contact, the boot loader resumed to the "work software".

In the PC architecture, the CPU is configured to start executing code on physical memory address (0xffff0) after a power-up/reset, and let us hope that there are some working instructions at that memory location. If the hardware is there (ROM, Read Only Memory, containing the BIOS), the computer does a BIOS startup procedure, which leads to the standard next step: find a boot device, load the boot sector from that, and jump to the code in the boot sector. If the boot sector is configured OK, the normal boot/loading of the operating system resumes.

How about Raspberry PI? How does a Raspberry PI boot?
Here is the process

- When the Raspberry Pi is first turned on, the ARM core is off, and the GPU core is on. At this point the SDRAM is disabled.
- The GPU starts executing the first stage bootloader, which is stored in ROM on the SoC. The first stage bootloader reads the SD card, and loads the second stage bootloader (bootcode.bin) into the L2 cache, and runs it.
- bootcode.bin enables SDRAM, and reads the third stage bootloader (loader.bin) from the SD card into RAM, and runs it.
- loader.bin reads the GPU firmware (start.elf)
- start.elf reads "config.txt", where the SoC can be configured (CPU/CPU core frequencies, video encoding liceses etc)
- start.elf (in the GPU) reads kernel.img (which normally is a Linux kernel), and after that releases an ARM reset, which makes the ARM CPU start execution on physical address 0x0.

Hence, in order to make the ARM CPU execute the FreeRTOS system, we must create a file "kernel.img" with bytecode corresponding to what the ARM CPU can use. Fortunately, this is automated in the FreeRTOS port "make" system, the result of issuing the "make" command when in the FreeRTOS directory is a file "kernel.img". The easiest way to make the RaspberryPI run the FreeRTOS is using a standard Raspberry Pi installation, where the "kernel.img" file is replaced with the one from the FreeRTOS build system.
Hence

1. Use a standard tool for configuring a SD card for Raspberry PI. The easiest way is to load the raspbmc installer for windows, available on http://www.raspbmc.com/download/. Download the installer, run the installer and install.

2. Copy the file "kernel.img" from the FreeRTOS directory, and put it on the root directory on the SD card.

3. Install the SD card in Raspberry PI, attach power, and the FreeRTOS should be running
How do we know that it is running? Use the status led (green one next to the ethernet status led on the board), that is available on GPIO pin 16.