Inferno in Embedded Space: Porting to the Sun SPOT

Brian L. Stuart

ABSTRACT

We report, here, on a project to port the Inferno operating system to the Sun SPOT sensor platform. The port provides support for the temperature and light sensors, the three-axis accelerameter, LEDs, momentary contact switches, and GPIO pins. Communication among sensors is provided by an IPv6 implementation running over the integrated 802.15.4 radios.

1. Introduction

In 2004, Sun Labs developed a small sensor platform for research into sensor networks. The platform is called the Small Programmable Object Technology, or SPOT. Sun designed the system to be suitable to run a small implementation of the Java Virtual Machine, called Squawk. Naturally, we cannot help but see any platform targetted to Java as begging for a port of Inferno.

Despite some pretty severe resource constraints described in the next section, the results are promising.

2. Sun SPOT Hardware

The SPOT's main processor is a 180MHz Atmel AT91RM9200 ARM9 SOC with 16KB of instruction cache and 16KB of data cache. This SOC also contains an additional 16KB of internal RAM that can be used before the memory controller is initialized. External memory in the SPOT is provided by a single package combined flash/RAM device containing 4MB of flash and 512KB of RAM. Both memory types have the same read access time, allowing code to be executed directly from flash with no performance penalty. In addtion to the main processor, memory, USB interface, and power control, an eSPOT main board includes an 802.15.4 radio. The radio uses the same unlicensed 2.6GHz spectrum as 802.11 and 802.15.1 Bluetooth. It supports low data rate connections operating at 250Kb/S.

A module consisting of only an eSPOT main board is referred to as a base station. A SPOT development kit also contains two modules that include not only the main board but a battery and a sensor board called the eDemo board. The eDemo board includes eight RGB LEDs, a three-axis accelerometer, a temperature sensor, a light sensor, two momentary contact switches, and a set of analog and digital I/O pins.

Unlike most small system designs, the devices on the eDemo board are not directly addressable by the main ARM processor. Instead, the eDemo board has a small processor that directly accesses the I/O devices and communicates with the main processor via SPI. This, of course, reduces the number pins necessare on the connector between the two boards and also has the effect of making the driver software quite regular and straightforward.

3. Sun SPOT Inferno

As a starting point for the port, we took the files clock.c, dat.h, fns.h, fpi.h, fpiarm.c, l.s, main.c, and trap.c from the os/cerf1110 and os/sa1110 directories. For the files dat.h, fns.h, fpi.h, fpiarm.c, and main.c, the changes ranged from none to minimal. Differences in details of the SOC and of the execution environment in the SPOT required substantial changes to l.s and trap.c Although os/sa1110/clock.c was used as starting point, it ended up being more of a model as it was nearly rewritten from scratch, because the clock design of the AT91RM9200 is so different from that of the sa1110. Several new files were written from scratch, including archspot.c, devchipcon.c, devsunspot.c, and lrpanmedium.c. We discuss these files more in the following sections.

3.1. SPOT Execution Environment

Although the SPOT was developed with the intention of being used with a Java Virtual Machine, Sun did produce some documentation that contained the information necessary to run other bare-metal code. Probably the single most useful document in this regard is an 11-page application note titled "The Sun SPOT bootloader."[1] Unfortunately, this document is not included on the CD in the development kit and no longer appears to be available online. Similarly, the source code for the bootloader and the schematics for the boards used to be available through the SPOT forums that seem to have disappeared as part of the transition into Oracle.

At reset, the AT91RM9200 begins executing code at location 0, which prior to any memory remapping, is the beginning of the flash memory area. The end result of the early initialization is that the bootloader is copied into the internal RAM and memory is remapped such that internal memory is at address 0, flash is at address 0x10000000, and external RAM is at address 0x20000000. Once this environment is set up and control is transferred to the copy in internal RAM, the bootloader first waits a few seconds to see if attention is gained over the USB port. If it is, then control is transferred to a flash loader that provides a limited monitor, mostly useful for reflashing the device. If nothing arrives on the USB port during those few seconds, the bootloader looks at a page of configuration values in flash to find the location of the "VM" image. Althought the details of the configuration page and flash loader vary some among different versions, the cases we've seen all place the "VM" image at location 0x10010000. The first three words of the image are interpreted as follows (quoting from the bootloader application note):

- word 0: the address (in external RAM) of the top of the data and program sections that should execute in external RAM
- word 1: the address of the top of uninitialised data (BSS) in external RAM.
- word 2: the program's start address for execution.

This header and starting location are sufficient to modify 5l to produce a Sun SPOT image format. Because the flash memory is located in the processor's address space and because it operates as fast as external RAM, the Inferno kernel text space is run directly from flash rather than being transferred to RAM on boot. Of course, the data segment is copied to RAM

at boot time. Because the SPOT boot loader expects the part of the image to be copied to RAM precedes what is run out of flash, 5l also had to be modified to output the data and text segments in the opposite of the usual order.

Once the boot loader starts an image, it remains resident in the SOC's internal RAM and provides some basic services via the software interrupt instruction. Routines are included in l.s that use these software interrupt services for doing console I/O through the USB interface. These routines are in turn called by code in archspot.c that implements the underlying queue handling expected by devcons. Archspot.c also contains the SPI routines that are used to communicate with the eDemo board.

3.2. Memory Usage

As might be expected, the 512KB RAM space is a fairly limiting resource constraint, particularly when using TCP/IP. Keeping the kernel in flash and running it from there is the first step toward fitting Inferno in such limited resources. The next step in managing memory consumption involves reducing the sizes of tables used in the TCP/IP stack, especially the ARP table.

Overhead in memory allocation can also put pressure on limited memory resources. The standard Inferno pool allocator uses a 32-byte quantum and imposes 12 bytes of overhead in every allocation. Even with numerous very small accesses, these parameters work well when memory is measured in tens of megabytes or more. However, when everything must fit in half a megabyte, reducing internal fragmentation is very helpful. To that end, we implemented a more simplistic memory allocator with four bytes of overhead, a quantum of four bytes, and an eight-byte minimum allocation. Unfortunately, because of dependencies on the specifics of the pool allocator in libinterp, doing so is not quite as easy as simply replacing alloc.c with one with simpler malloc and free calls. Nevertheless, as long as a select set of definitions in pool.h are retained or emulated, we can get away with only having to recompile the libraries.

3.3. Devsunspot

Access to the sensors and other devices of the Sun SPOT is provided through the devsunspot driver. This driver serves the following seven files:

accel This file provides access to the three-axis accelerometer. Reads from accel return three values giving the acceleration in the x, y, and z directions, calibrated in hundreths of a G. Writes to accel are used to set the scale of the accelerometer. Valid values are "2" and "6".

leds Access to the eight RGB LEDs on the eDemo board is provided through the leds file. When accessed through devsunspot, each of the three colors is treated as an independent LED with LEDs 0–7 being blue, LEDs 8–15 being green, and LEDs 16–23 being red. Each LED can be illuminated at a brightness level specified from 0 to 255 with 0 being off and 255 being the brighest illumination. Writes to leds are a pair of numbers. The first number in the pair is the LED number to set, and the second value is the brightness level. Reads from leds return the 24 brightness levels.

light Reads from the lights file return the raw A/D converter output. Each unit of the raw output corresponds to approximately 2 lux.

pins At present, the pins files only support reading the values of eight GPIO pins on the eDemo board. The low-order two pins correspond to the two momentary contact switches on the board. For a switch that is depressed, the bit value is 0, and for a switch that is released, the bit value is 1.

power Writes of "on" and "off" set the power state of the eDEMO board. Reads from power return a pair of numbers with the first being the state of the eDEMO board power. The second number is reserved for a partially implemented deep sleep mode.

spotled The spotled file provides access to a red/green indicator LED on the main eSPOT board. Reads from the file return the hex representation of the GPIO register driving the LEDs. The red and green LEDs are on bits 23 and 24 of the register. Writes set the state of the LEDs with messages of the form "red on" and "green off".

temp Reads from the temp file return a measurement of the internal temperature of the eDEMO board in units of hundreths of a degree C.

3.4. Devchipcon

Unlike many of the 802.11 devices on the market, the CC2420 802.15.4 radio chip is well-documented enough that it's possible to write a driver for it from scratch[2]. As with the devices on the eDemo board, communication with the chipcon radio is over the SPI interface. However, its programming model is much like many other simple communications devices. Configuration and control are handled through a number of on-chip registers, as is transmit initiation. Receives generate interrupts. Transmit and receive data are buffered in on-chip FIFOs. The SPI interface is used to transfer data into and out of the registers and FIFOs.

In the larger context, the devchipcon driver is implemented as a netif instance. The only control messages recognized by devchipcon are "poweron" and "poweroff" which handle the chip power management.

3.5. Lrpanmedium

RFC 4944 specifies an encapsulation of IPv6 packets on 802.15.4[3]. The bulk of the RFC is concerned with fragmenting and reassembling packets, because the MTU for IPv6 over 802.15.4 is 1280 bytes but the raw frame size of the 802.15.4 physical layer is 127 bytes. The details of this encapsulation are implemented as a network medium type in the source file lrpanmedium.c.

The implementation of RFC 4944 here is, however, incomplete. First, the RFC specifies some elements for mesh networking, and those elements are not yet implemented. More serious, however, is a simplifying assumption made in the receiver. In particular, it assumes that all fragments from a single IPv6 packet arrive contiguously. The fragments don't have to be in order, but the initial implementation doesn't maintain multiple partial packets concurrently. Because the bulk of the development testing has been with only a pair of units communicating at any given time, these limitations have not been major during development.

4. Results

Several informal experiments have been conducted examining different architectures for sensor monitoring. There is, of course, an elegance to an architecture where various sensors are

imported by a central processing node. Beyond that, minimizing processing in sensor nodes helps to conserve battery charge. However, we have found that in practice, the limitations of the 802.15.4 radios are not suited to that type of operation. Although a 250Kb/S transmission rate is itself sufficient to poll a significant number of devices at a useful period, there is enough overhead that it doesn't take too many devices before the network becomes a bottleneck.

More fundamentally, the resource importation model doesn't fit well with the operational mode we would like to use for power management. In particular, we normally would like to have a battery operated sensor to spend most of its time in a deep sleep, being woken up periodically to take readings. This is somewhat at odds with polling by a central processing node that imported the sensor devices. For power management, we would prefer a push model where each sensor sends only what it needs when it needs.

Therefore, we have found that we are generally better off expending some processing resources at each node to minimize network traffic and to better fit a push model of data collection. Thus an architecture where each sensor wakes up, dials the central system, transmits updates, and goes back to sleep works quite well. However, these results suggest that in an environment where the remote devices are effectors rather than sensors, a model based on importing the effector resources would work quite well.

Overall, this work has demonstrated the feasability of an embedded Inferno running with as little as a half mega-byte of RAM. Althouth the SunSPOT has 4MB of flash, we only use a little over half a mega-byte of it for the Inferno kernel and root file system. However, this is still a somewhat larger footprint than available in many embedded microcontrollers. At the same time, the work on reducing Inferno's memory footprint here is somewhat cursory. A good next target for an embedded port would be the PIC32MX795F512 from Microchip[4]. This is a single chip device including a MIPS CPU, 512KB of flash and 128KB of RAM as well as USB, CAN, and Ethernet controllers. At the present time, these devices cost less than $10 in single-unit quantities. Atmel offers a similar device, but with only 64KB of RAM. Beyond that, a system ported from, or inspired by, Inferno for the widely-used 8-bit microcontrollers would be a interesting avenue of research.

5. References

[1] Cleal, D., and Daniels, J., "The Sun SPOT bootloader," Sun Microsystems, 2007.

[2] "CC2420 2.4 GHz IEEE 802.15.4/ZigBee-ready RF Tranceiver," Chipcon AS SmartRF CC2420 Preliminary Datasheet (rev 1.2), 2004.

[3] Montenegro, G., et al, "Transmission of IPv6 Packets over IEEE 802.15.4 Networks," Network Working Group RFC 4944, 2007.

[4] "32-bit Microcontrollers (up to 512 KB Flash and 128 KB SRAM) with Graphics Interface, USB, CAN, and Ethernet," Microchip PIC32MX5XX/6XX/7XX Datasheet, 2009–2013.