Final Report: Remote Water Level Monitoring: A Prototype of a Wireless Sensor Network for Water Monitoring Application

Dhananjay Palshikar

May 12, 2012

Abstract

For countries such as India, water is a precious resource; one whose source is often only the unreliable four-month season monsoon after the summers. This project would try to provide a solution to the problem of monitoring the water storage tanks by developing a prototype wireless sensor network that could monitor water levels of the water storage tanks. Fennec Fox [1] was decided to be used as the communication protocol stack for the project. For its low cost, TI-EZ430RF2500 [2] has been decided to be used as the sensor mote for this project. This document will focus on the progress made towards porting TinyOs [3] to the TI-EZ430RF2500 development tool. TinyOs serves like an operating system for Fennec Fox network protocol stack, which in itself is independent from any implementation. For more details on the specification of application and the over all research plan please refer to [4].

This document serves as a Report as well as a Technical documentation for the progress made in the project. The section on Porting TI-EZ430RF2500 has been in the form of a guide, elaborating the sequence of operations right from the installation of tools to the changes made to the Tiny-OS system for future reference.

1 Introduction

Fennec Fox network communication protocol stack would be used for this project [5]. Although the concept of the Fennec Fox network protocol stack is independent from any implementation, the current version requires knowledge of nesC [6] and TinyOS [3]. For its low cost, Texas Instruments EZ430-RF2500 [2] development tool was chosen for the development of the project. Despite its low cost, TinyOS has not been ported to the EZ430-RF2500 platform. Thus, the first challenge is to port TinyOS to the EZ430-RF2500 platform. As of writing this document porting Tiny OS to EZ430-RF2500 is still in progress, *section*3, gives detailed account of the same. The next section would talk about the progress made as per the proposed research plan [4].

2 Fennec Fox Framework

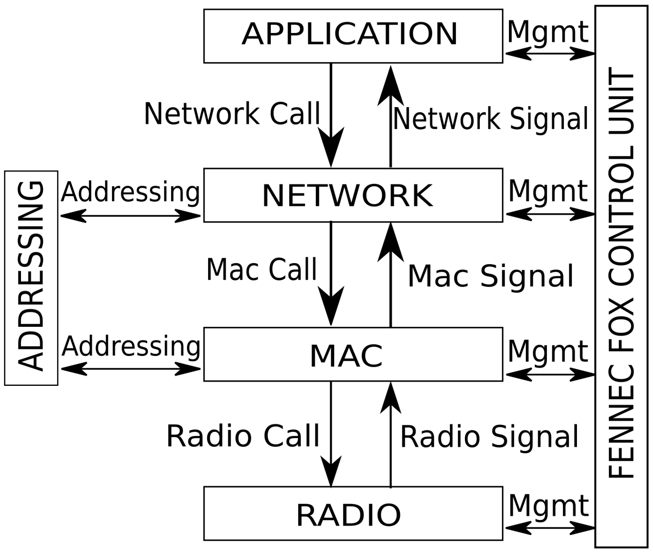

Figure 1: Current Fennec Fox Architecture

3 Summary of Milestones

The proposed research plan had targeted building a complete prototype based on Fennec Fox and using EZ430RF2500 wireless sensor. However, porting TinyOs to the EZ430RF2500 wireless development tool was not trivial, thus as of now the Porting TinyOs to the EZ430RF2500 is still in progress. The milestones relating to development of application in the setup that would then be built on EZ430RF2500 platform have not been worked on.

- **Literature Survey & Environment Setup** TinyOs tutorials were elaborate and easy to grasp. Although, most of the tutorials for are meant for application developers, the basic tutorials are a good read for the for tinyOs developers. Apart from the official tutorials on the tinyOs website, there are several high quality tutorials that can be found with rather simple searches on the Internet. All the tutorials/documentation that was referred to has been cited in the section on Porting EZ430RF2500 to TinyOs in this document. The environment setup has been documented in the section 4.3 of this document. It should be mentioned that Marcin Szczodrak's support setting up the environment was very useful.

- **Basic TinyOs support for Msp430f2274 microcontroller** Initially development was focused on developing the port without using an existing unofficial incomplete port for msp430f2274 controller [7]. The basic Makefile compilation setup along with some basic interfaces was created.
  While creating the setup it was realized that even an incomplete port for for TinyOs would save significant time. Thus the public repository [7] was used as base for Msp430f2274 development. The repository code could compile the basic applications for EZ430RF2500 without any issues. It had an incomplete support for buses (UART,SPI), it did not support the CC2500 radio.

- **Interface to New Radio RF2500** The natural next step was to interface the CC2500 radio chip with the microcontroller, although a modified version of CC2500 radio interface compiles with the tool chain, no radio activity is noticed. There can several reasons for no radio activity, they have been detailed in the section 4.3 *Future Development Ideas*. The CC2500 radio stack reached its present stage via detour to different platform called *tmote2500*, this has been explained below.

  - **Porting Blaze Radio CC2500 to EZ430RF2500** CC2500 radio stack was available as an unofficial implementation user contributed implementation [8]. Although the CCXX00 radio stack is in its stable release, CC2500 the stack did not compile with the existing release of TinyOs. In the official documentation the mention stack having been tested for a the radio chip CC2420 [9]. After efforts to change the radio stack implementation to make it suitable for EZ430RF2500 did not fetch results, it was decided first port the CC2500 Blaze Radio Stack to the *tmote2500 platform*. The tmote2500 platform is based on the same family of MSP430 processors and uses the CC2500 radio chip.
  - **Blaze Radio CC2500 to tmote2500** The motive was to first get the CC2500 to compile correctly with tmote2500, which could be then used with EZ430RF2500. Tmote2500 is the platform existing in the TinyOs release which uses a CC2500 radio stack. The details of modifications are given in the section 5.9.

- **Debug Interface -UART(Revised Milestone)** Work on the debug interface was started after no Radio Activity could be could be observed with the EZ430RF2500 motes. EZ430RF2500 -usb progammer could be used to debug application running on the mote via UART interface. The idea was to port the TinyOs Printf Library to EZ430RF2500. The debug interface is not perfectly complete as of now, it does not display the correct charterers to application listening to it.

- **Milestones that were not attempted**
  - Application Specifications
  - Application Development
  - Beta Implementation
  - Final Version

4 Lessons Learned

5 Porting Tiny-OS to EZ430-RF2500

Supporting a new platform translates to implementing the following:

- Microcontroller Specific Features e.g Clocks, Buses(USART,USCI,SPI..) etc..
- Radio Transceiver Interface: Configuration and Control of radio chip.
- Making the radio and microcontroller features accessible from the Hardware Independent Layer.

The above was just an overview,a more detailed explanation follows.

5.1 Introducing TI EZ430-RF2500

EZ430-RF2500 is a wireless development tool by Texas Instruments. Porting TinyOS to EZ430-RF2500 would require us to add support for the microcontroller and the radio chip on board the EZ430-RF2500.

- *MSP430F2274 microcontroller* belongs to the family of MSP430 microcontrollers as that of the popular telosb motes. Although this particular microcontroller is not supported by Tiny OS,being from the same family as telosb made it easier to port Tiny OS to MSP430F2274 microcontroller.

- *CC2500 radio transceiver* The CC2500 is a low-cost 2.4 GHz transceiver designed for very low-power wireless applications [10]. CC2500 is not directly supported by Tiny OS although its predecessor CC2420 radio transceiver is well supported.

A reader familiar with the working, installation and the basics of TinyOS development may jump to section 5.7 of the document.

5.2 Introducing Internals of TinyOS

TinyOS has been built to support a nesC application across a vast range of platforms, which means supporting multiple families of processors and radio transceivers. Also, these processors and radio transceivers could be in different combinations, you can have MSP430F1611 microcontroller with a CC2500 radio transceiver as a platform, the same processors could be connected to a CC2420 radio transceiver on another platform.
The popular slogan by Sun Microsytems for Java programming Language: *'Code once run everywhere'* could be used here. The difference being that unlike Java ,the nesC source code would have to be compiled for each platform.

5.2.1 The Tiny-OS 'notion' of an Operating System

Typical notion of a traditional operating system does not apply to TinyOs. Given the very limited memory and processing power of a microcontroller the Operating System along with the application runs as a single process. The application written in nesC is parsed along with the TinyOS nesC source code to result into a single 'C' file which is then compiled by the microcontroller specific compiler to the controller specific hex code. So, what goes into the microcontroller are just the components that your application refers to directly or indirectly, e.g. if your application does not use any serial communication protocol(USCI, SPI etc.) the compiled TinyOS Image will not include any of these components.

5.3 Supporting a new platform in TinyOS

Attention *From here on, I am assuming familiarity with the basics of TinyOS. If words phrases like 'wiring an interface', 'Hardware Description Layer(HDL)' , 'Interfaces, Modules,Configuration' are not clear, please go through previous sections and these tutorials*[11].

5.4 What TinyOS expects of a Microcontroller

Expectations of TinyOS would be similar to those for the any microcontroller specific C-application. The difference being that in order to be platform independent, the NesC interfaces need to be wired to the microcontroller specific interfaces. Effectively this translates to providing modules for existing TinyOS interfaces. Apart from just looking the TinyOs reference implementation, you could also look at the C-examples meant for the EZ430RF2500 [12]. The code is meant for IAR compiler , so you have to make some changes to IAR style interrupts to get it working with the msp430-gcc compiler. There is simple tutorial for writing interrupts for msp430-gcc here [13].

5.5 Porting TinyOS to MSP430F2274 microcontroller

This section details the processor specific changes that have to be implemented for the system. Pondering over the previous section and going through the MSP430F2274's family manual [14] will reveal the hardware features to lookout for. I have listed how, I went about configuring these features and got them working(or not working!).

5.6 Hardware features

You can relate some features directly with the MSP430F2274 controller, coding at *layer* is referred to in the TinyOS documentation as the HPL(Hardware Presentation Layer). The files names might also carry an 'HPL'prefix.

5.6.1 Assigning Pins & Registers

TinyOs defines internal names for the pins of the microcontroller, to maintain same set of names for all platforms. Each Platform directory has a 'hardware.h 'file which defines the pin assignments. There are as series of C macro calls which assign names to pins by declaring set of function for each pin. In the 'hardware.h' file for EZ430RF2500 the macro of the form $TOSH\_ASSIGN\_PIN(RED\_LED, 1, 0)$, which is defined in the file 'msp430hardware.h' declares functions of the form $TOSH\_SET\_\#\#name\#\#\_PIN()$. The 'Platforms 'section in the TinyOs tutorial [11] talks about it. The control registers for a microcontroller, which point to register address register file of the microcontroller are sometimes redefined. Redefinition is done sometimes to resolve a conflicting, or simply because the developer did not like the name.
While the hardware.h file is responsible for configuring the port directions and accessing pins at a lower level. The GeneralIO interface is implemented by each platform,to provide general Pin and port level IO ,to the the Hardware Independent Layer. It forms a part of the Hardware Presentation Layer. For EZ430RF2500, the GeneralIO has been borrowed from the msp430 family implementation in the $TOSDIR/chips/msp430/pins/Msp430GpioC.nc.

5.6.2 Microprocessor Clocks

Every microprocessor has multiple options of clock. There could be external oscillators, or there could be more than one internally generated clocks. The EZ430RF2500 does not have an onboard external oscillator, thus its clock is generated internally. For a microcontroller the clock needs to be configured before any applications code can start executing. Thus, if we were are coding in C [12] configuration of the clocks would be the first few lines of code that choose to change the default clock settings.
TinyOs does not expect an application developer to worry about clocks, thus, the configuration of clocks has been placed before *event Boot.booted* event in tinyOS returns. A platform that needs to be part of the operations before the *Boot.booted* event return has to implement Init interface, for EZ430RF2500 this implementation can be found in $TOSDIR/chips/msp430-small/timer.

5.6.3 Communication Buses

msp430f2274 microcontroller supports the UART bus protocol, it also supports another bus protocol - SPI. USCI is the generic bus interface implemented by Texas Instruments in the microcontroller. So, for msp430f2274 UART protocol runs over the USCI interface. UART bus is more importance to us as on the EZ430RF2500 ,as the pins for UART reception and transmission are connected to the USB interface. We can then use the USB-EZ430RF2500 to connect and communicate with any hardware/software implementation of a serial device.
Printf library also uses the UART protocol, which could be used as a debug interface. As of now the UART bus for the EZ430RF2500 does not works as expected, i.e. if we listen to the data sent of the UART bus, we receive unrecognized character symbol '?'. A possible idea for getting this to work has been discussed in the section 5.11. An explanation about the modification to the UART interface are given below:

- **Printf-library** The printf functionality for MSP430 microcontrollers in TinyOs exists in two forms. One of them is the standard tinyOS printf library implementation [11] ,which follows the standard practice keeping the library code hardware independent, and only wiring to the hardware dependent modules based on compilation time settings. The other prinft implementation is the PrintUart.h which is specifically meant for MSP430 microcontrollers. The prinfUart() function is directly called from the application without having through the hardware independent layers and the arbitration mechanism of tinyOS.
  The file $TOSDIR/chips/msp430-small/usci/PrintUart was modified for correctly configuring UART. For more on UART settings for msp430f2274 refer to [14]. The UART setting were taken from the example code [12]. Msp430rf2500 has UART state machine which is starts its execution after all the UART registers have been configure. The problem with the printf Library was that the program stopped execution after UART initialization. The testing was done with $TOSDIR../apps/tests/Testprintf.

- **Platform Serial** For supporting UART bus for EZ430RF2500 in tinyOs , in way that maintains the hardware independent nature of the application, the interface SerialActiveMessage was to be wired to UART bus via the intermediate layers of TinyOs. The hierarchy of interfaces to be connected has been documented in the applications here [?]. These are notes that were made during development of the project. It might help to view them as well. The modifications TestSerial application can be found here $TOSDIR/../apps/tests/TestSerial.

- **Uart Byte**

5.7 Adding Support for CC2500 radio transceiver

The CC2500 radio transceiver chip is not supported officially by TinyOs. However there is some support for the CC2500 radio under the name Blaze Radio Stack. Changes were made to the BlazeActiveMessage.nc file, which is the key stone in for radio communication and packet generation for CC2500 radio. In order to avoid the complexity of the Blaze radio stack, all the interfaces in this file were directly wired to their final implementations.

5.7.1 Reference Implementations

A good source for understanding how these wirings are take place is the Tiny OS source code for an already supported platform. In general for Porting EZ430-RF2500, the following platforms were quite useful:

- **telosb** The default TinyOs platform, it is based on the same family of TI msp430 microcontrollers. Almost everything that TinyOs supports, has been implemented for the system. Telosb is very elaborate in terms of the sheer number of interface it defines to provide maximum flexibility, for TinyOs applications. It is not you how you might want to structure the port for ez430rf2500 where the aim would be to "get it working".

- **tmote2500** It is based on msp430 family of microcontroller, and has support for CC2500 radio chip, but beware, there is no trace of the specifications of this device. This particular platform is probably incomplete,but was critical in getting the CC2500 radio for ez430 compiled without errors.

- **Platform Z1** Again, based on the msp430, this was used as reference for porting the tinyOS Printf Library to ez430rf2500.
- **Platform telosa** Belongs to the Telos family of platforms. Has a an implementation structure that is similar to other platforms. It was used as reference for porting the TinyOs Serial library ported to ez430rf2500.

5.8 Summary: State of Progress

As mentioned earlier, the port in still in complete. The following points capture the current status of the project:

- Supported microcontroller features:
  - Basic Tool Chain support
  -

5.9 Future development Ideas

There are a few ideas that could be useful to carrying this project forward; as it always is, due to time constraints some of these were overlooked. Each issue that is being faced and a direction for approaching the solution is being listed here. Future Development should start by addressing these issues.

- **Uart-Problem** The UART program in TinyOs does send some characters to program listening on the PC,but it they show up as a bunch of '?' symbols. The UART parameters on the node and the PC are the same.
  Microcontroller Clock $TOSDIR/chips/msp430-small/timer has the interfaces and implementation that configure the microcontroller clock every time on software Initialization. Refer to Boot Sequence in [11]. The clock of controller should match the UART settings that have been used in $TOSDIR/platforms/msp430f2274/UsciConf.nc .
- **Uart-receive problem** UART example program for C [12] does not receive data from the UART bus, but is able to send data to the UART bus, contents of which can be verified with a serial terminal client like minicom.
- **Blaze Radio CC2500** Applications with that use the Blaze Radio stack compile successfully, but the transmission or reception does not work. **Approach for Solution** First of all, the Blaze radio stack is not an official TinyOS port but rather an individual's port. The examples in the radio stack itself did not compile, also I could see several references to CC2420 radio stack, in the code. Blaze stack needs to be debugged for incorrect wirings.

6 Tips- Development Environment

A large part your time would be spent on finding reference to a particular variable, file, or simply a word. If your used to familiar with the luxury and comfort of 'IDEs', then it could take really long if you go about manually scanning each file. Unix-like systems have three tools, mostly pre-installed which were can be of great use to you:

- grep: prints lines matching a pattern on the shell, it can used recursively with '-r' to scan entire directories for pattern you want to look for.
- find: searches for files and directories. Generally used with '-name' for finding file names matching a pattern.

You can find more information in the 'man' pages for these commands.

7 Conclusion

Although the time it has taken to port TinyOS to the new platform has exceeded my expectations, it has also give me a deeper understanding in TinyOs which would be useful, for the continuation of this project in the next semester.

References

[1] M. Szczodrak and L. Carloni, "A complete framework for programming event-driven, self-reconfigurable low power wireless networks," in *Proceedings of the 9th ACM Conference on Embedded Networked Sensor Systems*, ser. SenSys '11. ACM, 2011, pp. 415–416.

[2] T. I. Inc., "Msp430 wireless development tools website," http://www.ti.com/tool/ez430-rf2500.

[3] "Tinyos documentation website," http://docs.tinyos.net/tinywiki/index.php/Main_Page.

[4] D. Palshikar, "Research plan-remote water level monitoring: A prototype of a wireless sensor network for water monitoring application," 2012.

[5] M. Szczodrak and L. Carloni, "Embedded system level design group," "http://embedded.cs.columbia.edu".

[6] D. C. David Gay, Philip Levis and E. Brewer, "nesc 1.1 language reference manual," http://nescc.sourceforge.net/papers/nesc-ref.pdf.

[7] T. Takaoka, "tinyos-msp430, tinyos port to small msp430 based microcontrollers," http://code.google.com/p/tinyos-msp430/.

[8] D. Moss, “Ccxx00 radio stack,” https://github.com/tyll/tinyos-2.x-contrib/tree/d205226feb156d8d815be0867277811ec6dd87e1/blaze/tos/chips/ccxx00, Rincon Research Corporation, 2009.

[9] ——, *CCxx00 Radio Stack*, https://github.com/tyll/tinyos-2.x-contrib/tree/d205226feb156d8d815be0867277811ec6dd87e1/blaze/tos/chips/ccxx00, Rincon Research Corporation, 2009.

[10] T. I. Inc., “CC2500 Data Sheet,” http://www.ti.com/lit/ds/symlink/cc2500.pdf.

[11] “Tinyos tutorials,” http://docs.tinyos.net/tinywiki/index.php/TinyOS_Tutorials.

[12] T. I. Inc., “Msp43022x4,msp43022x2 code examples,” http://www.ti.com/lit/zip/slac123.

[13] S. Underwood, “mspgcc: Writing interrupt services routines,” http://mspgcc.sourceforge.net/manual/x918.html.

[14] T. I. Inc., *MSP430x2xx Family User’s Guide (Rev. I)*, http://www.ti.com/litv/pdf/slau144i.