Appendix A List of Acronyms

3G	Third Generation
A/D	Analogue to Digital
AC	Alternating Current
ACK	Acknowledgement Packet
ADC	Analogue to Digital Converter
AODV	Ad Hoc On-Demand Distance Vector
ARC	Adaptive Transmission Rate Control
ARRL	American Radio Relay League
ASK	Amplitude Shift Keying
AWGN	Additive White Gaussian Noise
BER	Bit Error Rate
BPSK	Binary Phase Shift Keying
CCK	Complementary Code Keying
CDF	Cumulative Distribution Function
CDMA	Code Division Multiple Access
CRC	Cyclic Redundancy Check
CSMA/CA	Carrier Sense, Multiple Access with Collision Avoidance
DC	Direct Current
DCF	Distributed Co-Ordination Function
DS-CDMA	Direct Sequence Code Division Multiple Access
DSDV	Destination-Sequence Distance Vector
DSP	Digital Signal Processing
DSR	Dynamic Source Routing
DSSS	Direct Sequence Spread Sprectrum
ECE	Electrical and Computer Engineering (JCU)
EEPROM	Electronically Erasable Programmable Read Only Memory
EIRP	Effective Isotropic Radiated Power
EWMA	Exponentially Weighted Moving Average
FDMA	Frequency Division Multiple Access
FH-CDMA	Fast Hopping Code Division Multiple Access

FHSS	Frequency Hopping Spread Spectrum
FSK	Frequency Shift Keying
GFSK	Gaussian Frequencey Shift Keying
GPS	Global Positioning System
HF	High Frequency
I^2C	Inter-Integrated Circuit
IC	Integrated Circuit
IEEE	Institue of Electrical And Electronics Engineers
IF	Intermediate Frequency
IP	Internet Protocol
ISM	Industrial, Scientific And Medical
ITM	Irregular Terrain Model
ITU	International Telecommunication Union
JCU	James Cook University
LED	Light Emitting Diode
LF	Loop Filter
LNA	Low-Noise Amplifier
LOS	Line-of-Sight
LRNet	Long-Range Network
LRWSN	Long-Range Wireless Sensor Network
MAC	Medium Access Control
MCU	Microcontroller Unit
MIMO	Multiple-Input, Multiple-Output
MMX	Multimedia Extensions
MSK	Minimum Shift Keying
MT	Minimum Transmission
OFDM	Orthogonal Frequency Division Multiplexing
OQPSK	Offset Quadrature Phase Shift Keying
OS	Operating System
OSI	Open System Interconnection
PA	Power Amplifier
PAN	Personal Area Network
PCB	Printed Circuit Board
PCF	Point Co-Ordination Function
PDA	Personal Data Assistant
PDF	Probability Density Function
PLL	Phase-Locked Loop
PTP	Point-to-Point
QPSK	Quadrature Phase Shift Keying
RF	Radio Frequencies

RSSI	Received Signal Strength Indicator
RTS/CTS	Request To Send/Clear To Send
SDRAM	Synchronous Dynamic Random Access Memory
SMACS	Self-Organising Medium Access Control for Sensor Networks
SNR	Signal to Noise Ratio
SPI	Serial Peripherals Interface
SQL	Structured Query Language
SRAM	Static Random Access Memory
TCP	Transmission Control Protocol
TDMA	Time Division Multiple Access
TORA	Temporally-Ordered Routing Algorithm
TOSSIM	Tinyos Simulator
UART	Universal Asychronous Receiver Transmitter
UHF	Ultra High Frequency
USART	Universal Synchronous Asychronous Receiver Transmitter
USB	Universal Serial Bus
VCO	Voltage Controlled Oscillator
VHF	Very High Frequency
VSS	Visual System Simulator
VSWR	Voltage Standing Wave Ratio
WLAN	Wireless Local Area Network
WMEWMA	Window Mean with Exponentially Weighted Moving Average
WSN	Wireless Sensor Network
WSN Model	Wireless Sensor Network Propagation Model
ZC	Zigbee Coordinator
ZED	Zigbee End Device
ZR	Zigbee Router

Appendix B First Paper
Radio Propagation Model for Long-Range Wireless Sensor Network

Radio Propagation Model for Long-Range Ad-hoc Wireless Sensor Network

Simon L.Willis, *Student Member, IEEE,* Cornelis J. Kikkert, *Senior Member, IEEE*

***Abstract*—Recently there has been a growing interest in short-range, densely deployed wireless sensor networks. However, the range of applications for these networks is severely limited, as the sensor nodes are unable to communicate over long-distances. There are many applications involving environmental monitoring, which require ad-hoc communications over distances up to 10km. This paper describes a proposed long-range wireless sensor network and discusses the development of a suitable long-range ad-hoc propagation model to predict the signal-to-noise ratio (SNR) for radio links over irregular terrain. This paper shows that the SNR for such links can be approximated using a Rician distribution and concludes by providing an example of SNR prediction for a proposed rural test site.**

***Index Terms*—Ad-hoc networks, sensor networks, radio propagation.**

I. Introduction

RECENT advances in electronics and wireless technology have allowed for the development of low-cost, low-power sensor nodes that may be randomly deployed to form a multi-hop wireless sensor network. Multi-hop sensor networks have tremendous advantages over conventional 'base-station' networks since they are able to span distances much larger than the transmission range of a node and are able to adapt to network changes to improve network performance. In addition the transmitter power required by each node is less, because of the shorter distances involved, since the nodes have the ability to communicate with surrounding nodes so that messages may be passed across the network.

To date, the majority of research in this field involves the development of miniature, low-power nodes that are to be deployed in a dense sensor network. These nodes typically have a short transmission range of several of tens of metres, which excludes them from possible applications, such as the ability to monitor environmental conditions such as temperature, soil moisture and other critical environmental parameters that can be separated by long distances and are vital to the efficient operation of many farms.

The focus of our research is to investigate what changes are needed to existing short-range wireless sensor networks to enable communications over long distances. A distinctive environment where this capability would be desirable is on a typical Australian cattle property as shown in Fig. 1. On such a property it is highly desirable to monitor the level of water in cattle water troughs, as at present this needs to be checked manually on a daily basis. This is a laborious manual task since distances of up to 10km may separate the troughs.

Fig. 1. Typical Australian cattle property.

Our aim is to produce nodes that are low in cost, solar powered and easy to deploy. They require a transmission range of up to 10km. In order to obtain this goal we plan to modify the Berkeley Mote [1] by replacing the transceiver with a lower frequency, higher power unit. It is planned that the transceiver will operate in the Australian unlicensed 40.66 – 41 MHz frequency band with a maximum power output of 1W EIRP. To maximise power efficiency, a class-C RF amplifier will be utilised. However, due to the inherent non-linearity of this amplifier a modulation technique with a constant envelope must be used. Suitable techniques include frequency-shift keying (FSK) or some forms of quadrature phase-shift keying, such as $\pi/4$ QPSK, OQPSK or MSK.

In order develop the sensor network, it is necessary to utilise an accurate network simulator, such as Tossim, which is packaged with the TinyOS Mote operating system [1]. The main advantage of Tossim is that it conducts simulations using the same code that will be programmed into the nodes. However, Tossim has a very poor radio propagation model for the long distances required in our system. As a result we have

Manuscript received 15th March, 2005.

The authors are with the Department of Electrical & Computer Engineering, James Cook University, Townsville, QLD, Australia (email: simon.willis@jcu.edu.au; keith.kikkert@jcu.edu.au)

developed a propagation model that is suitable for the proposed sensor network and is presented in this paper.

Section II of this paper discusses the basic radio propagation mechanisms and the relevant equations that are implemented in our model. Sections III and IV present the final radio propagation model and describe the results of initial simulations. Section V concludes the paper and discusses future work.

II. Radio Propagation Modelling

Radio propagation is a complex and diverse phenomenon that has been the subject of research for a long period of time. Much of the recent work has been on predicting the signal strength at UHF frequencies in an urban environment for digital television and mobile phones [2]. For our application the sensor nodes operate at a frequency of 40MHz for which adequate radio propagation models fitting our application are difficult to obtain. In addition the system will be deployed in a sparsely vegetated, rural environment. The propagation mechanisms that operate at 40 MHz in a rural location are:

A. *Free-Space Loss*

In an ideal environment the power radiated by an antenna is spread uniformly over the surface of an imaginary sphere surrounding the antenna. Therefore, the power density at a point on the sphere decreases as the distance from the antenna increases. The power received at a distance, d from the antenna is given by equation (1).

$$P_r = \frac{P_t \lambda^2 G_t G_r}{16\pi^2 d^2} \qquad (1)$$

where P_r and P_t are the received and transmitted power, respectively, G_t and G_r are the gain of the transmitter and receiver antennas, respectively and d is the distance from the transmitter.

B. *Reflection*

When a transmitted signal is incident on an obstacle that is considerably larger than the wavelength, then a reflection of the signal will occur. Reflections may occur anywhere in an environment and cause multiple signals to reach the receiver, each with different magnitude and phase. This phenomenon is known as multipath propagation and is illustrated by Fig. 2. Here, we have 3 multipath signals that will have different phase to the direct signal and will therefore act to improve or degrade the overall signal quality.

It was stated above that for a reflection to occur, the obstacle must be larger than the wavelength. In the proposed sensor network the wavelength is 7.5m, but the terrain will be fairly barren for typical Australian outback applications. This means there will not be many reflections since obstacles with dimensions larger than 7.5m will be rare, as shown in Fig. 1.

Neskovic, Neskovic and Paunovic [2] showed that the magnitude of a reflected signal is determined by the reflection coefficient, Γ as shown by equation (2).

$$P_2 = \Gamma^2(\alpha)P_1 \qquad (2)$$

where $$\Gamma(\theta) = \frac{\cos\theta - a\sqrt{\varepsilon_r - \sin^2\theta}}{\cos\theta + a\sqrt{\varepsilon_r - \sin^2\theta}}$$

P_2 and P_1 are the power of the reflected and incident signals, respectively, ε_r is the relative dielectric constant of the ground $\theta = 90^o - \alpha$ (α is the angle of incidence shown in Fig. 2) and $a = 1/\varepsilon$ or 1 for vertical or horizontal polarisation, respectively.

C. *Two-Ray Model*

Nescovic *et al.* [2] presented the two-ray model which is commonly used for modelling a line-of-sight (LOS) radio channel. The signal strength is shown by equation (3) and is calculated by summing the contribution of each path. This equation is derived from the free-space loss equation (1), as well as equation (2), above.

$$P_r = P_t G_t G_r \left(\frac{\lambda}{4\pi}\right)^2 \left| \frac{1}{d_1}\exp(-jkd_1) + \Gamma(\alpha)\frac{1}{d_2}\exp(-jkd_2) \right|^2 \qquad (3)$$

where d_1 and d_2 represent the length of the 1st and 2nd paths, respectively.

The Two-Ray Model is often used to describe the propagation of a direct ray and a ground-reflected ray. If this is the case, the value of α is very small for horizontal polarisation and long links and the reflection coefficient is approximately –1. This means that the direct and the ground-reflected waves will cancel. However, in our case, the λ/4 antenna is vertically polarised and is positioned at ground-level, so that the effect of the ground plane reinforces the transmitted signal.

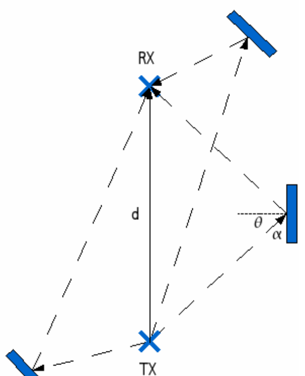


Fig. 2. Possible reflector positions.

D. *Scattering*

When a radio wave impinges on a rough surface the reflected wave is spread out or diffused. This phenomenon is know as scattering and has the affect of reducing the magnitude of the reflected signal. Gibson [3] provided an overview of scattering and stated that the roughness of a surface is often specified using the Rayleigh criterion (4).

$$h_c = \frac{\lambda}{8\cos\theta_i} \tag{4}$$

where θ_i is the angle of incidence. Gibson [3] presented the parameter h which describes the minimum to maximum protuberance about the mean terrain height. If $h < h_c$ then the surface is considered smooth and there is no scattering loss, otherwise the surface is considered rough and the reflection coefficient is multiplied by a scattering loss factor ρ_s. Gibson [3] presented Bothias' equation to calculate ρ_s (5).

$$\rho_s = \exp\left[-8\left(\frac{\pi\sigma_h\cos\theta_i}{\lambda}\right)^2\right] \mathrm{I}_0\left[8\left(\frac{\pi\sigma_h\cos\theta_i}{\lambda}\right)^2\right] \tag{5}$$

where σ_h is the standard deviation of surface height about the mean surface height and I_0 is the zeroth order Bessel function of the first kind.

At the proposed operating frequency of 40MHz, the minimum value of h_c would be 0.94m for an angle of incidence of approximately 0. This corresponds to extremely rough terrain which is unlikely to be experienced in a rural agricultural environment. As an indication, the value of σ_h is approximately 2.29cm for bare ploughed land [4].

E. Effect of Vegetation

As a radio wave passes through vegetation the signal strength is attenuated. However, this effect is more pronounced at higher frequencies. This is demonstrated by McLarlen [5] who summarised results from an ITU report [6], which stated that the attenuation caused by a forest is 0.4dB/m at 3GHz, 0.1dB/m at 1GHz and 0.05dB/m at 200MHz. Since we aim to operate the sensor network at 40MHz in barren terrain like shown in Fig. 1, the affect of vegetation would be negligible.

F. Refraction

When a radio signal passes over the Earth's surface it does not travel in a straight line. Due to a variation of the refractive index of the atmosphere with height above the surface, the radio signal follows a curved path. Hernando and Perez-Fontan [7] stated that normally, in propagation studies, straight lines are used and the Earth is assumed to have an effective radius given by kR_0. Here, R_0 is the radius of the earth (6370km) and k is a correction factor, which is normally 4/3. This effectively lengthens the range of a transmitter beyond the visible horizon.

G. Diffraction Loss

Diffraction loss occurs when the direct line between the transmitter and receiver is blocked by an obstacle, much bigger than the wavelength of the signal. At the edge of the obstacle, scattering occurs and the signal is attenuated. Fig. 3 shows the diffraction loss caused by a knife-edge obstruction.

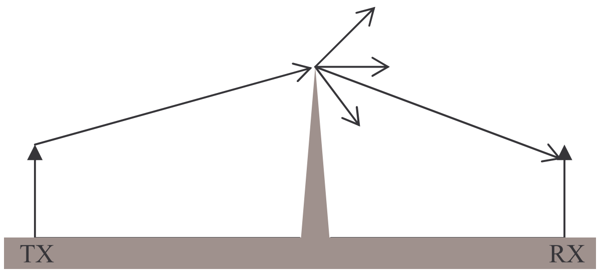


Fig. 3. Diffraction loss caused by a knife-edge obstruction.

The diffraction loss caused by a knife-edge is easy to model. However, knife-edges rarely occur in real-world scenarios and hence terrain features such as mountains and valleys need to be modelled using an approximation. For our application it is also important to model the diffraction loss due to propagation beyond the horizon.

Many empirical and theoretical models have been developed, most of these are designed for higher frequency transmissions in an urban environment. Suitable models for our application are the Longley-Rice model [8] or the Point-to-Point (PTP) model described by Wong [9]. The PTP model is based on the Longley-Rice model but gives results that are more accurately matched to real-world measurements. The PTP model calculates the diffraction loss by replacing a given terrain profile with an "equivalent rounded obstacle". Appropriate formulae are then applied to calculate the diffraction loss caused by that obstacle.

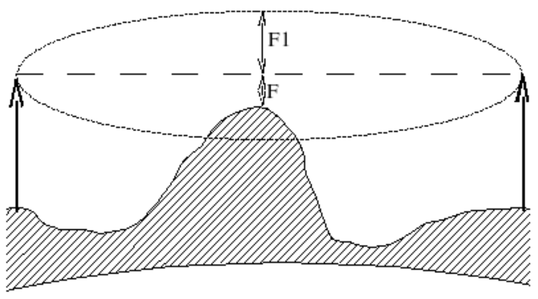


Fig. 4. Path clearance ratio.

In Fig. 4, the distance between the line-of-sight (LOS) path and the primary obstacle is shown as F. and the radius of the first Fresnel zone at this point is given as F1. The F/F1 ratio is called the "path clearance ratio", and indicates the amount that an obstacle protrudes into the Fresnel zone and is proportional to the diffraction loss. A negative path clearance ratio indicates that the obstacle is blocking the LOS path completely. The PTP model uses the path clearance ratio and the equivalent roundness factor to approximate the diffraction loss. Additionally to this, the PTP model also approximates the affect of the earth-bulge and diffraction beyond the horizon.

III. The Long Range Ad-hoc Propagation Model

In a wireless sensor network, nodes have the ability to control the path that data takes as it passes across the network. In reality it would be wise to choose the path with the lowest bit error rate (BER) to ensure that data remains uncorrupted. The BER of a given modulation scheme can be approximated using a formula such as (6), which applies to a simple Frequency Shift Keying (FSK) modulation scheme [10].

$$BER = \frac{1}{2}\operatorname{erfc}\left(\sqrt{\frac{E_b}{2N_0}}\right) \qquad (6)$$

where erfc is the complimentary error function and E_b / N_0 represents the signal-to-noise ratio (SNR). Similar expressions are available for other modulation techniques.

Equation (6) shows that the BER can be found quite easily, given the SNR. The propagation model must thus produce an equivalent SNR for the long-range ad-hoc radio system to be investigated.

To approximate the SNR, the signal component is assumed to consist solely of the direct ray. The noise is assumed to consist of the thermal noise received by the antenna, the receiver noise factor and signal contributions from the multipath rays, which are considered to be noise since they cannot easily be used as part of the received signal.

The direct signal is computed using the free-space loss equation (1) and the diffraction obtained from the PTP model for the direct signal component. The required terrain profile for the link can be generated using digital elevation maps provided by [11].

The multipath signals are computed using the free-space loss and reflection loss equations and the diffraction obtained from the PTP model for the reflected signal components. To calculate the reflection loss, a reflector is positioned at a random azimuth from the transceiver. The angle of incidence is chosen at random and checked to ensure that the orientation of the reflector will allow the reflection to reach the receiver. The reflection coefficient (2) is then computed using a random dielectric constant in the range of 2 to 7. This range is valid for dry soil and vegetation [12], which is typical of the proposed environment. In addition to the reflection loss, the scattering loss can be computed, however it was shown previously that this is negligible.

To compute the diffraction loss for the multipath signals, the terrain profile between the transmitter and receiver required for the PTP model is assumed uniform in the direction perpendicular to the direct signal path, as shown in Fig. 5.

The propagation model used for the long-range ad-hoc radio network presented here, uses a direct signal and a specified number of multipath signals, whose reflector positions and conditions are chosen at random. If one of these random multipath signals causes a received signal more than 30dB below the direct signal, then that path is discarded and a new path is generated.

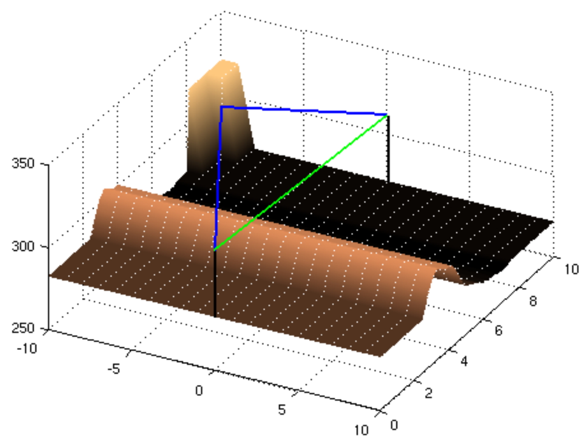


Fig. 5. Terrain used for multipath signals, the terrain along x=0 is the actual path profile.

This is demonstrated by Fig. 6, which shows the cumulative random positions of the reflectors after 1000 simulations of propagation conditions with a direct path and two multipath reflections, which is typical of the expected environment. Such a simulation allows the expected long-term SNR distribution to be determined.

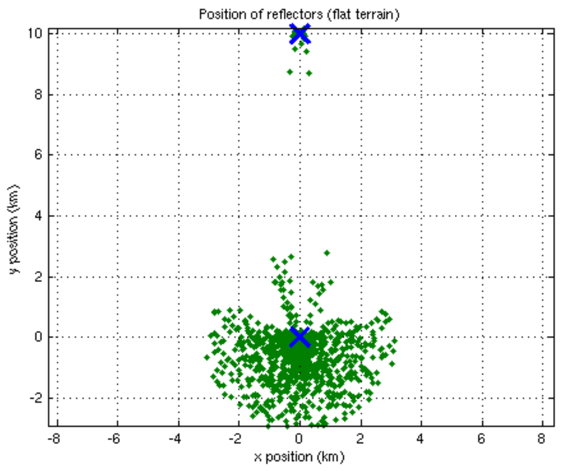


Fig. 6. Position of Reflectors.

In this figure, the transmitter and receiver are represented as crosses (the lower cross is the transmitter) and the reflectors are represented by dots. It is obvious in Fig. 6 that the large majority of possible reflector positions are behind the transmitter. At these positions the required angle of incidence will be large which incidentally causes a larger reflection coefficient and hence a stronger signal. It also evident that there are some areas in front of the transmitter where there are no reflectors. At these points, the angle of incidence (approximately 20 degrees) causes the reflection coefficient to be very small and the multipath signal to be more than 30 dB below the direct signal.

The reflectors are fairly close to the transmitter because the diffraction loss from the PTP model increases sharply at approximately 3km so that reflections which include two paths greater than 3 km, will result in negligible signals

IV. Propagation Model Predictions

A. *Effect of multipath signals*

For Benchmark testing, of the long-range ad-hoc propagation model, the results for propagation over a flat terrain are calculated. For flat terrain one would expect the results to have a Rician distribution. These simulations are carried out at a frequency of 40.8 MHz and an EIRP of 1 Watt.

For reliable communications, a BER less than 10^{-3} is required. Using equation (6), a BER of 10^{-3}, corresponds to a SNR of 9.80dB. The long-range ad-hoc propagation model for a 10 km path and flat terrain indicates a 35.7dB SNR, thus easily meeting the 9.80dB SNR requirement. When multipath signals are included in the long-range ad-hoc propagation model, the SNR will decrease, as shown in Fig. 7, which shows the cumulative SNR distribution obtained from 1000 simulations of the propagation model.

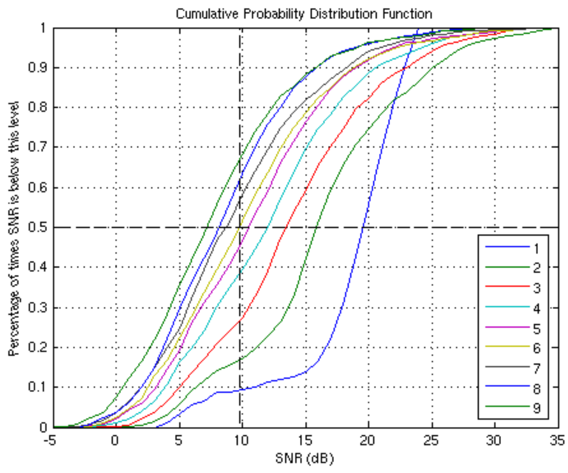


Fig. 7. Cumulative probability distribution for multipath signals.

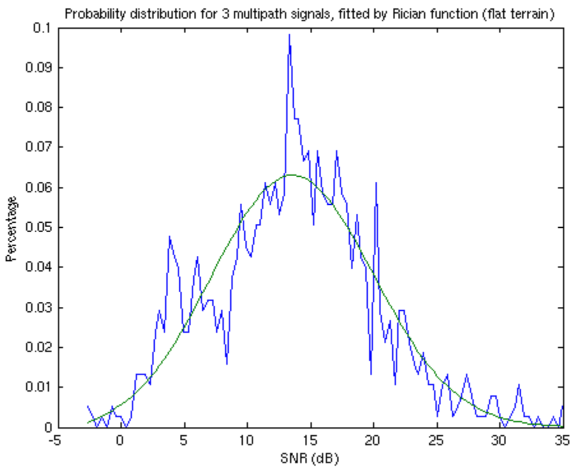


Fig. 8. SNR probability distribution, fitted with Rician function (3 multipath signals).

For flat terrain, and 6 multipath signals 50% of the received signals have a SNR of 9.8 dB or greater and will thus cause a BER of less than 10^{-3}. Most practical propagation paths at 40MHz will have less than 6 multipath signals, so that a 10 km path can be obtained reliably at the permitted operating power.

When the curves of Fig. 7 are plotted as a probability distribution, the data can be approximated using a Rician probability density function, as shown in Fig. 8. When there is only one multipath signal present, the Rician probability distribution function cannot be used to fit the data from the long-range ad-hoc propagation model.

B. *Effect of irregular terrain*

The propagation model is capable of analysing an entire test site such as the cattle property shown in Fig. 9. This is the same property shown in Fig.1. Each grid square is 1 km^2 and the blue circles represent cattle water troughs. At each cattle trough we propose to install a sensor node (labelled A-F) to monitor the water level. The green lines represent the possible links between sensor nodes and the data shown in brackets represents the distance and the minimum SNR that will be exceeded 50% of the time. For these calculations 2 multipath signals are used for communication between each node.

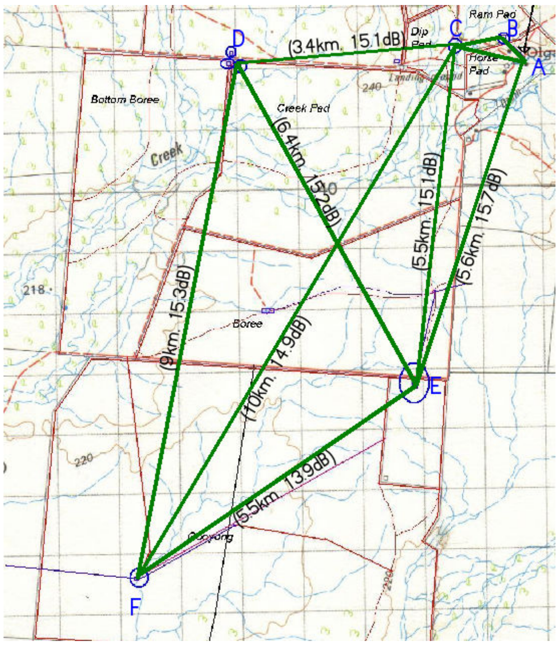


Fig. 9. Sensor network at cattle property.

The property shown in Fig. 9 is fairly flat. Therefore, as a case-study we will also analyse a link over more complex terrain. The nodes in this link are separated by a distance of 5.2 km and have a hill in the middle that is approximately 25m higher than the transmitter and 35m higher than the receiver.

Table 1 shows the minimum SNR exceeded 50% of the time for both flat terrain and the irregular terrain. It is obvious from this table that the irregular terrain causes more attenuation.

TABLE 1
COMPARISON OF MIINIMUM SNR OCCURRING 50% OF THE TIME FOR FLAT AND IRREGULAR TERRAIN

Number of multipath signals	SNR at 50% occurrence	
	Flat Terrain	Irregular Terrain
0	43.2	40.7
1	18.5	17.6
3	13.2	12.7
5	10.6	9.4
7	9.01	7.60
9	7.40	6.25

Fig. 10 shows a comparison of the probability distribution for flat and irregular terrain. It can be seen that paths over irregular terrain have a probability distribution that is similar to paths over flat terrain and can thus also be approximated by a Rician distribution for scenarios where the number of multipath signals is greater than 1.

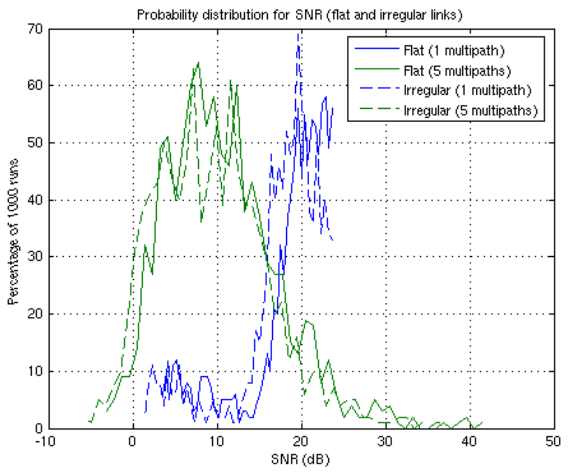


Fig. 10. Comparison of distribution for flat and irregular terrain

V. CONCLUSION

Many environmental monitoring applications can be implemented using ad-hoc networks where the distance between nodes can be up to 10 km. A detailed long-range ad-hoc radio propagation model for such networks has been developed and for both a flat terrain and an irregular terrain. A Rician distribution can be used to approximate these propagation paths. The long range ad-hoc propagation model also shows that reliable communications can be obtained for 10 km paths at 40 MHz and a 1 Watt EIRP.

This model will be used as an extension to the Tossim simulator, as well as an aid for the design and testing of prototype sensor nodes.

REFERENCES

[1] (2004, June 22). *TinyOS* [Online]. Available: http://www.tinyos.net/

[2] A. Neskovic, N. Neskovic, and G. Paunovic, "Modern Approaches in Modeling of Mobile Radio Systems Propagation Environment," in *IEEE Communications Surveys*, Third Quarter, 2000

[3] J. D. Gibson, *The Mobile Communications Handbook*. Florida, USA: CRC Press, Inc., 1996.

[4] Y. Oh and J. Stiles. (2005, February, 08). *Chapter IX. Surface Roughness Measurements* [Online]. Available: http://hydrolab.arsusda.gov/washita92/datarpt/chap9.htm

[5] B. McLarnon. (2005, March 01). *VHF/UHF/Microwave Radio Propagation: A Primer for Digital Experimenters* [Online]. Available: http://www.tapr.org/tapr/html/ve3jf.dcc97/ve3jf.dcc97.html

[6] *RECOMMENDATION ITU-R P.833-4 - Attenuation in vegetation*: ITU-R, 2003.

[7] J. M. Hernando and F. Perez-Fontan, *Introduction to Mobile Communications Engineering*. Boston, USA: Artech House, 1999.

[8] P. L. Rice, A. G. Longley, K. A. Norton, and A. P. Barsis, *Technical Note 101 - Transmission Loss Predictions for Troposheric Communication Circuits*: National Bureau of Standards, 1967.

[9] H. Wong. (2004, November 04). *Field Strength Prediction in Irregular Terrain - the PTP Model* [Online]. Available: http://www.fcc.gov/oet/fm/ptp/report.pdf

[10] S. Haykin, *Communications Systems, 4th Editiion*. New York, USA: John Wiley & Sons, 2001.

[11] (2005, February 04). *Geophysical Archive Data Delivery System* [Online]. Available: www.geoscience.gov.au/gadds

[12] (2004,October 26). *Dielectric Constant Table* [Online]. Available: http://zhangzc.jahee.com/chemcai/Dielectric%20Constant%20Table.htm

Appendix C Second Paper
Design of a Long-Range Wireless Sensor Node

Design of a Long-Range Wireless Sensor Node

Simon Willis and Cornelis J. Kikkert
Electrical and Computer Engineering
James Cook University
Townsville, Queensland, Australia
simon.willis@jcu.edu.au, keith.kikkert@jcu.edu.au

Abstract— **The use of wireless sensor network technology for environmental monitoring is becoming increasingly important. A major limitation of existing sensor nodes is their short transmission range. This paper introduces the JCUMote which is based on the Mica2 Mote, but has a transmission range up to 10 km. This paper discusses the design of the transceiver hardware, in particular the RF power amplifier and receiver isolation network for the JCUMote.**

Keywords— **sensor network, ad-hoc network, long-range**

I. INTRODUCTION

It is increasingly important to monitor environmental resources. Wireless sensor networks are ideal to perform this task. They consist of many small wireless sensor nodes that are self-configuring, self-maintaining and low in cost. Wireless sensor nodes are simple to install and can be configured to monitor any desired resource, hence they have a broad range of applications, one of which is environmental monitoring.

Existing sensor nodes such as the Berkeley [1] Mote are already being utilized for environmental monitoring, such as in the Great Duck Island project [2]. One major limitation of existing devices such as the Mote is that they have a short transmission range of less than several hundred meters. This limits them to applications involving small dense networks.

The aim of our research is to develop a sensor node that is capable of transmitting over large distances of up to 10 km. Such a sensor network has many applications in Australia. Typical examples are monitoring the condition of the Great Barrier Reef and monitoring the level of water in feeding troughs on a cattle station. A typical Australian cattle station is shown in Fig. 1. This property is approximately 800 km^2 in area, is inhabited by less than 10 people and is located 10 km from the nearest bitumen road and 20 km from the nearest neighbor. On this cattle station the water troughs are up to 10 km apart and it takes several days of travel to check all the cattle troughs. A wireless sensor network provides a continuous indication if all the troughs have water in them.

For the long-range applications we have developed a node called the JCUMote. It is based on the Berkeley Mica2 Mote so that it can use the same sensor boards and operate on the TinyOS Operating System [3] which has been designed specifically for sensor networks, is open-source and has been ported to many different sensor node applications.

Figure 1. A typical Australian Cattle Station.

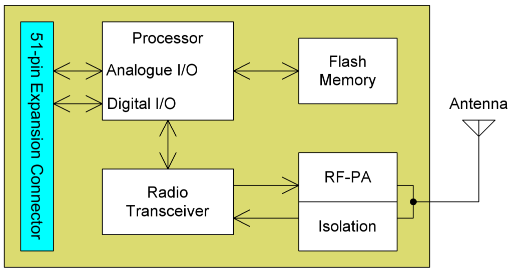


Figure 2. Block diagram of the JCUMote.

The JCUMote is designed so that it is low in cost, easy to install and has a single omni-directional antenna. It has been designed to operate in the Australian unlicensed frequency band of 40.66-41 MHz with 1 W EIRP. Existing nodes such as the Berkeley Mote operate at 315 MHz or higher with 10 mW of power. The advantage of using a lower frequency is that the attenuation due to free-space loss is less at lower frequencies, the radio signal will tend to bend around hills, pass easier through obstacles or vegetation and the large wavelength of the signal means that there will be less multipath reflections. A disadvantage of using higher power is that a larger battery is required. For this application, a 6V lead acid battery with a solar panel is used. The authors have developed a model [4] that simulates the propagation of a 40 MHz signal over irregular terrain.

II. DESIGN OF THE JCUMOTE

The JCUMote design is based on the Berkeley Mica2 Mote [1] and uses the same microcontroller (ATMEL Atmega128L), flash memory and sensor interface, but different radio hardware A block diagram of the JCUMote is shown in Fig. 2.

A. *Transceiver Design*

The Mica2 currently uses the Chipcon CC1000 radio transceiver chip which operates at frequencies of 315, 433, or 868/916 MHz with up to 10 dBm transmitted power. This transceiver is not suitable for operation at 40 MHz and the Melexis TH7122 was found to be a suitable replacement as it can operate at frequencies between 27 MHz and 1 GHz [5]. This transceiver includes a FSK demodulator, FSK/ASK modulator, programmable interface and an output power amplifier (PA) with 10 dBm maximum output.

For the JCUMote, the radio transceiver will have a maximum data rate of 19.2 kbps, which is the same as the Mica2. Manchester encoding is used to allow for easy synchronization at the receiver. The system transmits an FM signal with a frequency deviation of 38.4 kHz, this allows a class C power amplifier to be used, which has high efficiency, thus maximizing the battery life. The TH7122 uses a programmable phase-locked loop (PLL) so that the carrier frequency can be set from the microcontroller unit (MCU). To achieve narrowband FSK modulation, the voltage controlled oscillator (VCO) of the PLL is modulated directly via an external varactor diode. The signal is amplified by the internal PA which has a maximum power output of 10 dBm. An external amplifier is used to boost this signal to 30 dBm.

For demodulation, the received signal is applied to a low-noise amplifier (LNA) and mixed down to the intermediate frequency (IF). A 5.5 MHz IF was chosen, to permit a low-cost narrowband (100 kHz) Murata IF filter to be used. The IF signal is amplified and applied to a standard FM quadrature detector.

The CC1000 transceiver on the Mica2 Mote transmits a 10 dBm signal, which causes no harm to the receiver front-end. However, on the JCUMote the external PA boosts the signal to more than 30 dBm which will damage the receiver front-end. Hence, an external transmitter-receiver isolation network must be added to protect the receiver when the node is transmitting.

The CC1000 on the Mica2 normally uses a byte-level interface which allows a whole byte to be received from the MCU. The data is then Manchester encoded and transmitted. It is desirable that our software radio interface is identical to that of the Mica2 so that the TinyOS software for the upper layers of the network stack may be re-used. For the TH7122, byte-level transmissions can not be used and a bit-level interface must be used. This requires all the data generation and Manchester encoding to be done by the MCU. In order to achieve the required data rate the SPI is set to “slave mode” and the clock signal is generated by an output compare pin, as shown in Fig. 3. For reception, the data bits are processed by the MCU and formed into the required output bytes for passing to the upper layers of the network stack. The transceiver is programmed using a 3-wire synchronous serial system shown in Fig. 3.

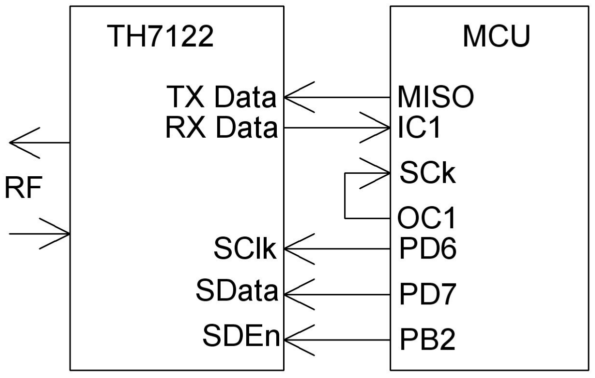


Figure 3. Microcontroller-Tranceiver Interface.

B. *Power Amplifier Design*

The external PA is required to boost the 10 dBm output signal from the transceiver to 1 Watt or +30 dBm. Since the amplifier is battery powered, it must be as efficient as possible, so that Class C operation is required. In addition, the amplifier must be stable, cost effective and matched to the impedance of the transceiver IC and antenna.

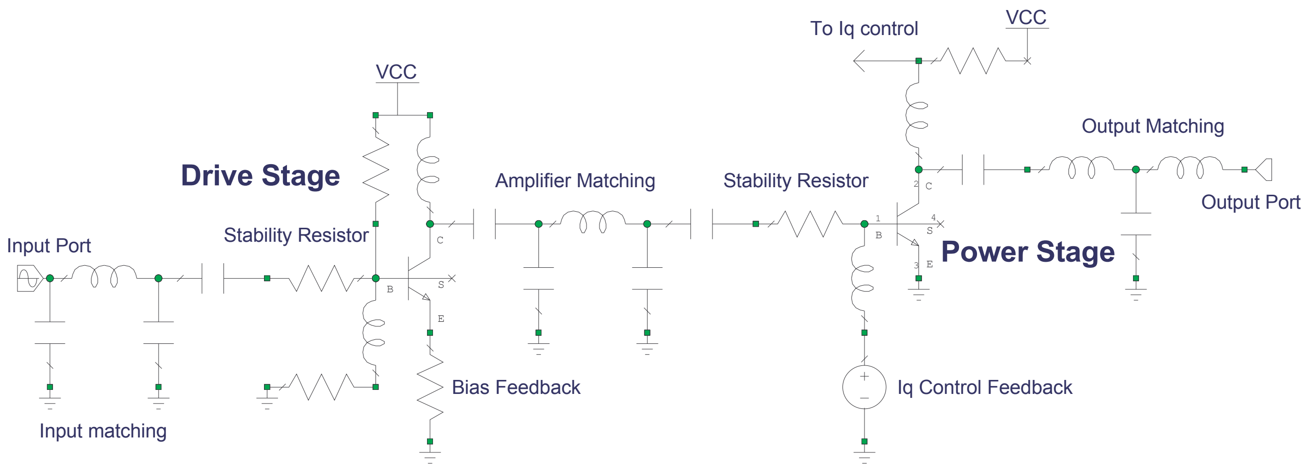


Figure 4. RF Power Amplifier Circuit

For 40 MHz operation, no suitable commercial amplifier IC is available, so the only cost effective approach is to design an amplifier using discrete components. The output transistor needs to be able to dissipate at least 2W of power, be able to supply a 500mA peak current, have a low ON resistance, have a low cost and have a gain of at least 10dB at 40MHz.

The Zetex FZT649 [6] was selected as a suitable power-stage transistor. A drive stage is needed to obtain the required 20 dB amplifier gain and provide sufficient drive for the FZT649 in class C operation under all temperature conditions. A general purpose transistor was selected for the drive stage and biased for class AB operation.

Fig. 4 shows the circuit diagram of the RF power amplifier. The input impedance is tuned to give a full voltage swing on the open-collector output of the transceiver. The driver transistor uses an emitter feedback network for bias control. A matching network is used between the driver and the output transistor to provide the optimum gain and stability. A T-matching network is used at the output as this provides the required impedance transformation with the lowest losses in the output transistor. For stability, low resistance resistors are required at the base of each transistor.

1) *Collector Current Control*

The power stage bias control voltage is provided by an operational amplifier circuit, which monitors the current flowing into the collector via a current sensing resistor in series with the supply and varies the bias voltage to ensure that the transistor remains correctly biased at all temperatures.

2) *Gain and Power Output*

The amplifier circuit in Fig. 4 was simulated and optimized in Microwave Office [7] using non-linear Gummel-Poon models for the transistors. Microwave Office was used to determine the linear gain shown in Fig. 5. The available gain is the amplifier gain assuming ideal matching. Since there are losses in the matching network, the actual gain (transducer gain) is slightly less.

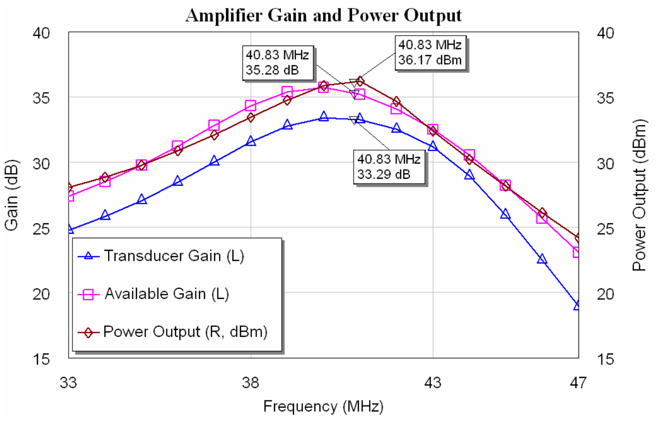


Figure 5. Amplifier Gain and RF Efficiency.

The power output graph in Fig. 5 was obtained from a non-linear simulation of the amplifier when a 10 dBm input was applied. Fig. 5 shows a 36 dBm saturated output from the power stage, which is a sufficient margin above the 30 dBm requirement. The saturated gain of 26 dB is less than the 33.3 dB transducer gain. The level of saturation is sufficient to ensure a constant power output with temperature and transistor parameter variations.

3) *Amplifier Stability*

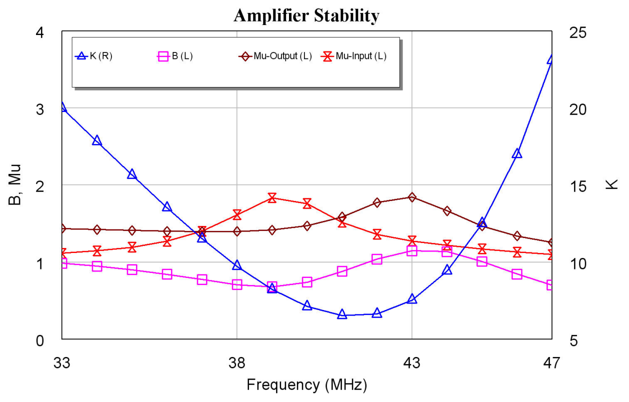


Figure 6. Amplifier Stability Measurements.

During the optimization of the amplifier, the stability factor K, the supplemental stability factor B and the geometric stability factor μ were all determined over the frequency range of 33 to 47 MHz. For unconditional stability, K and μ must be greater than 1 and B must be greater than 0. From Fig. 6 it can be seen that the amplifier is unconditionally stable over the desired frequency range.

In order to achieve unconditional stability small resistors were added in series with the base of each transistor. These resistors are 2.2 Ω and 1.8 Ω, respectively and improved the stability noticeably. These resistors have a negligible effect on the amplifier gain and efficiency.

C. *Isolation Network*

An isolation network is required to protect the receiver front-end when the device is transmitting. This network must attenuate the transmitted signal by at least 30 dB, but provide minimal loss when receiving. For efficiency, cost and reliability reasons, a solid state network rather than a relay controlled switch was used to provide the isolation. The solid state network must be able to handle the large voltages produced by the output amplifier. As a result large signal non-linear simulation models must be used to optimize the network. The resulting isolation network is shown in Fig. 7.

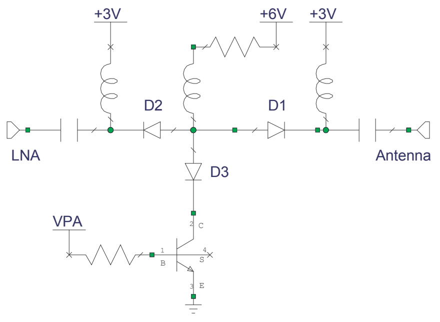


Figure 7. Isolation Network.

Simulations in Microwave Office showed that when in transmit mode the isolation network was able to attenuate a 30dBm signal by 40dB which is safe for the LNA. In receive mode a small drop of 0.8dB is evident in the received signal.

The operation of the isolation network is controlled by the voltage, V_{PA} which is set to 0V to receive data. In this state the transistor is off and D_1 and D_2 are forward biased, allowing the signal from the antenna to reach the LNA without much attenuation. During transmission, V_{PA} is set to 6V, causing the transistor to turn on and forward-bias D_3. Diodes D_1 and D_2 are reverse-biased, which means the transmitted signal is blocked by D_1. The small signal that gets past D_1 is then shorted to ground by D_3 and blocked by D_2.

On the JCUMote the power supply to the PA and the voltage V_{PA} are controlled by a spare pin on the TH7122. When in transmit mode the TH7122 shorts the pin to ground, switching on a transistor which provides 6V to V_{PA} in Fig. 7 and also powers the amplifier circuit of Fig. 4.

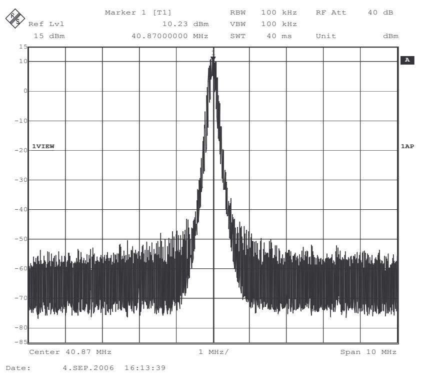


Figure 8. Frequency Spectrum (20 dB attenuation)

D. *PCB Design*

A PCB for the complete JCUMote, including the MCU, Memory, PA and isolation network was designed using Altium (Protel) DXP [8] and is shown in Fig. 9. This PCB can be attached to a Mica2 programmer board. The PCB is double-sided and is 160 mm X 100 mm in size. It is housed with the battery in a die-cast enclosure onto which a quarter-wavelength whip antenna is mounted. The only external parts required are the solar panel for powering the JCUMote.

III. Hardware Testing

A prototype network using four nodes has been constructed and each of the nodes produces 1 W of power, as shown in Fig. 8. A multi-hop routing algorithm has been installed on the nodes and is performing well. The prototype network is currently being field tested.

Acknowledgment

The authors wish to thank the staff at James Cook University for their support in manufacturing the hardware. The authors thank AWR for their University support program, which made it possible to use Microwave Office for the computer simulations presented in this paper. The Australian Research Council (ARC) is acknowledged for providing the grant for the equipment used for making the measurements of the hardware.

References

[1] (2004, June 23). Berkeley WEBS [Online]. Available: http://webs.cs.berkeley.edu/

[2] J. Kumagai, "The Secret Life of Birds," in Spectrum, IEEE, vol. 41, pp. 42-49, 2004.

[3] (2004, June 22). TinyOS [Online]. Available: http://www.tinyos.net/

[4] S. L. Willis and C. J. Kikkert, "Radio Propagation Model for Long-Range Ad Hoc Wireless Sensor Network," presented at 2005 International Conference on Wireless Networks, Communications and Mobile Computing, Maui, Hawaii, 2005.

[5] (2005, April 06). TH7122: 27 to 930MHz FSK/FM/ASK Transceiver [Online]. Available: http://www.melexis.com/prodmain.asp?family=TH7122

[6] (2006, April 25). ZTX649: NPN Silicon Planar Medium Power Transistor [Online]. Available: http://www.zetex.com/3.0/pdf/ZTX649.pdf

[7] (2006, April 27). Microwave Office [Online]. Available: http://www.mwoffice.com/products/mwoffice/

[8] (2006, April 27). Altium DXP [Online]. Available: http://www.altium.com

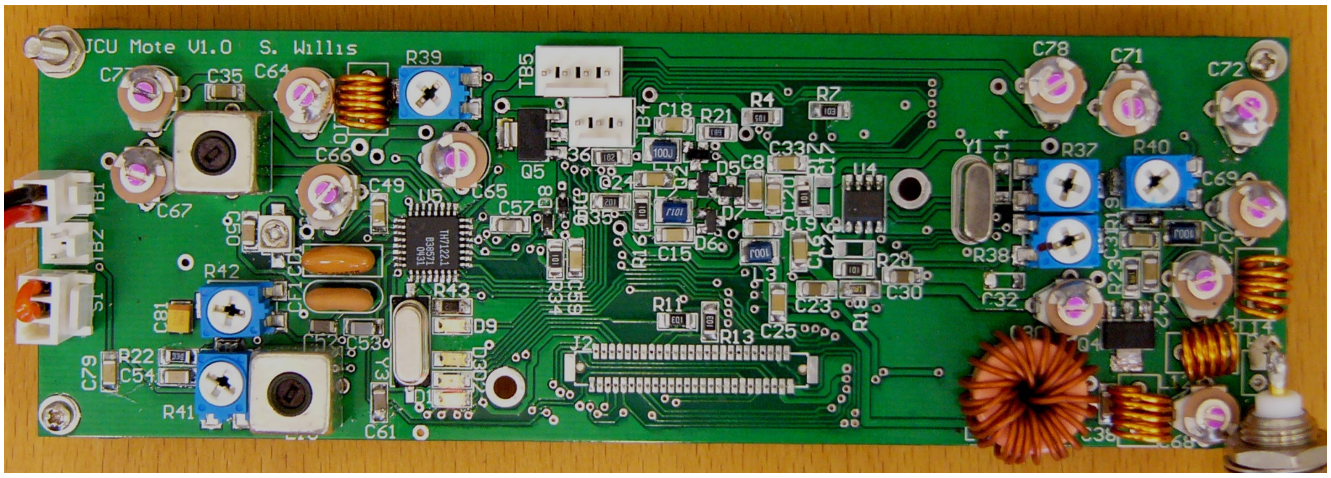


Figure 9. The JCUMote PCB

Appendix D Third Paper
Radio Propagation Model for Long-Range Wireless Sensor Networks

Radio Propagation Model for Long-Range Wireless Sensor Networks

Simon Willis and Cornelis Jan Kikkert
Electrical & Computer Engineering
James Cook University
Townsville, Australia
simon.willis@jcu.edu.au, keith.kikkert@jcu.edu.au

***Abstract*—Wireless Sensor Networks are ideal for monitoring environmental conditions, but typical nodes are limited by their short transmission range. This paper presents a long-range node with a tested range of 13.2 km. A novel radio propagation model is presented, which has been experimentally verified. This paper also presents a unique method to determining the number of multipath reflections in a suburban environment.**

***Keywords*—wireless sensor network, radio propagation model, long range**

I. INTRODUCTION

Wireless Sensor Networks (WSN) have emerged as a useful technology for a plethora of sensing applications. Manufacturers such as Crossbow [1] have produced many nodes such as the Mica2 Mote, which are being used by researchers worldwide. A major limitation of existing nodes such as the Mica2 is their maximum transmission range of several hundred meters. This restricts the node to applications that require the monitoring of small geographical areas only.

In countries such as Australia, there is a need to monitor environmental resources over very large areas. Typical examples are the monitoring of the water-level in a cattle feeding trough and the monitoring of the temperature and salinity of water on the Great Barrier Reef. To overcome the limitation in transmission range we have developed a wireless sensor node called the JCUMote, which operates in a licence free 40 MHz band, which in Australia permits a 1 W effective radiated power.

When operating a WSN over a large area it is important to use a good radio propagation model to determine the performance of the links. The model must include the effects of irregular terrain and multipath reflections. To perform this task we have developed a radio propagation model specifically for use with wireless sensor networks, called the JCU-WSN Model.

To verify the performance of the JCU-WSN model we have conducted field tests of the JCUMotes in rural and suburban areas and have compared the results with JCU-WSN model predictions. Using these results we have created an additional model that will estimate the number of multipath reflections in an urban environment.

II. RADIO PROPAGATION MODEL

Radio propagation has been the subject of research for a long period of time. The focus of more recent work has been on predicting the signal strength in an urban environment for digital television and mobile phones. Our scenario is very different, since the sensor nodes operate at a lower frequency and will often be deployed in a sparsely vegetated, rural environment. This section describes the typical propagation mechanisms that are likely to occur at the proposed frequency with elevated nodes.

A. Radio Propagation Modelling

1) Radio Propagation Over Flat Terrain

When a radio signal is transmitted over flat terrain it is subject to attenuation due to free-space loss, ground reflections and refraction due to the Earth's surface. The free-space model calculates the decrease in signal strength due to the distance away from the transmitter and shows that the received signal strength is proportional to the inverse square of the distance.

When an antenna is elevated, a ground reflection also occurs. The magnitude of the reflected power depends on the angle of incidence and the ground conductivity. At the receiver, the ground reflection is out of phase with the line-of-sight signal and causes cancellation. Hernando and Pérez-Fontán [2] shows that this can be approximated by the Plane-Earth model (equation 1), where the received signal is inversely proportional to distance raised to the fourth power.

$$P_r = P_t G_t G_r \left(\frac{h_t h_r}{d^2} \right)^2 \tag{1}$$

Due to a variation in the atmosphere's refractive index with height, the radio signal follows a curved path and tends to follow the curvature of the Earth. This effectively lengthens the range of a transmitter beyond the horizon. In propagation studies [2], straight lines are used for the propagation paths and the Earth is assumed to have an effective radius given by kR_0, where R_0 is the radius of the earth (6370km) and k is a correction factor, which is normally 4/3.

2) Radio Propagation Over Irregular Terrain

Irregular Terrain inside the Fresnel zone of the direct path causes attenuation of the signal due to scattering. Several models exist, which attempt to calculate these losses. The JCU-WSN model uses the point-to-point (PTP) model, developed by Wong [3]. The path loss due to scattering depends on refractive effects and is thus different from node A to node B and from node B to node A.

3) Multipath Reflections

Reflections can be caused by obstacles which are bigger than the signal wavelength. These reflections will arrive at the receiver with different phase and magnitude to the line-of-sight ray and can cause cancellation of the signal.

B. The JCU-WSN Model

The JCU-WSN model accounts for the above propagation characteristics as described by the authors in [4]. The model calculates the effects of the terrain, by using a terrain profile along the line of sight path, which can be obtained from digital elevation data provided by NASA [5]. The model assumes the terrain is symmetrical along the x-axis as shown in Fig. 1.

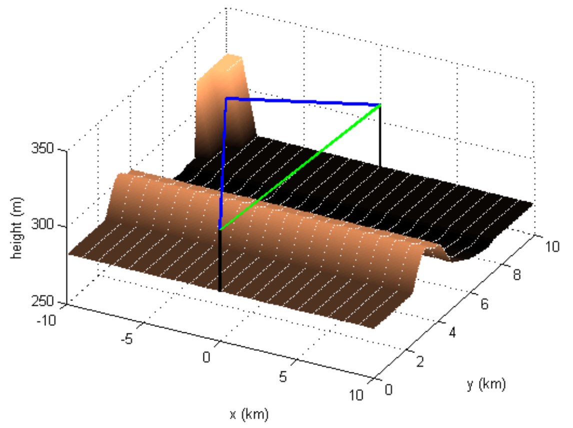


Figure 1. Terrain used for multipath signals, the terrain along x=0 is the actual path profile.

To represent multipath reflections, the model randomly places reflectors on the terrain, as shown in Fig. 1, and calculates the received signal strength. If the magnitude of the reflection is more than 30 dB below the direct signal, then it is discarded and recalculated for a new position. The number of multipath reflections is specified by the user and is affected by large terrain obstacles and buildings.

III. The Long-Range Wireless Sensor Node

A long-range wireless sensor node, called the JCUMote, was designed and is discussed by the authors in [6]. The JCUMote operates at a frequency of 40.66-41 MHz with 1 W EIRP. The node is based on the Crossbow Mica2 Mote [1], which is used extensively by other researchers. The Mica2 has an open-source schematic and can be programmed using TinyOS [7], which has been developed specifically for WSNs. By basing the JCUMote on this node we were able to re-use existing protocols and applications available in TinyOS.

The block diagram of the JCUMote is shown in Fig. 2. This node uses the same ATMEL ATmega128L microprocessor as the Mica2. The node contains 4Mb of flash memory and an expansion port for connecting programming or sensor boards.

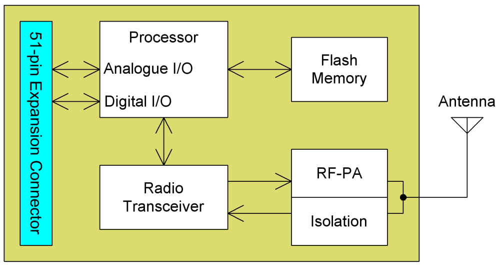


Figure 2. The JCUMote Block Diagram

For operation at 40 MHz, the JCUMote uses a different radio transceiver than the Mica2. A Melexis TH7122 was chosen as it contains an FSK modulator and demodulator. This IC is capable of operation between 27 MHz and 1 GHz and was configured for operation at 40 MHz using external circuitry. The TH7122 is capable of producing +10 dBm (10 mW) of RF power, which is amplified to +30 dBm (1W) using a custom-designed class-C power amplifier (PA) built from discrete components. The PA was designed using Microwave Office and is stable over a broad frequency range.

The JCUMote uses a single antenna to both transmit and receive. When transmitting, the 1 W power output is potentially damaging to the receiver front-end. Therefore, an isolation network was developed, which prevents the transmitted signal reaching the receiver circuitry.

The modulation hardware was configured for a frequency deviation of 38.4 kHz, so that the 9600 bps Manchester encoded data occupies a 150 kHz channel bandwidth. This allows 2 channels in the 40.66 – 41.0 MHz band, if required. The transmitted frequency spectrum is shown in Fig. 3. To protect the spectrum analyser, a 20 dB attenuator was used.

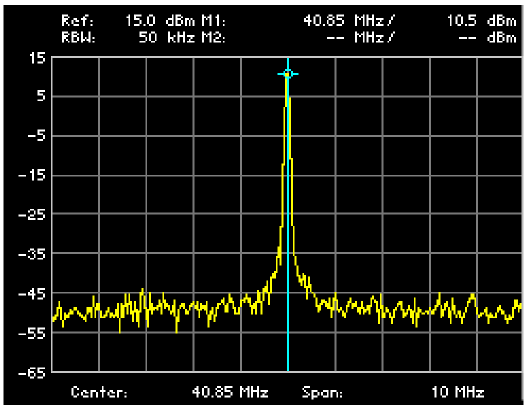


Figure 3. Frequency spectrum (20 dB attenuated)

For modulation, the digital waveform is injected into the control voltage input of the VCO (voltage controlled oscillator) of the Melexis TH7122. This form of modulation does not allow data with a DC component to be transmitted due to the feedback mechanism of the PLL. Hence, Manchester encoding was used. Since the TH7122 only provides a basic bit-level interface for transmitted and received data, the Manchester encoding was performed using algorithms in TinyOS.

The receiver was measured to have a sensitivity of -80 dBm at a 3×10^{-3} bit error rate, 20 kbps data rate and 20 kHz frequency deviation. At 40 MHz the background noise level on the outskirts of a suburban area was measured to be -95.8 dBm over a 100 kHz bandwidth, which is far higher than the sensitivity quoted for the Melexis TH7122, so that the range of the transmitter is primarily determined by man made noise. Fig. 4 shows the JCUMote PCB.

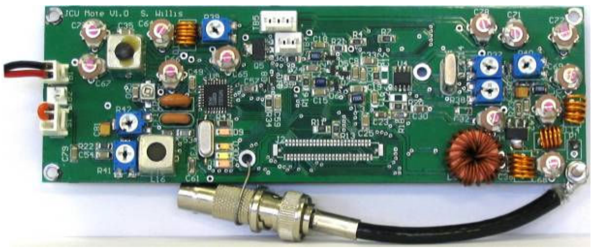


Figure 4. The JCUMote PCB

The design of the antenna was also investigated. The aim of this design was to be low in cost, highly efficient and suitable for WSNs. Nodes in a WSN must communicate with nodes in all directions, hence the antenna must be omni-directional. The final design consists of a quarter-wavelength whip antenna surrounded by 4 quarter-wavelength radials at a 30° declination to the horizontal (Fig. 5). This antenna has an input impedance of $49 + j3\ \Omega$, with a return loss of -31 dB at 40.84 MHz.

The JCUMote is powered by a 6 V lead-acid battery and has been tested with a solar panel. The node is housed in an aluminium enclosure as shown in Fig. 5. The antenna is mounted to the top of the enclosure with the radials extending from the base.

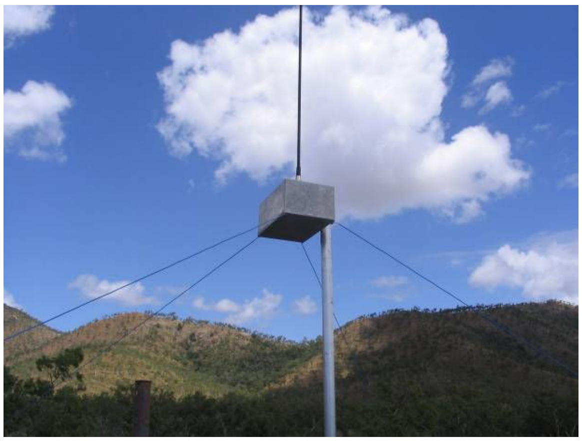


Figure 5. Node installed with radials

IV. Field Testing

A. Rural Environment

The range of the nodes was determined in a rural setting by measuring the received signal strength at set distances from the transmitter. A transmitting node was programmed using TinyOS to produce a simple known data stream, four times per second. A receiving node was programmed with a TinyOS application which logged the received signal strength, noise level and packet loss to a connected laptop computer.

The transmitter was installed at the top of a mountain range, which was 290 m above the final receiver testing position. The transmitter was mounted on a 1.8 m star-picket, surrounded by four quarter-wavelength radials. Periodic measurements were taken at the locations shown in Fig. 6, using the receiving node mounted on a car roof. The maximum range was measured to be 13.2 km.

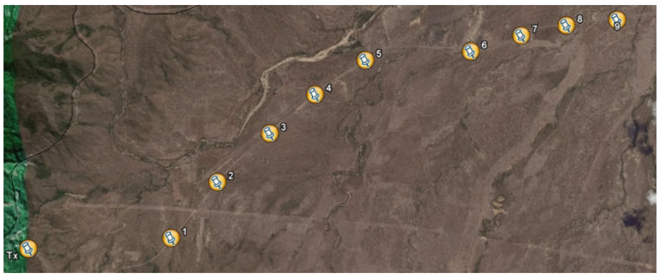


Figure 6. Receiver Test Positions (Google Earth)

The JCU-WSN model was used to calculate the mean signal strength (taken over 1000 runs) for cases where there are between 0 and 5 multipath reflections. A comparison between the JCU-WSN model predictions and the measured signal strength are shown in Table I. It is obvious that in all cases the model is most accurate for the case of a direct ray and one ground reflection only. This would be the expected case, because there are no buildings or large terrain features in the test area that may cause reflections.

TABLE I. Measured and Predicted Signal Strength

Posn.	**pR$_{meas}$ (dBm)**	**Dist. (km)**	**WSN Model (mean dBm)**			**Plane-Earth Model (dBm)**
			Num. reflections			
			0	***1***	***2***	
1	-56.3	2.8	-46.2	**-54.6**	-54.5	-55.1
2	-80.3	4.0	-50.6	**-61.0**	-60.9	-60.6
3	-67.3	5.5	-54.3	**-66.8**	-66.6	-65.8
4	-70.4	6.7	-56.5	**-70.4**	-70.2	-69.1
5	-73.8	8	-58.5	**-73.6**	-73.2	-72
6	-77.2	10.1	-60.9	**-77.6**	-77.0	-76
7	-76.3	11.2	-63.5	**-81.0**	-80.4	-77.7
8	-77.5	12.2	-64.7	**-82.8**	-81.8	-79.2
9	-84.2	13.2	-65.6	**-84.2**	-83.1	-80.4

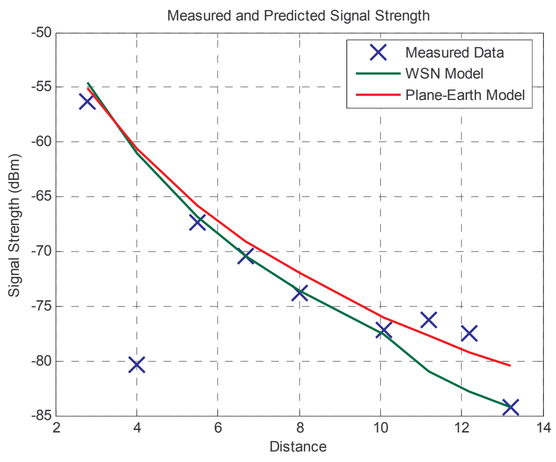


Figure 7. Comparison of Results for Rural Tests

A comparison between the measured results, the JCU-WSN model with 1 ground reflection, and the plane-earth model is shown in Fig. 7. It is evident that the JCU-WSN model gives the closest predictions. Fig. 7 shows an anomaly at 4 km, which is due to a line of sight obstruction that was not evident in the terrain data. At points 7 and 8, the measurements are higher than predicted. However these results are within the statistical variations observed for the very flat country at these points.

B. *Suburban WSN Testing*

Four nodes were constructed and installed in a suburban area to test the nodes as an ad-hoc network. A TinyOS application was written which allowed each node to log the received signal strength of its neighbours and forward the measurement to a base station node every eight seconds. The base station was connected to a PC running Crossbow's MoteView software [1] which logs all readings into a database. Over 930 000 measurements were logged during a period of 41 days.

The base station node was installed on the roof of the Electrical & Computer Engineering building at James Cook University at an elevation of 10 m above ground. Three other nodes were installed on houses in a neighbouring suburb on top of existing television antennae, approximately 4 m above ground. The full network is shown in Fig. 8 where node 0 is the base station. The link between nodes 0 and 4 is 3.1 km long and is shown as a dashed line because that path is obstructed by a small hill, so that the link only operated occasionally.

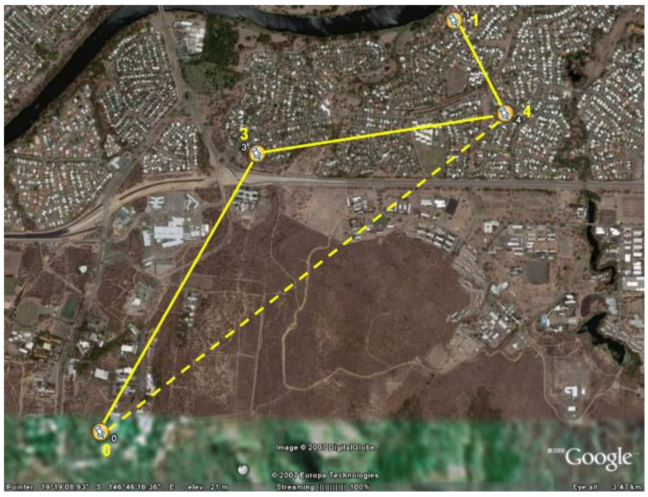


Figure 8. Suburban Wireless Sensor Network (Google Earth)

The measured signal strength for each link is shown in Table II, where pR_{meas} is the measured signal strength and pR_{mean} is the mean predicted signal strength after 1000 simulations of the JCU-WSN Model. For each link the optimum number of reflections is highlighted in grey. The worst prediction was within 2.5% of the measured value (4→1), all the others were more accurate. It is obvious that in a suburban environment there are a larger number of reflections than in a rural environment. This is to be expected due to the presence of surrounding buildings. The received power measurements show that links are asymmetrical, due to diffraction. For example, the link between nodes 3 and 4 has a difference of 2 dB in each direction. Note: Node 1 emits 4 dB less power than other nodes.

V. PREDICTING THE NUMBER OF MULTIPATH SIGNALS

Table II showed that the number of multipath signals due to reflection and diffraction has a strong influence on the accuracy of the model. It is also evident that the number of multipath signals is difficult to determine due to the paths being partially obstructed by houses, causing significant diffraction.

To enable the number of multipath signals to be estimated an empirical model was developed based on the results presented in Table II. The model uses three parameters to determine the number of multipath signals: 1) the number of buildings in the Fresnel zone, 2) the height of the receiving node and 3) the distance of the closest building to the receiver.

Table III shows the number of multipath signals (ground reflection excluded) for each link and the number of buildings in the Fresnel zone (counted using Google Earth). Generally, links with fewer buildings have less multipath signals.

Table III also includes the receiver height. It implies that when a receiver is elevated it receives additional multipath signals. An example is the link between nodes 0 and 4. When the higher node (0) is receiving, more multipath components are present than when the lower node (4) is receiving. However the shape of the hill partially obstructing that path and the diffraction that causes may be more dominant.

Table III also shows the distance from the receiver to the closest building in the Fresnel zone. It was noticed that additional multipath signals are received when a building is in close proximity to the receiver. An example of this is the link between nodes 3 and 4. Node 3 is closer to a building (10 m) than node 4 (50 m). When node 3 is receiving, more multipath signals are present than when node 4 is receiving.

TABLE II. MEASURED AND PREDICTED SIGNAL STRENGTH (SUBURBAN TESTS)

Link	**pR_{meas} (dBm)**	**Dist. (km)**		**WSN Model**						**Plane-Earth model**
			Num. signals	***0***	***1***	***2***	***3***	***4***	***5***	
0→3	-67.3	1.9	pR_{mean} (dBm)	-47.4	-69.5	**-67.8**	-66.1	-64.6	-63.3	-54.7
0→4	-77.0	3.1	pR_{mean} (dBm)	-53.2	-80.4	**-77.1**	-74.0	-71.8	-70.3	-64.4
1→3	-73.5	1.4	pR_{mean} (dBm)	-52.8	-81.8	-75.7	**-72.0**	-69.4	-68.4	-66.1
1→4	-68.6	0.6	pR_{mean} (dBm)	-40.2	**-68.8**	-64.2	-60.9	-59.0	-57.9	-61.0
3→0	-68.2	1.9	pR_{mean} (dBm)	-47.4	-69.6	**-68.0**	-65.9	-64.4	-63.7	-54.7
3→4	-67.7	1.5	pR_{mean} (dBm)	-47.8	-78.0	-71.2	**-67.8**	-65.7	-64.3	-63.7
4→0	-75	3.1	pR_{mean} (dBm)	-54.2	-81.4	-78.4	**-75.0**	-73.6	-71.9	-64.4
4→1	-62.9	0.6	pR_{mean} (dBm)	-40.2	**-64.5**	-59.8	-56.4	-54.4	-53.2	-61.0
4→3	-66.2	1.5	pR_{mean} (dBm)	-48.4	-78.6	-72.3	-67.9	**-65.6**	-64.3	-63.7

TABLE III. SUBURBAN LINKS SHOWING NUMBER OF BUILDINGS AND THE CLOSEST BUILDING

Num. Signals	Link	Dist. (km)	Num. Buildings	Closest Building (m)	Closest Building (%)	Rx Height (m)
0	1→4	0.6	9	40	6.7	3.6
	4→1	0.6	9	80	13	4
1	0→3	1.9	9	40	2.1	4.8
	3→0	1.9	9	120	6.3	13
	0→4	3.1	11	60	1.9	3.6
2	4→0	3.1	11	120	3.9	13
	1→3	1.4	16	10	0.71	4.8
	3→4	1.5	23	50	3.3	3.6
3	4→3	1.5	23	10	0.67	4.8

These observations were used to create an empirical model. The model uses three steps. The first step applies a building correction multiplier to account for the immediate buildings surrounding the receiver. The closeness of a building is expressed as a percentage relative to the path length. If the relative closeness is greater than 5%, then the node is classed as 'open'. Conversely, if the relative closeness is less than 2 % then the node is classed as 'confined'. To compensate, the number of buildings is reduced or increased by 20% for open and confined nodes, respectively.

Following the building compensation, the model calculates the number of multipath signals by applying the corrected number of buildings to equation (2). This equation was developed as the line of best fit for all links of normal height (links with a receiver dramatically above or below the mean building height were ignored). The resulting line of best fit is shown in Fig. 9 and is defined by equation (2).

$$m \approx {}^{b}\!/_{8} - 0.6 \qquad (2)$$

Where m is the number of multipath signals and b is the corrected number of buildings on the path.

The final stage of the model compensates for elevated receivers by adding or subtracting 1 multipath signal for high or low receivers, respectively. A high receiver is classed as having an elevation greater than twice the average height of the surrounding buildings. Low receivers are classed as having a height less than half of the average surrounding buildings.

This model was implemented as a MATLAB function and has been tested to accurately predict the correct number of multipath components for all links in Table II, as well as for some further "car-roof" tests that were performed in the same suburb.

ACKNOWLEDGMENT

The authors wish to thank the staff at James Cook University for their support in manufacturing the hardware. The authors thank AWR for their University support program, which made it possible to use Microwave Office for hardware simulations. The Australian Research Council (ARC) is acknowledged for providing the grant for the equipment used for making the measurements of the hardware.

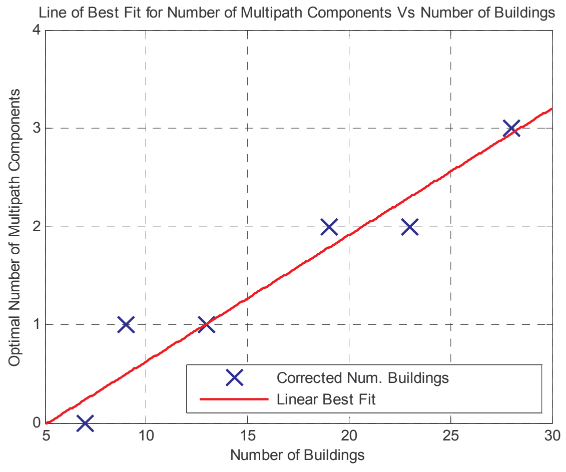


Figure 9. Line of Best Fit for Number of Multipath Signals Vs Number of Buildings

REFERENCES

[1] (2007, January 23). Crossbow Technology : Wireless Sensor Networks : Home Page [Online]. Available: http://www.xbow.com/Home/HomePage.aspx

[2] J. M. Hernando and F. Perez-Fontan, Introduction to Mobile Communications Engineering. Boston, USA: Artech House, 1999.

[3] H. Wong, (2004, November 04). Field Strength Prediction in Irregular Terrain - the PTP Model [Online]. Available: http://www.fcc.gov/oet/fm/ptp/report.pdf

[4] S. L. Willis and C. J. Kikkert, "Radio Propagation Model for Long-Range Ad Hoc Wireless Sensor Network," presented at 2005 International Conference on Wireless Networks, Communications and Mobile Computing, Maui, Hawaii, 2005.

[5] E. Ramirex, (2007, January 30). Shuttle Radar Topography Mission [Online]. Available: http://www2.jpl.nasa.gov/srtm/

[6] S. L. Willis and C. J. Kikkert, "Design of a Long-Range Wireless Sensor Node," presented at IEEE International Conference on Circuits and Systems, Singapore, 2006.

[7] (2004, June 22). TinyOS [Online]. Available: http://www.tinyos.net/

Appendix E Modulation Techniques

In order for an analogue or digital signal to be transmitted it must first be modulated onto a carrier wave. At the receiver, a corresponding demodulator is used to recover the original signal. The most commonly known modulation techniques are FM (frequency modulation) and AM (amplitude modulation). These techniques have been in use with common analogue radio systems for a long time. This report will focus on modulation techniques that are to be used for digital signals.

E.1 Amplitude Shift Keying

Kikkert [10] provided a summary of digital modulation techniques. The simplest technique is amplitude shift keying (ASK), which simply switches the carrier signal on or off corresponding to a 1 or 0 in the digital data. A problem with this scheme is that it momentarily switches off the amplifier which can cause nonlinearities in a class C amplifier. It is desirable to use a class C amplifier because it has high efficiency.

E.2 Frequency Shift Keying

Frequency Shift Keying (FSK) involves changing the frequency of the carrier in response to a 0 or 1 in the data. A 0 data bit will cause the carrier to switch to frequency f_1, whilst a 1 will cause the carrier to shift to frequency f_2. FSK can be used with a class C amplifier, because the signal has a constant amplitude.

E.3 Phase Shift Keying

The final basic modulation technique is Phase Shift Keying (PSK). In PSK the phase of the carrier is changed corresponding to a change in the input data. The simplest PSK scheme is Binary Phase Shift Keying (BPSK) where a change in the input data causes a 180^o phase change in the carrier signal.

E.4 QUADRATURE PHASE SHIFT KEYING

Quadrature Phase Shift Keying extends on BPSK by allowing four possible phase changes: 0^o, 90^o, 180^o and 270^o. Since there are four possible phase changes, then two data bits (a dibit) may be represented, as demonstrated by Table E-1.

TABLE E-1 QPSK PHASE SHIFTS

Dibit	Phase Shift
00	0^o
01	$+90^o$
11	$+180^o$
10	$+270^o$

One bit of the dibit causes the in-phase (I) component, whilst the other bit causes the quadrature (Q) component. The changes in phase may be represented on an I-Q diagram as shown in Figure E-1. The corners of the square represent the four possible phase positions and the blue lines represent changes between phases. Figure E-1 shows that some phase changes pass through the origin, which causes the amplifier to momentarily switch off. Therefore, QPSK is not suitable for use with a class C amplifier.

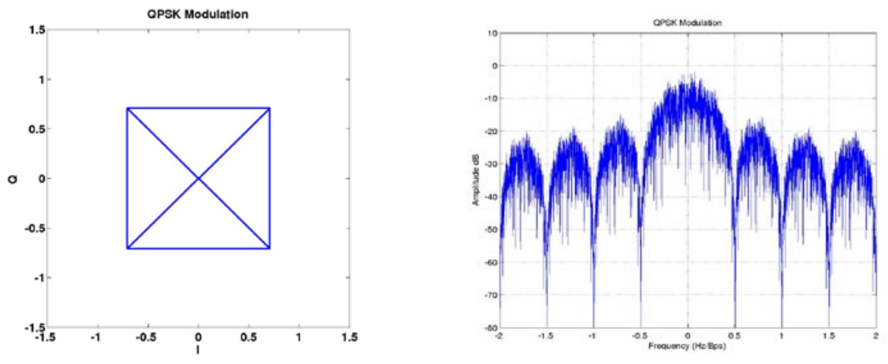


Figure E-1: QPSK I-Q Diagram and Spectrum [90]

Figure E-1 also shows the spectrum of the QPSK signal. It should be noted that QPSK requires a theoretical bandwidth of 0.5Hz/bps. Kikkert [10] stated that in practice this bandwidth is normally larger and may be reduced by using a filter. A common filtering technique is Nyquist filtering, also known as raised cosine filtering. The phase plane diagram and spectrum for a QPSK signal filtered with a Nyquist filter is shown in Figure E-2. It should be noticed that the I-Q diagram is now distorted, but the distinct phase points are still obvious.

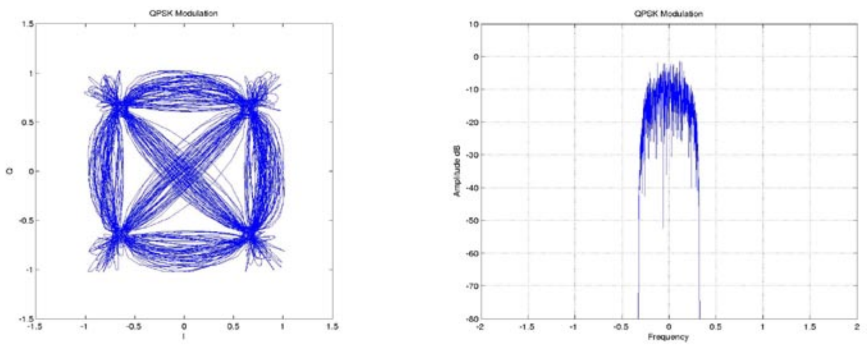


Figure E-2: Filtered QPSK I-Q Diagram and Spectrum [90]

In order to avoid 180^o phase changes, π/4 QPSK was introduced. This scheme has 8 locations in the phase plane with each location spaced by 45^o. The phase changes of this scheme are shown in Table E-2.

TABLE E-2: π/4 QPSK PHASE SHIFTS

Dibit	Phase Shift
00	$+45^o$
01	$+135^o$
10	$+225^o$
11	$+315^o$

The resulting I-Q diagram is shown in Figure E-3.

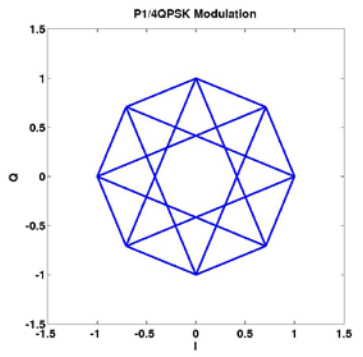


Figure E-3: π/4 QPSK Phase Plane Diagram [90]

An alternative method to avoid trajectories through the origin is to offset the transition of each bit of the dibit. If one bit changes before the other, then each bit can only cause a 90^o phase change at a time. This avoids 180^o phase changes, but doubles the number of changes that must occur. Figure E-4 shows the resulting I-Q diagram which demonstrates that there are no transitions through the origin. It should be noted that all QPSK schemes have a theoretical bandwidth utilisation of 0.5Hz/bps.

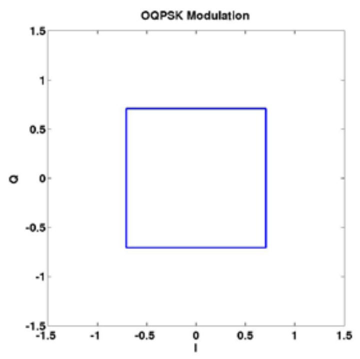


Figure E-4: OQPSK I-Q Diagram [90]

E.5 MINIMUM SHIFT KEYING

Another modulation scheme that avoids 180^o phase shifts is Minimum Shift Keying (MSK). The MSK scheme controls the rate of change of phase so that the correct I-Q location is just reached at the end of each data period. A constant rate of change of phase corresponds to a constant frequency. Since MSK produces slowly changing phase transitions it has less out of band energy. Kikkert [10] stated that 99% of its energy is within 1.2Hz/bps. This is demonstrated by the spectrum in Figure E-5. The I-Q diagram shows that there are no sharp phase changes and no transitions through the origin.

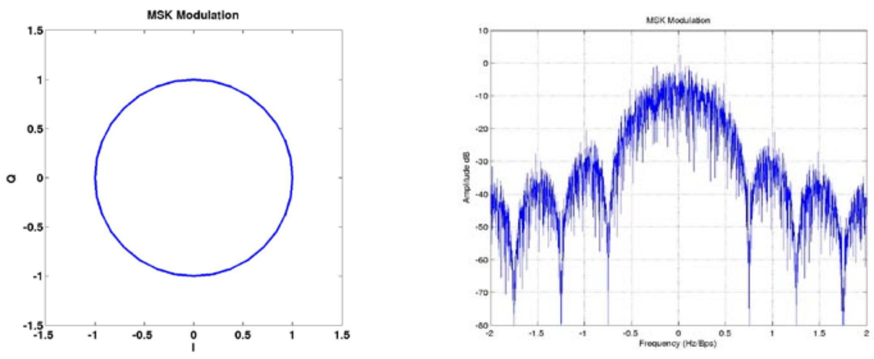


Figure E-5: MSK I-Q Diagram and Spectrum [90]

Filtering the data coming into the MSK modulator will reduce the out of band components, which is demonstrated by Figure E-6. By utilising a Gaussian filter, Gaussian Minimum Shift Keying (GMSK) is produced, which is used in GSM mobile telephones and has a normalised bandwidth of 0.738Hz/bps.

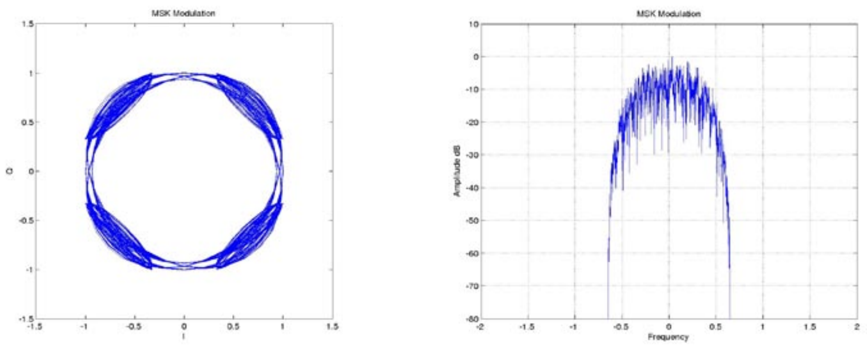


Figure E-6: GMSK I-Q Diagram and Spectrum [90]

E.6 Quadrature Amplitude Modulation

Quadrature Amplitude Modulation (QAM) alters the amplitude and phase of the carrier at the same time. This allows the I and Q data to be mapped directly onto the I-Q plane. By having more than two levels in each direction a larger amount of data may be transmitted. Figure E-7 shows the I-Q diagram for 64-QAM.

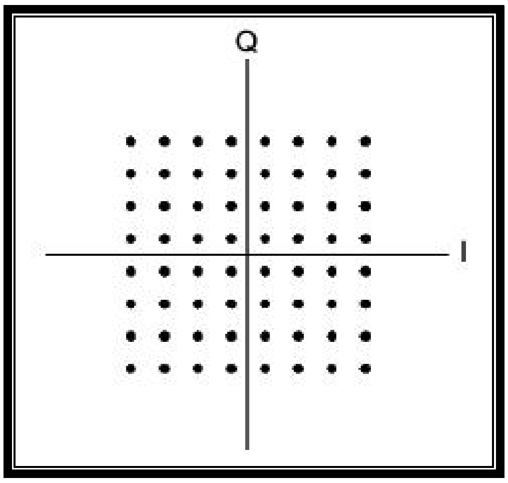


Figure E-7: 64-QAM I-Q Diagram [10]

By using more levels, a larger amount of data can be transmitted. However, by doing so the signal has lower noise immunity.

E.7 Spread-Spectrum Techniques

Spread-spectrum techniques spread the data signal over a broad range of frequencies. The spread-spectrum data is modulated using one of the conventional modulation techniques reviewed above. Common spread spectrum techniques are: frequency hopping, OFDM or CDMA [10].

E.7.1 Frequency Hopping

The Frequency Hopping technique involves changing the carrier frequency over time. There are two basic types of frequency hopping: slow hopping and fast hopping.

In slow hopping, the carrier frequency is changed slower than the data rate. This means that the bandwidth around each carrier is proportional to the data rate. The advantage of slow hopping is that any interference on a single carrier will only affect the data when that carrier is being used.

With fast hopping, the carrier frequency is changed faster than, or equal to the data rate. When the signal is demodulated, the average is taken from each carrier. Hence, giving better noise immunity.

E.7.2 OFDM and COFDM

Orthogonal Frequency Division Multiplexing (OFDM) or Coded Orthogonal Frequency Division Multiplexing (COFDM) refers to a system where the individual data bits of a word are transmitted on individual carriers. The carrier frequencies are chosen to be mutually orthogonal over one symbol period. The carriers are then modulated using a conventional technique such as PSK, MSK or QAM. RF interference such as multipath fading will tend to reduce the magnitude of one carrier. This means that one data bit will be constantly affected. To make the system robust against these errors, an error correction code is transmitted with the modulated data and a correction scheme is used to reduce the number of errors. This technique is referred to as Coded OFDM (COFDM) and in practice all OFDM schemes are coded.

E.7.3 Code Division Multiple Access

Code Division Multiple Access (CDMA) modulates the data with a set of codes so that the signal is spread over a large number of frequencies. The same code that is used by the transmitter to modulate the data must also be used by the receiver to demodulate the data. The spreading of the signal across the frequencies can be done using two techniques. The first technique, known as Frequency Hopping CDMA (FH-CDMA) uses the code to select the carrier frequency. The second technique called Direct Sequence CDMA (DS-CDMA) uses the code to directly modulate the data.

This technique is commonly referred to as just CDMA and is used in some mobile telephone systems. DS-CDMA operates by multiplying the data by a code before it is transmitted. Upon reception, the data is multiplied by the same code to obtain the original data. This system adds complexity to the hardware, because the transmitter and receiver must be time synchronised so that they use the same code.

Kikkert [10] stated that the clocking rate of the coded sequence is referred to as the "chip rate" and is normally many times the digital data rate. Hence, the chip rate determines the spectral bandwidth. This is demonstrated by Figure E-8. Kikkert [10] explained that if the chip rate is N, then the signal will be spread over a bandwidth of N. At the receiver, when the signal is demodulated, the energy that was spread over the bandwidth of N is now combined into a reduced spectrum by a factor of N and this hence improves the signal to noise ratio by a factor of N. This is known as Process Gain. At the receiver when any unwanted signals are multiplied by the coding sequence the resultant is zero. This means that the unwanted signals will be ignored and the system is hence very robust against interference.

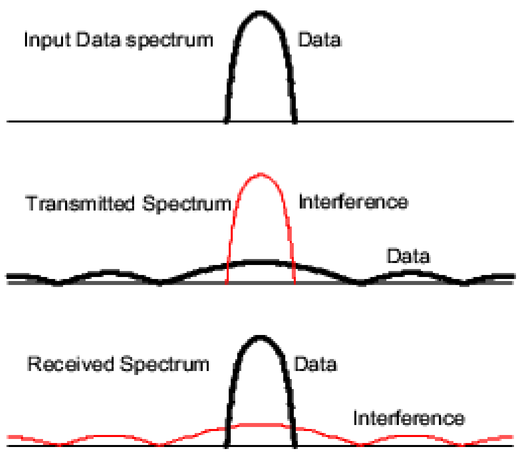
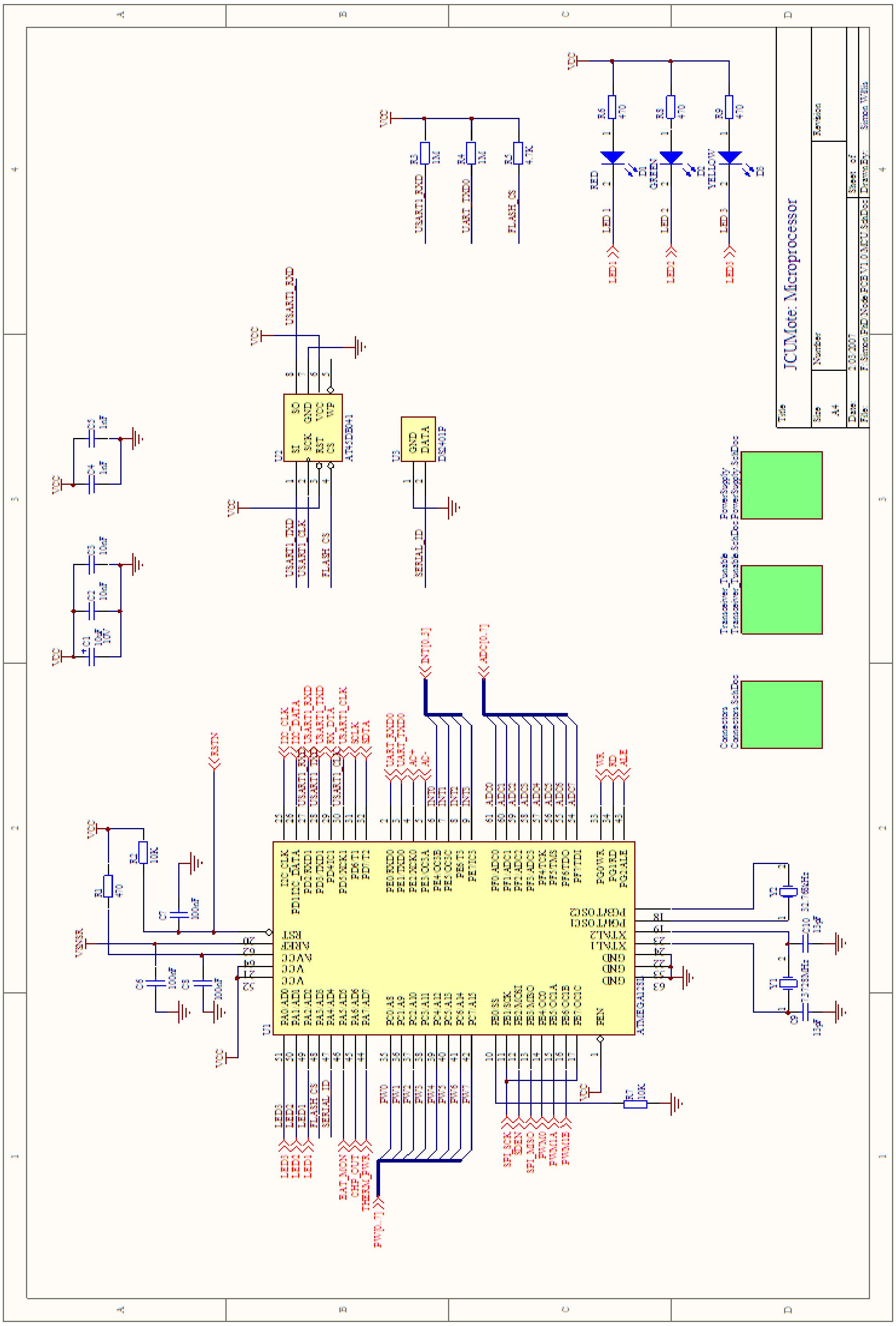
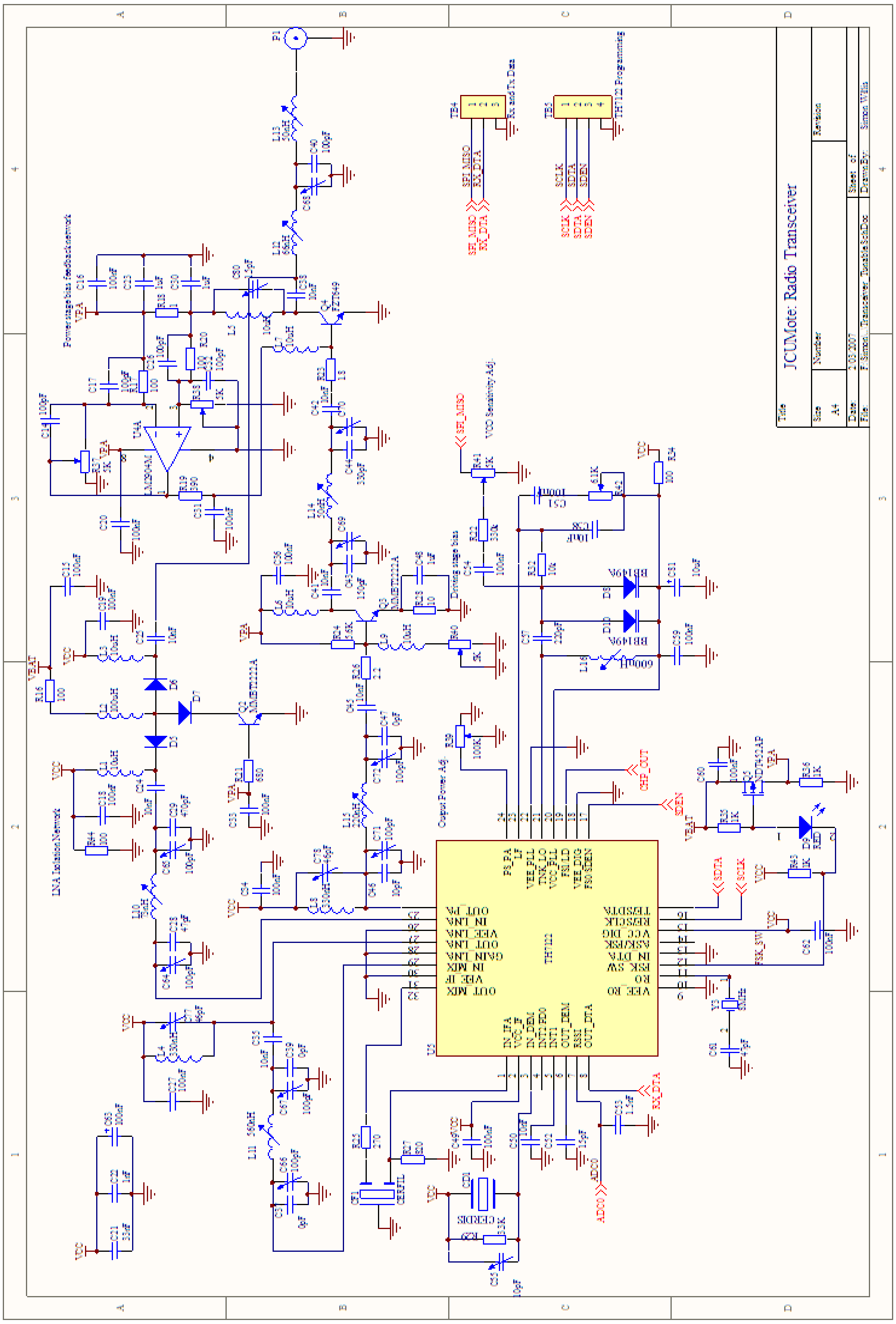
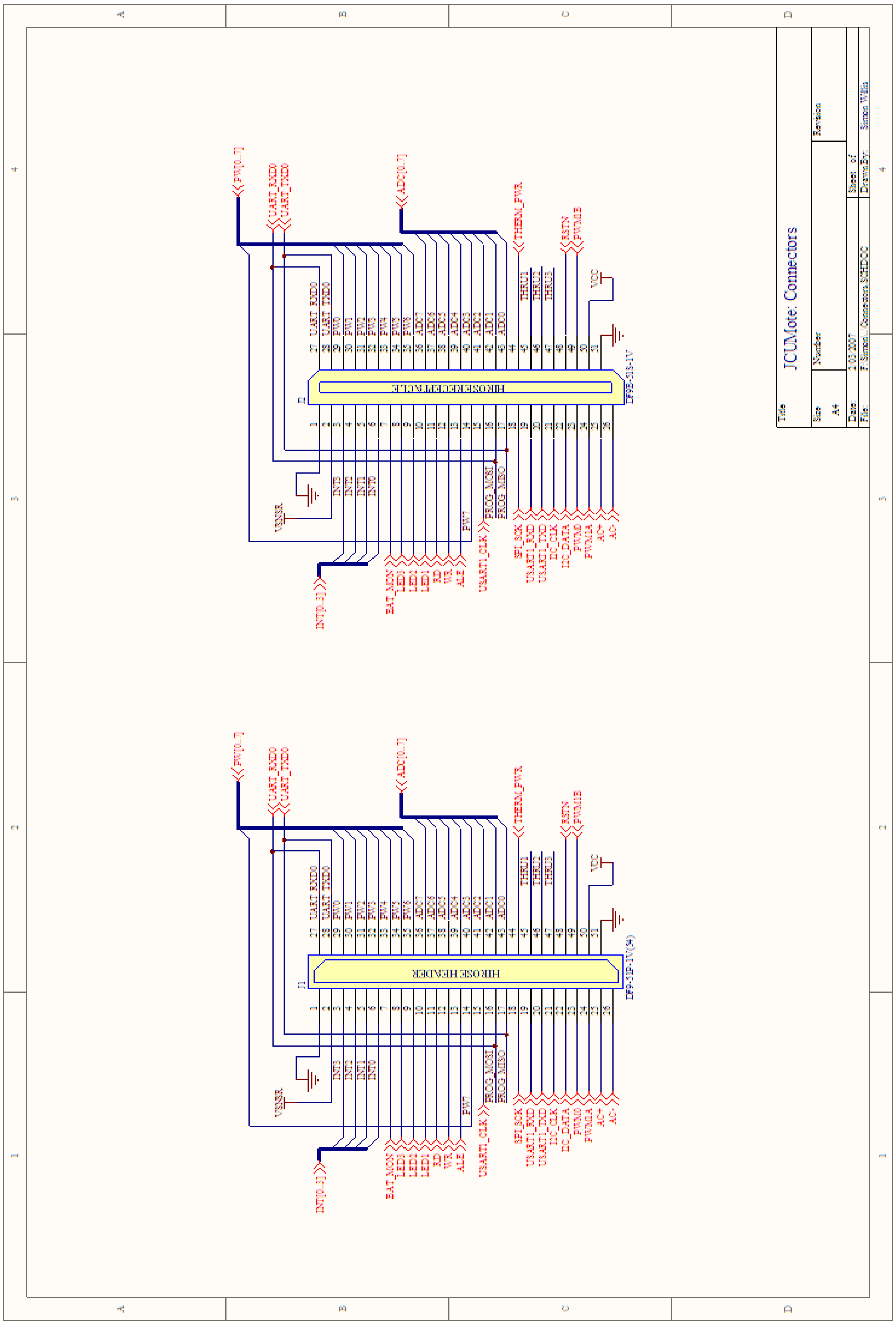


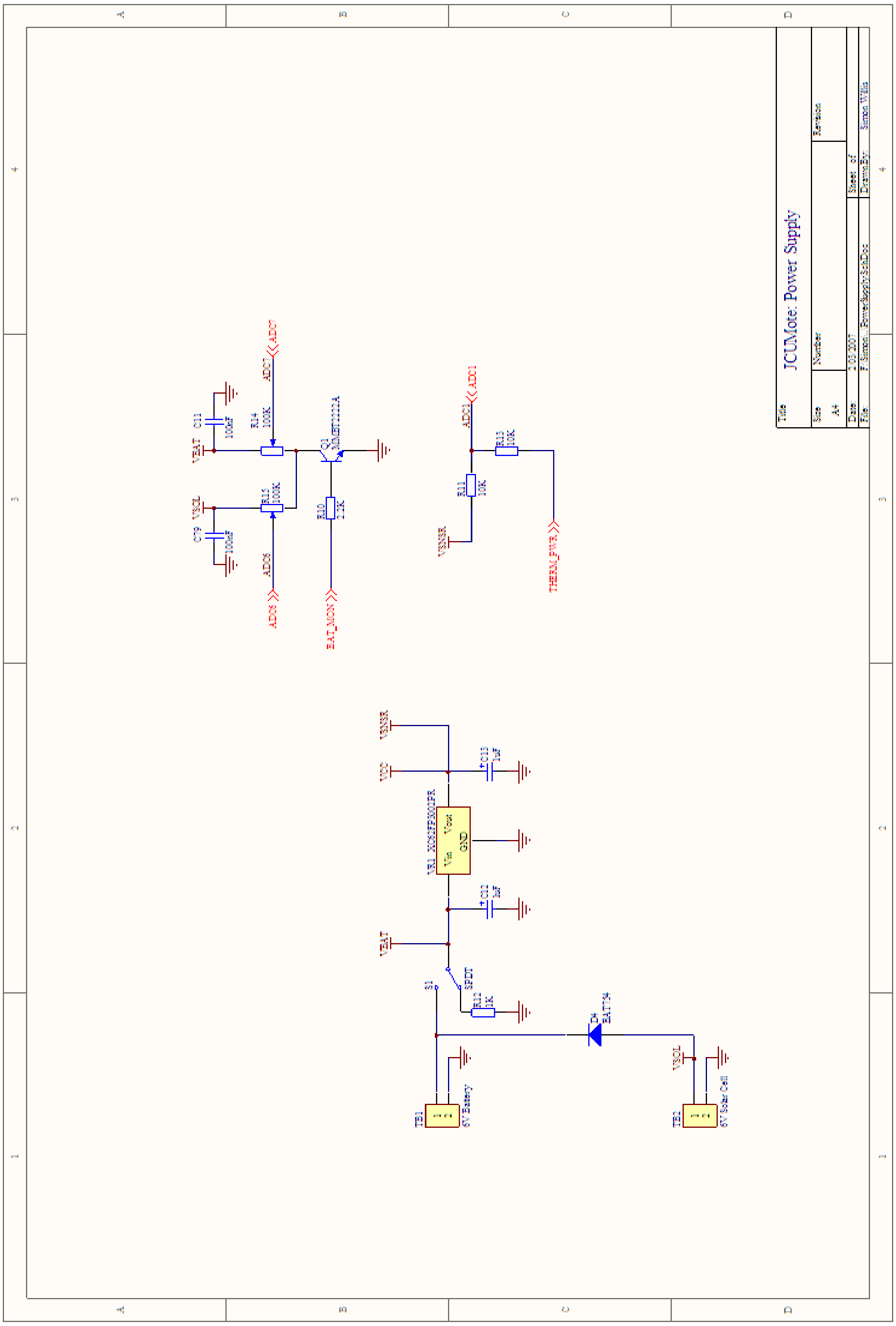
Figure E-8: CDMA Spectrum [10]

Appendix F JCUMote Schematic



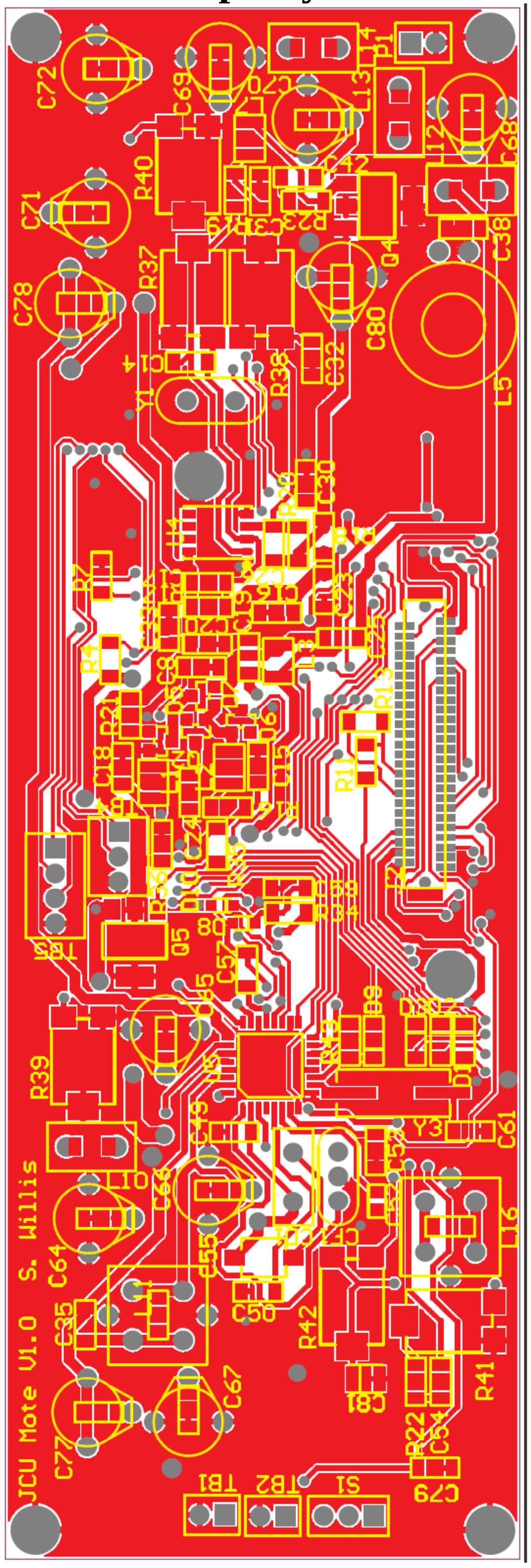

JCUMote: Radio Transceiver
TH7122
LNA Isolation Network
Power stage bias feedback network
Output Power Adj.
Driving stage bias
VCO Sensitivity Adj.
TH7122 Programming
Rx and Tx Data


JCUMote: Connectors
HIROSE RECEPTACLE
HIROSE HEADER
J1
J2
DF9B-51S-1V
DF9-51P-1V(54)
PW[0..7]
ADC[0..7]
UART_RXD0
UART_TXD0
INT[0..3]
THERM_PWR
RSTN
PWM1B
VCC
VSNSR
PROG_MOSI
PROG_MISO
USART1_CLK
BAT_MON
SPI_SCK
USART1_RXD
USART1_TXD
I2C_CLK
I2C_DATA
PWM0
PWM1A
AC+
AC-
A4

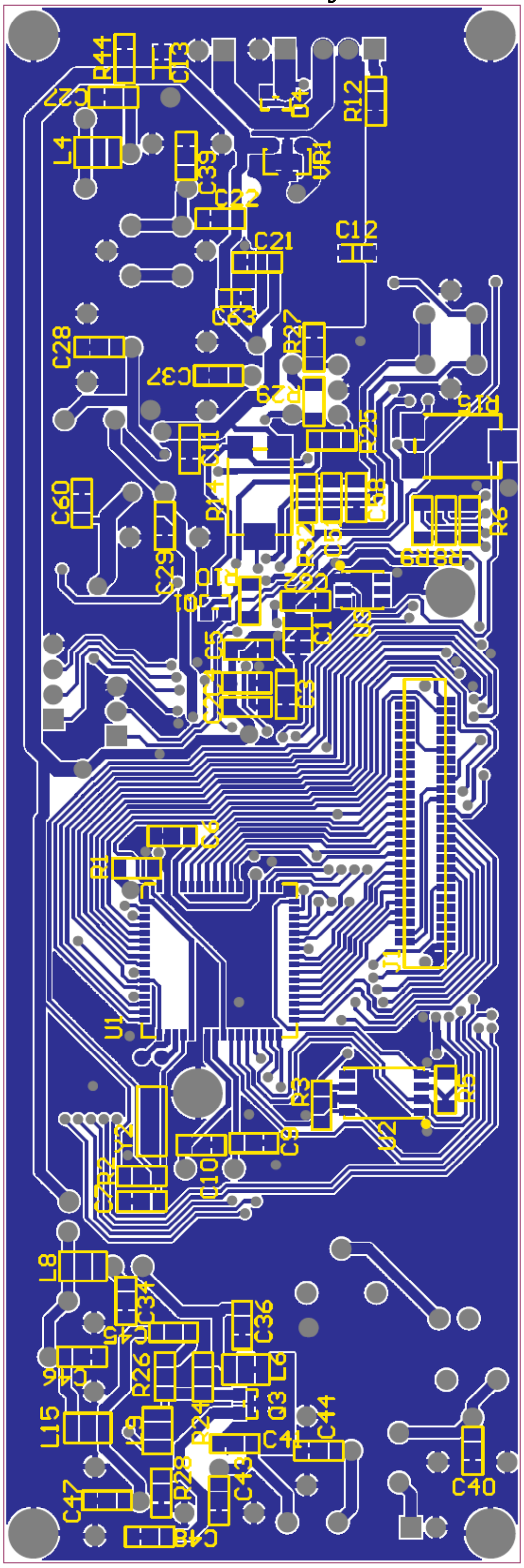

JCUMote: Power Supply
Title
Size
A4
Number
Revision
Date:
Sheet of
File:
Drawn By:
Samuel Willis
VBAT
C11
100nF
R14
100K
ADC7
VSOL
C79
100nF
R15
100K
ADC6
Q1
MMBT2222A
R10
2.2K
BAT_MON
ADC1
R13
10K
R11
10K
VSNSR
THERM_PWR
VSNSR
VCC
C13
1uF
VR1
Vin
Vout
GND
C12
1uF
VBAT
S1
SPDT
R12
1K
D4
BAT54
VSOL
TB1
6V Battery
TB2
6V Solar Cell

Appendix G JCUMote PCB

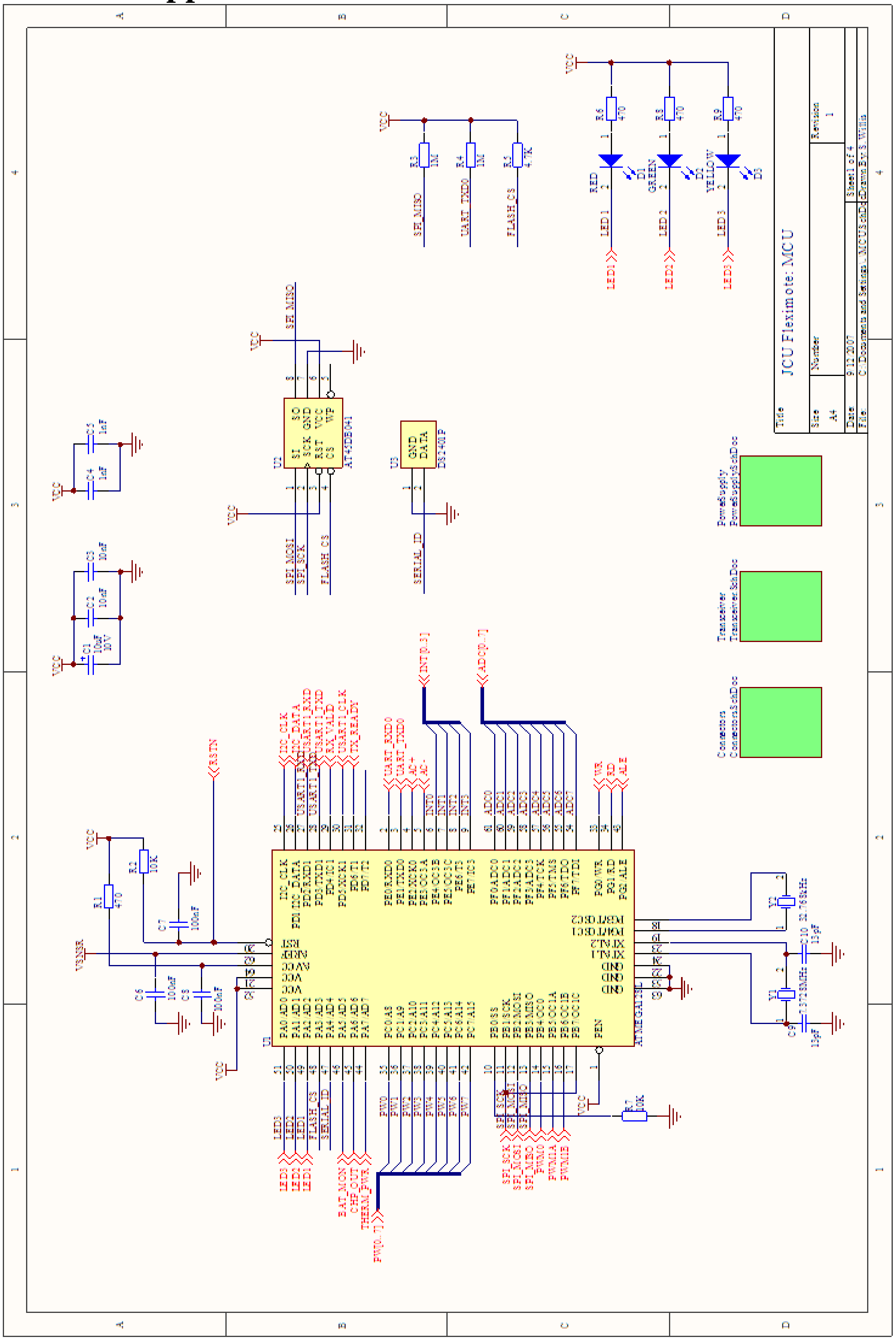
Top Layer

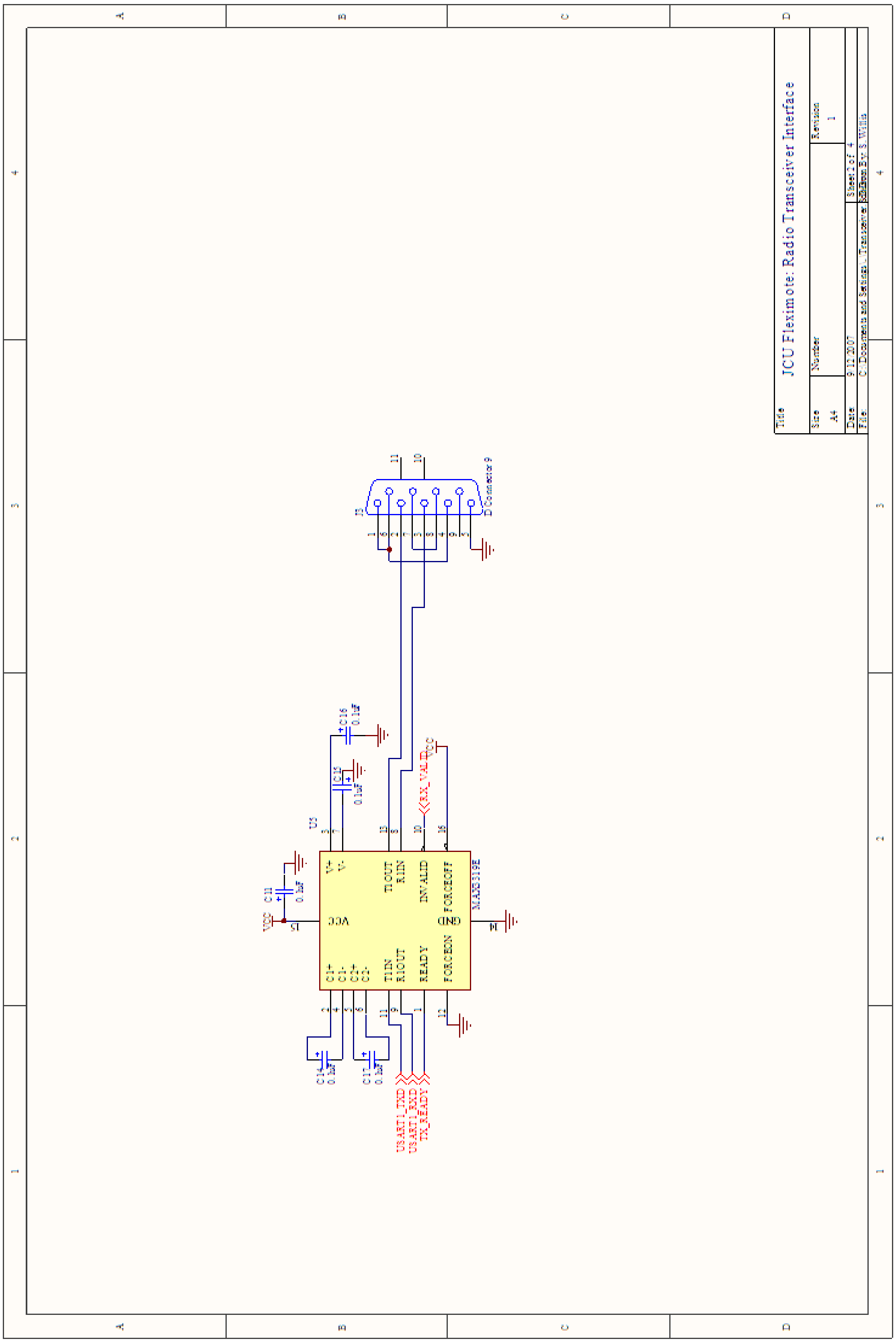


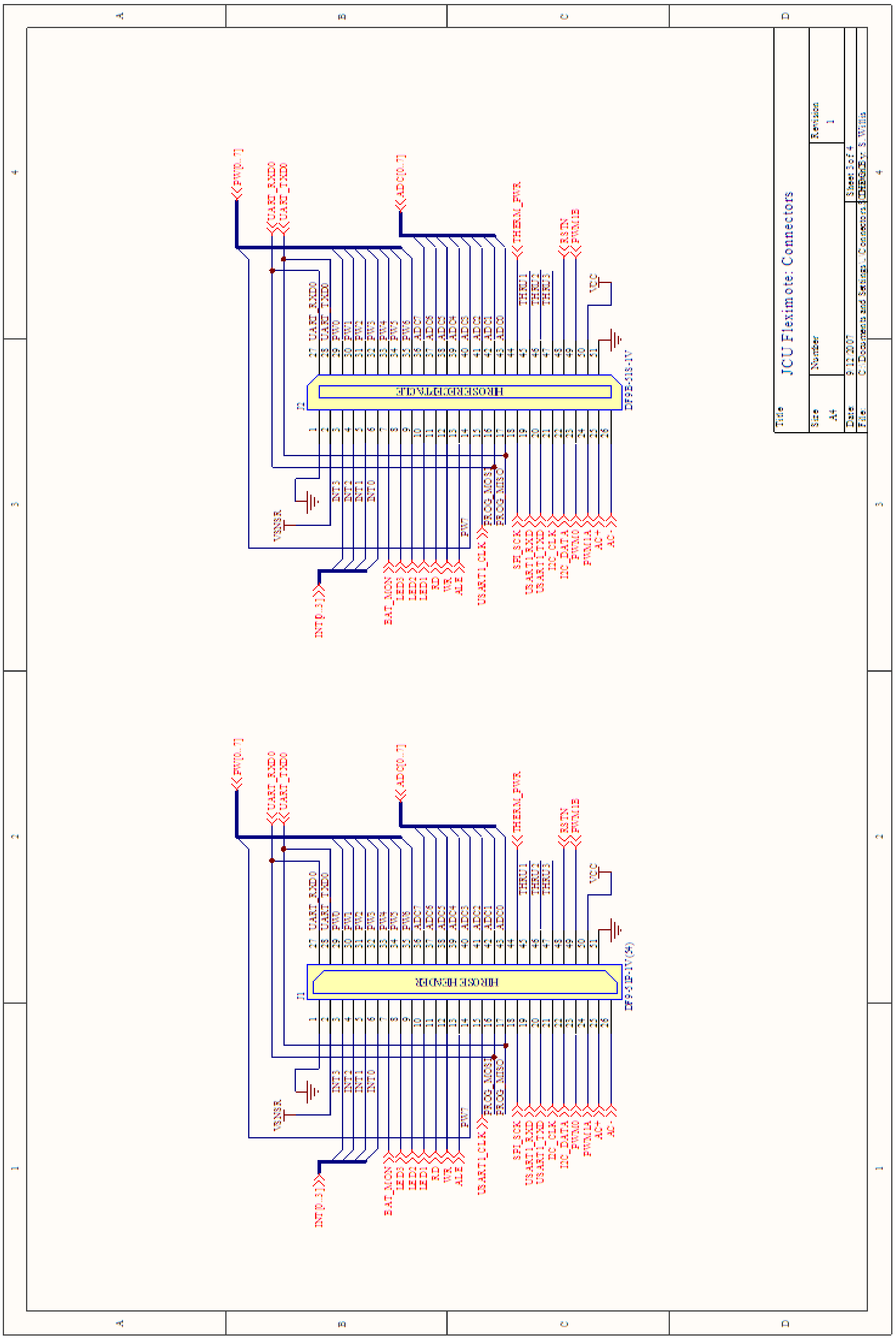
Bottom Layer

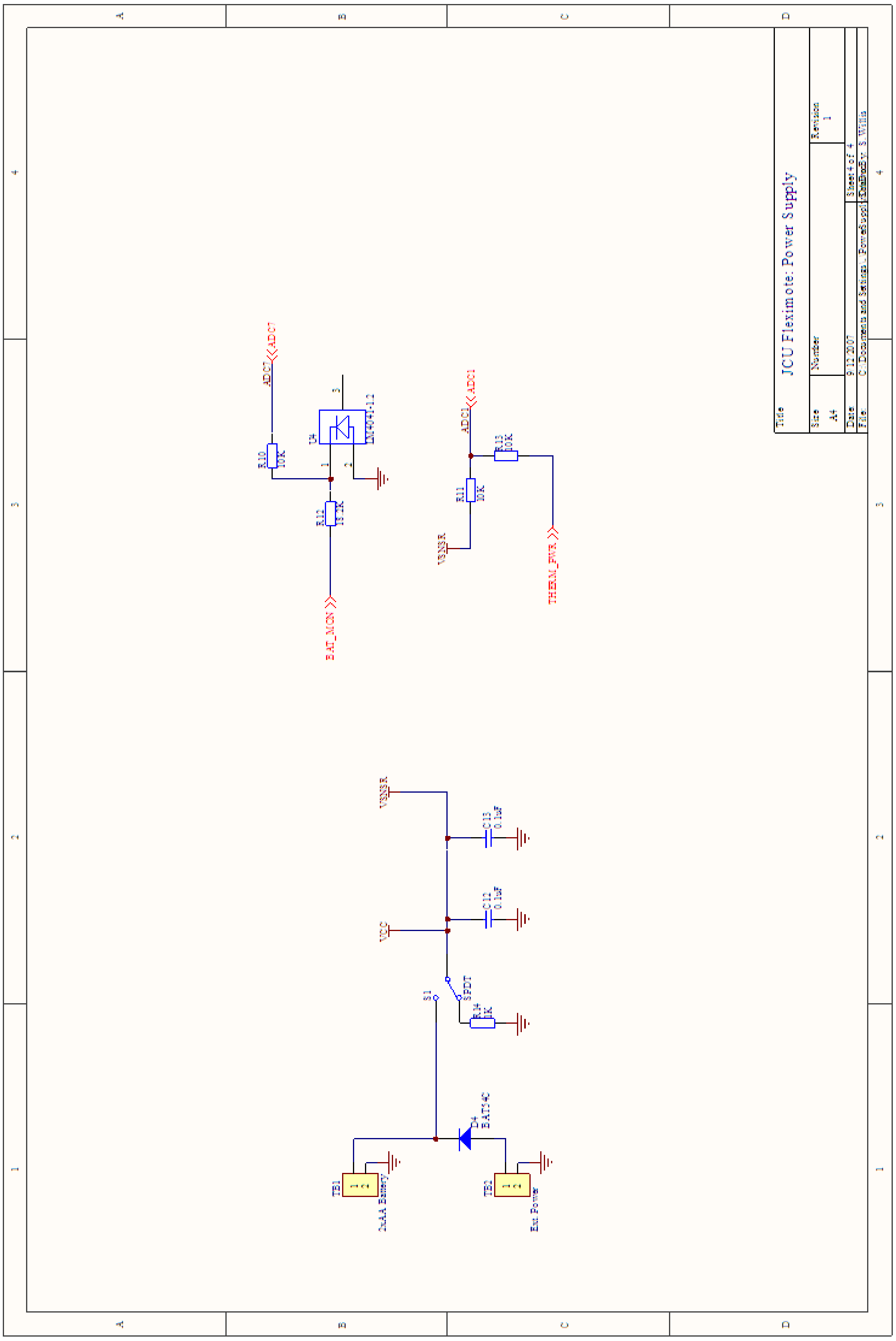


Appendix H JCU Fleximote Schematic

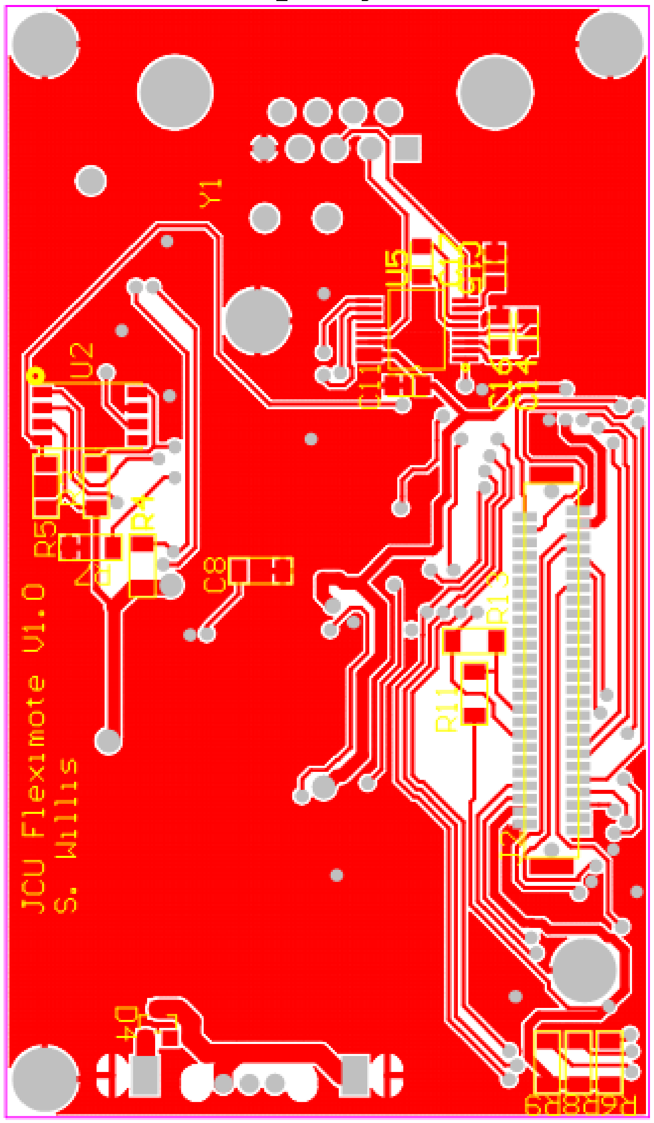



Title
JCU Fleximote: Radio Transceiver Interface
Size
A4
Number
Revision
1
Date
9/12/2007
Sheet 2 of 4
File
C:\Documents and Settings\...\Transceiver...
Drawn By: S. Willis
U3
MAX3319E
VCC
V+
V-
C1+
C1-
C2+
C2-
T1OUT
R1IN
T1IN
R1OUT
INVALID
READY
FORCEOFF
FORCEON
GND
C11 0.1uF
C13 0.1uF
C14 0.1uF
C15 0.1uF
C16 0.1uF
J5
D Connector 9
RX_VALID
USART1_TXD
USART1_RXD
TX_READY

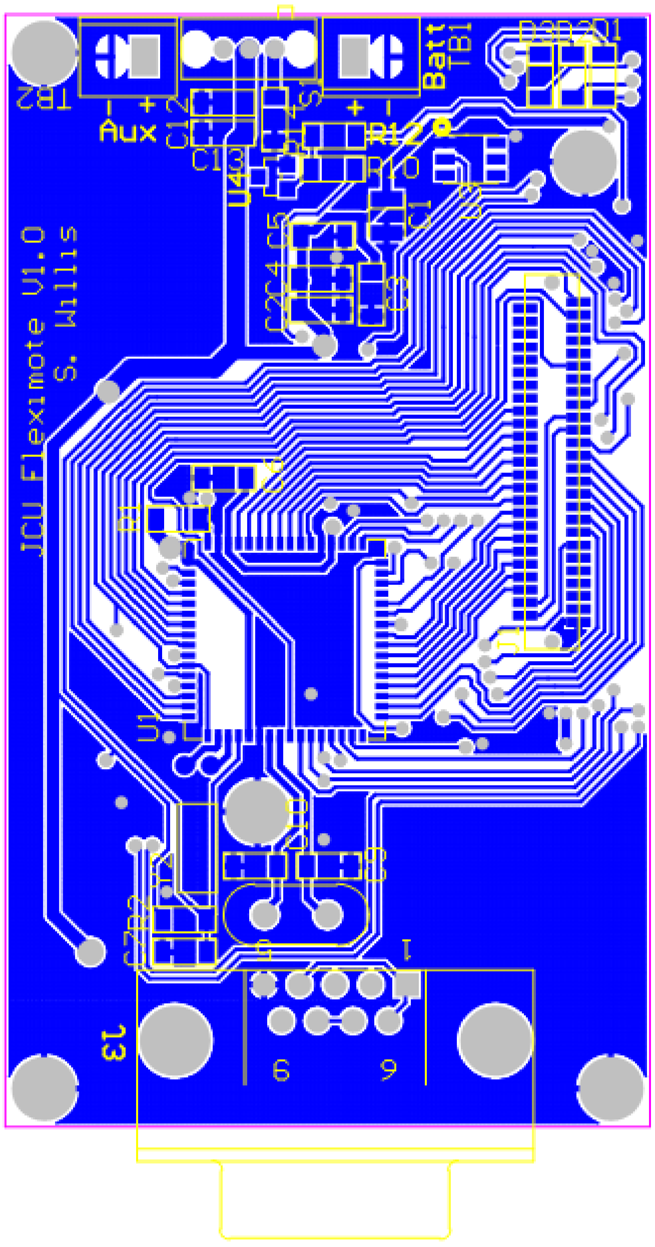

JCU Fleximote: Connectors
Title
Size A4
Number
Revision 1
Date 9/12/2007
Sheet 3 of 4
File C:\Documents and Settings\...\Connectors
Drawn By: S. Willis
HIROSE RECEPTACLE
DF9B-51S-1V
HIROSE HEADER
DF9-51P-1V(54)
J1
J2
FW[0..7]
UART_RXD0
UART_TXD0
ADC[0..7]
THERM_PWR
RSTN
PWR1B
VCC
VSNSR
INT[0..3]
BAT_MON
LED3
LED2
LED1
RD
WR
ALE
PW7
PROG_MOSI
PROG_MISO
USART1_CLK
SPI_SCK
USART1_RXD
USART1_TXD
I2C_CLK
I2C_DATA
PWR0
PWR1A
AC+
AC-
THRU1
THRU2
THRU3


JCU Fleximote: Power Supply
Title
Size
A4
Number
Revision
1
Date
9/12/2007
Sheet 4 of 4
File
C:\Documents and Settings\..\PowerSupply.SchDoc
Drawn By: S. Willis
ADC7
R10
10K
U4
LM4041-1.2
R12
19.1K
BAT_MON
ADC1
R11
10K
R13
10K
VSENSR
THERM_PWR
VCC
C12
0.1uF
C13
0.1uF
S1
SPDT
R14
1K
D4
BAT54C
TB1
2xAA Battery
TB2
Ext. Power

Appendix I JCU Fleximote PCB Top Layer



Bottom Layer



Appendix J TinyOS Components and Concurrency Model

J.1 Components

Applications in NesC consist of components that are linked together. Communication between components is specified by defined interfaces. Each component may be a provider or a user of one or more interfaces. An interface defines a set of functions called commands that are implemented by the provider. The interface also defines a set of functions called events that are implemented by the user.

There are two types of components in NesC: modules and configurations. Modules provide actual code and implement one or more interfaces. Configurations are used to connect the components together, connecting interfaces used by components to interfaces provided by others.

Operations in TinyOS are frequently split-phase and are started by an interface user calling a command function. The interface provider then commences the operation and signals the completion using an event.

An application can be represented as a component graph. On a component graph, configurations and modules are represented by boxes and interfaces are represented by an arrow which points towards the provider. Interfaces that are provided by a component, but not used, are also represented by a box.

Figure J-1 shows the component graph for a simple application called Blink which toggles the red LED every second. Blink is included in the TinyOS

tinyos-1.x/apps/Blink directory and consists of the Blink.nc configuration file and the BlinkM.nc module file. The configuration file specifies the wiring of the components and the module file implements the program.

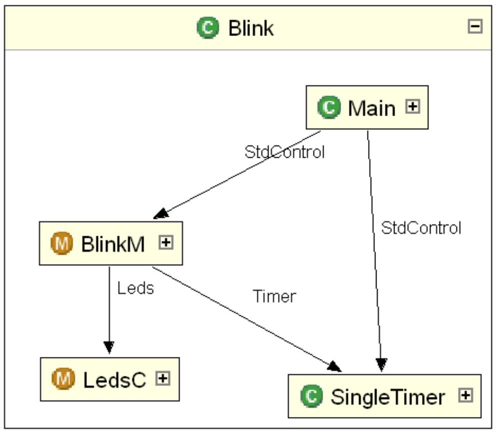


Figure J-1: Component Graph for the Blink Application (figure generated using [82])

The program code contained in the Blink.nc configuration file is shown in Figure J-2. The BlinkM module uses two interfaces: Timer and Leds and provides an interface called StdControl. This interface is used by the Main configuration and is also provided by the SingleTimer component. The StdControl interface contains three commands: init(), start() and stop(). When the microprocessor starts, the Main.StdControl.init() command is called, which causes the BlinkM and SingleTimer components to execute the code contained in the StdControl.init() command. This allows all components to execute any initialisation code before the program starts running.

```
configuration Blink {
}
implementation {
  components Main, BlinkM, SingleTimer, LedsC;
  Main.StdControl -> SingleTimer.StdControl;
  Main.StdControl -> BlinkM.StdControl;
  BlinkM.Timer -> SingleTimer.Timer;
  BlinkM.Leds -> LedsC;
}
```

Figure J-2: Configuration Code for Blink Application (Blink.nc)

Following this, TinyOS calls the `Main.StdControl.start()` command which starts the program. Figure J-3 shows the `BlinkM` code for `StdControl.start()`, which starts a timer that fires every second. A `StdControl.stop()` command is also provided which allows a component to be stopped.

```
command result_t StdControl.start() {
  // Start a repeating timer that fires every 1000ms
  return call Timer.start(TIMER_REPEAT, 1000);
}
```

Figure J-3: BlinkM.nc code for StdControl.start()

`BlinkM` uses a timer which generates a `Timer.fired()` event. The event handler (Figure J-4) uses the `redToggle()` command of the `Leds` interface to toggle the LED.

```
event result_t Timer.fired() {
  call Leds.redToggle();
  return SUCCESS;
}
```

Figure J-4: Timer.fired() event in BlinkM

J.2 Concurrency Model

In section 5.1.1 it was stated that TinyOS provides two threads of execution called tasks and hardware event handlers. Tasks have a deferred execution that is controlled by the scheduler. Tasks run to completion once started and do not pre-empt other tasks or events. Hardware event handlers are executed in response to a hardware event. They may pre-empt another task or event and always run to completion.

Hardware event handlers should finish execution as quickly as possible because they interrupt other task or events. If a hardware event handler requires complex calculations then a task can be used to perform the calculation at a later time. This allows the hardware event handler to end quickly so that the processor may resume executing the interrupted process.

Tasks are generated using the `post` command and are often used for complex operations such as processing sensor readings after they have been returned by the analogue to digital converter (ADC). As a trivial example, Blink could use a task to operate the red LED, as shown in Figure J-5.

```
task void toggleLed() {
  call Leds.redToggle();
}

event result_t Timer.fired() {
  post toggleLed();
  return SUCCESS;
}
```

Figure J-5: Example program using a Task

Appendix K MATLAB Code of Wireless Sensor Network Propagation Model

K.1 ANALYSESITE.M

```
%File: AnalyseSite.m
%Author: Simon Willis (simon.willis@jcu.edu.au)
%Date: 28/02/2005
%Modified: 14/02/2007
%
%This file carries out a study of the radio propagation on a given site.
%The site will contain nodes specified by letters (eg A -> H). This file
%uses AnalyseLink.m to find the propagation characteristics of each link
%Inputs: -site: Name of the site. Program looks to ..\..\ for directory
%           with same name.
%        -startNode: Name of first node
%        -endNode: Name of last node. Program will look in site directory
%           for terrain profile. Files must be of form for A_B.xz
%           where A is the first node and B is the second node
%        -numPathsMax: Maximum number of multipath signals to simulate
%
%This program stores all results in global variables. These can later be
%activated using ActivateGlobals.m
%
%---------------------------------------------------------------------------
function AnalyseSite (site, startNode, endNode, numPathsMax)

tic
for start = startNode:endNode
    for stop=start:endNode
        if (start~=stop)
            disp([setstr(start) ' <--> ' setstr(stop)]) %Display current link
            i=start-startNode+1;
            j=stop-startNode+1;
            %Create global variables (available after function execution)
            eval(['global SNRdirect' setstr(start) ' pDirect' setstr(start) '
dLossDirect' setstr(start) ' d' setstr(start) ' SNRmean' setstr(start) ' pR'
setstr(start) '_' setstr(stop) ' offsetFit' setstr(start) ' stdevFit' setstr(start) '
aFit' setstr(start) ' diffFit' setstr(start) ' SNR' setstr(start) '_' setstr(stop)])
            eval(['global SNRdirect' setstr(stop) ' pDirect' setstr(stop) '
dLossDirect' setstr(stop) ' d' setstr(stop) ' SNRmean' setstr(stop) ' pR' setstr(stop)
'_' setstr(start) ' offsetFit' setstr(stop) ' stdevFit' setstr(stop) ' aFit'
setstr(stop) ' diffFit' setstr(stop) ' SNR' setstr(stop) '_' setstr(start)])
            %Call AnalyseLink for forward direction
            eval(['[SNRdirect' setstr(start) '(j,:), pDirect' setstr(start) '(j,:),
dLossDirect' setstr(start) '(j,:), d' setstr(start) '(j,:), SNRmean' setstr(start)
'(j,:), pR' setstr(start) '_' setstr(stop)  ', offsetFit' setstr(start) '(j,:),
stdevFit' setstr(start) '(j,:), aFit' setstr(start) '(j,:), diffFit' setstr(start)
'(j,:),' ' SNR' setstr(start) '_' setstr(stop) '] =
AnalyseLink(site,setstr(start),setstr(stop),numPathsMax,1);'];
            %Call AnalyseLink for revese direciton
            eval(['[SNRdirect' setstr(stop) '(i,:), pDirect' setstr(stop) '(i,:),
dLossDirect' setstr(stop) '(i,:), d' setstr(stop) '(i,:), SNRmean' setstr(stop)
'(i,:), pR' setstr(stop) '_' setstr(start) ', offsetFit' setstr(stop) '(i,:),
stdevFit' setstr(stop) '(i,:), aFit' setstr(stop) '(i,:), diffFit' setstr(stop)
'(i,:),' ' SNR' setstr(stop) '_' setstr(start) '] =
AnalyseLink(site,setstr(stop),setstr(start),numPathsMax,1);']);
        end
    end
end
toc %Display computation time
```

K.2 ANALYSELINK.M

```
% File: AnalyseLink.m
% Author: Simon Willis (simon.willis@jcu.edu.au)
% Date: 28/02/2005
% Modified: 26/02/2007
%
% The function will analyse the radio propagation between two points. Each
% point is represented as a character (eg. A and B). This function will use
% the distance/elevation file given as A_B.xz.
%
% This program calculates the distance, signal power, diffraction loss and
% SNR for the direct path. It also calculates the SNR (distribution and
% mean) and fits a Rayleigh, Rician or Weibull distrubution to this using a
% least-squares fit (if relevant section uncommented).
%
% Inputs:   -site: Name of site (see AnalyseSite.M for explanation)
%           -pt1: First node
%           -pt2: Second node
%           -numpathsMax: Maximum number of multipath signals to simulate
%           -diffractionOn: boolean value. Enables the PTP model
%               diffraction loss calculations
%           -data: Terrain data (if not defined the program will load
%               terrain data from a file)
%
% Outputs:  -SNRdirect: SNR of direct signal
%           -pDirect: Strength of direct signal
%           -dLossDirect: Diffraction loss of direct siganl
%           -d:distance between nodes (km)
%           -SNRmean: Mean SNR with given number of multipath signals
%           -pRdB: Received signal strength for each simulated scenario
%               (dB)
%           -pRmean: Mean received signal strength with given number of
%               multipath signals
%           -offsetFit: Offset factor for statistical distribution fit.
%           -stdevFit: Standard deviation for statistical distribution fit.
%           -aFit: 'a' factor for statistical distribution fit.
%           -diffFit: sum of squares difference between fitted and real
%               data
%           -SNR: Signal to noise ratio of each simulated scenario
%           -reflectiondB: Magnitude of reflected signals
%
% Array format:-Means such as SNRmean, pRmean etc hold the mean of all
%               runs. Where the index is the number of paths. Eg. index 1
%               is direct ray, index 2 is direct + ground reflection, index
%               3 is direct + ground + 1 multipath reflection.
%              -Arrays that contain the values of all runs (such as pRdB)
%               are a table where each row represents the given number of
%               multipath signals and columns hold each run.
%
%-----------------------------------------------------------------------
function [SNRdirect, pDirect, dLossDirect, d, SNRmean, pRdB, pRmean,
offsetFit,stdevFit,aFit,diffFit,SNR,reflectiondB] =
AnalyseLink(site,pt1,pt2,numpathsMax,diffractionOn,data)

if (exist('numpathsMax') ~=1)
    %Default maximum number of multipath reflections.
    numpathsMax = 6;    %6=direct+ground+4 multipath reflections
end

if (exist('data')==1)   %We are already given the terrain data
    x=data(1,:);
    z=data(2,:);
    d=x(length(x));
else
    if (strcmp(site,'test')==1)     %test case (flat terrain)
```

```
        dd=0.1;
        d=3.2;
        ht=1.8;
        hr=1.8;
        x=dd:dd:d;
        z=zeros(1,length(x));
    else        %read in a file
        if (pt1 < pt2)  %we are looking at forwards direction
            filename=['../../Sites/' site '/' pt1 '_' pt2 '.xz']; %Filename
            [x,z,d]=readPTPData(filename);  %Read in elevation data
        else        %Looking at reverse direction
            filename=['../../Sites/' site '/' pt2 '_' pt1 '.xz']; %Filename
            [x,z,d]=readPTPData(filename);  %Read in elevation data
            z=fliplr(z);                    %Flip the elevation data
            z=circshift(z,[0 -1]);      %Shift data to left so terrain data is
symmetrical when used in PTP model
            z(length(z))=z(length(z)-1);
        end
    end
end

%Define nodes heights for the Townsville (Annandale) test site.
if strcmpi(site,'townsville')
    switch (pt1)
        case {'a', 'A'}
            ht=13;      %Node at JCU (0)
        case {'c','C'}
            ht=3.6;     %Node at KK (4)
        case {'d','D'}
            ht=4.8;     %Node at MJ (3)
        case {'l','L'}
            ht=4;       %Node at SW (1)
        otherwise
            ht=NaN;
    end
    switch (pt2)
        case {'a', 'A'}
            hr=13;      %Node at JCU (0)
        case {'c','C'}
            hr=3.6;     %Node at KK (4)
        case {'d','D'}
            hr=4.8;     %Node at MJ (3)
        case {'l','L'}
            hr=4;       %Node at SW (1)
        otherwise
            hr=NaN;
    end
else
    ht=NaN;
    hr=NaN;
end

%Call calcSNRPTP to calculate signal strength, SNR etc of direct ray
[SNR,pRdB,signaldB,dLossDirect] = CalcSNRPTP(d,1,x,z,diffractionOn,ht,hr);
SNRdirect=SNR;
pDirect=signaldB;
SNRmean=SNRdirect;
pRdB=pDirect;
pRmean=pDirect;

%Analyse the path for numpaths=2->numpathsMax
for numpaths=2:numpathsMax
    for i=1:1000
        %Uncomment the following line to determine the postion of the
        %reflectors
        %[SNR(i),pRdB(i),signaldB,dLossDirect,reflectiondB(i),xx(i,:),yy(i,:)] =
CalcSNRPTP(d,numpaths,x,z);
        %Call CalcSNRPTP to calculate the SNR etc for given number of paths
        %on given terrain data.

[SNR(numpaths,i),pRdB(numpaths,i),signaldB,dLossDirect,reflectiondB(numpaths,i)] =
CalcSNRPTP(d,numpaths,x,z,diffractionOn,ht,hr);
    end
    %Calculate means
    pRmean(numpaths)=mean(pRdB(numpaths,:));
    SNRmean(numpaths)=mean(SNR(numpaths,:));
    %Uncomment the following lines to enable statistical fit
    %[offsetFit(numpaths),stdevFit(numpaths),aFit(numpaths),diffFit(numpaths)] =
fitrician(SNR);
        %[offsetFit(numpaths),stdevFit(numpaths),aFit(numpaths),diffFit(numpaths)] =
fitweibull(SNR);
    %[offsetFit(numpaths),stdevFit(numpaths),diffFit(numpaths)]=
fitrayleigh(SNR,numpaths);
    %Statistical fit was not used. Set values to nan.
    offsetFit(numpaths)=nan;
    stdevFit(numpaths)=nan;
    diffFit(numpaths)=nan;
    aFit(numpaths)=nan;
end

%If we have a direct path only, set statistical values to NaN
if numpathsMax<2
    offsetFit=nan;
    stdevFit=nan;
    diffFit=nan;
    aFit=nan;
end

beep %Beep to indicate link analysis is finished.
```

K.3 READPTPDATA.M

```
% File: ReadPTPData.m
% Author: Simon Willis (Simon.willis@jcu.edu.au)
% Date: 22/02/2005
% Modified: 01/02/2007
%
% Reads in terrain data for the PTP model. Interpolates data to determine
% elevation at 100m spacing.
%
% Inputs:   -filename: Filename of terrain data
%           -all: Boolean (default 0). Causes program to also read in
%               the testing data contained in taso_Data.txt. This is used
%               for comparing the PTP model on a known test.
% Outputs:  -x: Array of horizontal terrain points
%           -z: Elevation at point x
%           -d: Distance between nodes.
%
%---------------------------------------------------------------------------
function [x,z,d] = ReadPTPData(filename,all)

if (exist('all')~=1)
    all=0;      %Default: Don't load test data
end

%Get x and z co-ordinates from file
[distMap,elev]=textread(filename,'%f%f','delimiter',',');

if (all==1)

    %Read data from the taso_data.txt file for the PTP model.

[call,source,lat,long,freq,city,state,ERP,txElev,txAntHt,txAntHtTerrain,azimuth,date,t
ime,dist,rxElev,rxAntHt,strength]=textread('taso_data.txt','%s%s%d%d%f%s%s%f%d%d%d%f%s
%d%f%d%d%f','delimiter','|');

    %Extract distance data for Fresno, 209.8MHz, 175.5deg azimuth (for testing)
    dist2=dist(find(lat==133477 & long==429960 & freq==209.8 & azimuth==175.5));
    %Alternative test site:
    %dist2=dist(find(lat==109557 & long==328268 & freq==59.8 & azimuth==317));

    %Extract elevation data
    elevTest=rxElev(find(lat==133477 & long==429960 & freq==209.8 & azimuth==175.5));
    %Alternative test site:
    %elevTest=rxElev(find(lat==109557 & long==328268 & freq==59.8 & azimuth==317));

    %Extract strength data
    strength2=strength(find(lat==133477 & long==429960 & freq==209.8 &
azimuth==175.5));
```

```
    %Alternative test site
    %strength2=strength(find(lat==109557 & long==328268 & freq==59.8 & azimuth==317));

    %Extract tx elevation
    txElev2 = txElev(find(lat==133477 & long==429960 & freq==209.8 & azimuth==175.5));
    %Alternative test site
    %txElev2 = txElev(find(lat==109557 & long==328268 & freq==59.8 & azimuth==317));

    txElev2 = txElev2(1);
else
    txElev2=elev(1);
end

distMap=distMap/1000; %in km

%break elevation data up into a gap size specified by dd.
dd=0.1;

%generate equi-spaced x data
x=0:dd:max(distMap);

index=1;

%generate y data for points between 0 and first elev point
while x(index) < distMap(1)
    z(index) = (elev(1) - txElev2) / distMap(1) * x(index) + txElev2;
    index = index + 1;
end

for index=index:length(x)
    %find elev points to use
    if x(index) > 0
        temp = distMap / x(index);
        [pts] = find(temp >= 1);
    else
        pts = 2;
    end

    %find 2 points used to calculate slope
    %j -> 2nd point
    j = pts(1);
    %i -> 1st point
    i = j - 1;
    %calculate elevation
    z(index) = (elev(j) - elev(i)) / (distMap(j) - distMap(i)) * (x(index) -
distMap(i)) + elev(i);
end
%Remove the value for elevation at position 0. PTPModel works from 'dd'
%onwards.
%x=x(2:length(x));          %Commented out to allow the transmitter height to be
determined. PTP model scraps first piece of terrain data
%z=z(2:length(z));
d=(length(x)-1)*dd;
```

K.4 CALCSNRPTP.M

```
% File: CalcSNRPTP.m
% Author: Simon Willis (Simon.willis@jcu.edu.au)
% Date: 15/02/2005
% Modified: 26/02/2007
%
% The function will calculate the SNR given the distance between Rx and Tx
% and the relevant details of the Rx and Tx. Uses the thermal noise,
% multipath signal, receiver noise factor and direct signal.
% Uses MultiRayPTPModel.m
%
% Inputs:   -d: Distance between nodes (km)
%           -numpaths: Number of multipath signals simulate. 1 = direct
%               only, 2 = direct + ground, 3 = direct + ground + 1
%               reflection
%           -x: Terrain points in x direction
%           -z: Elevation at terrain points (m)
%           -diffractionOn: Calculate diffraction loss (boolean)
%           -ht: Height of transmitter (m)
%           -hr: Height of receiver (m)
%
% Outputs:  -SNR: Signal to noise ratio (dB)
%           -pRdB: Received Signal strength (db)
%           -signaldB: Same as pRdB. Contained strength of direct ray in
%               earlier version
%           -dLossDirect: Diffraction loss of direct ray
%           -reflectiondB: Total magnitude of reflected signals
%           -xx: x positions of reflectors (normally unused)
%           -yy: y positions of reflectors (normally unused)
%
%-----------------------------------------------------------------------
function [SNR,pRdB,signaldB,dLossDirect,reflectiondB,xx,yy] =
CalcSNRPTP(d,numpaths,x,z,diffractionOn,ht,hr)

%d=10;       %distnance in km.
fc=40.83;    %Carrier Frequency in MHz
%fc=550;     %Bellenden-Ker
K=1.38e-23;  %Boltzmann's constant (J/K)
t=30;        %Temperature (degrees C)
b=100e3;     %Receiver Bandwidth (Hz)
%b=7e6;      %Bellenden-Ker
%NF=6;       %Receiver noise factor (dB)
%NF=36.8;    %Approximated from measurements on node 4
%NF=35.4;    %Node 3
%NF=33.3;    %Node 2
%NF=45;      %Node 1
NF=10;       %For receiver sensitivity of -104dBm
%numpaths=4; %Number of paths to the receiver (other than direct path)
maxLoss=30;  %Maximum loss of a multipath (dB)

if exist('numpaths')==0
    numpaths=4;
end

if (exist('x') ~= 1)        %If no terrrain data is given
    dd=0.1;
    x = dd:dd:d;
    z=zeros(1,length(x));  %Flat terrain
end

dd=x(2)-x(1);
tAbs=273+t; %Absolute temperature

%Call MultiRayPTPModel
if (numpaths > 2)
    %Uncomment following line to determine reflector positions

%[pDirect,pR,pNoise,dLossDirect,xx,yy]=MultiRayPTPModel(x,z,numpaths,length(z)*dd);

[pDirect,pR,pNoise,dLossDirect]=MultiRayPTPModel(x,z,numpaths,length(z)*dd,diffraction
On,ht,hr);
else

[pDirect,pR,pNoise,dLossDirect]=MultiRayPTPModel(x,z,numpaths,length(z)*dd,diffraction
On,ht,hr);
end

%Direct signal
%signaldB=10*log10(pDirect);    %Used in old version of program
signaldB=10*log10(pR);   %New version assumes direct + reflection is signal
%Thermal noise (Haykin)
pNt=K*tAbs*b;

noisedB=10*log10(pNt);

%Strength of the reflection:
if pNoise~=0
    reflectiondB=10*log10(pNoise);
else
    reflectiondB=NaN;
end

%Calculate the SNR
SNR=signaldB-noisedB-NF;
```

```
%SNR=SignaldB-NoisedB;        %test
pRdB=signaldB;
```

K.5 MULTIRAYPTPMODEL.M

```
% File MultiRayPTPModel.m
% Author: Simon Willis
% Date: 09/02/2005
% Modified: 14/02/2007
%
% This program is a multi-ray point-to-point channel model. It models a
% transmission where a signal may be reflected from a set number of points.
% Each path is also subject to diffraction loss, as determined by the PTP
% model (PTPModel.m). Terrain may be flat (default) or irregular. In this
% case elevation data must be provided (see readPTPData.m). This model
% assumes that the receiver is positioned above (North) the transmitter and
% any point on a given y-coordinate has the same elevation. Therefore any
% path that is at an angle from North travels over terrain that is
% elongated.
% In addition to this, paths beyond the line of sight have extra loss.

% This model was primarily designed for use at 40 MHz. At this frequency
% the surface must be very rough (standard deviation of at least 0.94m) for
% any scattering affects to occur upon a reflection. Therefore the surface
% is considered smooth and reflections do not have loss due to scattering.
% Scattering has been implemented, but no surface roughness is given so it
% assumes the surface is smooth.

% Inputs:   -x,z: x and z data for elevation profile
%           -numpaths: Number of paths (including direct and ground
%               reflection)
%           -d: distance between TX and RX (km)
%           -diffractionOn: Boolean: Diffraction loss model (on/off)
%           -ht,hr: Elevation of TX and RX antennas, respectively
%           -hAnt: Height of antennas
%           -freq: Frequency of operation (MHz)
%           -EIRP: Effective Isotropic Radiated Power of Transmitter (dB)
%           -gR: Gain of receiver antenna
%
% Outputs:  -pDirect: Power of the direct component
%           -pR: Received signal strength (dB)
%           -pNoise: Strength of multipath components (older version
%               of program modelled these as noise)
%           -dLossDirect: Diffraction loss of direct path
%           -xx,yy: position of reflectors
%
%-----------------------------------------------------------------------
function [pDirect,pR,pNoise,dLossDirect,xx,yy] =
MultiRayPTPModel(x,z,numpaths,d,diffractionOn,ht,hr,hAnt,freq,EIRP,gR)

ef = 4/3;   %Earth radius correct factor
r0 = 6370;  %Earth radius (km)

dd=x(2)-x(1);     %Terrain resolution (km)

%Min and Max dielectric constant of land (2-7 is typical for dry sandy
%terrain)
MINDIELECTRICCONST=2;
%MINDIELECTRICCONST=5;    %Bellenden-Ker
MAXDIELECTRICCONST=7;
%MAXDIELECTRICCONST=20;   %Bellenden-Ker

%If x does not exist then generate default flat terrain
if (exist('x') ~= 1)
    x = dd:dd:d;
    z=zeros(1,length(x));  %Flat terrain
end

%If d is not specified then determine from data
if (exist('d') ~=1)
    d=length(x)*dd;
end
```

```
if (exist('diffractionOn')~=1)
    diffractionOn=1;        %Default: diffraction loss is modelled
end

if (exist('freq') ~= 1)
    freq = 40.83;       %Default Carrier frequency (MHz)
    %freq=550;          %Bellenden-Ker
end

wavelength = 3e8/(freq*1e6);      %calculate wavelength

if (exist('numpaths') ~= 1)
    numpaths=2;         %Just direct path + ground reflection
end

%Default elevation of transmitting antenna
if ((exist('ht') ~= 1) | isnan(ht))
    %ht = wavelength/4;  %Quarter wavelength whip
    %ht=33;      %Bellenden-Ker
    %ht=3.6;     %KK height (node C)
    %ht=13;       %BS height (node A)
    %ht=4.8;     %MJ height (node D)
    %ht=4;        %SW height (node L)
    ht=1.7;        %On star picket
end

%Default elevation of receiving antenna
if ((exist('hr') ~= 1) | isnan(hr))
    %hr = wavelength/4;         %Quarter wavelength whip
    %hr=3;       %Bellenden-Ker
    %hr=4.8;     %MJ height (node D)
    %hr=3.6;     %KK height (node C)
    %hr=13;       %BS height (node A)
    %hr=4;        %SW height (node L)
    %hr=1.5;     %Node on car.
    hr=1.7;        %On star picket
end

if (exist('hAnt') ~= 1)
    hAnt = 0;                 %hAnt is added to ht and hr
end

%Default EIRP of 1W (0dB)
if (exist('EIRP') ~= 1)
    EIRP=0;          %EIRP (dB)
    %EIRP=-4.5;      %Node 1
    %EIRP=47;        %Bellenden-Ker
    %EIRP=21;        %Graham Range
end

if (exist('gR') ~= 1)
    gR = 1.5;           %Gain of quarter wavelength whip (dBi)
end

%Determine if terrain is flat or not
diff=z-mean(z);
if (max(abs(diff))>hr)   %If a terrain feature is bigger than Rx antenna
    flat = 0;   %not flat
else
    flat = 1;   %flat
end

%Adjust the node heights to include terrain.
heightDiff=z(1)-z(length(z));
if (abs(heightDiff) > 50)   %Adjust node heights if the height difference is
substantial
    if heightDiff>0         %Tx is higher
        htTer=ht+heightDiff;
        hrTer=hr;
    else
        htTer=ht;
        hrTer=hr-heightDiff;
    end
    %Straight-line distance from Tx to Rx
    distStraight=sqrt((d*1e3)^2 + heightDiff^2)/1e3;
else
```

```
    htTer=ht;
    hrTer=hr;
    distStraight=d;
end

%Free-space loss for direct path
lfsDirect = 32.4 + 20*log10(freq) + 20*log10(d);

%Find effective radio horizon
rE = ef * r0;

rRH = sqrt(2*rE*1e3*(htTer+hAnt)) + sqrt(2*rE*1e3*(hrTer+hAnt));    %radio horizon (m)
rRH = rRH / 1e3;        %in km

%Apply diffraction loss
if diffractionOn==1
    dLossDirect=PTPModel(z,round(d/dd),freq,z(1)+ht+hAnt,hr+hAnt,dd);
else
    dLossDirect=0;
end

%Calculate power of direct path
pDirect=EIRP + gR - lfsDirect + dLossDirect;

%Calculate angle of incidence of ground reflection
thetaReflect=atan(((hrTer + hAnt) + (htTer+hAnt))/(d*1e3));

pathLength(1)=(((htTer+hAnt)+(hrTer+hAnt))/sin(thetaReflect))/1e3;    %path length
(km)

%Calculate phase of other paths
if ((hr>0) | (ht>0))
    phase=[((pathLength(1)-distStraight)*1e3)*2*pi/wavelength rand(1,numpaths-
2)*2*pi];
else    %There is an obstruction or antennas are on ground
    phase=[0 rand(1,numpaths-2)*2*pi];    %No ground reflected ray
end

%Calculate delay of ground reflected path
if ((hr>0) | (ht>0))
    delay =((pathLength(1) - distStraight)*1e3)/3e8;
    dLossW(1)=10^(dLossDirect/10);
else
    delay = 0;
    dLossW(1)=0;
end

%Reflection coefficient of ground reflected path. small angle of
%incidence. Only applies antennas are above ground level.
if ((hr>0) | (ht>0))
    %Dielectric Constant
    dielectricConst=rand(1,1)*(MAXDIELECTRICCONST-
MINDIELECTRICCONST)+MINDIELECTRICCONST;
    a=1./dielectricConst;        %Vertical polarisation
    alpha=(pi/2)-thetaReflect;
    rc=(cos(alpha)-a.*sqrt(dielectricConst-
((sin(alpha)).^2)))./(cos(alpha)+a.*sqrt(dielectricConst-((sin(alpha)).^2)));
    %apply scattering loss (not used due to low frequency)
%    scatterScale = ScatteringLoss(0,0,freq,thetaI(i));
%    rc = rc * scatterScale;
else
    rc=0;
end

gr=10^(gR/10);          %Antenna gain in W
eirp = 10^(EIRP/10);    %EIRP in W

%Strength of ground reflected path (free-space loss)
if ((hr>0) | (ht>0))
    pathStr(1)=10*log10(eirp*dLossW*gr*(3e8*rc(1)/(4*pi*pathLength*1e3*freq*1e6))^2);
else
    pathStr(1)=0;
end

%Generate a new path and calculate signal strength
for i=2:numpaths-1      %For each extra path
```

```
    pathStr(i) = -inf;  %initialise path loss
    while pathStr(i) < pDirect - 30;    %ensure path is at most 30dB down on direct
        azimuth(i)=rand(1)*2*pi;        %random azimuth from TX
        reflectorWest=0;
        %If the reflector will be to the 'west' alter it so calculations
        %are correct
        if (azimuth(i) > pi)
            reflectorWest=1;
            azimuth(i)=2*pi-azimuth(i);
        end
        pathLength(i) = inf;
        %Choose angle of incidence so that the path length is at most 5
        %times d.
        while (pathLength(i) > 5*d)
            thetaI(i)=rand(1)*pi/2;     %angle of incidence
            %Calcuate the reflector distance from the transmitter
            distFromTX(i)=d*sin(2*thetaI(i)-azimuth(i))./sin(pi-2*thetaI(i));
            if (distFromTX(i) > 0)      %path is physically possible
                xx(i)=distFromTX(i)*sin(azimuth(i));    %x position of reflector
                %Adjust for a 'west' reflector
                if (reflectorWest==1)
                    xx(i)=xx(i)*-1;
                end
                yy(i)=distFromTX(i)*cos(azimuth(i));    %y position
                %Calculate the reflector distance from the reflector
                distFromRX(i)=sqrt((yy(i)-d)^2+xx(i)^2);
                %Find the total path length.
                pathLength(i)=distFromTX(i)+distFromRX(i);
            else
                pathLength(i)=-inf;    %Path not possible. Choose a new azimuth
            end
        end

        %If we have found a valid path, then continue
        if pathLength~=-inf

            %Scattering loss of reflection
            scatterScale = ScatteringLoss(0,0,freq,thetaI(i));

            %Dielectric Constant
            dielectricConst=rand(1,1)*(MAXDIELECTRICCONST-
MINDIELECTRICCONST)+MINDIELECTRICCONST;

            %Determine the reflection coefficient
            a=1./dielectricConst;
            theta=pi/2-thetaI(i);
            rc(i)=(cos(theta)-a.*sqrt(dielectricConst-
((sin(theta)).^2)))./(cos(theta)+a.*sqrt(dielectricConst-((sin(theta)).^2)));
            %apply scattering loss
            rc(i) = rc(i) * scatterScale;

            %Delay of path
            delay(i) = ((pathLength(i)-d)*1e3)/3e8;

            %Calculate signal strength
            pathStr(i)=eirp*gr*(3e8*rc(i)/(4*pi*pathLength(i)*1e3*freq*1e6))^2;

            %if the pathStr at this point (before computing diffraction loss)
            %is more than 25dB down on the direct path then there is no point
            %continuing to find the diffraction loss.
            if ((pDirect-25) < (10*log10(pathStr(i))))
                %Assume that the obstacle is between the receiver and trasmitter
                dLoss1(i) = 0;       %initialisation
                dLoss2(i) = 0;

                %Compute the new terrain profile to the reflector
                newZ1 = StretchTerrain(z,dd,xx(i),yy(i),1);

                %If the terrain is irregular or the link is beyond the horizon,
                %compute diffraction loss
                if (length(newZ1)>2)
                    if (diffractionOn==1)
                        %Call the PTP model to find diffraction loss on
                        %path to reflector from transmitter
                        dLoss1(i) = PTPModel(newZ1, length(newZ1), freq,
newZ1(1)+ht+hAnt,0,dd);    %Reflector is at ground level
```

```
                    else
                        dLoss1(i)=0;
                    end
                end
                %Compute the terrain profile from the reflctor to RX
                newZ2 = StretchTerrain(z, dd, xx(i), yy(i), 0);
                %Compute diffraction loss for irregular terrain of beyond LOS
                if (length(newZ2) > 2)
                    if (diffractionOn==1)
                        %Call the PTP model to find diffraction loss on
                        %path from reflector to receiver
                        dLoss2(i) = PTPModel(newZ2, length(newZ2), freq, newZ2(1),
hr+hAnt,dd);
                    else
                        dLoss2(i) = 0;
                    end
                end
                %Calculate diffraction loss in W.
                dLossW(i)=10^((dLoss1(i) + dLoss2(i))/10);
                %Apply diffraction loss
                pathStr(i)=pathStr(i)*dLossW(i);
            else
                pathStr(i)=10^((pDirect-40)/10);    %forces it to loop again
            end
            %Calculate path strength in dB
            pathStr(i)=10*log10(pathStr(i));
        end
        %disp(['1:' num2str(dLoss1(i)) ' 2: ' num2str(dLoss2(i))])  %Debugging
    end
end

%disp(['direct: ' num2str(pDirect-30) ' path: ' num2str(pathStr)])  %Debugging

%Find strength of multipath (noise) components.
pNoise=eirp*gr*(3e8/(4*pi*freq*1e6))^2*((abs(sum(rc.*sqrt(dLossW)./(pathLength*1e3).*e
xp(phase*-j))))^2);
%Find total strength (add data for direct path)
%rc(1)=0;        %temp: remove ground reflection

%Form arrays (including direct path)
rc=[1 rc];
phase=[0 phase];
pathLength=[d pathLength];
%pathLength=[distStraight pathLength];
dLossW=[10^(dLossDirect/10) dLossW];
%Total strength
pR=eirp*gr*(3e8/(4*pi*freq*1e6))^2*((abs(sum(rc.*sqrt(dLossW)./(pathLength*1e3).*exp(p
hase*-j))))^2);

%Convert pDirect to Watts (Since all other strength are given in Watts)
pDirect=10^(pDirect/10);

%if numpaths=1 there are no multipath components (no noise)
if (numpaths==1)
    pR=pDirect;
    pNoise=0;
end
```

K.6 PTPModel.m

```
% File: PTPModel.m
% Author: Simon Willis (simon.willis@jcu.edu.au)
% Date: 09/12/2004
% Last Modified: 01/02/2007
%
% MATLAB implementation of the PTP model by Wong (www.fcc.gov/oet/fm/ptp/)
% This is the FORTRAN PTP code given by the FCC ported to MATLAB
% (http://www.fcc.gov/oet/fm/ptp/diffract.for)

% Inputs:   -iElevat: Array of Elevation along the path (m)
%           -iDist:   Index to distance of point under study
%           -freq:    Carrier frequency (MHz)
%           -htamsl:  height of TX including terrain
%           -hr:      Height of receiving antenna
%           -dd:      Distance between points in elevation profile
% Outputs:  -dLoss:   Diffraction loss at point on the earth
%           -clutter: Clutter loss at point on the path
%
%-----------------------------------------------------------------------
function [dLoss, ratio, clutter] = PTPModel(iElevat, iDist, freq, htamsl,hr,dd)

ek=4/3;          %equiv. earth curvature
%hr = 9.1;       %testing

%Initialise variables
dist=iDist*dd;   %distance in km
fresMin = hr +1.0;   %Minimum fresnel radius
dLoss = 0;
roundFact = 0;
adj=0;

%Defaults
if (exist('hr')~=1)
    hr=1.875;
end
if (exist('dd')~=1)
    dd=0.1; %scaling (km per index point)
end
hramsl= iElevat(iDist)+hr;  %Rx height including terrain

%-------------------------
% Determine clearance ratio
%-------------------------

obstacle=1;          %initialisation
ratio=1;
clearMin=1000;       %Minimum clearance

slope=(hramsl - htamsl) / dist;     %Slope between receiver and transmitter

%l=1:iDist-1;   %pre-1/2/07
l=2:iDist;      %The first position is the transmitter location. We don't want to
compute Fresnel radius etc for this location.
xd=dd*(l-1);
earthB=0.0785 * xd .* (dist - xd) / ek; %Earth bulge term (Hernando pp.160)
cel=iElevat(l) + earthB;                %equivalent elevation
hlos=htamsl + slope * xd;               %height of LOS path
clearance=hlos - cel;                   %clearance to obtacle
fresnel=548 * sqrt(xd .* (dist - xd) / (dist * freq));              %Fresnel radius
(Wong pp. 5)
fresnel(find((fresnel < fresMin) & (clearance <= 0))) = fresMin;    %set min fresnel
radius to fresMin

%Clearance ratio
rat=clearance ./ fresnel;
[ratio,loc]=min(rat);                   %The obstacle is at the min ratio
cLen=dist-loc*dd;                       %path length to RX
[clearMin,obstacle]=min(clearance);     %Miniumum clearance
obsElev=cel(obstacle);                  %Obstacle elevation

if (ratio>0.8)
    obstacle = iDist - 1;               %There is no obstacle
end

obDist= obstacle * dd;                  %distance to obstacle
rLength = dist - obDist;                %Distance to receiver
pLength = rLength/dd;                   %in point counts

%----------------------------
% Smooth earth horizon distance
%----------------------------

elMin = htamsl - 30;
elMin = min(iElevat);       %Find min elevation

ref = elMin(1);

hrZ= 4.122 *sqrt(hr);       %Horizon from RX
ht = htamsl - ref;          %Height of TX antenna
```

```
if (ht < 30)
    %ht = 30;       %Set min antenna height to 30
end
%horizon= 4.122 * sqrt(ht);  %Horizon from TX
horizon= 4.122 * (sqrt(ht) + sqrt(hr));  %Horizon from TX (modified from original)
fract = dist/horizon;       %Distance in terms of horizon

%-------------------------------------------------
% beyond Horizon Adjustment
% (Wong is unsure why, but indicated such a need
% to fit collected data closer)
%-------------------------------------------------

adj=0;
blend = 100/sqrt(freq);

if (ratio <= 0.2)       %Path clearance ratio <= 0.2
    freqFact = 4.5 - 25.57/(log10(freq) + 5.53);
    beyond = dist - horizon;        %How far beyond horizon?
    if ((beyond+blend) >= 0)        %Beyond the horizon
        horzFact=224 / (horizon + 70) - 0.6;
        if (horzFact < 0.2)
            horzFact=0.2;
        end
        if (horzFact > 1.8)
            horzFact=1.8;
        end
        beyond=horzFact * beyond;

        beyFact = 11.7 - 222.3/(beyond +19);

        if (beyond <=20)
            blend = 100 /sqrt(freq);
            x1=-blend;
            x2=20;
            y2=6;
            bSlope=y2 / (x2 - x1);
            beyFact = bSlope * (beyond - x1);
        end
        adj = beyFact * freqFact;
        if (adj < 0)
            adj=0;
        end
    end
end

%-------------------------------------------------
% Smoorth-earth and knife-edge diffraction losses
% (from Wong's PTP paper)
%-------------------------------------------------
edge = -1.377 * ratio.^2 + 11.31 * ratio - 6;
if (ratio < -0.5)
    edge = 50.4/(1.6 - ratio) -36;      %knife-edge diffraction loss
end
if (edge > 0)
    edge = 0;
end
smooth = 38.68 * ratio - 21.66;         %Smooth-earth diffraction loss
if (ratio < -0.8)
    smooth=3030/(11.2 - ratio) -305.7; %Not in paper
end
if (smooth > edge)
    smooth=edge;
end

if (ratio < 0.57)      %Obstacle in path
    %------------------------------
    % Varying terrain (non isolated)
    %------------------------------
    init=obstacle - 100;
    last=obstacle + 100;
    if (init < 1)
        init = 1;
    end
    if (last >= iDist)
        last=iDist-1;
    end

    n = last - init + 1;

    x = init:last;
    y = iElevat(x);
    sumx = sum(x);
    sumx2 = sum(x.^2);
    sumy = sum(y);
    sumxy = sum(x.*y);

    % Least-squares fit
    % y = tSlope * x + tb
    den = n * sumx2 - sumx * sumx;
    tSlope = (n *sumxy - sumx *sumy) / den;
    tb = (sumy * sumx2 - sumx * sumxy) / den;

    %----------------------------------------------------
    % Determine fit through median (translate fitted line)
    %----------------------------------------------------
    % Uses a 'divide by 2' search.
    mpts = (n+1) / 2;       %Half the number of points
    boundUp=1000;           %Upper boundary
    boundLow=-1000;         %Upper bounday
    test=0;

    while (boundUp-boundLow > 1)
        fit = tSlope * x + tb + test;   %find the fitted line
        diff = y - fit;                 %find the difference
        countL = length(find(diff > test)); %we would like line moved up
        countH = length(find(diff >= test)); %we would like line moved down
        if (countH < mpts)      %Move line down
            boundUp=test;
            test = (boundUp + boundLow) / 2;
            if (boundLow <= -1000)
                test = boundUp - 10;    %out of boundaries
            end
        else
            if (countL > mpts)  %Move line up
                boundLow = test;
                test = (boundUp + boundLow) / 2;
                if (boundUp >= 1000)
                    test = boundLow + 10;   %out of boundaries
                end
            else
                boundUp=boundLow;        %finished
            end
        end
    end
    trans = test;

    %---------------------------------------------------------
    % Determine Delta H
    %---------------------------------------------------------

    x=init:last;
    y = iElevat(x);
    fit = tSlope * x + tb + trans;  %fitted line
    diff = y - fit;         %difference
    tMin = min(diff);       %Minimum difference
    tMax = max(diff);       %Maximum difference

    sumD = sum(diff);       %Total difference

    %find sum of (diff^2). Break up upper and lower parts.
    sumH2 = sum(diff(find(diff >= 0)) .* diff(find(diff >= 0)));
    sumL2 = sum(diff(find(diff < 0)) .* diff(find(diff < 0)));
    sumD2 = sumH2 + sumL2;

    hill = 1.6 * tMax;      %Hill at maximum difference point

    sigma = std(diff);
    delta = 2 * 1.282 * sigma;  %90% probability (two tails)

    deltaH = delta;
```

```
if (sumD2 > 0.1)        %not flat terrain
    if (delta > hill)
        deltaH = hill;
    else
        deltaH = hill / 2 + delta * sumL2/sumD2;
    end
end

tBlock = -deltaH;

%-----------------------------------------------------------
% Receiver very near obstacle
%-----------------------------------------------------------

if (rLength <= 5)       %distance from obs to receiver < 5
    diff = obsElev - iElevat(iDist);    %difference in elevation
    if (diff > 0)
        proj = 0.35 + 3 / (rLength + 0.4);
        cliff = diff * proj;
        if (deltaH > cliff)
            deltaH = (deltaH + cliff) / 2;
        end
    end
end

%-----------------------------------------------------------
% Short paths (no secondary obstacles & line-of-sight)
%-----------------------------------------------------------

if (ratio >= 0)
    if (fract < 0.3)
        deltaH = deltaH + 100 * (1.125 + 0.0422 / (fract - 0.3375));
    end
end

%-----------------------------------------------------------
% Section not mentioned in paper, but contained in Fortran
% code.
% Recalculates losses for NLOS links
% I think it reduces the clearance ratio due to a sharp edge
% being close to the receiver.
%-----------------------------------------------------------

fLim = 3413.8 / (deltaH + 266.7) - 5.8;
if ((cLen < fLim) & (ratio < 0)) %Non LOS
    plow = dist / 2;
    if (dist > 2*fLim)
        plow = fLim;
    end
    %Fresnel radius at central point.
    f5 = 548 * sqrt(plow * (dist - plow) / (dist * freq));
    primLim = -deltaH / f5;
    if (ratio < primLim)
        ratio = primLim;
    end
    %Calculate knife-edge and smooth-sphere loss
    edge = -1.377 * ratio ^ 2 + 11.31 * ratio - 6;
    if (ratio < -0.5)
        edge = 50.4 / (1.6 - ratio) - 36;
    end
    if (edge > 0)
        edge = 0;
    end
    smooth= 38.68 * ratio - 21.66;
    if (ratio < -0.8)
        smooth = 3030 / (11.2 - ratio) -305.7;
    end
    if (smooth > edge)
        smooth = edge;
    end
end

%------------------------------------------------------------
% Calculate roundness factor and diffraction loss
%------------------------------------------------------------
```

```
roundFact = 75 / (deltaH + 75);         %roundness factor
if (roundFact < 0)
    roundFact = 0;
end
if (roundFact > 1)
    roundFact = 1;
end

range = edge - smooth;
dLoss = edge - roundFact * range;       %diffraction loss

%-----------------------------------------------------------
% Adjust roundness near the horizon
%-----------------------------------------------------------

pRound = roundFact;


if (ratio >= -1.1)      %if not big obstruction
    part = fract;       %dist/horz
    if (part > 0.7)     %If we are approaching the horizon
        power = 0.17 + 0.166 / (fract - 0.5);
        if (power > 1)
            power = 1;
        end
        diff = 1 - power;
        %Recalculate the path clearance ratio
        rats = 1.33 - 0.266 / (ratio + 1.3);
        if (rats < 0)
            rats = 0;
        end
        if (rats > 1)
            rats = 1;
        end
        power = 1 - rats * diff;

        power = power * log10(roundFact);
        %Modify the roundness factor
        pRound = 10^power;
        % Recalculate diffraction loss
        dLoss = edge - pRound * range;
    end
end

%----------------------------------------------------------

ptLim = 5 * roundFact;      %set dist to look for 2nd obstacle
ptLim = round(ptLim / dd);     %In point counts
if (ptLim < 10)
    ptLim = 10;
end

%----------------------------------------------------------
% Beyond line-of-sight
%----------------------------------------------------------
%Transmitter side of the obstacle

tRound = pRound;
round2 = pRound;

if ((roundFact < 0.8) & (ratio <= 0))   %non LOS
    tAdj = 0;
    tObs = 1;
    tRatio = 0.57;
    clearMin = 1000;
    %slope between transmitter and obstacle
    slope = (obsElev - htamsl)/obDist;

    lastPt = obstacle - ptLim;
    if (lastPt > 1)        %not in original Fortran code
        %Calculate the path clearance ratio
        %l=10:lastPt;
        l=1:lastPt; %modified
        xd = l * dd;
        %Earth-bulge term
        earthB = 0.0785 * xd .* (dist - xd) / ek;
        %Terrain elevation including earth bulge
        cel = iElevat(l) + earthB;
        %height of LOS path to obstacle
        hLOS = htamsl + slope * xd;
        %clearance
        clearance = hLOS - cel;
        %Fresnel radius
        fresnel = 548 * sqrt (xd .* (obDist - xd) ./ (obDist * freq));
        %Set fresnel radius to fresMin for large obstacles where
        %fresnel<fresMin
        fresnel(find((clearance <= 0) & (fresnel < fresMin)))=fresMin;
        %Path clearance ratio
        rat = clearance ./ fresnel;
        %Find minimum path clearace ratio (TX side of obs)
        tRatio = min(rat);
        [clearMin, tObs] = min(clearance);           %Minumum clearance

        aRatio = tRatio - 0.3;

        if (aRatio > 0)
            tFactor = 4.95 / (aRatio + 4.5) - 1.1;
        else
            tFactor = 2 + 11 / (aRatio - 5.5);
        end

        %recalculate the roundness factor
        tAdj = tFactor * (1 -pRound);
        tRound = pRound + tAdj;

        if (tRound < 0)
            tRound = 0;
        end
        if (tRound >1)
            tRound = 1;
        end

        % End of transmitter side of primary obstacle
        %-----------------------------------------------------------
        % Adjust roundesss for long beyond line-of-sight paths
        % (loss between primary obstacle & receiver)
        %-----------------------------------------------------------
        round2 = tRound;
    end

    %if (rLength > 10)   %Non LOS
    if ((rLength/dd - ptLim -5) > 0)
        rObs = obstacle + 1;
        rRatio = 0.57;  %path-clearance ratio (rx side of obs)
        %Slope from obstacle to RX
        slope = (hramsl - obsElev) / rLength;

        %start and stop points for investigation
        initPt = obstacle + ptLim;  %after obstacle
        lastPt = iDist - 5;         %before receiver

        %find path clearance ratio
        l=initPt:lastPt;
        xd = l * dd;
        axd = xd - obDist;
        %earth bulge term
        earthB = 0.0785 * xd .* (dist - xd) / ek;
        %elevation, including earth bulge
        cel = iElevat(l) + earthB;
        %elevation of LOS path between obs and RX
        hLOS = obsElev + slope * axd;
        clearance = hLOS - cel;  %path clearance
        fresnel = 548 * sqrt(axd .* (rLength - axd) ./ (rLength * freq));
        %Adjust min. fresnel
        fresnel(find((clearance <= 0) & (fresnel < fresMin)))=fresMin;
        rat = clearance ./ fresnel; %Path clearance ratio;
        [rRatio, cNum] = min(rat);          %min path clearance ratio (RX side)
        cLen = rLength - axd(cNum);  %Dist from RX
        cNum=clearance(cNum);           %Clearance
        [clearMin, rObs] = min(clearance);                   %Minimum clearance

        if ((cLen < 5) & (rRatio < 0))  %obstacle closer to RX
```

```
            plow = rLength / 2;
            if (rLength > 10)
                plow = 5;
            end
            %Fresnel radius at point 'plow'
            f5 = 548 * sqrt(plow * (rLength - plow)/(rLength * freq));
            rxLim = cNum / f5;  %path clearance ratio
            if (rRatio < rxLim)
                rRatio = rxLim;
            end
        end

        %undo 2nd obstacle gain TX side
        un = 1;
        if (roundFact <= 0.5)   %if overall roundness factor is less than 0.5
            %if we have a knife-edge obstacle on RX side, approx beyond
            %horz and far from RX then adjust tRound
            if ((fract >= 0.8) & (rRatio <= 0.1) & (rLength >= 15))
                un = 1 - 10 * rRatio;
                if (un > 1)
                    un = 1;
                end
            end
        else
            if (tAdj <= 0)
                %Adjust roundness factor for TX side
                tRound = tRound - un * tAdj;
            end
        end


        %------------------------------------------------------------
        aRatio = rRatio - 0.5;

        if (aRatio > 0)
            rFactor = 4.95 / (aRatio + 4.5) - 1.1;
        else
            rFactor = 2 + 11 / (aRatio - 5.5);
        end

        rAdj = rFactor * (1 - roundFact);   %RX side adjustment
        round2 = tRound + rAdj;     %adjusted for both side;

        if (round2 < 0)
            round2 = 0;
        end
        if (round2 > 1)
            round2 = 1;
        end

    end
end

%End of RX side of primary obstacle
%---------------------------------------------------------------
% Terrain variations near receiver (limit smoothness)
%---------------------------------------------------------------

second = roundFact;

if ((roundFact >= 0.4) & ((dist/horizon) >= 0.8))
    if (rLength >= 10)
        init = iDist - 200;
        if (init < (obstacle + 50))
            init = obstacle + 50;
        end
        last = iDist;
        if ((last - init) >= 50)
            x = init:last;
            y=iElevat(x);
            n = last - init + 1;
            sumx = sum(x);
            sumx2 = sum(x.^2);
            sumy = sum(y);
            sumxy = sum(x.*y);
```

```
                % Least-squares fit
                % y = tSlope * x + tb
                den = n * sumx2 - sumx * sumx;
                tSlope = (n *sumxy - sumx *sumy) / den;
                tb = (sumy * sumx2 - sumx * sumxy) / den;

                %------------------------------------------------------
                % Determine fit through median (translate fitted line)
                %------------------------------------------------------
                % Uses a 'divide by 2' search.
                mpts = (n+1) / 2;       %Half the number of points
                boundUp=1000;           %Upper boundary
                boundLow=-1000;         %Upper bounday
                test=0;

                while (boundUp-boundLow > 1)
                    fit = tSlope * x + tb + test;   %find the fitted line
                    diff = y - fit;                 %find the difference
                    countL = length(find(diff > test)); %we would like line moved up
                    countH = length(find(diff >= test)); %we would like line moved
down
                    if (countH < mpts)      %Move line down
                        boundUp=test;
                        test = (boundUp + boundLow) / 2;
                        if (boundLow <= -1000)
                            test = boundUp - 10;    %out of boundaries
                        end
                    else
                        if (countL > mpts)  %Move line up
                            boundLow = test;
                            test = (boundUp + boundLow) / 2;
                            if (boundUp >= 1000)
                                test = boundLow + 10;   %out of boundaries
                            end
                        else
                            boundUp=boundLow;        %finished
                        end
                    end
                end
                trans = test;

                %Determine delta H
                fit = tSlope * x + tb + trans;
                diff = y - fit;
                sumD = sum(diff);
                sumD2 = sum(diff.^2);

                sigma = std(diff);
                delta2 = 2 * 1.282 * sigma;     %delta H

                second = 75 / (delta2 + 75);    %Roundness near RX
            end
        end
        ref = obDist;               %Obstacle distance
        if (obDist > horizon)       %if obs is beyond horizon
            ref = horizon;          %reference obs becomes horizon
        end
        x=dist/ref;
        fact = 0.4 * x + 0.6;       %calculate correction factor
        if (fact < 1)
            fact = 1;
        end
        slim = fact * second;       %modify roundness factor
        if (slim > 1)
            slim = 1;
        end

        if (roundFact / second >= 0.9)
            %similar terrain throughout radial
            if (exist('rRatio') ~= 1)  %Fix for short LOS links between obs and RX
                rRatio = 0.57;
            end
            aRatio = rRatio - ratio / 4;    %path clearance ratio
            if (aRatio > 1)
                aRatio = 1;
            end
            if (aRatio > 0)
                rFactor = 4.95 / (aRatio + 4.5) - 1.1;
            else
                rFactor = 2 + 11 / (aRatio - 5.5);
            end

            rAdj = rFactor * (1 - roundFact);
            round2 = roundFact + rAdj;      %adjust roundness factor
            if (round2 > slim)          %adjusted roundness > horizon corrected
                round2 = slim;
            end
        end
    end
    dLoss = edge + round2 * (smooth - edge);    %adjusted diffraction loss
end
%diffraction loss
dLoss = dLoss - adj;

%-----------------------------------------------------------------------------
%Clutter loss. Was commented out in Fortran code
%-----------------------------------------------------------------------------
%Clutter loss as function of frequency (smoothed out near LOS)
% cMod = 1.0;     %Clutter modifier (from land-usage?)
% c = 4.5 * log10(freq) - 4;
% c = cMod * c;
%
% span = 0.28 - 0.144 * (roundFact - 1.2);
% para = c / span^2;
% if (ratio > 0.4)
%     para = 6.25 * (c - 1);
% end
% clutter = para * (ratio - 0.4) ^ 2 - c;
% if (clutter > 0)
%     clutter = 0;
% end
%
% %Adjustment for steep andle of arrival
% cSlope = sqrt(freq) / 350;
% steep = 1 + cSlope * (dist - horizon);
% if (steep < 0)
%     steep = 0;
% end
% if (steep > 1)
%     steep = 1;
% end
% %clutter loss at point on path.
% clutter = clutter * steep;

%-----------------------------------------------------------------------------
% Troposcattering
%-----------------------------------------------------------------------------

tropo100 = 20.34 - 0.077 * dist;
freqAdj = 23.978 - 58026.76 / (freq + 2320);
if (freq > 1000)
    freqAdj = 24.5 - 7200 / (freq + 3000);
end
tropoFd = tropo100 - freqAdj;

fS_field = 106.9 - 20 * log10(dist);
scatter = tropoFd - fS_field;
diffL = scatter - dLoss;
combine = 150 / (20 - diffL) - 5;
if (diffL < -10)
    combine = 0;
end
if (diffL > 10)
    combine = diffL;
end

%Final diffraction loss!
dLoss = dLoss + combine;
```

K.7 ACTIVATEGLOBALS.M

```
% File: AcivateGlobals
% Author: Simon Willis
% Date: 23/03/2005

% Actvates global variables generated by AnalyseSite.m
%
%-----------------------------------------------------------------------
for start = startNode:endNode
    for stop=start:endNode
        if (start~=stop)
            eval(['global SNRdirect' setstr(start) ' pDirect' setstr(start) '
dLossDirect' setstr(start) ' d' setstr(start) ' SNRmean' setstr(start) ' pR'
setstr(start) '_' setstr(stop) ' offsetFit' setstr(start) ' stdevFit' setstr(start) '
aFit' setstr(start) ' diffFit' setstr(start) ' SNR' setstr(start) '_' setstr(stop)]);
            eval(['global SNRdirect' setstr(stop) ' pDirect' setstr(stop) '
dLossDirect' setstr(stop) ' d' setstr(stop) ' SNRmean' setstr(stop) ' pR' setstr(stop)
'_' setstr(start) ' offsetFit' setstr(stop) ' stdevFit' setstr(stop) ' aFit'
setstr(stop) ' diffFit' setstr(stop) ' SNR' setstr(stop) '_' setstr(start)]);
        end
    end
end
```

Appendix L TinyOS Code Developed for the JCUMote

L.1 Testing Applications

L.1.1 LRNet – Long Range Wireless Sensor Network

LRNet.nc

```
/*                                                                 tab:4
 * "Copyright (c) 2000-2003 The Regents of the University  of California.
 * All rights reserved.
 *
 * Permission to use, copy, modify, and distribute this software and its
 * documentation for any purpose, without fee, and without written agreement is
 * hereby granted, provided that the above copyright notice, the following
 * two paragraphs and the author appear in all copies of this software.
 *
 * IN NO EVENT SHALL THE UNIVERSITY OF CALIFORNIA BE LIABLE TO ANY PARTY FOR
 * DIRECT, INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES ARISING OUT
 * OF THE USE OF THIS SOFTWARE AND ITS DOCUMENTATION, EVEN IF THE UNIVERSITY OF
 * CALIFORNIA HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
 *
 * THE UNIVERSITY OF CALIFORNIA SPECIFICALLY DISCLAIMS ANY WARRANTIES,
 * INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY
 * AND FITNESS FOR A PARTICULAR PURPOSE.  THE SOFTWARE PROVIDED HEREUNDER IS
 * ON AN "AS IS" BASIS, AND THE UNIVERSITY OF CALIFORNIA HAS NO OBLIGATION TO
 * PROVIDE MAINTENANCE, SUPPORT, UPDATES, ENHANCEMENTS, OR MODIFICATIONS."
 *
 * Copyright (c) 2002-2003 Intel Corporation
 * All rights reserved.
 *
 * This file is distributed under the terms in the attached INTEL-LICENSE
 * file. If you do not find these files, copies can be found by writing to
 * Intel Research Berkeley, 2150 Shattuck Avenue, Suite 1300, Berkeley, CA,
 * 94704.  Attention:  Intel License Inquiry.
 *
 *-------------------------------------------------------------------------------------
 */
/** File: LRNet.nc (Long-Range WSN testing application)
 *
 * Long-Range Wireless Sensor Network (LRNet). Periodically samples the
 * signal strength of neighbouring nodes and forwards these measurements through the
 * network to the base station, along with measurements of battery and solar voltage.
 * This application is based on the Surge application that is distributed with TinyOS
 * and has been modified by S Willis for the JCUMote long-range WSN testing. All code
 * is copyright of TinyOS and is bound by the above copyright message. Sections of
 * modified or added code are indicated by an "SW" comment and are copyright of
 * S Willis.
 *
 * @author Simon Willis
 * @author Unknown (Original Surge Author not noted in original file)
 * @modified 19/12/2006
 **/
 //--------------------------------------------------------------------------------------
includes LRNet;       //SW: modified
includes SurgeCmd;
includes MultiHop;


configuration LRNet {
}
implementation {
  components Main, LRNetM, TimerC, LedsC, NoLeds, ADCC, RandomLFSR,
    GenericCommPromiscuous as Comm, Bcast, EWMAMultiHopRouter as multihopM,
    //Uncomment DelugeC for Deluge functionality (un-tested)
    QueuedSend, /*DelugeC,*/ TH7122RadioIntM, VoltageRaw, SolarRaw; //SW: modified

//Uncomment for Deluge functionality (un-tested)
//  Main.StdControl -> DelugeC;
  Main.StdControl -> LRNetM.StdControl;            //Start LRNet
  LRNetM.SubControl -> Bcast.StdControl;           //Broadcast for message from base
station to nodes
  LRNetM.SubControl -> multihopM.StdControl;       //Multihop routing
  LRNetM.SubControl -> QueuedSend.StdControl;      //Queued send used by multihop
protocol
  LRNetM.SubControl -> TimerC;                     //General timer
  LRNetM.SubControl -> Comm;                       //GenericCommPromiscuous
  //  multihopM.CommControl -> Comm;

#ifndef PLATFORM_PC
  LRNetM.Batt   -> VoltageRaw;
#else
  LRNetM.Batt   -> ADCC.ADC[10];                          //Battery monitor
#endif
  LRNetM.Solar -> SolarRaw;                               //SW: Solar voltage
monitor
  LRNetM.Timer -> TimerC.Timer[unique("Timer")];          //Wire a timer instance
  LRNetM.WDTimer -> TimerC.Timer[unique("Timer")];        //Wire a timer instance
  LRNetM.Leds  -> LedsC; // NoLeds;                       //LEDs for diagnostics

  //Wire received surge command messages to LRNet
  LRNetM.Bcast -> Bcast.Receive[AM_SURGECMDMSG];
  Bcast.ReceiveMsg[AM_SURGECMDMSG] -> Comm.ReceiveMsg[AM_SURGECMDMSG];

  //Wire MultihopM's RouteControl and send interfaces to LRNet
  LRNetM.RouteControl -> multihopM;
  LRNetM.Send -> multihopM.Send[AM_SURGEMSG];

  //Multihop protocol handle for received surge messages
  multihopM.ReceiveMsg[AM_SURGEMSG] -> Comm.ReceiveMsg[AM_SURGEMSG];
  //Received messages
  LRNetM.ReceiveMsg -> Comm.ReceiveMsg;            //SW: Added. Monitor incoming
packets

  LRNetM.RadioMonitor -> TH7122RadioIntM;          //SW: Added. Monitor RSSI (noise)
}
```

LRNETM.NC

```
/*                                                                  tab:4
 * "Copyright (c) 2000-2003 The Regents of the University  of California.
 * All rights reserved.
 *
 * Permission to use, copy, modify, and distribute this software and its
 * documentation for any purpose, without fee, and without written agreement is
 * hereby granted, provided that the above copyright notice, the following
 * two paragraphs and the author appear in all copies of this software.
 *
 * IN NO EVENT SHALL THE UNIVERSITY OF CALIFORNIA BE LIABLE TO ANY PARTY FOR
 * DIRECT, INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES ARISING OUT
 * OF THE USE OF THIS SOFTWARE AND ITS DOCUMENTATION, EVEN IF THE UNIVERSITY OF
 * CALIFORNIA HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
 *
 * THE UNIVERSITY OF CALIFORNIA SPECIFICALLY DISCLAIMS ANY WARRANTIES,
 * INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY
 * AND FITNESS FOR A PARTICULAR PURPOSE.  THE SOFTWARE PROVIDED HEREUNDER IS
 * ON AN "AS IS" BASIS, AND THE UNIVERSITY OF CALIFORNIA HAS NO OBLIGATION TO
 * PROVIDE MAINTENANCE, SUPPORT, UPDATES, ENHANCEMENTS, OR MODIFICATIONS."
 *
 * Copyright (c) 2002-2003 Intel Corporation
 * All rights reserved.
 *
 * This file is distributed under the terms in the attached INTEL-LICENSE
 * file. If you do not find these files, copies can be found by writing to
 * Intel Research Berkeley, 2150 Shattuck Avenue, Suite 1300, Berkeley, CA,
 * 94704.  Attention:  Intel License Inquiry.
 *
 *-------------------------------------------------------------------------------------
 */
/** File: LRNet.nc (Long-Range WSN testing application)
 *
 * Long-Range Wireless Sensor Network (LRNet). Periodically samples the
 * signal strength of neighbouring nodes and forwards these measurements through the
 * network to the base station, along with measurements of battery and solar voltage.
 * This application is based on the Surge application that is distributed with TinyOS
 * and has been modified by S Willis for the JCUMote long-range WSN testing. All code
 * is copyright of TinyOS and is bound by the above copyright message. Sections of
 * modified or added code are indicated by an "SW" comment and are copyright of
 * S Willis.
 *
 * @author Simon Willis
 * @author Unknown (Original Surge Author not noted in original file)
 * @modified 19/12/2006
 **/
 //------------------------------------------------------------------------------------

includes LRNet;       //SW: modified
includes SurgeCmd;

module LRNetM {
  provides {
    interface StdControl;
  }
  uses {
    interface StdControl as SubControl;      //SW: modified
    interface ADC as Batt;                   //SW: Battery voltage monitor
    interface ADC as Solar;                  //SW: Solar voltage monitor
//    interface ADCControl;
    interface Timer;
    interface Leds;
    interface Send;                          //Send messages
    interface Receive as Bcast;              //Receive broadcast SURGECMD messages
    interface RouteControl;                  //Determine parent for new messages
    interface ReceiveMsg[uint8_t id]; //SW: Receive all messages (monitor neighbours)
    interface RadioMonitor;                  //SW: Monitor RSSI (noise)
    interface Timer as WDTimer;              //SW: Added
  }
}

implementation {

//Component Variables
  enum {
    TIMER_GETADC_COUNT = 1,            //Timer ticks for ADC
    TIMER_CHIRP_COUNT = 10,            //Timer on/off chirp count
    NUM_NEIGHBOURS = 5                   //SW: Number of neighbours to record
  };

  bool sleeping;                               //application command state
  bool focused;

  TOS_Msg latestPacket;                        //SW: Added

  NeighbourStats latestNeighbour;                   //SW: Added. Latest neighbour
heard
  NeighbourStats neighbours[NUM_NEIGHBOURS];        //SW: Added. Array of strongest
neighbours
  norace uint16_t gSensorData;                      //SW: Added. protected by
gfSendBusy flag
  norace uint16_t gSolarData;                       //SW: Added. protected by
gfSendBusy flag
  uint8_t sendQueue[NUM_NEIGHBOURS];                //SW: Added. Queue of neighbour
details to be send
```

```
  uint8_t sendPutter;                              //SW: Added. Putter for send queue
  uint8_t sendGetter;                              //SW: Added. Getter for send queue
  uint8_t queueLen;                                //SW: Added. Length of queue

  TOS_Msg gMsgBuffer;
  bool gfSendBusy;

  uint32_t timer_rate;
  uint16_t timer_ticks;

 /**
  * Initialise timers, node state and neighbour details queue
  */
  static void initialize() {
       int i;
#ifndef PLATFORM_PC
    outp(0x00, DDRF);
#endif
    timer_rate = INITIAL_TIMER_RATE - (TOS_LOCAL_ADDRESS << 3);
    atomic gfSendBusy = FALSE;
    sleeping = FALSE;
    sendPutter=0;                    //SW: Added.
    sendGetter=0;                    //SW: Added.
    queueLen=0;                      //SW: Added.
    //SW: Added. Initialise neighbour details
    for (i=0; i<NUM_NEIGHBOURS; i++) {
       sendQueue[i]=0xFF;     //empty queue contains 0xff, because index 0 can exist
       neighbours[i].address=0xFF;
       neighbours[i].strength=0;
       neighbours[i].noise=0;
    }
  }

  // The following block of code is written by and is copyright of S Willis
  // *** General Queue Functions ***
 /**
  * Add a neighbour to the sending buffer
  *
  * @attrib neighbourIndex Address of neighbour
  * @return An indication if value was successfully added
  **/
  result_t addNeighbour (uint8_t neighbourIndex) {
    if (queueLen<NUM_NEIGHBOURS) {
         sendQueue[sendPutter]=neighbourIndex;
         sendPutter++;                      //increment the putter
         sendPutter %= NUM_NEIGHBOURS;      //circular buffer.
        queueLen++;
        return SUCCESS;
    }
    else
         return FAIL;
  }

 /**
  * Take a neighbour off the sending buffer
  *
  * @return The address of the neighbour at the front of the queue
  **/
  uint8_t nextNeighbour () {
       uint8_t neighbour = 0xFF;

       if (queueLen != 0) {
              neighbour = sendQueue[sendGetter];
              sendQueue[sendGetter]=0xFF;          //take this index out of the queue
              sendGetter++;
              sendGetter %=NUM_NEIGHBOURS;
              queueLen--;
       }
       return neighbour;
  }

 /**
  * Check if a neighbour is already on the queue
  *
  * @attrib neighbourIndex Address of neighbour
  * @return TRUE/FALSE
  **/
  bool inQueue (uint8_t neighbourIndex) {
        int i;
        bool found = FALSE;

        for (i=0;i<NUM_NEIGHBOURS;i++) {      //Search through the queue
          if (sendQueue[i]==neighbourIndex) {
                found = TRUE;
          }
        }
        return found;
  }
  // End of code block by S Willis

 /**
  * Task to send a LRNet data packet
  **/
  task void SendData() {
    LRNetMsg *pReading;
    uint16_t len;
    uint32_t batt;            //SW: Battery voltage
    uint8_t neighbourIndex;   //SW: Index to neighbour in queue
    dbg(DBG_USR1, "LRNetM: Sending sensor reading\n");

    if ((pReading = (LRNetMsg *)call Send.getBuffer(&gMsgBuffer,&len))) {
        //Set up the packet
      pReading->type = SURGE_TYPE_SENSORREADING;
        //Set up the parent
      pReading->parentaddr = call RouteControl.getParent();
      neighbourIndex=nextNeighbour();        //SW: Get the next neighbour
      pReading->solar = gSolarData;                 //SW: set the solar value
      pReading->seq_no ++;                          //Sequence number
//        batt = ((uint32_t)(gSensorData)) << 23; //Old usage
        //Jcumote has bigger ADC values than Mica2. Mica2 never used MSB
        batt = ((uint32_t)(gSensorData)) << 22;      //SW: altered
//        pReading->seq_no &= 0x7fffff;              //Old usage
        pReading->seq_no &= 0x3fffff;               //SW: altered
        pReading->seq_no += batt;                   //Store battery value
        //SW: If we don't have a neighbour then send all 0s
        if (neighbourIndex == 0xff) {
          pReading->neighbourAddr = 0;
          pReading->strength = 0;
         pReading->noise = 0;
        }
        //SW: Load the details of the neighbour into the packet
        else {
          pReading->neighbourAddr = neighbours[neighbourIndex].address;
          pReading->strength = neighbours[neighbourIndex].strength;
          pReading->noise = neighbours[neighbourIndex].noise;
      }

      //Set the neighbour node strength to 0 in the neighbour table.
        //This will allow another node to take that position in the
        //table if required
      neighbours[neighbourIndex].strength=0; //SW: Added.

      if ((call Send.send(&gMsgBuffer,sizeof(LRNetMsg))) != SUCCESS)
          atomic gfSendBusy = FALSE;
    }

  }

 /**
  * Task to process the latest neighbour details that have been received
  *
  * The following task is written by and is copyright of S Willis
  **/
  void task processNeighbour() {
        bool bKnownNeighbour = FALSE;
        uint8_t neighbourIndex=0xFF;
        uint16_t weakestSignal;
        int i;

        //Do we have this neighbour in the table already?
        for (i=0; i<NUM_NEIGHBOURS; i++) {
               if (latestNeighbour.address == neighbours[i].address) {
```

```
                        bKnownNeighbour=TRUE;
                        neighbourIndex=i;
                }
        }

        //If this neighbour is known update the details in the array
        if (bKnownNeighbour) {
                neighbours[neighbourIndex]=latestNeighbour;
                //add the neighbour to the sending queue if it is not already there
                if (!(inQueue(neighbourIndex))) {
                        //add the neighbour to the sending queue if we can fit it in
                        addNeighbour(neighbourIndex);
                }
        }

        //this is not a known neighbour. If there are any neighbours in the table
        //with a weaker signal (or 0 - not used yet) then replace them.
        else {
                weakestSignal=0xFFFF;
                //find the weakest neighbour
                for (i=0; i<NUM_NEIGHBOURS; i++) {
                        if (neighbours[i].strength < weakestSignal) {
                                weakestSignal=neighbours[i].strength;
                                neighbourIndex=i;
                        }
                }
                //if the new neighbour has a stronger signal then replace it in the
table
                if (weakestSignal<=latestNeighbour.strength) {
                        neighbours[neighbourIndex]=latestNeighbour;
                        //add this neighbour to the sending list if this index is not in
the sending queue already
                        if (!(inQueue(neighbourIndex)))
                                addNeighbour(neighbourIndex); //add neighbour if it will
fit
                }
        }
  }

 /**
  * Standard Control initialise command
  *
  * @return Always returns SUCCESS.
  **/
  command result_t StdControl.init() {
    call SubControl.init();
    initialize();
    return SUCCESS;
  }

 /**
  * Standard Control start command
  *
  * @return Always returns SUCCESS.
  **/
  command result_t StdControl.start() {
    call SubControl.start();
    call Timer.start(TIMER_REPEAT, timer_rate);
    call WDTimer.start(TIMER_ONE_SHOT, WATCHDOG_MISSED_PACKETS * timer_rate);
        //SW: start the watchdog timer
    return SUCCESS;
  }

 /**
  * Standard Control stop command
  *
  * @return Always returns SUCCESS.
  **/
  command result_t StdControl.stop() {
    call SubControl.stop();
    return call Timer.stop();
  }

 /**
  * Main LRNet timer fires. Causes node to sample data and send a packet
  *
  * @return Always returns SUCCESS.
  **/
  event result_t Timer.fired() {
    dbg(DBG_USR1, "LRNetM: Timer fired\n");
    timer_ticks++;
    if (timer_ticks % TIMER_GETADC_COUNT == 0) {
      //TOSH_SET_BAT_MON_PIN();
      //TOSH_uwait(250);
      call Solar.getData();   //SW: Start the ADC getting the solar voltage
    }
    // If we're the focused node, chirp
    if (focused && timer_ticks % TIMER_CHIRP_COUNT == 0) {
      call Leds.greenOn();    //SW: modified
      call Leds.redOn();      //SW: modified
      call Leds.yellowOn();   //SW: modified
    }
    // If we're the focused node, chirp (toggle LEDs)
    if (focused && timer_ticks % TIMER_CHIRP_COUNT == 1) {
      call Leds.greenOff();   //SW: modified
      call Leds.redOff();     //SW: modified
      call Leds.yellowOff();  //SW: modified
    }
    return SUCCESS;
  }

 /**
  * Handler for the watchdog timer. If this timer goes off then it has
  * not been reset. The timer should be reset whenever a new packet is
  * received. Reset everything. Used in troubleshooting to ensure a node
  * does not get jammed in an erroneous state
  *
  * The following event handler was written by and is copyright of S Willis
  *
  * @return Always returns SUCCESS.
  **/
  event result_t WDTimer.fired() {
    call Leds.redOn();
    call StdControl.init();
    call StdControl.start();
    call WDTimer.start(TIMER_ONE_SHOT, WATCHDOG_MISSED_PACKETS * timer_rate);
       //reset the watchdog timer
    return SUCCESS;
  }

 /**
  * Once solar data has been collected then measure battery voltage
  *
  * The following event handler was written by and is copyright of S Willis
  *
  * @attrib data ADC reading of solar voltage
  * @return Always returns SUCCESS
  **/
  async event result_t Solar.dataReady(uint16_t data) {
    //LRNetMsg *pReading;
    //uint16_t len;
    //TOSH_CLR_BAT_MON_PIN();
    dbg(DBG_USR1, "LRNetM: Got ADC reading: 0x%x\n", data);
    atomic {
              gSolarData = data;
    }
    call Batt.getData();      //Get the battery voltage
    return SUCCESS;
  }

 /**
  * Once the battery voltage has been measured we are ready to send a packet
  *
  * @attrib data ADC reading
  * @return Always returns SUCCESS
  **/
  async event result_t Batt.dataReady(uint16_t data) {
    //LRNetMsg *pReading;
    //uint16_t len;
    //TOSH_CLR_BAT_MON_PIN();
    dbg(DBG_USR1, "LRNetM: Got ADC reading: 0x%x\n", data);
    atomic {
      if (!gfSendBusy) {
        gfSendBusy = TRUE;
        gSensorData = data;
        post SendData();              //Fill a packet and send it
      }
    }
    return SUCCESS;
  }

 /**
  * Packet was sent successfully
  *
  * @attrib pMsg Pointer to TinyOS message
  * @attrib success success of transmission
  * @return Always returns SUCCESS
  **/
  event result_t Send.sendDone(TOS_MsgPtr pMsg, result_t success) {
    dbg(DBG_USR2, "LRNetM: output complete 0x%x\n", success);
    //call Leds.greenToggle();       //SW: troubleshooting
    atomic gfSendBusy = FALSE;
    return SUCCESS;
  }

 /**
  * Command interpreter for broadcasts
  *
  * @attrib pMsg Pointer to received message
  * @attrib payload Pointer to payload
  * @attrib payloadLen Length of pointer
  * @return Pointer to message
  **/
  event TOS_MsgPtr Bcast.receive(TOS_MsgPtr pMsg, void* payload, uint16_t payloadLen)
{
    SurgeCmdMsg *pCmdMsg = (SurgeCmdMsg *)payload;

    dbg(DBG_USR2, "LRNetM: Bcast  type 0x%02x\n", pCmdMsg->type);

    if (pCmdMsg->type == SURGE_TYPE_SETRATE) {       // Set timer rate
      timer_rate = pCmdMsg->args.newrate;
      dbg(DBG_USR2, "LRNetM: set rate %d\n", timer_rate);
      call Timer.stop();
      call Timer.start(TIMER_REPEAT, timer_rate);

    }
    else if (pCmdMsg->type == SURGE_TYPE_SLEEP) {
      // Go to sleep - ignore everything until a SURGE_TYPE_WAKEUP
      dbg(DBG_USR2, "LRNetM: sleep\n");
      sleeping = TRUE;
      call Timer.stop();
      call Leds.greenOff();
      call Leds.yellowOff();

    }
    else if (pCmdMsg->type == SURGE_TYPE_WAKEUP) {
      dbg(DBG_USR2, "LRNetM: wakeup\n");

      // Wake up from sleep state
      if (sleeping) {
         initialize();
        call Timer.start(TIMER_REPEAT, timer_rate);
         sleeping = FALSE;
      }
    }
    else if (pCmdMsg->type == SURGE_TYPE_FOCUS) {
      dbg(DBG_USR2, "LRNetM: focus %d\n", pCmdMsg->args.focusaddr);
      // Cause just one node to chirp and increase its sample rate;
      // all other nodes stop sending samples (for demo)
      if (pCmdMsg->args.focusaddr == TOS_LOCAL_ADDRESS) {
        // OK, we're focusing on me
        focused = TRUE;
        call Leds.greenOn();   //SW: added
      call Leds.redOn();       //SW: added
      call Leds.yellowOn();    //SW: added
        call Timer.stop();
        call Timer.start(TIMER_REPEAT, FOCUS_TIMER_RATE);
      }
      else {
          // Focusing on someone else
          call Timer.stop();
          call Timer.start(TIMER_REPEAT, FOCUS_NOTME_TIMER_RATE);
      }
    }
    else if (pCmdMsg->type == SURGE_TYPE_UNFOCUS) {
      // Return to normal after focus command
      dbg(DBG_USR2, "LRNetM: unfocus\n");
      focused = FALSE;
      call Leds.greenOff();   //SW: added
      call Leds.redOff();     //SW: added
      call Leds.yellowOff();  //SW: added
      call Timer.start(TIMER_REPEAT, timer_rate);
    }
    return pMsg;
  }

  /**
   * Event handler for all received packets
   *
   * Intercept event signalled when a packet is received. Wired to snoop
   * and intercept so that packet is always received whether it is being
   * forwarded or not. Used to determine the strength of neighbours' signals
   *
   * This is event handler is written by and is copyright of S Willis
   * @attrib pMsg Pointer to received message
   * @return Pointer to message
   **/
   event TOS_MsgPtr ReceiveMsg.receive[uint8_t id](TOS_MsgPtr pMsg) {
    TOS_MHopMsg *pMhopMsg = (TOS_MHopMsg *) pMsg->data;

    if ((id==AM_SURGEMSG) || (id==AM_MULTIHOPMSG)) {
        latestNeighbour.address = pMhopMsg->sourceaddr;             //store address
        latestNeighbour.strength = pMsg->strength;                  //signal strength
        latestNeighbour.noise = call RadioMonitor.getSquelch();     //Noise
        //restart the watchdog timer once a packet has been received
        call WDTimer.start(TIMER_ONE_SHOT, WATCHDOG_MISSED_PACKETS * timer_rate);
        //Add this neighbour to the queue if applicable
        post processNeighbour();
    }
    return pMsg;
  }
}
```

LRNET.H

```
/*                                                                      tab:4
 * "Copyright (c) 2000-2003 The Regents of the University  of California.
 * All rights reserved.
 *
 * Permission to use, copy, modify, and distribute this software and its
 * documentation for any purpose, without fee, and without written agreement is
 * hereby granted, provided that the above copyright notice, the following
 * two paragraphs and the author appear in all copies of this software.
 *
 * IN NO EVENT SHALL THE UNIVERSITY OF CALIFORNIA BE LIABLE TO ANY PARTY FOR
 * DIRECT, INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES ARISING OUT
 * OF THE USE OF THIS SOFTWARE AND ITS DOCUMENTATION, EVEN IF THE UNIVERSITY OF
 * CALIFORNIA HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
 *
 * THE UNIVERSITY OF CALIFORNIA SPECIFICALLY DISCLAIMS ANY WARRANTIES,
 * INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY
 * AND FITNESS FOR A PARTICULAR PURPOSE.  THE SOFTWARE PROVIDED HEREUNDER IS
 * ON AN "AS IS" BASIS, AND THE UNIVERSITY OF CALIFORNIA HAS NO OBLIGATION TO
 * PROVIDE MAINTENANCE, SUPPORT, UPDATES, ENHANCEMENTS, OR MODIFICATIONS."
 *
 * Copyright (c) 2002-2003 Intel Corporation
 * All rights reserved.
 *
 * This file is distributed under the terms in the attached INTEL-LICENSE
 * file. If you do not find these files, copies can be found by writing to
 * Intel Research Berkeley, 2150 Shattuck Avenue, Suite 1300, Berkeley, CA,
 * 94704.  Attention:  Intel License Inquiry.
 *
 *-------------------------------------------------------------------------------------
 */
/** File: LRNet.h (Long-Range WSN testing application)
 *
 * Header file for Long-Range Wireless Sensor Network (LRNet). Periodically samples
 * the signal strength of neighbouring nodes and forwards these measurements through
 * the network to the base station, along with measurements of battery and solar
 * voltage. This application is based on the Surge application that is distributed
 * with TinyOS and has been modified by S Willis for the JCUMote long-range WSN
 * testing. All code is copyright of TinyOS and is bound by the above copyright
 * message. Sections of modified or added code are indicated by an "SW" comment and
 * are copyright of S Willis.
 *
 * @author Simon Willis
 * @author Unknown (Original Surge Author not noted in original file)
 * @modified 18/12/2006
 **/
 //------------------------------------------------------------------------------------

enum{
    INITIAL_TIMER_RATE = 1024 * 8,    //8 second transmit rate
    FOCUS_TIMER_RATE = 1000,
    FOCUS_NOTME_TIMER_RATE = 1000,
    WATCHDOG_MISSED_PACKETS = 15      //15*8sec=2 minutes watchdog timer
};

//Surge message types
enum {
  SURGE_TYPE_SENSORREADING = 0,
  SURGE_TYPE_ROOTBEACON = 1,
  SURGE_TYPE_SETRATE = 2,
  SURGE_TYPE_SLEEP = 3,
  SURGE_TYPE_WAKEUP = 4,
  SURGE_TYPE_FOCUS = 5,
  SURGE_TYPE_UNFOCUS = 6
};

//SW: altered this struct
//LRNet message type definition (identical size to Surge packet)
typedef struct LRNetMsg {
  uint8_t type;
  uint16_t solar;
  uint16_t parentaddr;
  uint32_t seq_no;
  uint16_t neighbourAddr;
  uint16_t strength;
  uint16_t noise;
} __attribute__ ((packed)) LRNetMsg;

//SW: Struct for Neighbour data
typedef struct NeighbourStats {
  uint16_t address;
  uint16_t strength;
  uint16_t noise;
} NeighbourStats;

//LRNet messages have same AM type as Surge.
//Allows applications such as SurgeView to be used
enum {
  AM_SURGEMSG = 17
};
```

L.1.2 SIGNALSTRENGTHTEST – NODE RANGE TESTING

SIGNALSTRENGTHTEST.NC – INSTALLED ON RECEIVER

```
/**
 * File: SignalStrengthTest.nc
 *
 * Used for testing the transmission range of nodes. The transmitter is loaded with
 * CntToLedsAndRfm, which periodically outputs a counter value. This value is
 * received by SignalStrengthTest, which is installed on the receiver. This program
 * stores the signal strength, noise and packet loss in a measurements buffer and
 * then outputs these measurements once 10 have been collected. Oscilloscope is used
 * to output the measurements over the UART. The program also displays the counter
 * value on the LEDs. SimpleCmd is used to allow a 'Flush' command to be issued from
 * the PC. This causes the program to output the measurements buffer immediately.
 * This file is written by and is copyright of S Willis. It uses IntToLeds, Oscope,
 * SimpleCmd and Deluge which are distributed with TinyOS.
 *
 * @author Simon Willis
 * @modified 29/11/2006
 *
 **/
 //------------------------------------------------------------------------------------
includes IntMsg;
includes SimpleCmdMsg;


configuration SignalStrengthTest {
         provides interface ProcessCmd;
}

implementation
{
  components Main
              , SignalStrengthTestM        //The module
             , TimerC                      //General timer
             , GenericComm                 //Output over the UART
             , IntToLeds                   //Outputs the counter on LEDs
             , TH7122RadioIntM             //SW:Radio interface
             , OscopeC                     //Outputs values to the Oscilloscope
application
             , DelugeC                             //Remote programmiing
             , SimpleCmdM;                 //Receives commands from UART

  //StdControl Interfaces
  Main.StdControl -> DelugeC;
  Main.StdControl -> SimpleCmdM;
  Main.StdControl -> SignalStrengthTestM;
  Main.StdControl -> TimerC;
  Main.StdControl -> IntToLeds.StdControl;
  Main.StdControl -> OscopeC.StdControl;

  //Wire the provided ProcessCmd interface to SimpleCmd
  ProcessCmd = SimpleCmdM.ProcessCmd;

  //Outputs counter value on LEDs
  SignalStrengthTestM.IntOutput -> IntToLeds.IntOutput;
  //Communications control interface
  SignalStrengthTestM.CommControl -> GenericComm;
  //event handler for received messages
  SignalStrengthTestM.ReceiveIntMsg -> GenericComm.ReceiveMsg[AM_INTMSG];
  //Determine the noise level
  SignalStrengthTestM.RadioMonitor -> TH7122RadioIntM;
  //Three oscilloscope channels (strength, noise and packet loss)
  SignalStrengthTestM.OscopeCh0 -> OscopeC.Oscope[0];
  SignalStrengthTestM.OscopeCh1 -> OscopeC.Oscope[1];
  SignalStrengthTestM.OscopeCh2 -> OscopeC.Oscope[2];
  //Event handler of command messages (flush)
  SimpleCmdM.ReceiveCmdMsg -> GenericComm.ReceiveMsg[AM_SIMPLECMDMSG];
  //SimpleCmdM requires access to Oscope to flush the buffer
  SimpleCmdM.OscopeCh0 -> OscopeC.Oscope[0];
  SimpleCmdM.OscopeCh1 -> OscopeC.Oscope[1];
  SimpleCmdM.OscopeCh2 -> OscopeC.Oscope[2];
}
```

SIGNALSTRENGTHTESTM.NC

```
/**
 * File: SignalStrengthTest.nc
 *
 * Used for testing the transmission range of nodes. The transmitter is loaded with
 * CntToLedsAndRfm, which periodically outputs a counter value. This value is
 * received by SignalStrengthTest, which is installed on the receiver. This program
 * stores the signal strength, noise and packet loss in a measurements buffer and
 * then outputs these measurements once 10 have been collected. Oscilloscope is used
 * to output the measurements over the UART. The program also displays the counter
 * value on the LEDs. SimpleCmd is used to allow a 'Flush' command to be issued from
 * the PC. This causes the program to output the measurements buffer immediately.
 * This file is written by and is copyright of S Willis. It uses IntToLeds, Oscope,
 * SimpleCmd and Deluge which are distributed with TinyOS.
 *
 * @author Simon Willis
 * @modified 29/11/2006
 *
 **/
 //------------------------------------------------------------------------------------

includes IntMsg;

module SignalStrengthTestM
{
  provides interface StdControl;
  uses {
    interface Leds;
    interface StdControl as CommControl;
    interface ReceiveMsg as ResetCounterMsg; //Reset counter messages
    interface ReceiveMsg as ReceiveIntMsg;   //Counter value
    interface IntOutput;                     //Output to LEDs
    interface RadioMonitor;                  //Radio noise
    interface Oscope as OscopeCh0;           //Oscope channels - strength
    interface Oscope as OscopeCh1;           //                    - noise
    interface Oscope as OscopeCh2;           //                    - lost packets
  }
}

implementation
{
  //Component variables
  uint16_t readingNumber;
  uint16_t lostPackets;       //Number of packets lost
  uint16_t lastCount;         //Last integer received

 /**
  * Initialises the CommControl component and variables.
  * @return Always returns SUCCESS.
  **/
  command result_t StdControl.init() {
        //Turn on radio and UART
        call CommControl.init();

      //Reset all variables
        atomic {
        readingNumber = 0;
              lastCount=0;
        lostPackets=0;
        }

        dbg(DBG_BOOT, "OSCOPE initialized\n");
        return SUCCESS;
  }

 /**
  * Starts the CommControl component.
  * @return Always returns SUCCESS.
  **/
  command result_t StdControl.start() {
    call CommControl.start();
    return SUCCESS;
  }

 /**
  * Stops the CommControl component.
  * @return Always returns SUCCESS.
  **/
  command result_t StdControl.stop() {
    call CommControl.stop();
    return SUCCESS;
  }

 /**
  * Signalled when an integer is received from the radio
  * Output the integer on the LEDs and store to the oscope buffer
  *
  * @attrib m pointer to received message
  * @return a pointer to a message (location to store next msg)
  **/
  event TOS_MsgPtr ReceiveIntMsg.receive(TOS_MsgPtr m) {
        IntMsg *message = (IntMsg *)m->data;  //Data component

        if ((((message->val)-lastCount) > 1) & (lastCount!=0))
               lostPackets+=(message->val)-lastCount-1;
        lastCount=message->val;

        call IntOutput.output(message->val);
        atomic {
               //Output strength, noise and lost packets to oscope buffers
               call OscopeCh0.put(m->strength);
               call OscopeCh1.put(call RadioMonitor.getSquelch());
               call OscopeCh2.put(lostPackets);
        }
        return m;
  }

 /**
  * Signalled once an integer has been output to the LEDs.
  *
  * @attrib success Success of int output
  * @return Always returns SUCCESS.
  **/
  event result_t IntOutput.outputComplete(result_t success) {
    return SUCCESS;
  }

 /**
  * Signalled when the reset message counter AM is received.
  *
  * @attrib Pointer to received message
  * @return Pointer to a message (location to store next msg)
  **/
  event TOS_MsgPtr ResetCounterMsg.receive(TOS_MsgPtr m) {
    atomic {
      readingNumber = 0;
    }
    return m;
  }
}
```

SIMPLECMDM.NC

```
/*                                                                tab:4
 * "Copyright (c) 2000-2003 The Regents of the University  of California.
 * All rights reserved.
 *
 * Permission to use, copy, modify, and distribute this software and its
 * documentation for any purpose, without fee, and without written agreement is
 * hereby granted, provided that the above copyright notice, the following
 * two paragraphs and the author appear in all copies of this software.
 *
 * IN NO EVENT SHALL THE UNIVERSITY OF CALIFORNIA BE LIABLE TO ANY PARTY FOR
 * DIRECT, INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES ARISING OUT
 * OF THE USE OF THIS SOFTWARE AND ITS DOCUMENTATION, EVEN IF THE UNIVERSITY OF
 * CALIFORNIA HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
 *
 * THE UNIVERSITY OF CALIFORNIA SPECIFICALLY DISCLAIMS ANY WARRANTIES,
 * INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY
 * AND FITNESS FOR A PARTICULAR PURPOSE.  THE SOFTWARE PROVIDED HEREUNDER IS
 * ON AN "AS IS" BASIS, AND THE UNIVERSITY OF CALIFORNIA HAS NO OBLIGATION TO
 * PROVIDE MAINTENANCE, SUPPORT, UPDATES, ENHANCEMENTS, OR MODIFICATIONS."
 *
 * Copyright (c) 2002-2003 Intel Corporation
 * All rights reserved.
 *
 * This file is distributed under the terms in the attached INTEL-LICENSE
 * file. If you do not find these files, copies can be found by writing to
 * Intel Research Berkeley, 2150 Shattuck Avenue, Suite 1300, Berkeley, CA,
 * 94704.  Attention:  Intel License Inquiry.
 */
/**
 * File: SimpleCmdM.nc
 *
 * Receives command messages and executes a corresponding command. Modified by
 * S Willis for use with SignalStrengthTest. When command '1' is received this
 * program will dump 10 zeros into the oscilloscope buffer, causing oscilloscope to
 * output any data remaining in the buffer. All code is copyright of TinyOS and is
 * bound by the above copyright message. Sections of modified or added code are
 * indicated by an "SW" comment and are copyright of S Willis.
 *
 * @author Simon Willis
 * @author Robert Szewczyk (Original SimpleCmdM author)
 * @author Su Ping (Original SimpleCmdM author)
 * @modified 29/11/2006
 **/
 //-----------------------------------------------------------------------------------


includes SimpleCmdMsg;

module SimpleCmdM {
  provides {
    interface StdControl;
    interface ProcessCmd;
  }

  uses {
    interface Oscope as OscopeCh0;           //SW: Added. Oscilloscope channel 0
    interface Oscope as OscopeCh1;           //SW: Added. Oscilloscope channel 1
    interface Oscope as OscopeCh2;           //SW: Added. Oscilloscope channel 2
    interface ReceiveMsg as ReceiveCmdMsg;
  }
}

implementation
{

  // declare module static variables here
  TOS_MsgPtr cur_msg;  // The current command message
  TOS_Msg buf;         // Free buffer for message reception

 /**
  * This task evaluates a command and executes it.
  * Signals ProcessCmd.sendDone() when the command has completed.
  *
  * @return Return: None
  **/
  task void cmdInterpret() {
    struct SimpleCmdMsg *cmd = (struct SimpleCmdMsg *)cur_msg->data;
    result_t status = SUCCESS;
    uint8_t i;

    // do local packet modifications: update the hop count and packet source
    cmd->hop_count++;
    cmd->source = TOS_LOCAL_ADDRESS;

    // Execute the command
    switch (cmd->action) {
    case DUMP_BUF:            //SW: Added. DUMP_BUF command
```

```
      for (i=0;i<10;i++) {    //Add 10 zeros to the oscilloscope buffer
        call OscopeCh0.put(0);
          call OscopeCh1.put(0);
          call OscopeCh2.put(0);
      }
    break;
    //SW: Removed LED_ON, LED_OFF, RADIO_QUIETER, RADIO_LOUDER, START_SENDING,
READ_LOG commands
    default:
      status = FAIL;
    }

    signal ProcessCmd.done(cur_msg, status); //Signal completion
  }

  /**
   *  Initialize the application.
   *  @return Success of component initialization.
   **/
  command result_t StdControl.init() {
    cur_msg = &buf;   //SW: deleted some initialisation code that was not needed
    return SUCCESS;
  }

  /**
   * Start the application.
   * @return Always returns <code>SUCCESS</code>
   **/
  command result_t StdControl.start(){
    return SUCCESS;   //SW: deleted comms start code that was not needed
  }

  /**
   * Stop the application.
   * @return Always returns <code>SUCCESS</code>
   **/
  command result_t StdControl.stop(){
    return SUCCESS;   //SW: deleted comms stop code that was not needed
  }

  //SW: deleted LoggerRead.readDone event. No longer needed
  /**
   * Post a task to process the message in 'pmsg'.
   * Called from receiver event handler
   *
   * @attrib pmsg Pointer to command message
   * @return Always returns <code>SUCCESS</code>
   **/
  command result_t ProcessCmd.execute(TOS_MsgPtr pmsg) {
    cur_msg = pmsg;           //Save the command message
    post cmdInterpret();      //Interpret the command
    return SUCCESS;
  }

  /**
   * Called upon message receive; invokes ProcessCmd.execute().
   *
   * @attrib pmsg Received command message
   * @return pointer to message
   **/
  event TOS_MsgPtr ReceiveCmdMsg.receive(TOS_MsgPtr pmsg){
    result_t retval;
    TOS_MsgPtr ret = cur_msg;

    retval = call ProcessCmd.execute(pmsg);  //Call execute command above
    if (retval==SUCCESS) {
      return ret;
    } else {
      return pmsg;
    }
  }

  /**
   * Default event handler for <code>ProcessCmd.done</code>.
   *
   * @attrib pmsg Pointer to command message
   * @attrib Result (success)of ProcessCmd
   * @return The value of 'status'.
   **/
  default event result_t ProcessCmd.done(TOS_MsgPtr pmsg, result_t status) {
    return status;
  }

  //SW: Removed SendLogMsg.sendDone event

} // end of implementation
```

SIMPLECMDMSG.H

```
/*                                                                  tab:4
 * "Copyright (c) 2000-2003 The Regents of the University  of California.
 * All rights reserved.
 *
 * Permission to use, copy, modify, and distribute this software and its
 * documentation for any purpose, without fee, and without written agreement is
 * hereby granted, provided that the above copyright notice, the following
 * two paragraphs and the author appear in all copies of this software.
 *
 * IN NO EVENT SHALL THE UNIVERSITY OF CALIFORNIA BE LIABLE TO ANY PARTY FOR
 * DIRECT, INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES ARISING OUT
 * OF THE USE OF THIS SOFTWARE AND ITS DOCUMENTATION, EVEN IF THE UNIVERSITY OF
 * CALIFORNIA HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
 *
 * THE UNIVERSITY OF CALIFORNIA SPECIFICALLY DISCLAIMS ANY WARRANTIES,
 * INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY
 * AND FITNESS FOR A PARTICULAR PURPOSE.  THE SOFTWARE PROVIDED HEREUNDER IS
 * ON AN "AS IS" BASIS, AND THE UNIVERSITY OF CALIFORNIA HAS NO OBLIGATION TO
 * PROVIDE MAINTENANCE, SUPPORT, UPDATES, ENHANCEMENTS, OR MODIFICATIONS."
 *
 * Copyright (c) 2002-2003 Intel Corporation
 * All rights reserved.
 *
 * This file is distributed under the terms in the attached INTEL-LICENSE
 * file. If you do not find these files, copies can be found by writing to
 * Intel Research Berkeley, 2150 Shattuck Avenue, Suite 1300, Berkeley, CA,
 * 94704.  Attention:  Intel License Inquiry.
 */

/**
 * File: SimpleCmdMsg.h
 *
 * This header file defines the AM_SIMPLECMDMSG message type for the SimpleCmd
 * application. This file has been modified by S Willis to suit the SignalStrengthTest
 * program. All code is copyright of TinyOS and is bound by the above copyright
 * message. Sections of modified or added code are indicated by an "SW" comment and
 * are copyright of S Willis.
 *
 * @author Simon Willis
 * @author Robert Szewczyk (Original SimpleCmdM author)
 * @author Su Ping (Original SimpleCmdM author)
 * @modified 29/11/2006
 **/
 //------------------------------------------------------------------------------------

enum {
 AM_SIMPLECMDMSG = 8,
 AM_LOGMSG=9
};

enum {
  DUMP_BUF = 1 //SW: Removed LED_ON, LED_OFF, RADIO_LOUDER, RADIO_QUIETER,
                    //START_SENSING and READ_LOG. Added DUMP_BUF
};

/*                    //SW: Commented out this struct. Not needed in
SignalStrengthTest
typedef struct {
    int nsamples;
    uint32_t interval;
} start_sense_args; */
```

```
/*                      //SW: Commented out this struct. Not needed in
SignalStrengthTest
typedef struct {
    uint16_t destaddr;
} read_log_args; */

// SimpleCmd message structure
typedef struct SimpleCmdMsg {
    int8_t seqno;       //Sequence number
    int8_t action;      //Action/command number
    uint16_t source;    //Source address
    uint8_t hop_count; //Hop count
    /*union {           //SW: Removed union. No longer required.
      start_sense_args ss_args;
      read_log_args rl_args;
      uint8_t untyped_args[0];
    } args; */
} SimpleCmdMsg;

// Log message structure

/*                      //SW: Commented out this struct. Not needed in
SignalStrengthTest
typedef struct LogMsg {
    uint16_t sourceaddr;
    uint8_t log[16];
} LogMsg; */
```

CNTTOLEDSANDRFM – INSTALLED ON TRANSMITTER

```
/*                                                                  tab:4
 * "Copyright (c) 2000-2003 The Regents of the University  of California.
 * All rights reserved.
 *
 * Permission to use, copy, modify, and distribute this software and its
 * documentation for any purpose, without fee, and without written agreement is
 * hereby granted, provided that the above copyright notice, the following
 * two paragraphs and the author appear in all copies of this software.
 *
 * IN NO EVENT SHALL THE UNIVERSITY OF CALIFORNIA BE LIABLE TO ANY PARTY FOR
 * DIRECT, INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES ARISING OUT
 * OF THE USE OF THIS SOFTWARE AND ITS DOCUMENTATION, EVEN IF THE UNIVERSITY OF
 * CALIFORNIA HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
 *
 * THE UNIVERSITY OF CALIFORNIA SPECIFICALLY DISCLAIMS ANY WARRANTIES,
 * INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY
 * AND FITNESS FOR A PARTICULAR PURPOSE.  THE SOFTWARE PROVIDED HEREUNDER IS
 * ON AN "AS IS" BASIS, AND THE UNIVERSITY OF CALIFORNIA HAS NO OBLIGATION TO
 * PROVIDE MAINTENANCE, SUPPORT, UPDATES, ENHANCEMENTS, OR MODIFICATIONS."
 *
 * Copyright (c) 2002-2003 Intel Corporation
 * All rights reserved.
 *
 * This file is distributed under the terms in the attached INTEL-LICENSE
 * file. If you do not find these files, copies can be found by writing to
 * Intel Research Berkeley, 2150 Shattuck Avenue, Suite 1300, Berkeley, CA,
 * 94704.  Attention:  Intel License Inquiry.
 */

/**
This application blinks the LEDS as a binary counter and also send
a radio packet sending the current value of the counter.
**/


configuration CntToLedsAndRfm {
}
implementation {
  components Main, Counter, IntToLeds, IntToRfm, TimerC;

  Main.StdControl -> Counter.StdControl;
  Main.StdControl -> IntToLeds.StdControl;
  Main.StdControl -> IntToRfm.StdControl;
  Main.StdControl -> TimerC.StdControl;
  Counter.Timer -> TimerC.Timer[unique("Timer")];
  IntToLeds <- Counter.IntOutput;
  Counter.IntOutput -> IntToRfm;
}
```

L.1.3 TXRXTEST

TXRXTEST.NC

```
/**
 * File: TxRxTest.nc
 *
 * Periodically switches the TH7122 on the JCUMote between transmit and receive mode.
 * Used for tuning the PLL. This file is written by and is copyright of S Willis.
 *
 * @author Simon Willis
 * @modified 7/11/2006
 **/
 //------------------------------------------------------------------------------------

configuration TxRxTest {
}
implementation {
	components Main
		, TxRxTestM		//Module
		, TH7122ControlM	//Radio control
		, TimerC		//Timer
		, HPLTH7122M;		//Radio hardware presentation level
		//, LedsC;		//Uncomment to activate LEDs

	//Wire the timer and radio standard control interfaces
	Main.StdControl -> TxRxTestM.StdControl;
	Main.StdControl -> TH7122ControlM;
	Main.StdControl -> TimerC;

	TxRxTestM.TH7122Control -> TH7122ControlM;		//Radio control interface
	TxRxTestM.Timer -> TimerC.Timer[unique("Timer")];	//Radio timer
	TH7122ControlM.HPLMelexis -> HPLTH7122M;			//HPL
	//TH7122ControlM.Leds -> LedsC;				//Uncomment to
activate LEDs
}
```

TXRXTESTM.NC

```
/**
 * File: TxRxTestM.nc
 *
 * Periodically switches the TH7122 on the JCUMote between transmit and receive mode.
 * Used for tuning the PLL. This file is written by and is copyright of S Willis.
 *
 * @author Simon Willis
 * @modified 31/08/2006
 **/
 //------------------------------------------------------------------------------------

module TxRxTestM {
	provides {
		interface StdControl;		//Standard Control
	}
	uses {
		interface Timer;		//Timer
		interface TH7122Control;	//Radio control
	}
}

implementation {

	const uint32_t TIMER_INTERVAL = 500;	//Switch every 500ms
	bool bTxOn;				//Current state

	/**
```

```
         * Init command of standard control interface
         *
         * @return Always returns SUCCESS
         **/
        command result_t StdControl.init() {
                return SUCCESS;
        }

        /**
         * Start command of standard control interface
         * Switch radio to Rx mode, start the timer
         *
         * @return Always returns SUCCESS
         **/
        command result_t StdControl.start() {
                call Timer.start(TIMER_REPEAT, TIMER_INTERVAL);      //Start timer
                call TH7122Control.rxMode();          //Switch to receiver mode
                bTxOn=FALSE;                          //Not in TX mode
                return SUCCESS;
        }

        /**
         * Stop command of standard control interface
         * Stop the timer
         *
         * @return Always returns SUCCESS
         **/
        command result_t StdControl.stop() {
                call Timer.stop();                    //Stop the timer
                return SUCCESS;
        }

        /**
         * Timer fired event
         * Toggle between Tx and Rx modes
         *
         * @return Always returns SUCCESS
         **/
        event result_t Timer.fired() {
                if (bTxOn) {   //If in transmit mode
                        call TH7122Control.rxMode();  //Switch to Rx mode
                        bTxOn=FALSE;                  //Change the status bit
                }
                else {  //Else in Rx mode
                        call TH7122Control.txMode();  //Switch to Tx mode
                        bTxOn=TRUE;                   //Change the status bit
                }
                return SUCCESS;
        }

}
```

L.1.4 TXTEST2

TXTEST2.NC

```
/**
 * File: TxTest2.nc
 *
 * Constantly outputs a bit stream to test the radio transceiver and the manchester
 * encoder. This file is written by and is copyright of S Willis.
 *
 * @author Simon Willis
 * @modified 24/12/2006
 **/
 //-----------------------------------------------------------------------------------

configuration TxTest2 {
}
implementation {
        components Main
                , TxTest2M              //Module
                , TH7122ControlM        //Transceiver control
```

```
		, HPLTH7122M		//Transceiver HPL
		, ManchesterByteFIFOM	//Manchester encoder
		, HPLPowerManagementM	//Power management
		, HPLTimer1M		//Timer used for manchester encoder
		, LedsC;		//LEDs

	Main.StdControl -> TxTest2M.StdControl;			//Standard Control
	TxTest2M.SubControl -> TH7122ControlM;			//Standard Control
	TxTest2M.TH7122Control -> TH7122ControlM;		//Transceiver control
	TxTest2M.ManchesterByteFIFO -> ManchesterByteFIFOM;	//Manchester encoder
	TH7122ControlM.HPLMelexis -> HPLTH7122M;		//Transceiver HPL
	ManchesterByteFIFOM.Clock16 -> HPLTimer1M;		//Manchester clock
	ManchesterByteFIFOM.PowerManagement -> HPLPowerManagementM; //Power management
(manchester encoder)
	ManchesterByteFIFOM.TimerCapture -> HPLTimer1M;		//Input capture
(manchester decoder)
	TxTest2M.Leds->LedsC;					//LEDs
}
```

TXTEST2M.NC

```
/**
 * File: TxTest2.nc
 *
 * Constantly outputs a bit stream to test the radio transceiver and the manchester
 * encoder. This file is written by and is copyright of S Willis.
 *
 * @author Simon Willis
 * @modified 24/12/2006
 **/
 //-----------------------------------------------------------------------------------

module TxTest2M {
	provides {
		interface StdControl;			//Standard Control
	}
	uses {
		interface TH7122Control;		//Transceiver control
		interface StdControl as SubControl;	//Standard control
		interface ManchesterByteFIFO;		//Manchester encoder
		interface Leds;				//LEDs
	}
}
implementation {
	uint8_t byteNum;		//Variables (current byte number)

	/**
	 * Init command of standard control interface
	 * Initialise radio interface and LEDs
	 *
	 * @return Always returns SUCCESS
	 **/
	command result_t StdControl.init() {
		call SubControl.init();
		call Leds.init();
		return SUCCESS;
	}

	/**
	 * Start command of standard control interface
	 * Send a byte.
	 *
	 * @return Always returns SUCCESS
	 **/
	command result_t StdControl.start() {
		call SubControl.start();
		call ManchesterByteFIFO.initSlave();	//Initialise Manchester encoder
(Rx mode)
		call ManchesterByteFIFO.enableIntr();
		call ManchesterByteFIFO.writeByte(0xaa);	//Write a byte
		call TH7122Control.txMode();		//Switch radio to transmit
mode
		call ManchesterByteFIFO.txMode();	//Send the byte
		byteNum=0;				//Current byte number
		return SUCCESS;
	}

	/**
	 * Stop command of standard control interface
	 * Shutdown the radio
	 *
	 * @return Always returns SUCCESS
	 **/
	command result_t StdControl.stop() {
		call SubControl.stop();			//Stop the radio IC
		return SUCCESS;
	}

	/**
	 * dataReady event generated by Manchester Encoder
	 * Indicates that a transmission has completed. Time to send another byte
	 *
	 * @attrib data The sent data
	 * @return Always returns SUCCESS
	 **/
	async event result_t ManchesterByteFIFO.dataReady(uint8_t data) {
		call Leds.greenToggle();	//Toggle the green LED
		byteNum++;			//Increment the current byte number
//		if (byteNum<=10) {		//If it's less than 10 then send AA
			call ManchesterByteFIFO.writeByte(0xAA);
//		}
/*		else {				//Else send some other bytes
			switch (byteNum) {
				case 11:
					call ManchesterByteFIFO.writeByte(0xE8);
					break;
				case 12:
					call ManchesterByteFIFO.writeByte(0x46);
					break;
				case 13:
					call ManchesterByteFIFO.writeByte(0xED);
					break;
				case 14:
					call ManchesterByteFIFO.writeByte(0xDB);
					break;
				case 15:
					call ManchesterByteFIFO.writeByte(0x5B);
					break;
				default:
					call ManchesterByteFIFO.writeByte(0xAA);
			}
		}
		if (byteNum==15)
			byteNum=0; */
		return SUCCESS;
	}

	/**
	 * Manchester code violation. Only used in receive mode.
	 * Do nothing
	 *
	 * @return Always returns SUCCESS
	 **/
	async event result_t ManchesterByteFIFO.manchesterViolation() {
		return SUCCESS;			//Do nothing
	}

}
```

L.1.5 RXTEST2 – RECEIVER TEST PROGRAM

RXTEST2.NC

```
/**
 * File: RxTest2.nc
 *
 * Used for testing and tuning the receiver and checking the manchester decoder.
 * This program sends each received byte over the UART. This file is written by and
 * is copyright of S Willis.
 *
 * @author Simon Willis
 * @modified 29/11/2006
 **/
 //-------------------------------------------------------------------------------------

includes IntMsg;       //Used for transmitting the byte

configuration RxTest2 {
}

implementation {
       components Main
               , RxTest2M             //Module
               , TH7122ControlM       //Radio Control
               , HPLTH7122M           //Radio hardware presentation layer
               , ManchesterByteFIFOM  //Manchester encoder/decoder
               , HPLPowerManagementM  //Power management (machester encoder)
               , HPLTimer1M           //Timer used by Manchester encoder
               , UARTComm as Comm     //UART communications
               , LedsC;               //LEDs

       //Wire standard control interfaces
       Main.StdControl -> RxTest2M.StdControl;
       RxTest2M.SubControl -> TH7122ControlM;
       RxTest2M.TH7122Control -> TH7122ControlM;
       Main.StdControl -> Comm;

       //Manchester decoder
       RxTest2M.ManchesterByteFIFO -> ManchesterByteFIFOM;
       RxTest2M.Leds -> LedsC.Leds;                  //LEDs
       TH7122ControlM.HPLMelexis -> HPLTH7122M;      //HPL for radio control
       ManchesterByteFIFOM.PowerManagement -> HPLPowerManagementM; //Power management
for manchester decoder
       ManchesterByteFIFOM.TimerCapture -> HPLTimer1M;             //Timer for
manchester decoder
       ManchesterByteFIFOM.Clock16 -> HPLTimer1M;            //Timer for manchester
decoder
       ManchesterByteFIFOM.Leds->LedsC;                      //LEDs for manchester
decoder (diagnostic)

       RxTest2M.Send -> Comm.SendMsg[AM_INTMSG];             //Send and int message
over UART

}
```

RXTEST2M.NC

```
/**
 * File: RxTest2M.nc
 *
 * Used for testing and tuning the receiver and checking the manchester decoder.
 * This program sends each received byte over the UART. This file is written by and
 * is copyright of S Willis.
 *
 * @author Simon Willis
 * @modified 28/12/2006
 **/
 //-------------------------------------------------------------------------------------

includes IntMsg;       //Used for transmitting the byte
```

```
module RxTest2M {
        provides {
                interface StdControl;
        }
        uses {
                interface TH7122Control;              //Radio control
                interface ManchesterByteFIFO;         //Manchester decoder
                interface StdControl as SubControl;   //Standard control of radio
                interface SendMsg as Send;            //Send message (UART)
                interface Leds;                       //LEDs
        }
}
implementation {
        //Define variables
        bool pending;           //Transmission pending
        TOS_Msg data;           //data packet
        uint8_t newData;        //New number received
        uint8_t temp[100];      //Temporary Buffer (used for JTAG debugging)
        uint8_t bitCounter;     //Count of bits received

        /**
         * Init command of Standard Control interface
         *
         * @return Always returns SUCCESS
         **/
        command result_t StdControl.init() {
                call SubControl.init();               //Initialise radio
                call Leds.init();                     //Initialised LEDs
                return SUCCESS;
        }

        /**
         * Start command of Standard Control interface
         * Starts the radio and manchester decoder
         *
         * @return Always returns success
        command result_t StdControl.start() {
                call SubControl.start();                      //Start the radio
                call TH7122Control.biasOn();                  //Transceiver to IDLE mode
                call ManchesterByteFIFO.initSlave();          //Initialise manchester
decoder
                call TH7122Control.rxMode();                  //Transceiver to RECEIVE
mode
                call ManchesterByteFIFO.rxMode();             //Manchester decoder to
receive mode
                call ManchesterByteFIFO.enableIntr();         //Start the manchester
decoder
                call Leds.greenOn();                          //LED on
                return SUCCESS;
        }

        /**
         * Stop command of the Standard Control interface
         * Stops everything
         *
         * @return Always returns SUCCESS
         **/
        command result_t StdControl.stop() {
                call Leds.redOn();            //Red LED on - shows status
                call Leds.greenOff();         //Green LED off
                call SubControl.stop();       //Stop the radio
                return SUCCESS;
        }

        /**
         * Task to send the received integers
         **/
        task void sendData () {
                IntMsg *message = (IntMsg *)data.data;

//              call Leds.greenToggle();
        if (!pending)           //If no transmission is pending then assemble the packet
        {
                        atomic {
                                //Assemble the packet
```

```
                              message->val = (uint16_t) newData;    //Latest received
value
                              pending = TRUE;
        //Transmission is now pending
                              message->src = TOS_LOCAL_ADDRESS;     //Source address
                      }

        }
                call Send.send(TOS_UART_ADDR, sizeof(IntMsg), &data);
        //Send the data
        }

        //SPI byte has received. Send it off.
        /**
         *Event handler for byte received from manchester decoder
         *Stores the byte in the buffer and sends it over the UART
         *
         * @attrib value Received byte
         * @return Always returns SUCCESS
         **/
        async event result_t ManchesterByteFIFO.dataReady(uint8_t value) {
                atomic newData=value;
                temp[bitCounter]=newData;       //Store the value in the temp buffer
                bitCounter++;                   //Increment the counter
                if (bitCounter==99)             //Roll the counter to the start
                        bitCounter=0;
                post sendData();                //Send the received byte over the UART

                if ((value==(0xAA)) || (value==(0x55)))       //If we have a preamble
then toggle green LED
                        call Leds.greenToggle();
                else
                        call Leds.yellowToggle();             //Else toggle yellow LED

                return SUCCESS;
        }

        /**
         * Event handler for a completed transmission over the UART
         *
         * @attrib msg Pointer to transmitted message
         * @return Always returns SUCCESS
         **/
        event result_t Send.sendDone(TOS_MsgPtr msg, result_t success) {
                atomic {
                        pending=FALSE; //Transmission is complete. Reset the pending bit
                }
                return SUCCESS;
        }

        /**
         * Event handler for a violation detected by the Manchester decoder
         *
         * @return Always returns SUCCESS
         **/
        async event result_t ManchesterByteFIFO.manchesterViolation() {
                call Leds.redToggle(); //Toggle the red LED to indicate Manchester
violation
                return SUCCESS;
        }

}
```

L.1.6 REVERSEUARTEEPROM – UART AND EEPROM TEST PROGRAM

REVERSEUARTEEPROMC.NC

```
/* "Copyright (c) 2000-2003 The Regents of the University of California.
 * All rights reserved.
 *
 * Permission to use, copy, modify, and distribute this software and its
 * documentation for any purpose, without fee, and without written agreement
 * is hereby granted, provided that the above copyright notice, the following
 * two paragraphs and the author appear in all copies of this software.
 *
 * IN NO EVENT SHALL THE UNIVERSITY OF CALIFORNIA BE LIABLE TO ANY PARTY FOR
 * DIRECT, INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES ARISING OUT
 * OF THE USE OF THIS SOFTWARE AND ITS DOCUMENTATION, EVEN IF THE UNIVERSITY
 * OF CALIFORNIA HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
 *
 * THE UNIVERSITY OF CALIFORNIA SPECIFICALLY DISCLAIMS ANY WARRANTIES,
 * INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY
 * AND FITNESS FOR A PARTICULAR PURPOSE.  THE SOFTWARE PROVIDED HEREUNDER IS
 * ON AN "AS IS" BASIS, AND THE UNIVERSITY OF CALIFORNIA HAS NO OBLIGATION TO
 * PROVIDE MAINTENANCE, SUPPORT, UPDATES, ENHANCEMENTS, OR MODIFICATIONS."
 *
 *---------------------------------------------------------------------------------
 */
/** File: ReverseUARTEEPROMC.nc (Long-Range WSN testing application)
 *
 * Program to test the UART and EEPROM on the Mica2 / JCUMote. This program receives
 * a string from the UART, reverses it and stores in on the EEPROM. The program then
 * extracts the string from the EEPROM and outputs it on the UART. This program can
 * be used with TestReverseUART.java which sends and receives the strings. The
 * program is based on ReverseUART which is distributed with TinyOS and has been
 * modified by S Willis to include the EEPROM interface. All code is copyright of
 * TinyOS and is bound by the above copyright message. Sections of modified or added
 * code are indicated by an "SW" comment and are copyright of S Willis.

 *
 * @author Simon Willis
 * @author Cory Sharp (Original ReverseUART Author)
 * @modified 22/07/2006
 **/
 //--------------------------------------------------------------------------------

includes ReverseUART;

configuration ReverseUARTEEPROMC
{
}
implementation
{
  components Main
       , ReverseUARTEEPROMM   //Module
       , UARTComm             //UART Communications
       , LedsC                //LEDs for indication
       , Logger;              //SW: Added. Logger interface to EEPROM

  // Wire standard control interface
  Main.StdControl -> UARTComm;
  Main.StdControl -> ReverseUARTEEPROMM;
  Main.StdControl -> Logger;         //SW: Added. Standard Control interface on logger

  ReverseUARTEEPROMM.LoggerWrite -> Logger.LoggerWrite;     //SW: Added. Logger write
interface
  ReverseUARTEEPROMM.LoggerRead -> Logger.LoggerRead;       //SW: Added. Logger read
interface
  ReverseUARTEEPROMM.SendMsg -> UARTComm.SendMsg[ AM_REVERSEUARTMSG ];     //Send
message on UART
  ReverseUARTEEPROMM.ReceiveMsg -> UARTComm.ReceiveMsg[ AM_REVERSEUARTMSG ];
       //Receive message on UART
  ReverseUARTEEPROMM.Leds -> LedsC;          //LEDs
}
```

REVERSEUARTEEPROMM.NC

```
// $Id: ReverseUARTM.nc,v 1.2 2005/07/12 07:38:43 cssharp Exp $

/* "Copyright (c) 2000-2003 The Regents of the University of California.
 * All rights reserved.
 *
 * Permission to use, copy, modify, and distribute this software and its
 * documentation for any purpose, without fee, and without written agreement
 * is hereby granted, provided that the above copyright notice, the following
 * two paragraphs and the author appear in all copies of this software.
 *
 * IN NO EVENT SHALL THE UNIVERSITY OF CALIFORNIA BE LIABLE TO ANY PARTY FOR
 * DIRECT, INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES ARISING OUT
 * OF THE USE OF THIS SOFTWARE AND ITS DOCUMENTATION, EVEN IF THE UNIVERSITY
 * OF CALIFORNIA HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
 *
 * THE UNIVERSITY OF CALIFORNIA SPECIFICALLY DISCLAIMS ANY WARRANTIES,
 * INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY
 * AND FITNESS FOR A PARTICULAR PURPOSE.  THE SOFTWARE PROVIDED HEREUNDER IS
 * ON AN "AS IS" BASIS, AND THE UNIVERSITY OF CALIFORNIA HAS NO OBLIGATION TO
 * PROVIDE MAINTENANCE, SUPPORT, UPDATES, ENHANCEMENTS, OR MODIFICATIONS."
 *
 *----------------------------------------------------------------------------------
 */
/** File: ReverseUARTEEPROMM.nc (Long-Range WSN testing application)
 *
 * Program to test the UART and EEPROM on the Mica2 / JCUMote. This program receives
 * a string from the UART, reverses it and stores in on the EEPROM. The program then
 * extracts the string from the EEPROM and outputs it on the UART. This program can
 * be used with TestReverseUART.java which sends and receives the strings. The
 * program is based on ReverseUART which is distributed with TinyOS and has been
 * modified by S Willis to include the EEPROM interface. All code is copyright of
 * TinyOS and is bound by the above copyright message. Sections of modified or added
 * code are indicated by an "SW" comment and are copyright of S Willis.

 *
 * @author Simon Willis
 * @author Cory Sharp (Original ReverseUART Author)
 * @modified 22/07/2006
 **/
 //---------------------------------------------------------------------------------

module ReverseUARTEEPROMM
{
  provides interface StdControl;      //Standard Control
  uses interface SendMsg;             //Send message over UART
  uses interface ReceiveMsg;          //Receive message from UART
  uses interface Leds;                //LEDs for indication/debugging
  uses interface LoggerWrite;         //SW: Added. Write to the EEPROM
  uses interface LoggerRead;          //SW: Added. Read from the EEPROM
}
implementation
{
  TOS_Msg m_msg;                      //TinyOS message
  bool m_sending;                     //Sending state variable
  char buff[16];                      //SW: Added. 16 bit received text buffer from
UART
  char * ptr = &buff[0];              //SW: Added. Pointer to receive buffer
  char readBuff[16];                  //SW: Added. Received text buffer from EEPROM
  char* readPtr = &readBuff[0];       //SW: Added. Pointer to read buffer

 /**
  * Standard Control initialisation command
  *
  * @return Always returns SUCCESS
  **/
  command result_t StdControl.init()
  {
    m_sending = FALSE;        //Set the sending status to false
    return SUCCESS;
  }

 /**
  * Standard Control start command
  * Does nothing. Program waits for text to be received from UART
  *
  * @return Always returns SUCCESS
  **/
  command result_t StdControl.start()
  {
    return SUCCESS;
  }

 /**
  * Standard Control stop command
  *
  * @return Always returns SUCCESS
  **/
  command result_t StdControl.stop()
  {
    return SUCCESS;
  }

 /**
  * Event handler called once UART message has been sent
  * Clears the data component of the TinyOS message
  *
  * @attrib msg Pointer to transmitted message
  * @attrib success Indicates whether the transmission was successful
  * @return Always returns SUCCESS
  **/
  event result_t SendMsg.sendDone( TOS_MsgPtr msg, result_t success )
  {
    int i;
    call Leds.greenToggle();  //Toggle the green LED to indicate that the transmission
was successful
    m_sending = FALSE;        //Clear the sending status variable
    for (i=0;i<16;i++)
      m_msg.data[i]=0;        //SW: Added. Reset the transmitted messsage (debugging)
    return SUCCESS;
  }

 /**
  * Task to send the message over UART
  *
  **/
  task void sendMsg()
  {
    //Transmit the message
    if( call SendMsg.send( TOS_UART_ADDR, m_msg.length, &m_msg ) == FAIL )
      m_sending = FALSE;      // Clear the sending status variable
  }

 /**
  * Function to determine the length of an array
  * Searches for non-zero characters
  *
  * @attrib data Data array
  * @attrib maxlen Maximum length of the array to search through
  * @return Returns the length of the data/text in the array
  **/
  int szLen( int8_t* data, int maxlen )
  {
    int i;
    for( i=0; i<maxlen; i++ )
    {
      if( data[i] == 0 )      //If the value at the current index is 0 then this is
the end of the string
        return i;             //The current index is the same as the string length
    }
    return maxlen;            //String fills whole array
  }

 /**
  * Event handler called when a message is received from the UART
  * Reverses the received data and calls task to write it to the EEPROMM
  *
  * @attrib msg Pointer to the received message
  * @return Returns pointer to received message
  **/
  event TOS_MsgPtr ReceiveMsg.receive( TOS_MsgPtr msg )
  {
    call Leds.redToggle();    //Toggle the red LED for indication
    if( m_sending == FALSE )  //If we're not already sending then continue
    {
      const int n = szLen( msg->data, sizeof(msg->data) ) - 1;     //Determine the
size of the string
      int i;

      //SW: Removed. Section of code that saved data into a second buffer for sending

          for (i=0; i<16; i++) //SW: Added. Copy and reverse first 16 characters into
buffer
            buff[i]=msg->data[n-i];
```

```
         for (i=0; i<16; i++) {      //SW: Added. Clear the received message (debug)
              msg->data[i]=0;
              readBuff[i]=0;
         }

         i=n;         //SW: Added. Length of data
         if (n>6)     //SW: Added. Limit to 7 character (plus null)
              i=6;

      buff[i+1] = 0;   //SW: Added. Add the null character at the end.

      m_msg.length = msg->length;

      if( call LoggerWrite.append(ptr) == TRUE )     //SW: Modified. Add the data to
the EEPROM
              m_sending = TRUE;                    //Change the sending state
    }
    return msg;               //Return pointer for storage of next received message
  }

 /**
  * Event handler called when the data has been written to the EEPROM
  * Read data back from EEPROM. Event handler added by SW.
  *
  * @attrib status Status of write operation
  * @return Always returns SUCCESS
  **/
  event result_t LoggerWrite.writeDone( result_t status ) {
    call LoggerRead.readNext(readPtr);       //Now text is stored, read it back.
    //Text that is read back is stored in readBuff (pointed to by readPtr)
    return SUCCESS;
  }

 /**
  * Event handler called when data has been read from the EEPROM
  * Send the received data over the UART. Event handler added by SW
  *
  * @attrib packet Pointer to packet
  * @attrib success Success of read operation
  * @return Always returns SUCCESS
  event result_t LoggerRead.readDone(uint8_t * packet, result_t success) {
    // Send message only if read was successful
       //Copy data into m_msg and send

    int i;
    for (i=0;i<16;i++) //Copy the EEPROM data into the message to be sent
       m_msg.data[i]=readBuff[i];

    if( post sendMsg() == TRUE )      //Send the message over the UART
       m_sending = TRUE;                    //Set the sending status
    return SUCCESS;
  }
}
```

L.2 JCUMote Platform Specific Files

TH7122RadioC.nc

```
/*                                                                   tab:4
 *
 *
 * "Copyright (c) 2000-2002 The Regents of the University  of California.
 * All rights reserved.
 *
 * Permission to use, copy, modify, and distribute this software and its
 * documentation for any purpose, without fee, and without written agreement is
 * hereby granted, provided that the above copyright notice, the following
 * two paragraphs and the author appear in all copies of this software.
 *
 * IN NO EVENT SHALL THE UNIVERSITY OF CALIFORNIA BE LIABLE TO ANY PARTY FOR
 * DIRECT, INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES ARISING OUT
 * OF THE USE OF THIS SOFTWARE AND ITS DOCUMENTATION, EVEN IF THE UNIVERSITY OF
 * CALIFORNIA HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
 *
 * THE UNIVERSITY OF CALIFORNIA SPECIFICALLY DISCLAIMS ANY WARRANTIES,
 * INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY
 * AND FITNESS FOR A PARTICULAR PURPOSE.  THE SOFTWARE PROVIDED HEREUNDER IS
 * ON AN "AS IS" BASIS, AND THE UNIVERSITY OF CALIFORNIA HAS NO OBLIGATION TO
 * PROVIDE MAINTENANCE, SUPPORT, UPDATES, ENHANCEMENTS, OR MODIFICATIONS."
 *
 */
/*                                                                 tab:4
 *  IMPORTANT: READ BEFORE DOWNLOADING, COPYING, INSTALLING OR USING.  By
 *  downloading, copying, installing or using the software you agree to
 *  this license.  If you do not agree to this license, do not download,
 *  install, copy or use the software.
 *
 *  Intel Open Source License
 *
 *  Copyright (c) 2002 Intel Corporation
 *  All rights reserved.
 *  Redistribution and use in source and binary forms, with or without
 *  modification, are permitted provided that the following conditions are
 *  met:
 *
 *      Redistributions of source code must retain the above copyright
 *  notice, this list of conditions and the following disclaimer.
 *      Redistributions in binary form must reproduce the above copyright
 *  notice, this list of conditions and the following disclaimer in the
 *  documentation and/or other materials provided with the distribution.
 *      Neither the name of the Intel Corporation nor the names of its
 *  contributors may be used to endorse or promote products derived from
 *  this software without specific prior written permission.
 *
 *  THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
 *  ``AS IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
 *  LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A
 *  PARTICULAR PURPOSE ARE DISCLAIMED.  IN NO EVENT SHALL THE INTEL OR ITS
 *  CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL,
 *  EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
 *  PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR
 *  PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
 *  LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING
 *  NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
 *  SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
 *
 *-----------------------------------------------------------------------------------
 */
/**
 * Configuration for TH7122 radio transceiver IC as used on the JCUMote. This
 * file is based on the CC1000 configuration (CC1000RadioC.nc) that is distributed
 * with TinyOS and has been modified by S Willis for the JCUMote. All code is
 * copyright of TinyOS and is bound by the above copyright message. Sections of
 * modified or added code are indicated by an "SW" comment and are copyright of
 * S Willis.
 *
 * @author Simon Willis
 * @author Philip Buonadonna (original CC1000RadioC.nc author)
 * @modified 17/01/2007
 **/

configuration TH7122RadioC
{
  provides {
    interface StdControl;
    interface BareSendMsg as Send;
    interface ReceiveMsg as Receive;
    interface TH7122Control;         //SW: Modified. For TH7122
    interface RadioCoordinator as RadioReceiveCoordinator;
    interface RadioCoordinator as RadioSendCoordinator;
    interface MacControl;
    interface MacBackoff;
  }
}
implementation
{
  components TH7122RadioIntM as TH7122RadioM //SW: Modified. For TH7122. Data-link
layer implementation
```

```
       , TH7122ControlM      //SW: Modified. For TH7122. Control interface
       , HPLTH7122M          //SW: Modified. For TH7122. Hardware presentation layer
       , RandomLFSR          //Random number generator
       , ADCC                //Analogue to digital converter
       , ManchesterByteFIFOM //SW: Added. Manchester encoder/decoder
       , TimerC              //Timer interface
       , HPLPowerManagementM //Power management interface
       , LedsC               //LEDs for debugging
       , HPLTimer1M;         //SW: Added. Timer1 interface used for Manchester
encoder/decoder

  //Interfaces provided by the TH7122RadioIntM file
  StdControl = TH7122RadioM;          //SW: Modified. For TH7122. Standard Control
interface
  Send = TH7122RadioM;                //SW: Modified. For TH7122. Send interface
  Receive = TH7122RadioM;             //SW: Modified. For TH7122. Receive interface
  MacControl = TH7122RadioM;          //SW: Modified. For TH7122. MAC control interface
  MacBackoff = TH7122RadioM;          //SW: Modified. For TH7122. MAC backoff interface
  TH7122Control = TH7122ControlM;     //SW: Modified. For TH7122. Radio IC control
interface
  RadioReceiveCoordinator = TH7122RadioM.RadioReceiveCoordinator;   //SW: Modified. For
TH7122. Radio receive coordinator
  RadioSendCoordinator = TH7122RadioM.RadioSendCoordinator;         //SW: Modified. For
TH7122. Radio receive interface

  //SW: Modified. For TH7122. Standard Control interface of radio IC control wired to
data-link layer
  TH7122RadioM.TH7122StdControl -> TH7122ControlM;
  //SW: Modified. For TH7122. Radio IC control interface wired to data-link layer
  TH7122RadioM.TH7122Control -> TH7122ControlM;
  TH7122RadioM.Random -> RandomLFSR;  //SW: Modified. For TH7122. Random used for MAC
backoff periods etc
  TH7122RadioM.ADCControl -> ADCC;           //SW: Modified. For TH7122. Analogue-
digital converter interface
  TH7122RadioM.RSSIADC -> ADCC.ADC[TOS_ADC_TH_RSSI_PORT];   //SW: Modified. For
TH7122. Analogue-digital converter used for RSSI measurements
  TH7122RadioM.ManchesterByteFIFO -> ManchesterByteFIFOM;   //SW: Modified. For
TH7122. Manchester encoder/decoder

  TH7122RadioM.TimerControl -> TimerC.StdControl;                   //SW: Modified. For
TH7122. Timer used in data link layer
  TH7122RadioM.SquelchTimer -> TimerC.Timer[unique("Timer")];       //SW: Modified. For
TH7122. Squelch timer used in data link layer
  TH7122RadioM.WakeupTimer -> TimerC.Timer[unique("Timer")];        //SW: Modified. For
TH7122. Wakeup timer used in data link layer
  TH7122RadioM.Leds -> LedsC;                                       //SW: Modified. For
TH7122. LEDs Used for debugging

  //SW: Modified. For TH7122. Hardware presentation layer for TH7122 Control module
  TH7122ControlM.HPLMelexis -> HPLTH7122M;
  //SW: Modified. For TH7122. MCU power management
  TH7122RadioM.PowerManagement ->HPLPowerManagementM.PowerManagement;
  //SW: Added. For TH7122. MCU Power management used in manchester encoder/decoder
  ManchesterByteFIFOM.PowerManagement ->HPLPowerManagementM.PowerManagement;

  ManchesterByteFIFOM.Clock16 -> HPLTimer1M;        //SW: Added. Timer used for
Manchester encoder/decoder
  ManchesterByteFIFOM.TimerCapture -> HPLTimer1M;   //SW: Added. Timer input capture
used for Manchester decoder
  ManchesterByteFIFOM.Leds->LedsC;                  //SW: Added. LEDs for debugging
}
```

TH7122CONTROL.NC

```
/*                                                                    tab:4
 *
 *
 * "Copyright (c) 2000-2002 The Regents of the University  of California.
 * All rights reserved.
 *
 * Permission to use, copy, modify, and distribute this software and its
 * documentation for any purpose, without fee, and without written agreement is
 * hereby granted, provided that the above copyright notice, the following
 * two paragraphs and the author appear in all copies of this software.
 *
 * IN NO EVENT SHALL THE UNIVERSITY OF CALIFORNIA BE LIABLE TO ANY PARTY FOR
 * DIRECT, INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES ARISING OUT
 * OF THE USE OF THIS SOFTWARE AND ITS DOCUMENTATION, EVEN IF THE UNIVERSITY OF
 * CALIFORNIA HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
 *
 * THE UNIVERSITY OF CALIFORNIA SPECIFICALLY DISCLAIMS ANY WARRANTIES,
 * INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY
 * AND FITNESS FOR A PARTICULAR PURPOSE.  THE SOFTWARE PROVIDED HEREUNDER IS
 * ON AN "AS IS" BASIS, AND THE UNIVERSITY OF CALIFORNIA HAS NO OBLIGATION TO
 * PROVIDE MAINTENANCE, SUPPORT, UPDATES, ENHANCEMENTS, OR MODIFICATIONS."
 *
 */
/*                                                                 tab:4
 *  IMPORTANT: READ BEFORE DOWNLOADING, COPYING, INSTALLING OR USING.  By
 *  downloading, copying, installing or using the software you agree to
 *  this license.  If you do not agree to this license, do not download,
 *  install, copy or use the software.
 *
 *  Intel Open Source License
 *
 *  Copyright (c) 2002 Intel Corporation
 *  All rights reserved.
 *  Redistribution and use in source and binary forms, with or without
 *  modification, are permitted provided that the following conditions are
 *  met:
 *
 *      Redistributions of source code must retain the above copyright
 *  notice, this list of conditions and the following disclaimer.
 *      Redistributions in binary form must reproduce the above copyright
 *  notice, this list of conditions and the following disclaimer in the
 *  documentation and/or other materials provided with the distribution.
 *      Neither the name of the Intel Corporation nor the names of its
 *  contributors may be used to endorse or promote products derived from
 *  this software without specific prior written permission.
 *
 *  THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
 *  ``AS IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
 *  LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A
 *  PARTICULAR PURPOSE ARE DISCLAIMED.  IN NO EVENT SHALL THE INTEL OR ITS
 *  CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL,
 *  EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
 *  PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR
 *  PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
 *  LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING
 *  NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
 *  SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
 *
 *-----------------------------------------------------------------------------------
 */
/**
 * Control interface of TH7122. Based on CC1000Control.nc and modified for the
 * JCUMote which has a TH7122 transceiver with external PA. This file is based on the
 * CC10000Control.nc file that is distributed with TinyOS and has been modified by
 * S Willis for the JCUMote TH7122 radio transceiver IC. All code is copyright of
 * TinyOS and is bound by the above copyright message. Sections of modified or added
 * code are indicated by an "SW" comment and are copyright of S Willis.
 *
 * @author Simon Willis
 * @author Philip Buonadonna (original CC1000Control author)
 * @author Jaein Jeong (original CC1000Control author)
 * @modified 17/01/2007
 **/

/**
 * TH7122 Radio Control interface.
 **/
interface TH7122Control
{
  /**
   * Tune the radio to one of the frequencies available in the TH7122_Params table.
   * Calling Tune will also reset the rfpower and LockVal selections to the table
   * values.
   *
   * @param freq The index into the TH7122_Params table that holds the desired preset
   * frequency parameters.
   *
   * @return Status of the Tune operation.
   **/
  command result_t tunePreset(uint8_t freq);

  /**
   * Shift the TH7122 Radio into transmit mode.
   *
   * @return SUCCESS if the radio was successfully switched to TX mode.
   **/
  async command result_t txMode();

  /**
   * Shift the TH7122 Radio in receive mode.
   *
   * @return SUCCESS if the radio was successfully switched to RX mode.
   **/
  async command result_t rxMode();

  /**
   * Turn off the BIAS power on the TH7122 radio, but leave the core
   * and crystal oscillator powered.  This will result in power savings.
   *
   * @return SUCCESS when the BIAS powered is shutdown.
   **/
  command result_t biasOff();

  /**
   * Turn the BIAS power on. This function must be followed by a call
   * to either RxMode() or TxMode() to place the radio in a recieve/transmit
   * state respectively.
   *
   * @return SUCCESS when BIAS power has been restored.
   **/
  command result_t biasOn();

  /**
   * Set the transmit RF power value.  The input value is simply an arbitrary
   * value that is programmed into the TH7122 registers.  Consult the TH7122
   * datasheet for the resulting power output/current consumption values.
   *
   * @param power A power value between 0 and 3.
   *
   * @result SUCCESS if the radio power was adequately set.
   *
   **/
  command result_t setRFPower(uint8_t power);

  /**
   * Get the present RF power index.
   *
   * @result The power index value.
   **/
  command uint8_t  getRFPower();

  /**
   * Enable the LD signal on the TH7122.
   *
   * @param LockVal The boolean value for the TH7122_LOCKMODE register bit.
   *
   * @result SUCCESS if the selected signal was programmed into the TH7122
   *
   **/
  command result_t selectLock(bool LockVal);

  /**
   * Get the binary value from the LD pin.  Analog signals cannot be read using
   * function.
   *
   * @result 1 - Pin is high or 0 - Pin is low
   *
   **/
  command bool  getLock();
}
```

TH7122CONTROLM.NC

```
/*                                                                    tab:4
 *
 *
 * "Copyright (c) 2000-2002 The Regents of the University  of California.
 * All rights reserved.
 *
 * Permission to use, copy, modify, and distribute this software and its
 * documentation for any purpose, without fee, and without written agreement is
 * hereby granted, provided that the above copyright notice, the following
 * two paragraphs and the author appear in all copies of this software.
 *
 * IN NO EVENT SHALL THE UNIVERSITY OF CALIFORNIA BE LIABLE TO ANY PARTY FOR
 * DIRECT, INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES ARISING OUT
 * OF THE USE OF THIS SOFTWARE AND ITS DOCUMENTATION, EVEN IF THE UNIVERSITY OF
 * CALIFORNIA HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
 *
 * THE UNIVERSITY OF CALIFORNIA SPECIFICALLY DISCLAIMS ANY WARRANTIES,
 * INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY
 * AND FITNESS FOR A PARTICULAR PURPOSE.  THE SOFTWARE PROVIDED HEREUNDER IS
 * ON AN "AS IS" BASIS, AND THE UNIVERSITY OF CALIFORNIA HAS NO OBLIGATION TO
 * PROVIDE MAINTENANCE, SUPPORT, UPDATES, ENHANCEMENTS, OR MODIFICATIONS."
 *
 */
/*                                                                    tab:4
 *  IMPORTANT: READ BEFORE DOWNLOADING, COPYING, INSTALLING OR USING.  By
 *  downloading, copying, installing or using the software you agree to
 *  this license.  If you do not agree to this license, do not download,
 *  install, copy or use the software.
 *
 *  Intel Open Source License
 *
 *  Copyright (c) 2002 Intel Corporation
 *  All rights reserved.
 *  Redistribution and use in source and binary forms, with or without
 *  modification, are permitted provided that the following conditions are
 *  met:
 *
 *      Redistributions of source code must retain the above copyright
 *  notice, this list of conditions and the following disclaimer.
 *      Redistributions in binary form must reproduce the above copyright
 *  notice, this list of conditions and the following disclaimer in the
 *  documentation and/or other materials provided with the distribution.
 *      Neither the name of the Intel Corporation nor the names of its
 *  contributors may be used to endorse or promote products derived from
 *  this software without specific prior written permission.
 *
 *  THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
 *  ``AS IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
 *  LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A
 *  PARTICULAR PURPOSE ARE DISCLAIMED.  IN NO EVENT SHALL THE INTEL OR ITS
 *  CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL,
 *  EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
 *  PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR
 *  PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
 *  LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING
 *  NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
 *  SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
 *
 *-----------------------------------------------------------------------------------
 */
/**
 * Control interface of TH7122. Based on CC1000ControlM.nc and modified for the
 * JCUMote which has a TH7122 transceiver with external PA. This file is based on the
 * CC10000ControlM.nc file that is distributed with TinyOS and has been modified by
 * S Willis for the JCUMote TH7122 radio transceiver IC. All code is copyright of
 * TinyOS and is bound by the above copyright message. Sections of modified or added
 * code are indicated by an "SW" comment and are copyright of S Willis.
 *
 * @author Simon Willis
 * @author Philip Buonadonna (original CC1000Control author)
 * @author Jaein Jeong (original CC1000Control author)
 * @modified 17/01/2007
 **/
```

```
includes HPLTimer1;    //SW: Added

module TH7122ControlM {
  provides {
    interface StdControl;             //Standard control interface
    interface TH7122Control;          //Radio transceiver control interface
  }
  uses {
    interface HPLTH7122 as HPLMelexis;       //Hardware presentation layer
  }
}
implementation
{
  norace uint8_t gCurrentParameters[12];     //SW: Modified. Array containing radio IC
settings
  //
  // PRIVATE Module functions
  //

 /**
  * Write current state to the given register
  * Function added by S Willis
  *
  * @param addr TH7122 Register to be written to.
  **/
  void writeMelexis(uint8_t addr) {
       switch (addr) {
              case (TH7122_A):      //Write to register A
                     call
HPLMelexis.write(TH7122_A,(((uint32_t)gCurrentParameters[0]<<16) |
((uint32_t)gCurrentParameters[1]<<8) | ((uint32_t)gCurrentParameters[2])));
                     break;
              case (TH7122_B):      //Write to register B
                     call
HPLMelexis.write(TH7122_B,(((uint32_t)gCurrentParameters[3]<<16) |
((uint32_t)gCurrentParameters[4]<<8) | ((uint32_t)gCurrentParameters[5])));
                     break;
              case (TH7122_C):      //Write to register C
                     call
HPLMelexis.write(TH7122_C,(((uint32_t)gCurrentParameters[6]<<16) |
((uint32_t)gCurrentParameters[7]<<8) | ((uint32_t)gCurrentParameters[8])));
                     break;
              default:              //Write to register D
                     call
HPLMelexis.write(TH7122_D,(((uint32_t)gCurrentParameters[9]<<16) |
((uint32_t)gCurrentParameters[10]<<8) | ((uint32_t)gCurrentParameters[11])));
       }
  }

 /**
  * Write current state to all registers
  * Modified by S Willis
  **/
  void th7122SetFreq() {
       uint8_t i;
       //Write registers A,B,C and D
    for (i=0;i<4;i++) {
       writeMelexis(i);             //SW: Modified. Write to the TH7122
    }
    return;

  }


  //
  // PUBLIC Module Functions
  //

 /**
  * Standard Control initialisation command
  *
  * @return Always returns SUCCESS
  **/
  command result_t StdControl.init() {
    call HPLMelexis.init();   //SW: Modified for TH7122. Initialise the IC
    //SW: Added for TH7122. Set the OPMODE to standby
    gCurrentParameters[1]=gCurrentParameters[1] & ~(0x03<<TH7122_OPMODE);
    //SW: Added for TH7122. Call TunePreset to set the parameters to default freq
    call TH7122Control.tunePreset(TH7122_DEF_PRESET);
    return SUCCESS;
  }

 /**
  * tunePresent command: Set the TH7122 to the default state
  * Considerably modified by SW for TH7122. All code in this command is written by
  * and is copyright of S Willis unless stated otherwise.
  *
  * @param freq Index to defined frequencies (See th7122.h)
  * @return Always returns SUCCESS
  **/
  command result_t TH7122Control.tunePreset(uint8_t freq) { //Line originally in
CC1000Control
    uint8_t state1;   //State of Reg B
    uint8_t state0;   //State of Reg A
    uint8_t temp;
    int i;                    //Line originally in CC1000Control

    //Save the current state, so we restore the OPMODE, LOCKMODE
    state1=gCurrentParameters[1];
    state0=gCurrentParameters[0];
    state1=state1 & (0x03<<TH7122_OPMODE);
    state0=state0 & (0x01<<TH7122_LOCKMODE);
    for (i=0;i < 12;i++) {    //SW: Modified. Line originally in CC1000Control, but
modified by SW
       gCurrentParameters[i]=PRG_RDB(&TH7122_Params[freq][i]);     //SW: Modified.
Line originally in CC1000Control, but modified by SW
    }
    //Clear OPMODE and LOCKMODE
    gCurrentParameters[1]=gCurrentParameters[1] & ~(0x03<<TH7122_OPMODE);
    gCurrentParameters[0]=gCurrentParameters[0] & ~(0x01<<TH7122_LOCKMODE);
    //Restore settings
    temp = gCurrentParameters[0] | state0;
    gCurrentParameters[0]=temp;
    temp = gCurrentParameters[1] | state1;
    gCurrentParameters[1]=temp;

    th7122SetFreq();  //SW: Modified. Line originally in CC1000Control, but modified
by SW
    TOSH_uwait(500);  //Wait switching time
    return SUCCESS;
  }

 /**
  * txMode command: Switch the TH7122 to transmit state
  * Considerably modified by SW for TH7122. All code in this command is written by
  * and is copyright of S Willis unless stated otherwise.
  *
  * @return Always returns SUCCESS
  **/
  async command result_t TH7122Control.txMode() {   //Line originally in CC1000Control
    uint8_t curPower;
    //TX power will have already been set in gCurrentParameters by SetRFPower
    //Change gCurrentParameters to transmit mode and turn on PA
    //Save the current transmit power
    curPower=gCurrentParameters[1] & (0x03<<TH7122_TXPOWER);
    //Clear the bits to be changed
    gCurrentParameters[1]=gCurrentParameters[1] & ~((0x03<<TH7122_OPMODE) |
(0x03<<TH7122_TXPOWER));
    gCurrentParameters[0]=gCurrentParameters[0] | (1<<TH7122_DTAPOL);      //Turn off
external PA
    //Now set them
    gCurrentParameters[1]=gCurrentParameters[1] | (0x02<<TH7122_OPMODE);   //Tx mode
    gCurrentParameters[6]=gCurrentParameters[6] | (0x03<<TH7122_VCOCUR);   //VCOCUR
high
    writeMelexis(TH7122_C);                          //Write to C-register
    writeMelexis(TH7122_A);                          //Write to A-register
    //TOSH_uwait(5000);                              //Wait 5ms for LF to settle
    while (!(call HPLMelexis.getLOCK())) {           //Wait for lock before sending
data
       //call Leds.redOn();                          //Debug
    }
    //call Leds.redOff();                            //Debug
    // Now start the external and internal PAs.
    gCurrentParameters[0]=gCurrentParameters[0] & ~(1<<TH7122_DTAPOL);     //Turn on
external PA
    gCurrentParameters[1]=gCurrentParameters[1] | (curPower);              //Internal
PA
    writeMelexis(TH7122_A);
       //Write to the IC
    TOSH_uwait(5000);                                //Wait for the external PA to
settle
    return SUCCESS;
  }

 /**
  * rxMode command: Switch the TH7122 to receive state
  * Considerably modified by SW for TH7122. All code in this command is written by
  * and is copyright of S Willis unless stated otherwise.
  *
  * @return Always returns SUCCESS
  **/
  async command result_t TH7122Control.rxMode() {    //Line originally in CC1000Control
    //Change gCurrentParameters      to RX mode
    //Clear the bits to be changed
    gCurrentParameters[1]=gCurrentParameters[1] & ~(0x03<<TH7122_OPMODE);
    //Now set them
    gCurrentParameters[1]=gCurrentParameters[1] | (0x01<<TH7122_OPMODE);   //Rx mode
    //VCOCUR low (set to 00 by ANDing)
    gCurrentParameters[6]=gCurrentParameters[6] & ~(0x03<<TH7122_VCOCUR);
    writeMelexis(TH7122_C);                      //Write to C-register
    writeMelexis(TH7122_A);                      //Write to A-register
    TOSH_uwait(1500);                            //Wait 1.5ms for switching time
    return SUCCESS;
  }

 /**
  * biasOff command: Switch the TH7122 to idle state  and turns off the PLL
  * Considerably modified by SW for TH7122. All code in this command is written by
  * and is copyright of S Willis unless stated otherwise.
  *
  * @return Always returns SUCCESS
  **/
  command result_t TH7122Control.biasOff() {         //Line originally in CC1000Control
    //Turn off external PA, set IDLESEL to 0 (turns off PLL)
    gCurrentParameters[0]=gCurrentParameters[0] & ~(1<<TH7122_IDLESEL);
    //switch to idle mode
    gCurrentParameters[1]=gCurrentParameters[1] | (0x03<<TH7122_OPMODE);
    writeMelexis(TH7122_A);                      //Write data
    TOSH_uwait(200);
    return SUCCESS;
  }

 /**
  * biasOn command: Switch the TH7122 to idle state
  * Considerably modified by SW for TH7122. All code in this command is written by
  * and is copyright of S Willis unless stated otherwise.
  *
  * @return Always returns SUCCESS
  **/
  command result_t TH7122Control.biasOn() {  //Line originally in CC1000Control
    //set OPMODE to idle
    gCurrentParameters[1]=gCurrentParameters[1] | (0x03<<TH7122_OPMODE) |
    //Set IDLESEL to power up the PLL
    (1<<TH7122_IDLESEL);
    writeMelexis(TH7122_A);
    TOSH_uwait(200);                                 //Line originally in CC1000Control
    return SUCCESS;
  }

 /**
  * Standard Control Stop command
  * Considerably modified by SW for TH7122. All code in this command is written by
  * and is copyright of S Willis unless stated otherwise.
  *
  * @return Always returns SUCCESS
  **/
  command result_t StdControl.stop() {               //Line originally in CC1000Control
```

```
    // MAIN register to power down mode. Shut everything off
    //switch to standby mode, turn off PA, only RO is active in IDLE
    gCurrentParameters[1]=gCurrentParameters[1] & ~(0x03<<TH7122_OPMODE);
    gCurrentParameters[0]=gCurrentParameters[0] & ~(1<<TH7122_IDLESEL);
    //Write data
    writeMelexis(TH7122_A);

    return SUCCESS;
  }

 /**
  * Standard Control Start command
  * Considerably modified by SW for TH7122. All code in this command is written by
  * and is copyright of S Willis unless stated otherwise.
  *
  * @return Always returns SUCCESS
  **/
  command result_t StdControl.start() {              //Line originally in CC1000Control
    //All code commented out by S Willis. Upper layers start the radio
    //Switch to idle mode, but just wake the xtal oscillator
    //gCurrentParameters[1]=gCurrentParameters[1] | (0x03<<TH7122_OPMODE);
    //gCurrentParameters[0]=gCurrentParameters[0] & ~(1<<TH7122_IDLESEL);
    // Write data
    //writeMelexis(TH7122_A);
    //Wait for oscillator to start
    //TOSH_uwait(1000);
    return SUCCESS;
  }

 /**
  * Set the desired RF power output. Number between 0 and 3 (see TH7122 datasheet)
  * The desired power output will be set on the next transmission
  * Considerably modified by SW for TH7122. All code in this command is written by
  * and is copyright of S Willis unless stated otherwise.
  *
  * @param power Desired power output 0 to 3 (see TH7122 datasheet)
  * @return Always returns SUCCESS
  **/
  command result_t TH7122Control.setRFPower(uint8_t power) {       //Line originally
in CC1000Control
    // RF power level is set on every transmit, so set our state variable
    // and the TX power level will be sent to the radio on the next TX
    //Clear the TXPOWER bits
    gCurrentParameters[1] = gCurrentParameters[1] & ~(0x03<<TH7122_TXPOWER);
    //Set the bits
    gCurrentParameters[1] = gCurrentParameters[1] | (power<<TH7122_TXPOWER);
    return SUCCESS;
  }

 /**
  * Get the current RF power output. Number between 0 and 3 (see TH7122 datasheet)
  * Considerably modified by SW for TH7122. All code in this command is written by
  * and is copyright of S Willis unless stated otherwise.
  *
  * @return The current set RF power level
  **/
  command uint8_t TH7122Control.getRFPower() {              //Line originally in
CC1000Control
    uint32_t state;           //Holds the current state
    //Get the value and mask other bits
    state=gCurrentParameters[1] & (0x03<<TH7122_TXPOWER);
    return ((uint8_t)state>>TH7122_TXPOWER); //Return power (bit shift)
  }

 /**
  * Set the current PLL lock mode. Remain high once locked or constantly reflect the
  * PLL lock state (see TH7122 datasheet)
  * Considerably modified by SW for TH7122. All code in this command is written by
  * and is copyright of S Willis unless stated otherwise.
  *
  * @param Value 0 or 1 representing lock mode
  * @return Always returns SUCCESS
  **/
  command result_t TH7122Control.selectLock(bool Value) {          //Line originally
in CC1000Control
    //Get the A-register, clear the bit we want to set
    gCurrentParameters[0]=gCurrentParameters[0] & ~(1<<TH7122_LOCKMODE);
    //Set the bit
    gCurrentParameters[0]=gCurrentParameters[0] | (Value<<TH7122_LOCKMODE);
    writeMelexis(TH7122_A);
    return SUCCESS;
  }

 /**
  * Get the current state of the TH7122 lock detect pin
  * Considerably modified by SW for TH7122. All code in this command is written by
  * and is copyright of S Willis unless stated otherwise.
  *
  * @return 0 - unlocked, 1 - locked
  **/
  command bool TH7122Control.getLock() {            //Line originally in CC1000Control
    bool retVal;
    retVal = call HPLMelexis.getLOCK();             //Get the current level on the LD
pin
    return retVal;
  }

}
```

HPLTH7122.NC

```
/*                                                                  tab:4
 * "Copyright (c) 2000-2003 The Regents of the University  of California.
 * All rights reserved.
 *
 * Permission to use, copy, modify, and distribute this software and its
 * documentation for any purpose, without fee, and without written agreement is
 * hereby granted, provided that the above copyright notice, the following
 * two paragraphs and the author appear in all copies of this software.
 *
 * IN NO EVENT SHALL THE UNIVERSITY OF CALIFORNIA BE LIABLE TO ANY PARTY FOR
 * DIRECT, INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES ARISING OUT
 * OF THE USE OF THIS SOFTWARE AND ITS DOCUMENTATION, EVEN IF THE UNIVERSITY OF
 * CALIFORNIA HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
 *
 * THE UNIVERSITY OF CALIFORNIA SPECIFICALLY DISCLAIMS ANY WARRANTIES,
 * INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY
 * AND FITNESS FOR A PARTICULAR PURPOSE.  THE SOFTWARE PROVIDED HEREUNDER IS
 * ON AN "AS IS" BASIS, AND THE UNIVERSITY OF CALIFORNIA HAS NO OBLIGATION TO
 * PROVIDE MAINTENANCE, SUPPORT, UPDATES, ENHANCEMENTS, OR MODIFICATIONS."
 *
 * Copyright (c) 2002-2003 Intel Corporation
 * All rights reserved.
 *
 * This file is distributed under the terms in the attached INTEL-LICENSE
 * file. If you do not find these files, copies can be found by writing to
 * Intel Research Berkeley, 2150 Shattuck Avenue, Suite 1300, Berkeley, CA,
 * 94704.  Attention:  Intel License Inquiry.
 *
 *-------------------------------------------------------------------------------------
 */
 /**
 * Hardware Presentation Layer for the TH7122. Based on HPLCC1000.nc and modified for
 * the JCUMote which has a TH7122 transceiver with external PA. This file is based on
 * the HPLCC10000.nc file that is distributed with TinyOS and has been modified by
 * S Willis for the JCUMote TH7122 radio transceiver IC. All code is copyright of
 * TinyOS and is bound by the above copyright message. Sections of modified or added
 * code are indicated by an "SW" comment and are copyright of S Willis.
 *
 * @author Simon Willis
 * @author Jason Hill(original HPLCC1000 author)
 * @author David Gay(original HPLCC1000 author)
 * @author Philip Levis (original HPLCC1000 author)
 * @modified 17/01/2007
 **/

interface HPLTH7122 {
  /**
   * Initialize TH7122 pins
   *
   * @return SUCCESS if successful
   */
  command result_t init();

  /**
   * Transmit 22-bit data for 2-bit register address
   *
   * @param addr 2-bit address of register
   * @param data 22-bit data word to write to register
   * @return SUCCESS if successful
   */
  async command result_t write(uint8_t addr, uint32_t data);

  /**
   * Read a BINARY value from the TH_LD pin
   *
   * @return TRUE or FALSE
   */
  async command bool getLOCK();
}
```

HPLTH7122M.NC

```
/*                                                                    tab:4
 * "Copyright (c) 2000-2003 The Regents of the University  of California.
 * All rights reserved.
 *
 * Permission to use, copy, modify, and distribute this software and its
 * documentation for any purpose, without fee, and without written agreement is
 * hereby granted, provided that the above copyright notice, the following
 * two paragraphs and the author appear in all copies of this software.
 *
 * IN NO EVENT SHALL THE UNIVERSITY OF CALIFORNIA BE LIABLE TO ANY PARTY FOR
 * DIRECT, INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES ARISING OUT
 * OF THE USE OF THIS SOFTWARE AND ITS DOCUMENTATION, EVEN IF THE UNIVERSITY OF
 * CALIFORNIA HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
 *
 * THE UNIVERSITY OF CALIFORNIA SPECIFICALLY DISCLAIMS ANY WARRANTIES,
 * INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY
 * AND FITNESS FOR A PARTICULAR PURPOSE.  THE SOFTWARE PROVIDED HEREUNDER IS
 * ON AN "AS IS" BASIS, AND THE UNIVERSITY OF CALIFORNIA HAS NO OBLIGATION TO
 * PROVIDE MAINTENANCE, SUPPORT, UPDATES, ENHANCEMENTS, OR MODIFICATIONS."
 *
 * Copyright (c) 2002-2003 Intel Corporation
 * All rights reserved.
 *
 * This file is distributed under the terms in the attached INTEL-LICENSE
 * file. If you do not find these files, copies can be found by writing to
 * Intel Research Berkeley, 2150 Shattuck Avenue, Suite 1300, Berkeley, CA,
 * 94704.  Attention:  Intel License Inquiry.
 *
 *-------------------------------------------------------------------------------------
 */
 /**
 * Hardware Presentation Layer for the TH7122. Low level hardware access to the
 * TH7122. The TH7122 does not include a read function so the status of the registers
 * has to stored in memory. Based on HPLCC1000M.nc and modified for the JCUMote which
 * has a TH7122 transceiver with external PA. This file is based on the HPLCC10000M.nc
 * file that is distributed with TinyOS and has been modified by  S Willis for the
 * JCUMote TH7122 radio transceiver IC. All code is copyright of TinyOS and is bound
 * by the above copyright message. Sections of modified or added code are indicated by
 * an "SW" comment and are copyright of S Willis.
 *
 * @author Simon Willis
 * @author Jason Hill (original HPLCC1000M author)
 * @author David Gay (original HPLCC1000M author)
 * @author Philip Levis (original HPLCC1000M author)
 * @modified 17/01/2007
 **/

module HPLTH7122M {           //SW: Modified. For TH7122
  provides {
    interface HPLTH7122;      //SW: Modified. For TH7122
  }
}
implementation
```

```
{

  norace bool bProgMode;             //SW: Added. In programmable user mode.

 /**
  * Intialise the MCU to transceiver interface pins
  *
  * @return Always returns SUCCESS
  **/
  command result_t HPLTH7122.init() {
    bProgMode=FALSE;                 //SW: Added. Set programmable user mode status to
false
    TOSH_MAKE_TH_LD_INPUT();         //SW: Added. Make the lock detect pin an input
    TOSH_MAKE_TH_SDEN_OUTPUT();      //SW: Modified. SDEN to suit TH7122
    TOSH_MAKE_TH_SCLK_OUTPUT();      //SW: Modified. SCLK to suit TH7122
    TOSH_MAKE_TH_SDTA_OUTPUT();      //SW: Modified. SDATA to suit TH7122
    TOSH_MAKE_SPI_OC1C_OUTPUT();     //SW: Added. Output compare to drive Manchester
encoder
    TOSH_CLR_TH_SDEN_PIN();          //SW: Modified. Set SDEN low
    TOSH_CLR_TH_SDTA_PIN();          //SW: Modified. Set SCLK low
    TOSH_CLR_TH_SCLK_PIN();          //SW: Modified. Set SDTA low
    return SUCCESS;
  }

 /**
  * Write to a TH7122 control register
  * Accepts a 2 bit address and 32 bit data, creates an array of ones and zeros
  * for each, and uses a loop counting through the arrays to get consistent timing
  * for the TH7122 radio control interface.  Set SDEN pin high to switch to
  * programmable user mode, then send 2 bits of address, followed by 22 bits of data
  *
  * @param addr 2 bit address of register
  * @param data 22 bits of data
  * @return Always returns SUCCESS
  **/
  async command result_t HPLTH7122.write(uint8_t addr, uint32_t data) {
    uint8_t cnt = 0;

       //SW: Added. Switch to programmable user mode if not already in that mode
       if (!bProgMode) {
               TOSH_SET_TH_SDEN_PIN();              //Logic change on pin SDEN
switches to programmable user mode
               TOSH_SET_TH_SDEN_PIN();
               TOSH_CLR_TH_SDEN_PIN();
               bProgMode=TRUE;
       }

    // address cycle starts here
    for (cnt=0;cnt<2;cnt++)          //SW: Modified. Send addr PDATA msb first
    {
      if (addr&0x02)                 //SW: Modified. Send 2 address bits
        TOSH_SET_TH_SDTA_PIN();
      else
        TOSH_CLR_TH_SDTA_PIN();
      TOSH_CLR_TH_SCLK_PIN();        //SW: Modified. Pulse the SCLK
      TOSH_SET_TH_SCLK_PIN();
      TOSH_SET_TH_SCLK_PIN();
      TOSH_CLR_TH_SCLK_PIN();
      addr <<= 1;                    //bit shift the address
    }

    // data cycle starts here
    for (cnt=0;cnt<22;cnt++)         //SW: Modified. Send data SDATA msb first
    {
      if (data&0x200000)
        TOSH_SET_TH_SDTA_PIN();
      else
        TOSH_CLR_TH_SDTA_PIN();
      TOSH_CLR_TH_SCLK_PIN();        //SW: Modified. Pulse the SCLK
      TOSH_SET_TH_SCLK_PIN();
      TOSH_SET_TH_SCLK_PIN();
      TOSH_CLR_TH_SCLK_PIN();
      data <<= 1;                    //bit shift the data
    }
    TOSH_CLR_TH_SDTA_PIN();          //SW: Modified for TH7122 pins
    TOSH_SET_TH_SDEN_PIN();
    TOSH_CLR_TH_SDEN_PIN();
    return SUCCESS;
  }

 /**
  * Get the current status of the TH7122 lock detect pin
  *
  * @return 0->PLL unlocked, 1->PLL locked
  **/
  async command bool HPLTH7122.getLOCK() {
    bool val;

    val = TOSH_READ_TH_LD_PIN();      //SW: Modified to suit TH7122 pins

    return val;
  }
}
```

MANCHESTERBYTEFIFO.NC

```
/**
 * Manchester encodor/decoder.
 * Uses the SPI to transmit Manchester encoded bytes. The SPI is run in slave mode
 * and the OC1C output compare pin is used to generate a clock pulse to drive the
 * SPI at the desired data rate. The IC1 input capture pin is used for the
 * Manchester decoder. All code in this fule is written by and copyright of S
 * Willis.
 *
 * @author Simon Willis
 * @modified 17/01/2007
 **/

interface ManchesterByteFIFO {

 /**
  * Transmit a byte over the SPI
  * Breaks the byte into 2 halves and transmits LSB first
  *
  * @param data Data to transmit
  * @return True if data was stored to the transmit buffers successfuly
  **/
  async command result_t writeByte(uint8_t data);

 /**
  * Determines if the transmit buffer is empty or not
  *
  * @return 0 if the buffer is full, 1 if the buffer is empty
  **/
  async command result_t isBufBusy();

 /**
  * Get the last received byte (gives compatibility with a standard SPI as used by the
Mica2)
  *
  * @return The last received byte
  **/
  async command uint8_t readByte();

 /**
  * Enable the interrupts. Will generate interrupts when bytes are received or
transmitted
  * Starts the SPI or input capture
  *
  * @return TRUE if interrupts were stated successfully
  **/
  async command result_t enableIntr();

 /**
  * Disable interrupts, turn off output compare and input capture
  *
  * @return Always returns SUCCESS
  **/
  async command result_t disableIntr();

 /**
```

```
  * Initialise slave (compatible with SPI for Mica2)
  * Switch to receive state and enable interrupts
  *
  * @return Always returns SUCCESS
  **/
  async command result_t initSlave();

 /**
  * Switch to transmit mode. Enable interrupts if already on (Mica2 compatible)
  *
  * @return Always returns SUCCESS
  **/
  async command result_t txMode();

 /**
  * Switch to receive mode. Enable interrupts if already on
  *
  * @return Always returns SUCCESS
  **/
  async command result_t rxMode();

 /**
  * Set the data rate
  *
  * @param dataRate Desired datarate (/8 prescaler)
  * @return Always returns SUCCESS
  **/
  async command result_t setDataRate(uint8_t dataRate);

 /**
  * dataReady event. Signalled when a byte has been received or sent.
  *
  * @param data The data that was sent or received.
  * @return success
  **/
  async event result_t dataReady(uint8_t data);

 /**
  * manchesterViolation event. Signalled when an error has occurred in the
  * Manchester decoding.
  *
  * @return success
  */
  async event result_t manchesterViolation();

}
```

MANCHESTERBYTEFIFOM.NC

```
/**
 * Manchester encodor/decoder.
 * Uses the SPI to transmit Manchester encoded bytes. The SPI is run in slave mode
 * and the OC1C output compare pin is used to generate a clock pulse to drive the
 * SPI at the desired data rate. The IC1 input capture pin is used for the
 * Manchester decoder. All code in this fule is written by and copyright of S
 * Willis.
 *
 * @author Simon Willis
 * @modified 17/01/2007
 **/

module ManchesterByteFIFOM {
  provides {
    interface ManchesterByteFIFO;     //Byte level interface
  }
  uses {
    interface PowerManagement;        //MCU power management
    interface Clock16;                //Clock for output compare 1
    interface TimerCapture;           //Input capture for received bytes
    interface Leds;                   //Debugging
  }
}
implementation {
  enum {               //States of encoder/decoder
    INIT,              //Initialisation
    TXSTATE_H,         //ready to send the high byte
    TXSTATE_L,         //ready to send the low byte
    RXSTATE            //Receive state
  };

  enum {               //Receive states
    IDLE,              //Idle
    GETLOCK,           //Get locked onto preamble
    DETECT             //Sychronised, getting data
  };

  const uint8_t BUFFLEN=10;   //Buffer length

  uint8_t FIFOState;          //Current state of the FIFO
  uint8_t manchesterByteH;    //Manchester encoded bytes
  uint8_t manchesterByteL;
  uint8_t outgoingByte;       //The current outgoing byte (buffer)
  uint8_t rxByte;             //The latest received byte
  bool bTxPending;            //Transmission pending
  uint8_t decState;           //State of the decoder
  uint8_t bitCounter;         //Number of bits detected
  uint16_t lastTime;          //Holds the time of the last transition
  uint16_t elapsed;           //Time since last transition
  uint8_t dataIn[2];          //Input data. Use 2 buffers
  bool curBuf;                //Index to current buffer
  uint8_t putter;             //Putter for dataIn buffer
  uint8_t getter;             //Getter for dataIn buffer.
  uint8_t chiptime;           //Half the period of 1 bit (a manchester chip)
  uint8_t bitLag;             //Allows for delay of transition.
  uint8_t gCurrentDataRate;   //Current datarate
  bool bInterruptOn;          //Interrupts are on.
  bool bViolation;            //A Manchester violation has occurred.
  bool centrePos;             //Manchester decoder. Last rising edge in centre of bit
  uint8_t numChips;           //Number of elapsed chips

//Public functions

 /**
  * Transmit a byte over the SPI
  * Breaks the byte into 2 halves and transmits LSB first
  *
  * @param data Data to transmit
  * @return True if data was stored to the transmit buffers successfuly
  **/
  async command result_t ManchesterByteFIFO.writeByte(uint8_t data) {
    //Temporary variables
    uint8_t i, tempH, tempL;
    bool txStateTmp;

    tempH=0;   //Temp buffer for MSB
    tempL=0;   //Temp buffer for LSB
    atomic {
      txStateTmp=bTxPending;          //Save current transmit state
    }
    if (!txStateTmp) {                //If we're not transmitting then continue
      atomic outgoingByte = data;     //Save the outgoing byte for the dataReady signal
        //Machester encoding
        for (i=0; i<4; i++) {
          if (data&0x80) //MSB              1 = 1->0 (2)
            tempH=tempH | (0x2<<(6-(2*i)));
          else                              //0 = 0->1 (1)
            tempH=tempH | (0x1<<(6-(2*i)));
          if (data&0x08) //LSB
            tempL=tempL | (0x2<<(6-(2*i)));  //1 = 1->0
          else
            tempL=tempL | (0x1<<(6-(2*i)));  //0 = 0->1
          data<<=1;   //Bit shift the data
        }
      atomic {
        manchesterByteH = tempH;      //Save the encoded data
        manchesterByteL = tempL;
        bTxPending=TRUE;              //Transmit pending
        if (FIFOState!=RXSTATE)       //Get ready to transmit it (if not in Rx state)
          FIFOState=TXSTATE_H;
      }
      return SUCCESS;
    }//Transmit is still pending
    else
      return FAIL;
    }

 /**
  * Determines if the transmit buffer is empty or not
  *
  * @return 0 if the buffer is full, 1 if the buffer is empty
  **/
  async command result_t ManchesterByteFIFO.isBufBusy() {
    return (!bTxPending);
  }

 /**
  * Get the last received byte (gives compatibility with a standard SPI as used by the
Mica2)
  *
  * @return The last received byte
  **/
  async command uint8_t ManchesterByteFIFO.readByte() {
    return (rxByte);
  }

 /**
  * Enable the interrupts. Will generate interrupts when bytes are received or
transmitted
  * Starts the SPI or input capture
  *
  * @return TRUE if interrupts were stated successfully
  **/
  async command result_t ManchesterByteFIFO.enableIntr() {
    result_t retValue;

    atomic {
      switch(FIFOState) {
        case (RXSTATE):       //Switch to receive mode, turn on interrupts
          outp(0,SPCR);       //Switch off SPI
          outp(0,SPDR);       //Clear any data left in the SPDR
          if ((bViolation) | (decState==IDLE)) {     //If we have had a violation then
restart
            decState=IDLE;    //Initialise state
            bitCounter=0;     //Reset number of bits received
            lastTime=0;
            bViolation=FALSE; //No Manchester violation yet.
            curBuf=0;         //Reset the current buffer
            centrePos=TRUE;   //The first rising edge will be in the centre
            chiptime=(gCurrentDataRate+1)<<1;       //Chiptime is twice the datarate
value.
            bitLag=chiptime>>1;                     //bitlag is 50% of chiptime
          }
          call TimerCapture.setEdge(FALSE);                //Capture on falling edge.
The transmitter inverts data, so by capturing on falling edge, we get around this
          call Clock16.setRate(TIMER1_DEFAULT_INTERVAL,TCLK_CPU_DIV8);     //Start the
clock source (/8 prescaler)
          call TimerCapture.clearPendingInterrupt(); //Clear any pending interrupts
          call TimerCapture.enableEvents();         //Enable interrupts, switch timer
to input capture mode
          bInterruptOn=TRUE;
          retValue=SUCCESS;
          break;
        case (TXSTATE_H):             //Transmit state. Turn on OC clock and SPI
        case (TXSTATE_L):
          //Disable the input capture
          call TimerCapture.disableEvents(); //stop the input capture interrupts
          call Clock16.intDisable();         //Disable the clock interrupts
          TOSH_MAKE_MISO_OUTPUT();
          call Clock16.setIntervalAndScale((uint16_t)gCurrentDataRate, TCLK_CPU_DIV8);
//fire up the clock
          outp(0xC0,SPCR);                   //Turn on the SPI and interrupts
          retValue=SUCCESS;
          bInterruptOn=TRUE;                 //Interupts are ON
          break;
        default:
          bInterruptOn=TRUE;
          retValue=FAIL;
      }
    }
    call PowerManagement.adjustPower();      //Adjust MCU power
    return retValue;                         //Return SUCCESS or FAIL
  }

 /**
  * Disable interrupts, turn off output compare and input capture
  *
  * @return Always returns SUCCESS
  **/
  async command result_t ManchesterByteFIFO.disableIntr() {

    atomic {
      outp(0,SPCR);           //Disable the SPI
      bInterruptOn=FALSE;     //Interrupts are OFF
    }
    call TimerCapture.disableEvents();       //Disable input capture
    call Clock16.intDisable();               //Disable output compare
    call Clock16.setIntervalAndScale(TIMER1_DEFAULT_INTERVAL, TCLK_CPU_OFF);  //Stop
the clock
    return SUCCESS;
  }

 /**
  * Initialise slave (compatible with SPI for Mica2)
  * Switch to receive state and enable interrupts
  *
  * @return Always returns SUCCESS
  **/
  async command result_t ManchesterByteFIFO.initSlave() {
    atomic {
      FIFOState=RXSTATE;
      decState=IDLE;
      gCurrentDataRate=TH7122_DEF_DATARATE;  //Load the default datarate.
      bInterruptOn=TRUE;
      call ManchesterByteFIFO.enableIntr();  //Call enableIntr to start the clock etc.

    }
    return SUCCESS;
  }

 /**
  * Switch to transmit mode. Enable interrupts if already on (Mica2 compatible)
  *
  * @return Always returns SUCCESS
  **/
  async command result_t ManchesterByteFIFO.txMode() {
    uint8_t temp;
    TOSH_MAKE_MISO_OUTPUT();
    call Clock16.intDisable();       //Disable the clock interrupts
    atomic {
      FIFOState=TXSTATE_H;           //Change the state
        if (bInterruptOn)            //Enable interrupts if applicable
          call ManchesterByteFIFO.enableIntr();
        else
          temp=0;
    }
    return SUCCESS;
  }

 /**
  * Switch to receive mode. Enable interrupts if already on
  *
  * @return Always returns SUCCESS
  **/
  async command result_t ManchesterByteFIFO.rxMode() {

    TOSH_MAKE_MISO_INPUT();
    atomic {
      FIFOState=RXSTATE;             //Change the state
      if (bInterruptOn)
        call ManchesterByteFIFO.enableIntr();       //Set up the manchester decoder
    }
    return SUCCESS;
  }
 /**
  * Set the data rate
  *
  * @param dataRate Desired datarate (/8 prescaler)
  * @return Always returns SUCCESS
  **/
  async command result_t ManchesterByteFIFO.setDataRate(uint8_t dataRate) {
    //Set the datarate
    atomic gCurrentDataRate=dataRate;
    return SUCCESS;
  }

//**********************************************************************************
//Event and interrupt handlers

 /**
  * Event handler for clock16.fire. This will fire if we have received rubbish
  *
  * @return Always returns SUCCESS
  **/
  async event result_t Clock16.fire() {
    signal ManchesterByteFIFO.dataReady(0);  //Signal data received, send 0
    call Clock16.intDisable();               //Disable the clock interrupts
    call ManchesterByteFIFO.enableIntr();    //Reinitialise the decoder.
    return SUCCESS;
  }

 /**
  * Interrupt data for SPIF. Signalled when a transfer has been completed on the SPI.
  **/
  TOSH_SIGNAL(SIG_SPI) {
    switch (FIFOState) {
      case TXSTATE_H:                 //Need to transmit the high byte
        outp(manchesterByteH,SPDR);   //Transmit the high byte
        manchesterByteH=0;            //Clear the buffer
        atomic FIFOState=TXSTATE_L;   //ready to transmit the low byte
        break;
      case TXSTATE_L:                 //Need to transmit the low byte
        outp(manchesterByteL,SPDR);   //Transmit the low byte
        manchesterByteL=0;            //Clear the buffer
        atomic {
          bTxPending=FALSE;           //We're ready for the next byte
          FIFOState=TXSTATE_H;
        }
        signal ManchesterByteFIFO.dataReady(outgoingByte);  //signal with the byte
just sent.
/*      break;
      default:
        call ManchesterByteFIFO.enableIntr();               */     //RXMODE? -
Reinitialise.
      }
    }

 /**
  * signalDone task. Called when a byte has been received
  **/
  task void signalDone() {
    uint8_t tempData;
    atomic tempData=dataIn[!curBuf]; //Buffer has already been incremented to next
buffer
    signal ManchesterByteFIFO.dataReady(tempData);   //Signal with data that has been
received
  }

 /**
  * Event handler for input capture
  * Locks onto preamble and then decodes the following data
  *
  * @param time Input capture counter
  **/
  async event void TimerCapture.captured(uint16_t time) {   //interrupt handler
    uint8_t currentRate;

    if (decState==IDLE)       {       //We are in the idle state. 1st bit received
      //call Leds.redOn();            //Debug
      lastTime=time;                  //Record the transition
      decState=GETLOCK;               //Change to the GETLOCK state
    }
    else {                            //2nd bit received. Determine elapsed time
      //call Leds.yellowOn();         //Debug
      if (lastTime>time) {            //timer overflow
        elapsed=0xFFFF-lastTime+time+1;
      }
      else {
        elapsed=time-lastTime;
      }
      lastTime=time;                  //Update lastTime
      if (elapsed < TH7122_MAXRATE) {        //Values in TH7122const corresepond to
chiptime * 2
        //Very short time since last transmission. Likely to be nouise
        //We are going to get rubbish data here, so just set the OC to generate an
interrupt at the point where the byte would end.
        call TimerCapture.disableEvents();   //disable the IC interrupts
        atomic {
          currentRate=gCurrentDataRate;
          bViolation=1;                      //Manchester violation has occurred
        }
        //Wait for the time equivalent to transmitting the rest of the byte
        call Clock16.setIntervalAndScale(((uint16_t)(currentRate<<2))*(8-
(bitCounter>>1)),TCLK_CPU_DIV8);

        call Clock16.setCounter(0);   //Reset the counter
        call Clock16.intEnable();     //Enable the clock interrupts
        return;                       //Done. Return
      }
      //Determine the number of elapsed chips
      //1 chip
      if (elapsed<(chiptime+bitLag)) {
        numChips=1;                   //Record the number of chips
        return;                       //Finish early
      }
      else {
        //2 chips
        if (elapsed<((chiptime<<1) + bitLag))       //Chiptime * 2
          if (numChips==1)            //If we previosuly detected 1 chip, then add it
            numChips=3;
          else
            numChips=2;
        else {
          //3 chips
          if (elapsed<((chiptime*3)+bitLag))
            if (numChips==1)          //If we previosuly detected 1 chip, then add it
              numChips=4;
            else
              numChips=3;
          else {
            //4 chips
            if (elapsed<((chiptime<<2)+bitLag))
              if (numChips==1)        //If we previosuly detected 1 chip, then add it
                numChips=5;
              else
                numChips=4;
            else
              numChips=0xFF;
          }
        }
      }
      switch(decState) {
        //We need to get the lock
        case(GETLOCK):
          //If 4 chips have elapsed, then we have a lock
          if (numChips==4) {
            dataIn[curBuf]=dataIn[curBuf]<<2 | ((uint8_t) 2);      //We have detected
10
            bitCounter+=2;            //2 bits detected
            centrePos=TRUE;           //Rising edge occurred in middle of bit.
            if (bitCounter>=TH7122_MANCHESTER_PREAMBLE)     {//We are locked (detected
preamble)
              decState=DETECT;               //Change state.
              //call Leds.yellowToggle();    //Debug
            }
          }
          else {              //Else reset the bit counter and keep waiting.
            bitCounter=0;
          }
          break;
        case (DETECT):
          //If the last rising edge was in the middle of a bit
          if (centrePos) {
            switch(numChips) {
              case (2):
                //If we have 2 chips and we're in the centre, we have another 0
                dataIn[curBuf]=dataIn[curBuf]<<1;  //Bit shift
                bitCounter++;
                break;
              case (3):
                //If we have 3 chips, then we have a bit change (1)
                dataIn[curBuf]=dataIn[curBuf]<<1 | ((uint8_t)1);
                bitCounter++;
                centrePos=FALSE;  //Currently not at the centre.
                break;
              case(4):
                //If we have 4 chips, then we have a 10
                dataIn[curBuf]=dataIn[curBuf]<<1 | ((uint8_t)1);    //Append the 1
                bitCounter++; //Incremement bit counter
                if (bitCounter==8)    {//If we have a full byte
                  curBuf=!curBuf;     //Change buffers
                  post signalDone();  //Signal it.
                  bitCounter=0;       //Reset bit counter
                }
                dataIn[curBuf]=dataIn[curBuf]<<1;           //Append the 0
                bitCounter++;         //Increment bit counter
                break;
              //Any other number of bits is a manchester error
              default:
                signal ManchesterByteFIFO.manchesterViolation();    //Signal manchester
violation
                //call Leds.redToggle();                    //Debug
                bViolation=TRUE;                            //Violation has occurred
                call ManchesterByteFIFO.enableIntr();       //Initialise the system
again
            }
          }//end if(centrePos)
          else {       //Last edge not in the centre position
            switch(numChips) {
              case(2): //2 chips means we have another 1.
                dataIn[curBuf]=dataIn[curBuf]<<1 | ((uint8_t)1);
                bitCounter++;
                break;
              case(3): //3 chips means we have a 10 and back to the centre of the bit
                dataIn[curBuf]=dataIn[curBuf]<<1 | ((uint8_t)1);    //Add the one
                bitCounter++;
                if (bitCounter==8)    {       //If we have a byte now.
                  curBuf=!curBuf;             //Change buffers
                  post signalDone();          //Signal it.
                  bitCounter=0;               //Reset bit counter
                }
                dataIn[curBuf]=dataIn[curBuf]<<1;    //Add the 0
                bitCounter++;                 //Increment the bit counter
                centrePos=TRUE;               //Currently in the centre of the bit.
                break;
              //Manchester error
              default:
                signal ManchesterByteFIFO.manchesterViolation();
                //call Leds.yellowToggle();                 //Debug
                bViolation=TRUE;                            //Violation has occurred
                call ManchesterByteFIFO.enableIntr();       //Initialise the system
again
            }
          }//End else - !centrePos
        break; //End case(detect)
      }//end switch(decState) */
      //If we have a byte, send it.
      if (bitCounter==8)      {       //If we have a byte now.
        curBuf=!curBuf;               //Change buffers
        post signalDone();            //Signal it.
        bitCounter=0;                 //Reset bit counter
      }
    }//End else(decState!=IDLE)
  }//End inputcapture interrupts

}
```

TH7122CONST.H

```
/*                                                                  tab:4
 *  IMPORTANT: READ BEFORE DOWNLOADING, COPYING, INSTALLING OR USING.  By
 *  downloading, copying, installing or using the software you agree to
 *  this license.  If you do not agree to this license, do not download,
 *  install, copy or use the software.
 *
 *  Intel Open Source License
 *
 *  Copyright (c) 2002 Intel Corporation
 *  All rights reserved.
 *  Redistribution and use in source and binary forms, with or without
 *  modification, are permitted provided that the following conditions are
 *  met:
 *
 *      Redistributions of source code must retain the above copyright
 *  notice, this list of conditions and the following disclaimer.
 *      Redistributions in binary form must reproduce the above copyright
 *  notice, this list of conditions and the following disclaimer in the
 *  documentation and/or other materials provided with the distribution.
 *      Neither the name of the Intel Corporation nor the names of its
 *  contributors may be used to endorse or promote products derived from
 *  this software without specific prior written permission.
 *
 *  THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
 *  ``AS IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
 *  LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A
 *  PARTICULAR PURPOSE ARE DISCLAIMED.  IN NO EVENT SHALL THE INTEL OR ITS
 *  CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL,
 *  EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
 *  PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR
 *  PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
 *  LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING
 *  NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
 *  SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE. *
 *-----------------------------------------------------------------------------------
 */
/**
 * Settings for TH7122 radio transceiver. Defines registers, frequency and data
 * rate settings. This file is based on the CC10000Const.h file that is distributed
 * with TinyOS and has been modified by S Willis for the JCUMote TH7122 radio
 * transceiver IC. All code is copyright of TinyOS and is bound by the above copyright
 * message. Sections of modified or added code are indicated by an "SW" comment and
 * are copyright of S Willis.
 *
 * @author Simon Willis
 * @author Philip Buonadonna (original CC1000Const.h author)
 * @modified 17/01/2007
 **/

#ifndef _TH7122CONST_H
#define _TH7122CONST_H

#include <avr/pgmspace.h>

/* Constants defined for TH7122 */
/* Register addresses */

//SW: Added. All register definitions have been changed to suit the TH7122

#define TH7122_A         0x00
#define TH7122_B               0x01
#define TH7122_C               0x02
#define TH7122_D               0x03

// Register A Bit Posititions
#define TH7122_IDLESEL       5
#define TH7122_DTAPOL        4
```

```
#define TH7122_MODSEL         3
#define TH7122_CPCUR          2
#define TH7122_LOCKMODE               1
#define TH7122_PACTRL         0
#define TH7122_TXPOWER        6
#define TH7122_SETTO1A        5
#define TH7122_LNAGAIN        4
#define TH7122_OPMODE         2
#define TH7122_RRMSB          0
#define TH7122_RRLSB          0

// Register B Bit Positions
#define TH7122_PKDET          5
#define TH7122_SETTO1B        4
#define TH7122_DELPLL         3
#define TH7122_LNAHYST        2
#define TH7122_AFC            1
#define TH7122_OA2            0
#define TH7122_ROMAX          5
#define TH7122_ROMIN          2
#define TH7122_RTMSB          0
#define TH7122_RTLSB          0

// Register C Bit Positions
#define TH7122_LNACTRL        5
#define TH7122_PFDPOL         4
#define TH7122_VCOCUR         2
#define TH7122_BAND           1
#define TH7122_NRMSB          0
#define TH7122_NRMID          0
#define TH7122_NRLSB          0

// Register D Bit Positions
#define TH7122_MODCTRL        5
#define TH7122_LDTM           3
#define TH7122_ERTM           1
#define TH7122_NTMSB          0
#define TH7122_NTMID          0
#define TH7122_NTLSB          0

/*
 * TH7122 Register Parameters Table
 *
 * This table follows the same format order as the TH7122 register
 * set EXCEPT for the last entry in the table which is the
 * CURRENT register value for TX mode.
 *
 * NOTE: To save RAM space, this table resides in program memory (flash).
 * This has two important implications:
 *     1) You can't write to it (duh!)
 *     2) You must read it using the PRG_RDB(addr) macro. IT CANNOT BE ACCESSED AS AN
ORDINARY C ARRAY.
 *
 * Add/remove individual entries below to suit your RF tastes.
 *
 */
#define TH7122_40_750_MHZ            0x00           //SW: Modified. Frequency
definitions changed to suit JCUMote
#define TH7122_40_850_MHZ            0x01
#define TH7122_40_900_MHZ            0x02

#ifdef TH7122_DEFAULT_FREQ
#define TH7122_DEF_PRESET (TH7122_DEFAULT_FREQ)
#endif
#ifdef TH7122_MANUAL_FREQ
#define TH7122_DEF_FREQ (TH7122_MANUAL_FREQ)
#endif

#ifndef TH7122_DEF_PRESET
#define TH7122_DEF_PRESET     (TH7122_40_850_MHZ)    //SW: Modified
#endif

//Data rates are calculated assuming a /8 prescaler on timer 1 (7.3728MHz). For
19200bps datarate we have
//38400bps Manchester encoder datarate. The SPI outputs on each falling edge, so the
OC1C pin must be toggled
```

```
//at twice the Manchester datarate (76.8kHz)->(7.3728e6/8)/76.8e3 -> 12
//SW: Added. All data rate definitions added by S Willis
#define TH7122_19200BPS              12-1
#define TH7122_9600BPS        24-1    //Data rate (assumes a /8 prescaler on timer 1)
#define TH7122_4800BPS        48-1
#define TH7122_2400BPS        96-1
#define TH7122_MAXRATE        (TH7122_19200BPS)

#ifndef TH7122_DEF_DATARATE
#define TH7122_DEF_DATARATE   (TH7122_9600BPS)
#endif

#define TH7122_MANCHESTER_PREAMBLE           8              //SW: Added. Number of
bits detected in preamble to get Manchester lock (max 8)


//#define TH7122_SquelchInit        0x02F8 // 0.90V using the bandgap reference
//#define TH7122_SquelchInit        0x138 (CC1K value)
//#define TH7122_SquelchInit      0x1F7      //(~1.47V -> -90dBm) TH7122 Value
#define TH7122_SquelchInit        0x270      //SW: Added. Measured with antenna on
JCUMote in lab
#define TH7122_SquelchTableSize   9
#define TH7122_MaxRSSISamples     5
#define TH7122_Settling           1
#define TH7122_ValidPrecursor     2
#define TH7122_SquelchIntervalFast 128
#define TH7122_SquelchIntervalSlow 2560
#define TH7122_SquelchCount       30
#define TH7122_SquelchBuffer      0

#define TH7122_LPL_STATES         7

#define TH7122_LPL_PACKET_TIME    16

// duty cycle         max packets        effective throughput
// -----------------  -----------------  -----------------
// 100% duty cycle    42.93 packets/sec  12.364kbps
// 35.5% duty cycle   19.69 packets/sec   5.671kbps
// 11.5% duty cycle    8.64 packets/sec   2.488kbps
// 7.53% duty cycle    6.03 packets/sec   1.737kbps
// 5.61% duty cycle    4.64 packets/sec   1.336kbps
// 2.22% duty cycle    1.94 packets/sec   0.559kbps
// 1.00% duty cycle    0.89 packets/sec   0.258kbps
static const prog_uchar TH7122_LPL_PreambleLength[TH7122_LPL_STATES*2] = {
    0, 8,      //28
    0, 94,      //94
    0, 250,     //250
    0x01, 0x73, //371,
    0x01, 0xEA, //490,
    0x04, 0xBC, //1212
    0x0A, 0x5E  //2654
};

static const prog_uchar TH7122_LPL_SleepTime[TH7122_LPL_STATES*2] = {
    0, 0,       //0
    0, 20,      //20
    0, 85,      //85
    0, 135,     //135
    0, 185,     //185
    0x01, 0xE5, //485
    0x04, 0x3D  //1085
};

static const prog_uchar TH7122_LPL_SleepPreamble[TH7122_LPL_STATES] = {
    0,
    8,
    8,
    8,
    8,
    8,
    8
};

//SW: The below settings (until end of file) have been completely changed by S Willis
to suit the JCUMote
static const prog_uchar TH7122_Params[4][12] = {    //Changed from prog_uchar - SW
```

```
//(0) 40.75 MHz channel
{     // A-register
      //Byte 2
      //Only RO is active in IDLE mode
      ((0<<TH7122_IDLESEL) |
      //Normal input data polarity (0->Fmin)
      (0<<TH7122_DTAPOL) |
      //FSK mode
      (1<<TH7122_MODSEL) |
      //Charge pump output current (260uA default)
      (0<<TH7122_CPCUR) |
      // LD signal is ascertainted once after signal remains in high state (default)
      (0<<TH7122_LOCKMODE) |
      // PA is always on in TX mode (default)
      (1<<TH7122_PACTRL)),
      //Byte 1
      //Output power attenuation (highest power setting-default)
      ((0x03<<TH7122_TXPOWER) | (1<<TH7122_SETTO1A) |
      //LNAGAIN max (default)
      (1<<TH7122_LNAGAIN) |
      //OPMODE - set to standby (default)
      (0x00<<TH7122_OPMODE) |
      //RR - reference divider ratio in RX mode (dec 160, 0xA0).
      //160
      (0x00<<TH7122_RRMSB)),
      //Byte 0
      (0xA0<<TH7122_RRLSB),  //fref=50kHz

      // B-register
      //Byte 2
      //RSSI peak detector outputs via OA1 (default)
      ((0<<TH7122_PKDET) | (1<<TH7122_SETTO1B) |
      //delayed start of PLL (default)
      (1<<TH7122_DELPLL) |
      //Hysteresis on pin GAIN-LNA (not used)
      (1<<TH7122_LNAHYST) |
      //Internal AFC feature on
      (1<<TH7122_AFC) |
      //OA2 disabled (default)
      (0<<TH7122_OA2)),
      //Byte 1
      //Start-up current of reference oscillator 525uA (default)
      ((0x07<<TH7122_ROMAX) |
      //Steady state of reference oscillator 150uA (default)
      (0x02<<TH7122_ROMIN) |
      //Reference divider in TX mode (dec 160, 0xA0)
      //160
      (0x00<<TH7122_RTMSB)),
      //Byte 0
      (0xA0<<TH7122_RTLSB),  //fref=50kHz

      //C-register
      //Byte 2
      //LNA will be controlled by LNAGAIN bit
      ((1<<TH7122_LNACTRL) |
      //negative phase detector polarity (default)
      (0<<TH7122_PFDPOL) |
      //Set VCO to low current (receive)
      (0<<TH7122_VCOCUR) |
      //Set to low frequency band
      (0<<TH7122_BAND) |
      //feedback divider ratio (RX mode) (dec 705, 0x2C1)
      //705 (35.25MHz)
      (0<<TH7122_NRMSB)),
      //Byte 1
      (0x02<<TH7122_NRMID),  //fref=50kHz
      //Byte 0
      (0xC1<<TH7122_NRLSB),  //fref=50kHz

      //D-register
      //Byte 2
      //Modulation is set by MODCTRL register
      ((1<<TH7122_MODCTRL) |
      //Lock condition of the PLL (16 clocks - default)
      (0x01<<TH7122_LDTM) |
      //Unlock condition of the PLL (2 clocks - default)
      (0x00<<TH7122_ERTM) |
      //Feedback divider ratio (TX mode) (dec 815, 0x32F)
      //815 (40.75MHz)
      (0x00<<TH7122_NTMSB)),
      //Byte 1
      (0x03<<TH7122_NTMID),

      //Byte 0
      (0x2F<<TH7122_NTLSB)
},
//(1) 40.85 MHz channel
{     // A-register
      //Byte 2
      //Only RO is active in IDLE mode
      ((0<<TH7122_IDLESEL) |
      //Normal input data polarity (0->Fmin)
      (0<<TH7122_DTAPOL) |
      //FSK mode
      (1<<TH7122_MODSEL) |
      //Charge pump output current (260uA default)
      (0<<TH7122_CPCUR) |
      // LD signal is ascertainted once after signal remains in high state (default)
      (0<<TH7122_LOCKMODE) |
      // PA is always on in TX mode (default)
      (1<<TH7122_PACTRL)),
      //Byte 1
      //Output power attenuation (highest power setting-default)
      ((0x03<<TH7122_TXPOWER) | (1<<TH7122_SETTO1A) |
      //LNAGAIN max (default)
      (1<<TH7122_LNAGAIN) |
      //OPMODE - set to standby (default)
      (0x00<<TH7122_OPMODE) |
      //RR - reference divider ratio in RX mode (dec 160, 0xA0).
      //160
      (0x00<<TH7122_RRMSB)),
      //Byte 0
      (0xA0<<TH7122_RRLSB), //fref=50kHz

      // B-register
      //Byte 2
      //RSSI peak detector outputs via OA1 (default)
      ((0<<TH7122_PKDET) | (1<<TH7122_SETTO1B) |
      //delayed start of PLL (default)
      (1<<TH7122_DELPLL) |
      //Hysteresis on pin GAIN-LNA (not used)
      (1<<TH7122_LNAHYST) |
      //Internal AFC feature on
      (1<<TH7122_AFC) |
      //OA2 disabled (default)
      (0<<TH7122_OA2)),
      //Byte 1
      //Start-up current of reference oscillator 525uA (default)
      ((0x07<<TH7122_ROMAX) |
      //Steady state of reference oscillator 150uA (default)
      (0x02<<TH7122_ROMIN) |
      //Reference divider in TX mode (dec 160, 0xA0)
      //160
      (0x00<<TH7122_RTMSB)),
      //Byte 0
      (0xA0<<TH7122_RTLSB), //fref=50kHz

      //C-register
      //Byte 2
      //LNA will be controlled by LNAGAIN bit
      ((1<<TH7122_LNACTRL) |
      //negative phase detector polarity (default)
      (0<<TH7122_PFDPOL) |
      //Set VCO to low current (receive)
      (0<<TH7122_VCOCUR) |
      //Set to low frequency band
      (0<<TH7122_BAND) |
      //feedback divider ratio (RX mode) (dec 707, 0x2C3)
      //707 (35.35MHz)
      (0<<TH7122_NRMSB)),
      //Byte 1
      (0x02<<TH7122_NRMID), //fref=50kHz
      //Byte 0
      (0xC3<<TH7122_NRLSB),  //fref=50kHz

      //D-register
      //Byte 2
      //Modulation is set by MODCTRL register
      ((1<<TH7122_MODCTRL) |
      //Lock condition of the PLL (16 clocks - default)
      (0x01<<TH7122_LDTM) |
      //Unlock condition of the PLL (2 clocks - default)
      (0x00<<TH7122_ERTM) |
      //Feedback divider ratio (TX mode) (dec 817, 0x331)
      //817 (40.85 MHz)
      (0x00<<TH7122_NTMSB)),
      //Byte 1
      (0x03<<TH7122_NTMID),
      //Byte 0
      (0x31<<TH7122_NTLSB)
},
//(3) 40.9 MHz channel
{     // A-register
      //Byte 2
      //Only RO is active in IDLE mode
      ((0<<TH7122_IDLESEL) |
      //Normal input data polarity (0->Fmin)
      (0<<TH7122_DTAPOL) |
      //FSK mode
      (1<<TH7122_MODSEL) |
      //Charge pump output current (260uA default)
      (0<<TH7122_CPCUR) |
      // LD signal is ascertainted once after signal remains in high state (default)
      (0<<TH7122_LOCKMODE) |
      // PA is always on in TX mode (default)
      (1<<TH7122_PACTRL)),
      //Byte 1
      //Output power attenuation (highest power setting-default)
      ((0x03<<TH7122_TXPOWER) | (1<<TH7122_SETTO1A) |
      //LNAGAIN max (default)
      (1<<TH7122_LNAGAIN) |
      //OPMODE - set to standby (default)
      (0x00<<TH7122_OPMODE) |
      //RR - reference divider ratio in RX mode (dec 160, 0xA0).
      //160
      (0x00<<TH7122_RRMSB)),
      //Byte 0
      (0xA0<<TH7122_RRLSB),  //fref=50kHz

      // B-register
      //Byte 2
      //RSSI peak detector outputs via OA1 (default)
      ((0<<TH7122_PKDET) | (1<<TH7122_SETTO1B) |
      //delayed start of PLL (default)
      (1<<TH7122_DELPLL) |
      //Hysteresis on pin GAIN-LNA (not used)
      (1<<TH7122_LNAHYST) |
      //Internal AFC feature on
      (1<<TH7122_AFC) |
      //OA2 disabled (default)
      (0<<TH7122_OA2)),
      //Byte 1
      //Start-up current of reference oscillator 525uA (default)
      ((0x07<<TH7122_ROMAX) |
      //Steady state of reference oscillator 150uA (default)
      (0x02<<TH7122_ROMIN) |
      //Reference divider in TX mode (dec 160, 0xA0)
      //160
      (0x00<<TH7122_RTMSB)),
      //Byte 0
      (0xA0<<TH7122_RTLSB),  //fref=50kHz

      //C-register
      //Byte 2
      //LNA will be controlled by LNAGAIN bit
      ((1<<TH7122_LNACTRL) |
      //negative phase detector polarity (default)
      (0<<TH7122_PFDPOL) |
        //Set VCO to low current (receive)
        (0<<TH7122_VCOCUR) |
        //Set to low frequency band
        (0<<TH7122_BAND) |
        //feedback divider ratio (RX mode) (dec 708, 0xDD1)
        //708 (35.4MHz)
        (0<<TH7122_NRMSB)),
        //Byte 1
        (0x02<<TH7122_NRMID),  //fref=50kHz
        //Byte 0
        (0xC4<<TH7122_NRLSB),  //fref=50kHz

        //D-register
        //Byte 2
        //Modulation is set by MODCTRL register
        ((1<<TH7122_MODCTRL) |
        //Lock condition of the PLL (16 clocks - default)
        (0x01<<TH7122_LDTM) |
        //Unlock condition of the PLL (2 clocks - default)
        (0x00<<TH7122_ERTM) |
        //Feedback divider ratio (TX mode) (dec 818, 0x332)
        //818 (40.9 MHz)
        (0x00<<TH7122_NTMSB)),
        //Byte 1
        (0x03<<TH7122_NTMID),
        //Byte 0
        (0x32<<TH7122_NTLSB)
  }
};

#endif /* _TH7122CONST_H */
```